Lecture Notes for Chapter 4: Divide and Conquer

The first method we present for solving problems in this book is the divide and conquer method. In this type of algorithm, the main problem is divided into sub-problems that are exactly similar to the main problem but smaller in size.

The first phase of this method, as its name suggests, is to break or divide the problem into sub-problems, and the second phase is to solve smaller problems and then integrate the answers aiming to find the answer to the main problem. The divide and conquer method is not a general solution to all problems and can only be used for problems that are inherently divisible into smaller problems. We use this method when the number of data is large and also the problem can be divided into k sub-problem.

In this case, k is a number between 1 and n. It is necessary, first, to solve the k sub-problem. Finally, we must have a way of combining these sub-problems so that we can solve the main problem. If, after dividing the problem, the sub-problems are still large and unsolvable, we divide them again into several other sub-problems. So problems have to be small enough to be solved by successive divisions.

Many algorithms have a recursive structure. These algorithms need to call themselves one or more to solve problems. By generating smaller sub-problems similar to the main problem and solving them, the answer to the main problem is obtained. In this case, we say that these algorithms use the divide and conquer method.

In summary, the divide and conquer algorithms follow the following three steps.

1. Division: divides the problem into sub-problems.
2. Conquer: Solves sub-problems recursively. (If the sub problems are small enough, it solves them only directly (not recursively).
3. Combination: integrates the solved sub-problems to solve the main problem.

The divide and conquer method is a top-down method. That is, solving a high-level sample of the problem is achieved by going down and solving smaller samples.

The recurrence relation of most algorithms designed by divide and conquer method is as follows:

$$T(n) = \begin{cases} T(1) & \text{n} = 1 \\ aT(\frac{n}{b}) + f(n) & n > 1 \end{cases}$$

where a is the number of sub-problems that must be solved recursively. That is, the number of divisions of the main problem into sub-problems. n is the

input size and b is the number of divisions of the input size, and *f(n)* is the time required to combine.

Example- Suppose algorithm A solves a sample problem by dividing it into four sub-problems of half the size, recursively solves each of these four sub-problems, and then combines their solutions to form the solution of the input problem in O(n log n) time. The recurrence for the algorithm is as follows:

$$a = 4, \quad b = 2, \quad f(n) = nlogn$$

$$T(n) = 4T(\frac{n}{2}) + O(nlogn)$$

1 Exercises for Chapter 3: Divide and Conquer

1.1 Preliminary

1. If $T(n) = aT(n/b) + O(n^d)$ for some constants $a > 0, b > 1$, and $d \geq 0$, prove

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

2. Suppose algorithm A solves a sample problem by dividing it into four sub-problems of half the size, recursively solves each of these four sub-problems, and then combines their solutions to form the solution of the input problem in O(n) time. The recurrence for the algorithm is as follows:

$$T(n) = 4T(\frac{n}{2}) + O(n)$$
$$T(1) = 1$$

a) Solve this recurrence using the Master Theorem.

b) Solve this recurrence using the Recursion Tree Method.

c) Solve this recurrence using the Substitution Method.

3. Suppose algorithm A solves a sample problem of size n by dividing it into two sub-problems of size n - 1, recursively solves each of these two sub-problems, and then combines their solutions to form the solution of the input problem in constant time. The recurrence for the algorithm is as follows:

$$T(n) = 2T(n-1) + c$$
$$T(1) = 1$$

a) Solve this recurrence using the Master Theorem.

b) Solve this recurrence using the Recursion Tree Method.

c) Solve this recurrence using the Substitution Method.

4. Suppose the following three algorithms can be solve a specific problem of size *n*. which one of the three algorithms would you choose to solve the problem? Explain your answer.

- Algorithm *M1* solves problem by dividing it into five subproblems of half the size, recursively solves each of these five sub-problems, and then combines their solutions to form the solution of the input problem in O(n) time (linear time).

- Algorithm *M2* solves problem by dividing it into two subproblems of size *n-1*, recursively solves each of these two sub-problems, and then combines their solutions to form the solution of the input problem in constant time.
- Algorithm *M3* solves problem by dividing it into nine subproblems of size $\frac{n}{3}$, recursively solves each of these nine sub-problems, and then combines their solutions to form the solution of the input problem in quadratic time (n^2).

1.2 Binary Search

5. *Binary Search.* In this search, we divide our array of size n into two parts (by taking a mid) and discard half of our search space at each iteration.

a) Describe an iterative algorithm for binary search and then calculate its time complexity.

b) Describe a recursive algorithm for binary search and then calculate its time complexity.

c) Implement both algorithms in a programming language and compare the speed of both algorithms on arrays of different sizes (e.g., n = 32, 64, 128, 512, 1024, ...).

6. Consider the following array:

$$A = [85\ 80\ 75\ 40\ 27\ 23\ 21\ 20\ 18\ 12\ 11]$$

For x= 18, x=40, x=84, apply binary search algorithm. In a table shows low, high, mid, and A[mid] in each iterative of the algorithm.

7. Consider the following array:

$$A = [-1\ 0\ 3\ 5\ 10\ 23\ 30\ 45]$$

a) How many comparisons does it take using a binary search to find the elements -1, 0, 5, 30?

b) In the array, find the average number of comparisons in both successful and unsuccessful search?

8. For binary search problem, analyze the average number of comparisons when

a) all elements in the array have an equal probability of being searched for.

b) each element in the array have different probability of being searched for.

9. Let n is not necessarily a power of 2. Show that the number of comparisons used by the binary search algorithm is $\lceil \log \rceil$?

10. Does the following binary search algorithm work properly?

```
Function search(A, x, ℓ , r)
comment find x in A[ℓ .r]
if ℓ = r then return (ℓ)
   else
    m := ⌊(ℓ+r) / 2⌋
    if x ≤ A[m]
       then return(search(A, x, ℓ ,m))
       else return(search(A, x, m, r))
```

11. Is the following algorithm for binary search correct? If so, prove it. If not, give an example on which it fails.

```
Function search(A, x, ℓ , r)
comment find x in A[ℓ .r]
if ℓ = r then return(ℓ)
  else
    m := ⌊ℓ + (r − ℓ + 1)/2⌋
    if x ≤ A[m]
       then return(search(A, x, ℓ ,m))
       else return(search(A, x, m+1, r))
```

12. Modify the binary search algorithm so that it splits the input not into two sets of almost-equal sizes, but into two sets of sizes approximately one-third and two-thirds.

13. Find number of rotations in a circularly sorted array. Given a circularly sorted array of integers, find the number of times the array is rotated. Assume there are no duplicates in the array and the rotation is in anti-clockwise direction.

For example: Input: arr = [8, 9, 10, 2 5, 6]

Output: The array is rotated 3 times

We can see that the number of rotations is equal to number of elements before minimum element of the array or index of the minimum element.

a) Describe a linear search algorithm and then calculate its time complexity.

b) Modify binary search algorithm to solve this problem and then calculate its time complexity.

14. Find first or last occurrence of a given number is sorted array. Given a sorted array of integers, find index of first or last occurrence of a given number. If the element is not found in the array, report that as well.

For example: Input: arr = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9]
Target = 5
Output: First occurrence of element 5 is found at index 1
Last occurrence of element 5 is found at index 3

a) Describe a linear search algorithm and then calculate its time complexity.

b) Modify binary search algorithm to solve this problem and then calculate its time complexity.

15. Count occurrence of a number in sorted array with duplicates. Given a sorted array of integers containing duplicates, count occurrence of a number provided. If the element is not found in the array, report that as well.

For example: Input: arr = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9]
Target = 5
Output: Element 5 occurs 3 times

a) Describe a linear search algorithm and then calculate its time complexity.

b) Modify binary search algorithm to solve this problem and then calculate its time complexity.

16. Find smallest missing element from a sorted array. Given a sorted array of distinct non-negative integers, find smallest missing in it.

Input: arr = [0, 1, 2, 6, 9, 11, 15]
Output: The smallest missing element is 3

a) Describe a linear search algorithm and then calculate its time complexity.

b) Modify binary search algorithm to solve this problem and then calculate its time complexity.

17. *Find peak element in an array.* Given an array, find an element in it that is greater than its neighbors.

a) Describe a linear search algorithm and then calculate its time complexity.

b) Modify binary search algorithm to solve this problem and then calculate its time complexity.

18. *Ternary Search.* In this search, we divide our array into three parts (by taking two mid) and discard two-third of our search space at each iteration.

a) Describe an iterative algorithm for ternary search and then calculate its time complexity.

b) Describe a recursive algorithm for ternary search and then calculate its time complexity.

c) Describe a recurrence relation for this algorithm and solve this recurrence relation.

d) Compare the efficiency of ternary and binary search algorithms in terms of time complexity.

19. Given a monotonically increasing function f(x) on the non-negative integers, find the value of x where f(x) becomes positives for the first time. In other words, find a positive integer x such that f(x-1), f(x-2), ... are negative and f(x+1), f(x+2), ... are positive. (Hint: You can solve this problem with the help of binary search algorithm.)

20. If an array is sorted from 1 to 1024 integers. How many comparisons are needed to find number 4 by binary search algorithm?

21. Consider the following array:

$$[-1, 0, 1, 2, 3, 4, 5, 6, 7]$$

What is the average number of comparisons for a successful search?

22. In an array, n numbers are in descending order. If we use binary search to find the number, what is the maximum number of comparisons?

23. Suppose a 200-element array is arranged. What is the worst-case comparison number to find the known element x in array A using binary search?

24. Prove in the binary search the following relations hold.

$$S(n) = (I + n)/n$$

$$U(n) = E/(n + 1)$$

where
S(n)= number of comparisons in case of successful search
U(n)= number of comparisons in case of unsuccessful search
I= internal path length of the binary tree, and
E= external path length of the binary tree.
n is the number of nodes in the binary tree.

25. Considering above question, show the following relations hold.

$$S(n) = \Theta(n)$$

$$S(n) = (1 + \frac{1}{n})U(n) - 1$$

26. Let T(n) denote the run time of the binary search algorithm on an array of size n. Show the recurrence relation for this algorithm is as follows:

$$T(n) = T(n/2) + 1$$

$$T(1) = 1$$

27. Use the binary search technique to devise an algorithm for the problem of finding square-roots of natural numbers: Given an n-bit natural number N, compute $\lceil \sqrt{n} \rceil$ using only O(n) additions and shifts.

1.3 Finding minimum and maximum

28. The following algorithm compute the maximum value in an array A[1..n].

```
Function maximum (n)
      Comment Return max of A[1..n].
1.    if  n ≤ 1  then  return (A[1])
2.    else return (max (maximum (n-1) , A[n]))

Function max (z, y)
      Comment Return largest of x and y.
3.    if x ≥ y then  return (x) else return (y)
```

a) Prove that this recursive algorithm for computing the maximum value in an array A[1..n] is correct.

b) Analyze this algorithm. How many comparisons does it use (that is, how many times is line 2 executed) in the worst case?

29. Consider the following procedure. How many the number of comparisons required in line 5 in terms of n?

```
FindMax (A, i, j)

1   n ← j - i + 1
2   if n=1 then return A[i]
3   m1 = FindMax (A, i , i + ⌊n/2⌋, j)
4   m1 = FindMax (A, i + ⌊n/2⌋, j)
5   if m1< m2
6      then return m2
7      else return m1
```

a) $n-1$

b) n

c) $n-(\lceil \frac{n}{2} \rceil - \lfloor \frac{n}{2} \rfloor)$

d) $\lceil logn \rceil$

30. *Finding minimum and maximum.* Considering the problem of finding minimum and maximum in an array of size n, answer the following questions.

a) Give a naive algorithm to find the maximum element in the array.

b) Give a naive algorithm to find the minimum element in the array.

c) Give a naive algorithm to simultaneously find both the minimum and the maximum in the array.

31. True or False, and Justify. Given an array of integers of size n, using a naive algorithm, the number of comparisons are required to determine the maximum element in the array is $n-1$; and the number of comparisons are required to determine the minimum element in the array is $n-1$. Therefore, the number of comparisons required to simultaneously find the smallest and largest elements is $2n-2$.

32. *Simultaneous finding minimum and maximum.* Design a divide and conquer algorithm to simultaneous find the minimum and the maximum in the array of size n. An application of this problem is in a graphics program. A graphics program may need to scale a set of(x,y) data to fit onto a rectangular display screen or other graphical output device. To do so, the program must first determine the minimum and maximum of each coordinate.

33. The following recursive algorithm is designed to find the maximum element in array S of size n (n is a power of 2).

```
Function maximum(x, y)
comment return maximum in S[x..y]
  if y − x ≤ 1 then return (max(S[x], S[y]))
  else
    max1:=maximum (x, ⌊(x + y)/2⌋)
    max2:=maximum (⌊(x + y)/2⌋ + 1, y)
  return (max(max1, max2))
```

a) Prove the correctness of the algorithm.

b) Derive a recurrence relation for the worst-case number of key comparisons used by this algorithm. Solve this recurrence relation. Explain your answer.

c) Find the time complexity of this algorithm?

34. Consider the problem of finding the minimum number of comparisons required to determine the minimum and the maximum of an integer array with n numbers using divide and conquer approach.

a) Draw recursion tree for following list.

$$[12, 13, 0, -3, 15, 43, 57, 31, 47]$$

b) The minimum number of comparisons required to find the *min* and the *max* in this array is ... .

35. Given an array of integers of size n, if n is even, how many the minimum number of comparisons required to find the minimum and the maximum of the array?

36. Given an array of integers of size n, if n is odd, how many the minimum number of comparisons required to find the minimum and the maximum of the array?

37. If T(n) is the number of comparisons, prove the recurrence relation for finding the minimum number of comparisons required to find the minimum and the maximum of an integer array with n numbers is

$$T(n) = \begin{cases} T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

38. The minimum number of comparisons required to find the minimum and the maximum of an array of size 200 is .... .

39. The following algorithm is designed to find maximum and minimum of the array A[1..n].

```
Procedure maxmin(A)
1. if n is odd then max:=A[n]; min:=A[n]
2.    else max:=−∞; min:=∞
3. for i := 1 to ⌊n/2⌋ do
4.     if A[2i − 1] ≤ A[2i]
5.         then small:=A[2i − 1]; large:=A[2i]
6.     else small:=A[2i]; large:=A[2i − 1]
7.     if small < min then min:=small
8.     if large > max then min:=small
```

a) Apply this algorithm to the array below.

$$[12, 13, 0, -3, 15, 43, 57, 31, 47]$$

b) In this algorithm, how many comparisons are required to find the minimum and the maximum of the array.

c) Is it likely to be faster or slower than the divide-and-conquer algorithm in practice?

Consider the maximum partial sum problem (MPS). This problem is defined as follows. Given an array A[1..n] of integers, find values of i and j with $1 \leq i \leq j \leq n$ such that

$$\sum_{k=i}^{j} A[k]$$

is maximized.

Example: For the array [4,-5,6,7,8,-10,5], the solution to MPS is i = 3 and j = 5 (sum 21).

To help us design an efficient algorithm for the maximum partial sum problem, we consider the *left position l maximal partial sum* problem ($LMPS_l$). This problem consists of finding value j with $l \leq j \leq n$ such that

$$\sum_{k=l}^{j} A[k]$$

is maximized. Similarly, the *right position r maximal partial sum* problem ($RMPS_r$), consists of finding value i with $1 \leq i \leq r$ such that

$$\sum_{k=i}^{r} A[k]$$

is maximized.

Example: For the array [4,-5,6,7,8,-10,5], the solution to e.g. $LMPS_4$ is j = 5 (sum 15) and the solution to $RMPS_7$ is i = 3 (sum 16).

Answer the following questions.

40. Describe O(n) time algorithms for solving $LMPS_l$ and $RMPS_r$ for given l and r.

41. Using an O(n) time algorithm for $LMPS_l$, describe a simple $O(n^2)$ algorithm for solving MPS.

42. Using O(n) time algorithms for $LMPS_l$ and $RMPS_r$, describe an $O(n \log n)$ divide and-conquer algorithm for solving MPS.

43. Design a divide-and-conquer algorithm to compute k^n for $k > 0$ and integer $n \geq 0$.

44. *K'th smallest element in union of two sorted arrays.* Let A and B are two sorted lists of size n and m. We need to find the k-th smallest element in the union of the two sorted lists. Design an algorithm with O(log n + log m) time.

45. *K'th smallest element in unsorted array.* Given a collection of data, the k-th order statistic is the k-th smallest value in the data set. Given an array and a number k where k is smaller than size of array, we need to find the k-th smallest element in the given array. Answer the following questions.

a) One way to solve this problem is to sort and then output the kth element. Explain why this simple algorithm is O(n log n).

b) Design an algorithm which uses Min Heap to find K'th smallest element, and then compute its time complexity.

c) Design an algorithm which uses Max Heap to find K'th smallest element, and then compute its time complexity.

d) Design an algorithm which uses median-of-medians algorithm to find K'th smallest element, and then compute its time complexity.

46. If QuickSort is used as a sorting algorithm in first step then selecting K'th smallest element. In QuickSort, we pick a pivot element, then move the pivot element to its correct position and partition the array around it. The idea is, not to do complete quicksort, but stop at the point where pivot itself is k'th smallest element. Also, not to recur for both left and right sides of pivot, but recur for one of them according to the position of pivot. Show the worst case time complexity of this method is $O(n^2)$, but it works in O(n) on average [Ref].

47. Let A denote an array of size n and integer $k \leq n$. Also, let T(n, k) denote the expected time to find the kth smallest in the array A, and let $T(n) = \max_k T(n, k)$. Show that $T(n) < 4n$.

48. Given array A of size n and integer $k \leq n$, consider the following algorithm to find K'th smallest element in an unsorted array.

1. Divide the array into n/5 parts of size 5 and find the median of each part.
2. Recursively, find the true median of the medians. Call this m.
3. Use m as a pivot to split the array into subarrays LESS and GREATER.
4. Recurse on the appropriate piece.

a) Apply this algorithm to the array below.

$$[12, 13, 0, -3, 15, 43, 57, 31, 47, 20, 1, -5, 42, 14, 15, 8, 100, 32, 85, 41]$$

b) Show this algorithm makes O(n) comparisons to find the kth smallest in an array of size n.

49. Let T(n) be maximum number of comparisons to find the kth smallest in an array of size n. Show

$$T(n) \leq cn + T(n/5) + T(7n/10)$$

50. Show $n + \lceil \log n \rceil - 2$ comparisons are sufficient to find the second-largest of n elements.

51. Prove the lower bound for finding kth smallest element in an array of size n (n > 1) is

$$n + (k-1)\lceil \log(\frac{n}{k-1}) \rceil - k$$

1.4 Greatest Common Divisor (gcd)

52. Let a and b be two positive integers. The *greatest common divisor* (gcd) of a and b is the largest integer which divides them both.

a) Check whether the following recursive relation can calculate the gcd correctly?

$$gcd(a,b) = \begin{cases} 2gcd(a/2, b/2) & \text{if } a, b \text{ are even} \\ gcd(a, b/2 & \text{if } a \text{ is odd, } b \text{ is even} \\ gcd((a-b)/2, b) & \text{if } a, b \text{ are odd} \end{cases}$$

b) Describe a recursive algorithm for gcd(a, b).

c) Let a and b are n-bit integers. Compare the algorithm described in (b) to Euclid's algorithm.

53. Explain how we can use the *gcd* function described in exercise 52 for more than two arguments. Give a recursive algorithm for this problem. (Hint: $gcd(a_0, a_1, ..., a_n) = gcd(a_0, gcd(a_1, a_2, ..., a_n)))$

54. Prove for any nonnegative integer a and any positive integer b,

$$gcd(a,b) = gcd(b, a \ mod \ b).$$

55. Let *lcm* denotes the least common multiple of the n integers. Explain how we can use *gcd* as a subroutine to compute $lcm(a_1, a_2, ..., a_n)$.

1.5 Mergesort

56. Let A and B are two sorted arrays of size n and m, respectively. The following algorithm is used to merge these to arrays and form a sorted array C of size p (where p = n + m).

```
void merge (int A[], int n, int B[], int m, int C[], int p)
{
  int i = 0, j = 0, k;
  for(k = 0; i < n && j < m; k++)
    if(A[i] < B[j])
      C[k] = A[i++];
    else
      C[k] = B[j++];
  while(i < n)
    C[k++] = A[i++];
  while(j < m)
    C[k++] = B[j++];
}
```

a) Using the merge algorithm, merge the following arrays.

$$A = [3, 5, 7, 8, 10], \quad B = [1, 2, 6, 12, 20]$$

b) Provide two sorted arrays A and B that require a maximum number of comparisons to merge (worst case happens)?

c) Provide two sorted arrays A and B that require a minimum number of comparisons to merge (best case happens).

d) How many comparisons would be required to merge two sorted arrays of lengths m and n in the worst case and the best case.

57. Suppose we have several sorted arrays. Give two multi-merge algorithms for merging them.

58. Let A and B are two sorted arrays so that the number of elements of array A is much less than the elements of array B. Provide an efficient algorithm for merging the two arrays. (Hint: use binary search algorithm.)

59. For the merge sort algorithm:

a) Describe an iterative algorithm for merge sort.

b) Describe a recursive algorithm for merge sort.

c) Implement both algorithms in a programming language and compare the speed of both algorithms on arrays of different sizes (e.g., n = 32, 64, 128, 512, 1024, ...).

d) When does the best-case of merge sort occur?

e) When does the worst-case of merge sort occur?

60. Consider the following merge sort algorithm.

```
Procedure mergeSort(low, high, A)
Comment A[low..high] is an array
  if (low < high)
    then  mid = ⌊(low + high)/2⌋
          mergeSort(low, mid, A)
          mergeSort(mid + 1, high, A)
          merge(low, mid, high, A)
```

Merge function combines the two sorted subarrays A[low .. mid] and A[mid + 1 .. high] into a sorted sequence. For the following array

$$310, 285, 179, 625, 351, 423, 861, 254, 450, 520$$

a) How many calls (mergeSort) are made?

b) How many calls are made for merge function?

c) What is the maximum number of comparisons between array elements?

61. Draw the recurrence tree for the following array.

$$20, 10, 7, 15$$

62. Let T(n) denotes the run time of merge sort on an array of size n. Show the recurrence relation of this algorithm is as follows:

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n - 1$$

63. Show the exact solution for the recurrence relation for merge sort is as follows:

$$T(n) = nlogn - (n - 1)$$

64. Solve the recurrence relation for merge sort using the Master Theorem, Recursion Tree Method, and the Substitution Method.

65. For the merge sort algorithm, justify:

a) Let T(n) denotes the number of recursive calls. We have

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1, \quad T(1) = 1$$

b) Let T(n) denotes the number of merge function calls in the merge sort algorithm. We have

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1, \quad T(1) = 0$$

c) Let T(n) denotes the number of comparisons in the merge function. We have

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n - 1, \quad T(1) = 0$$

66. We use the merge sort algorithm for sorting an array of size 10:

a) Get the number of recursive calls.

b) Get the number of merge function calls in the merge sort algorithm.

c) Get the number of comparisons in the merge function.

67. Apply merge sort to sort the list G, F, O, B, D, A, E in alphabetical order. Draw the tree of the recursive calls made.

68. Explain why for the following description, among sorting algorithms merge sort algorithm is preferred.

You have to sort 1 GB of data with only 100 MB of available main memory.

69. The algorithm Merge-1 is presented for merging the two arrays A and B with n and m elements, respectively. The result is stored in sorted array C of length m + n. The array index starts from 1. For which of the following statements does this algorithm work correctly instead of conditional statement?

```
Merge-1 (A, B, n, m)

1   i ← 1; j ← 1; k ← 0
2   while conditional statement
3         do k ← k + 1
4            if j > m or ((i ≤ n) and (A[i] ≤ B[j]))
5               then C[k] ← A[i]
6                    i ← i + 1
7               else C[k] ← B[j]
8                    j = j + 1
```

a) (i < n and j < m)

b) (i ≤ n and j ≤ m)

c) (i ≤ n or j ≤ m)

d) (i < n or j < m)

70. The algorithm Merge-2 is presented for merging the two arrays A and B with n and m elements, respectively. The result is stored in array C of length m + n. The array index starts from 1. For which of the following statements does this algorithm work correctly instead of conditional statement?

```
Merge-2 (A, B, n, m)

1   i ← 1; j ← 1; k ← 0
2   while  i  ≤ n or j ≤ m
3         do k ←  k + 1
4            if conditional statement
5              then if A[i] ← B[j]
6                        then  C[k] ← A[j]
7                              i ←  i + 1
8                        else  C[k] ← B[j]
9                              j ←  j + 1
10           else  if i >n
11                     then C[k] ← A[i]
12                          i ← i + 1
13                else C[k] ← B[j]
14                     j = j + 1
```

a) (i < n and j < m)

b) (i ≤ n and j ≤ m)

c) (i ≤ n or j ≤ m)

d) (i < n or j < m)

71. The algorithm Merge-3 is presented for merging the two arrays A and B with n and m elements, respectively. The result is stored in array C of length m + n. The array index starts from 1. When does this algorithm work?

```
Merge-3 (A, B, n, m)

1   i ← 1; j ← 1; k ← 0
2   while  i  ≤ n or j ≤ m
3         do if j > m or A[i] ≤ B[j]
4               then if C[k] ← A[j]
5                       i ←  i + 1
6                    else  C[k] ← B[j]
7                          j ←  j + 1
8            k = k + 1
```

a) It always works right.

b) It never works properly.

c) B[m] < A[n]

d) $\forall i$ A[i] < B[1]

72. To sort array S of length n containing real numbers, the following algorithm is suggested:

a) If n $\leq$ 2, sort it in a simple way (by comparison).

b) If n $\geq$ 3, divide S (left and right) by sub-arrays A, B, and C so that the sizes A and B are $\lceil \frac{n}{3} \rceil$ and $\lfloor \frac{n}{3} \rfloor$, respectively.

c) Recursively sort the sub-arrays A and B.

d) Recursively sort the sub-arrays B and C.

e) Recursively sort the sub-arrays A and B.

Does this algorithm always work correctly?

73. True or False? Merge sort outperforms heap sort in most of the practical situations.

74. Assume that a merge sort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes?

a) 256

b) 512

c) 1024

d) 2048

75. Which sorting algorithm is the best sorting method to use for an array with more than ten million elements? Merge sort or quicksort?

76. Write is the auxiliary space complexity of merge sort.

77. Suppose that in the merge sort, the list is divided into four equal parts at each step, and then in the merge step, these four lists are merged. What is the complexity of the algorithm?

78. Suppose that in the merge sort, we divide the list by 9 to 1 at each step. That is, one part is 9 times the other, and then in the merge phase, the two lists are merged. What is the complexity of the algorithm?

79. If in merge sort, insertion sort is used to sort lists with fewer than 20 elements, what will be the time complexity of the algorithm?

80. *A k-way merge operation.* Assume we have k sorted arrays of size n, and our aim is to merge them into a single sorted array of size kn.

a) Using the merge procedure, merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of k and n?

b) Using divide and conquer technique, devise an efficient algorithm to this problem.

81. Let A[1..n] be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A.

a) List the five inversions of the array $< 3, 5, 10, 7, 1 >$.

b) What array with elements from the set $\{1, 2, ..., n\}$ has the most inversions? How many does it have?

c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

d) Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \log n)$ worst-case time. (Hint: Modify merge sort.)

1.6 Quicksort

82. Quicksort is the widely used sorting algorithm. In Quicksort, first pick a pivot element and then partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element. Picking a good pivot is necessary for the fast implementation of quicksort. Some of the ways of choosing a pivot are as follows:

- Pivot can either be the first element or the last element of the given array.
- Pivot can be random, i.e. select the random pivot from the given array.
- Select median as the pivot element.

In the following function, the first element is selected as pivot.

```
Partition(A[1..n], int low, int high, int & pivot)
  right = high
  left = low + 1
  pivot = A[low]
  pivotIndex = low
  while(left < right)
  {
    while(A[left] < pivot)
      left = left + 1
    while(A[right] ≥ pivot)
      right = right − 1
    if(left < right)
      swap(A[left], A[right])
  }
  swap(A[pivotIndex], A[right])
```

Partition the following arrays using the *Partition* function.

a) $55, 11, 66, 10, 99, 22, 88, 33, 77, 44$

b) $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

c) $10, 9, 8, 7, 6, 5, 4, 3, 2, 1$

83. One way to improve the QUICKSORT is to partition around a pivot that is chosen more carefully than by picking a random element from the subarray. One common approach for computing *pivot* is called median-of-three: where *pivot* is chosen as the median of the first, last, and middle elements of the array A; i.e. median(A[0], A[(n - 1)/2], A[n - 1]). Modify the *Partition* function to support the median-of-three.

Perform the partitioning step of Quicksort on the following array, where the pivot is chosen using the median-of-three heuristic.

a) $55, 11, 66, 10, 99, 22, 88, 33, 77, 44$

b) $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

c) $10, 9, 8, 7, 6, 5, 4, 3, 2, 1$

84. Consider the following Quicksort algorithm.

```
QUICKSORT(A[], low, high)
  if low < high
    q = Partition(A, low, high)
    QUICKSORT(A, low, q − 1)
    QUICKSORT(A, q + 1, high)
```

Sort the following arrays using QuickSort algorithm (show the steps):

a) $55, 11, 66, 10, 99, 22, 88, 33, 77, 44$

b) $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

c) $10, 9, 8, 7, 6, 5, 4, 3, 2, 1$

85. For Quicksort algorithm:

a) When does the best-case of Quicksort occur? Show that its best-case running time on an array of size n is $\Theta(nlogn)$.

b) When does the worst-case of Quicksort occur? Show that its worst-case running time on an array of size n is $\Theta(n^2)$.

c) Show that its expected running time satisfies the recurrence relation

$$T(n) \leq O(n) + \frac{1}{n}\sum_{i=1}^{n-1}(T(i) + T(n-i))$$

Then, show that the solution to this recurrence is *O(n log n).*

86. Apply Quicksort to sort the list *S, A, Y, M, Y, L, S* in alphabetical order. Draw the tree of the recursive calls made.

87. True or False and Justify? The running time of the Quick Sort algorithm depends on how the *partition()* method will split the input array.

88. True or False and Justify? The QuickSort algorithm performs the worst when the pivot is the largest or smallest value in the array.

89. True or False and Justify? Is Quicksort a stable sorting algorithm?

90. Consider the following variant of quick sort. The values to be sorted are in an array S[1..n].

```
1.   Procedure quicksort (ℓ.r)
2.     comment sort S (ℓ..r)
3.     i := ℓ; j:= r
       a := some element from S [ℓ..r]
4.     repeat
5.       while S[i] < a do i := i + 1
6.       while S[j] > a do j := j + 1
7.       if i ≤ j then
8.         swap S[i] and S[j]
9.         i := i + 1; j:= j − 1
10.    until i > j
11.    if ℓ < j then quicksort (ℓ,j)
12.    if i < r then quicksort (i, r)
```

a) Prove that this variant of quicksort is correct.

b) Analyze that variant of quicksort. How many comparisons does it use (that is, how many times are lines 5 and 6 executed) in the worst case?

91. What is the running time of quicksort when all elements of array A have the same value?

92. Show that the running time of quicksort is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

93. There are several schemes that can be used in quicksort to partition the array. Consider Hoare's and Lomuto partition schemes:

a) Using an example, explain how Hoare's and Lomuto's partition schemes work?

b) Write high-level version of Hoare's quicksort algorithm.

c) Write high-level version of Lomuto's quicksort algorithm.

d) Implement Hoare's and Lomuto partition schemes in quicksort and compare them performance.

94. Banks often record transactions on an account in order of the times of the transactions, but many people like to receive their bank statements with checks listed in order by check number. People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch. The problem of converting time-of-transaction ordering to check-number ordering is therefore the problem of sorting almost-sorted input. Argue that the procedure INSERTION-SORT would tend to beat the procedure QUICKSORT on this problem [Ref].

95. True or False and Justify? For a randomized algorithm, we analyze the average-case, not worst-case.

96. Given an array of positive and negative numbers, we would like to rearrange them such that all negative numbers appear before all the positive numbers in the array.

a) Use an auxiliary array to rearrange the numbers in the array.

b) Use partition process of QuickSort to rearrange the numbers.

97. Let T(n) be the worst-case time for the procedure QUICKSORT on an input of size n. Show

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

where the parameter q ranges from 0 to n - 1 because the procedure PARTITION produces two subproblems with total size n - 1.

98. The running time of quicksort can be improved in practice by taking advantage of the fast running time of insertion sort when its input is "nearly" sorted. When quicksort is called on a subarray with fewer than k elements, let it simply return without sorting the subarray. After the top-level call to quicksort returns, run insertion sort on the entire array to finish the sorting process. Argue that this sorting algorithm runs in O(nk + n log(n/k)) expected time. How should k be picked, both in theory and in practice? [Ref]

99. Suppose quicksort were to always pivot on the $\lceil \frac{n}{3} \rceil$th smallest value. How many comparisons would be made then in the worst case?

100. The following is an implementation of quicksort on an array S[1..n].

```
1. Procedure quicksort(ℓ , r)
2. comment sort S[...r]
3. i := ℓ; j := r;
4. a := some element from S[ℓ..r];
5. repeat
6.     while S[i] < a do i := i + 1;
7.     while S[j] > a do j := j − 1;
8.     if i ≤ j then
9.     swap S[i] and S[j];
10.    i := i + 1; j := j − 1;
11. until i > j;
12. if ℓ < j then quicksort(ℓ, j);
13. if i < r then quicksort(i, r);
```

a) How many number of comparisons needed to sort n values in the worst case?

b) How many number of comparisons needed to sort n values on average?

c) Suppose the preceding algorithm pivots on the middle value, that is, line 4 is replaced by a:= $S[\lfloor(\ell + r)/2\rfloor]$. Give an input of 8 values for which quicksort exhibits its worst-case behavior.

d) Suppose the above algorithm pivots on the middle value, that is, line 4 is replaced by a:= $S[\lfloor(\ell + r)/2\rfloor]$. Give an input of n values for which quicksort exhibits its worst-case behavior.

101. In a new type of quicksort algorithm, to select the pivot from the n elements, we select the first $2\sqrt{n} + 1$ elements of the array and arrange them with a simple algorithm such as insertion sort, and then we select the middle element of them as the pivot of the algorithm. The rest of the algorithm is done in the classic way. Show the worst execution time of this algorithm with the

recursive relation.

102. For the quicksort:

a) Are arrays made up of all equal elements the worst-case input, the best-case input, or neither?

b) Are strictly decreasing arrays the worst-case input, the best-case input, or neither?

c) Assume that the algorithm uses the median-of-three pivot selection. Is the ascending array the worst-case input, the best-case input, or neither? Answer the same question for decreasing array.

d) Estimate how many times faster quicksort will sort an array of one million random numbers than insertion sort.

e) True or False and Explain? There are arrays that are sorted faster by insertion sort than by quicksort.

103. Show that any array of integers x[1..n] can be sorted in O(n + M) time, where

$$M = \max_i x_i - \min_i x_i$$

For small M, this is linear time: why doesn't the $\Omega(n \log n)$ lower bound apply in this case?

104. *Randomized-Select, or QuickSelect for finding K'th Smallest Element in Unsorted Array.* We want to select K'th smallest element using Quicksort. To this end, after the partitioning step, we choose the subarray that contains K'th smallest element just by looking at their sizes. This is repeated recursively to find the desired element. Let "LESS" and "GREATER" subarrys indicate the elements less than the pivot and elements greater than the pivot, respectively. The following algorithm is used for this aim.

QuickSelect: Given array A[1..n] and integer $m \leq n$,

1. Choose a random pivot p from A.
2. Use partition procedure in Quicksort to divide array A into subarrays LESS and GREATER by comparing each element to pivot p. Simultaneously count the number of elements (Q) entered into LESS.
3. (a) If $Q = m$ - *1*, then output p
   (b) If $Q > m$ - *1*, output QuickSelect(LESS, m).
   (c) If $Q < m$ - *1*, output QuickSelect(GREATER, m - Q - *1*)

a) Implement this algorithm and apply it on an array of size 547 to find 281'th smallest element.

b) Show the number of comparisons for QuickSelect is O(n).

105. In the finding the k smallest element of the array A [1..n], we first divide all the elements into s-parts (except possibly the last part). We get the median of each part and then we find the median of medians recursively. Select this element as the pivot element and apply the partition algorithm to the array elements, then run the same algorithm recursively (and for another k) on one of the parts to find the element. Write the recurrence relation for algorithm.

106. *Stooge sort.* This algorithm is a recursive sorting algorithm and was presented by professors Howard, Fine, and Howard.

a) Write stooge sort algorithm, then apply it on an array.

b) Write the recurrence relation of the stooge sort algorithm.

c) Write the running time of the stooge sort algorithm (worst-, average-, and best- cases).

1.7 Finding the median

107. Let A denote an array of size n containing distinct numbers and k is a positive integer $k \leq n$. Give an algorithm to find the k numbers in A that are closest to the median of A.

108. Let A[1..n] and B[1..n] are two sorted arrays. Give an algorithm to find the median of all 2n elements in arrays A and B. Analyze its running time.

109. The median of a set of n values is the $\lceil \frac{n}{2} \rceil$th smallest value. Suppose quicksort were to always pivot on the median value. How many comparisons would be made then in the worst case?

1.8 Integer multiplication

110. Assume n is a power of 2. We partition two numbers u and v into lower and upper digits as $u = x \times 10^{n/2} + y$ and $v = w \times 10^{n/2} + z$. Thus, the product becomes $A \times 10^n + (B + C) \times 10^{n/2} + D$, where $A = x \times w, B = y \times w, C = x \times z$ *and* $D = y \times z$. Consider the following recursive algorithm to multiply two numbers u and v.

Multiply(u, v):

(1) Assume n = length(u) = length(v), can pad 0's for shorter number

(2) if $length(u)$ *and* $length(v) = 1$ then return $u \times v$

(3) Partition u,v into $u = x \times 10^{n/2} + y$ and $v = w \times 10^{n/2} + z$

(4) A = Multiply(x, w)

(5) B = Multiply(y, w)

(6) C = Multiply(x, z)

(7) D = Multiply(y, z)

(8) Return $A \times 10^n + (B + C) \times 10^{n/2} + D$

Answer the following questions.

a) Let T(n) be the running time to multiply two n-digit numbers, u and v. Show
$$T(n) = 4T(\frac{n}{2}) + O(n)$$

b) Analyze the Multiply(u, v) algorithm running time.

c) Calculate the multiplication of two numbers 4301×2021 using the above algorithm.

111. Consider the following recursive algorithm to multiply two numbers u and v.

Multiply(u, v):

(1) Assume n = length(u) = length(v), can pad 0's for shorter number

(2) if $length(u) \geq 1$ then return $u \times v$

(3) Partition u,v into $u = x \times 10^{n/2} + y$ and $v = w \times 10^{n/2} + z$

(4) A = Multiply(x, w)

(5) B = Multiply(y, w)

(6) C = Multiply(x, z)

(7) D = Multiply(y, z)

(8) Return $A \times 10^n + (B + C) \times 10^{n/2} + D$

Multiply 2345 with 678 using the above algorithm.

112. Consider the following recursive algorithm to multiply two numbers u and v.

Multiply(u, v):

(1) Assume n = length(u) = length(v), can pad 0's for shorter number

(2) if $length(u) \geq 1$ then return $u \times v$

(3) Partition u,v into $u = x \times 10^{n/2} + y$ and $v = w \times 10^{n/2} + z$

(4) A = Multiply(x, w)

(5) B = Multiply(y, z)

(6) C = Multiply(x + y, z + w)

(7) Return $A \times 10^n + (C - A - B) \times 10^{n/2} + B$

Let T(n) be the running time to multiply two n-digit numbers, u and v. Show

$$T(n) = 3T(\frac{n}{2}) + O(n)$$

113. Similar to the standard integer multiplication using divide and conquer technique, we would like to design an algorithm for performing integer multiplication. Let x and y are two integer numbers. Instead of breaking the inputs x and y into two parts, we break them into three parts. Suppose x and y have n bits, where n is a power of 3. Break x into three parts, *a, b, c,* each with $\frac{n}{3}$ bits. Break y into three parts, *d, e, f,* each with $\frac{n}{3}$ bits [Ref]. Then,

$$xy = ad2^{4n/3} + (ae + bd)2^n + (af + cd + be)2^{2n/3} + (bf + ce)2^{n/3} + cf$$

We have:
r_1 := ad
r_2 := (a + b)(d + e)
r_3 := be
r_4 := (a + c)(d + f)
r_5 := cf
r_6 := (b + c)(e + f)

$$z := r_1 2^{4n/3} + (r_2 - r_1 - r_3)2^n + (r_3 + r_4 - r_1 - r_5)2^{2n/3} + (r_6 - r_3 - r_5)2^{n/3} + r_5$$

a) Show that z = xy.

b) Compute the multiplication of two numbers 4301 × 2021 by this divide and conquer algorithm.

c) Show that the running time of this algorithm is $O(n^{1.63})$.

114. In the large-integer multiplication algorithm, explain why we did not include multiplications by 10^n in the multiplication count.

115. Let $A(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$ and $B(x) = b_0 + b_1x + b_2x^2 + ... + b_nx^n$ are two polynomials. To compute the multiplication $C(x) = A(x) \times B(x)$:

a) Write a naive algorithm to compute this multiplication.

b) Give two divide and conquer algorithms to compute this multiplication.

116. Let $A = a^x + b$ and $B = c^x + d$ are two polynomials. Explain how we can multiply $A \times B$ using only three multiplications. (Hint: One of multiplications is $(a + b)(c + d)$.)

117. Let A and B are two polynomials of degree-bound n. Describe a divide and conquer algorithm that run in time $\Theta(n^{\log 3})$ so that it divide the input polynomial coefficients into a high half and a low half.

118. Let A and B are two polynomials of degree-bound n. Describe a divide and conquer algorithm that run in time $\Theta(n^{\log 3})$ so that it divide them according to whether their index is even or odd.

119. Let A and B be two n-bit integers:

a) Explain how we can use divide and conquer integer multiplication algorithm to multiply A and B.

b) Show that two n-bit integers can be multiplied in $O(n^{\log 3})$ steps, where each step operates on at most a constant number of 1-bit values.

c) Multiply the two binary integers 11110011 and 10001010 using the algorithm described in (a).

1.9 Matrix multiplication

120. Consider the following matrix multiplication algorithm.

```
Procedure matmultiply (Y, Z, n);
        Comment multiplies n × n matrices Y Z
1.      for i := 1 to n do
2.          for j := 1 to n do
3.              X [i, j] := 0;
4.              for k := 1 to n do
5.                  X [i, j] := X[i, j] + Y[i, k] . Z[k, j];
6.      return (X)
```

a) Prove that this matrix multiplication algorithm is correct.

b) Analyze the matrix multiplication algorithm. How many multiplications does it use (that is, how many times is line 5 executed) in the worst case?

121. Let us assume that n is a power of 2 in each of the $n \times n$ matrices for A and B. This simplifying assumption allows us to break a big $n \times n$ matrix into smaller blocks or quadrants of size $n/2 \times n/2$. Let A and B be two square $n \times n$ matrices, where n is even. Let A_{11}, A_{12}, A_{21}, and A_{22} represent the four $n/2 \times n/2$ submatrices of A that correspond to its four quadrants. For example, A_{11} consists of rows 1 through n/2 of A whose entries are restricted to columns 1 through n/2. Similarly, A_{12} consists of rows 1 through n/2 of A whose entries are restricted to columns n/2 + 1 through n. Finally, A_{21} and A_{22} represent the bottom half of A. Thus, A can be written as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad C = A \times B = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

The following divide and conquer algorithm is used to multiply the two matrices A and B.

MMult(A, B, n):

(1) If n = 1 Output A × B

(2) Else

(3) Compute A11, B11, ..., A22, B22

(4) X1 ← MMult(A11, B11, n/2)

(5) X2 ← MMult(A12, B21, n/2)

(6) X3 ← MMult(A11, B12, n/2)

(7) X4 ← MMult(A12, B22, n/2)

(8) X5 ← MMult(A21, B11, n/2)

(9) X6 ← MMult(A22, B21, n/2)

(10) X7 ← MMult(A21, B12, n/2)

(11) X8 ← MMult(A22, B22, n/2)

(12) C11 ← X1 + X2

(13) C12 ← X3 + X4

(14) C21 ← X5 + X6

(15) C22 ← X7 + X8

(16) Output C

(17) End If

Multiply the following matrices using the above algorithm.

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix} \qquad B = \begin{pmatrix} 9 & 8 & 7 & 5 \\ 2 & 3 & 1 & 5 \\ 6 & 6 & 3 & 4 \\ 7 & 9 & 1 & 2 \end{pmatrix}$$

122. Let T(n) be the total number of mathematical operations performed by MMult(A, B, n), show

$$T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$$

123. Strassen's algorithm to multiply the two matrices is based on the following observation:
C11 = P5 + P4 - P2 + P6
C12 = P1 + P2
C21 = P3 + P4
C22 = P1 + P5 - P3 - P7
where
P1 = A11(B12 - B22)
P2 = (A11 + A12)B22
P3 = (A21 + A22)B11
P4 = A22(B21 - B11)
P5 = (A11 + A22)(B11 + B22)
P6 = (A12 - A22)(B21 + B22)
P7 = (A11 - A21)(B11 + B12)

The above formulas can be used to compute A $\times$ B recursively as follows:

Strassen(A, B):

(1) If n = 1 Output A $\times$ B

(2) Else

(3) Compute A11, B11, ..., A22, B22

(4) P1 $\leftarrow$ Strassen(A11, B12 - B22)

(5) P2 $\leftarrow$ Strassen(A11 + A12, B22)

(6) P3 $\leftarrow$ Strassen(A21 + A22, B11)

(7) P4 $\leftarrow$ Strassen(A22, B21 - B11)

(8) P5 $\leftarrow$ Strassen(A11 + A22, B11 + B22)

(9) P6 $\leftarrow$ Strassen(A12 - A22, B21 + B22)

(10) P7 $\leftarrow$ Strassen(A11 - A21, B11 + B12)

(11) C11 ← P5 + P4 - P2 + P6

(12) C12 ← P1 + P2

(13) C21 ← P3 + P4

(14) C22 ← P1 + P5 - P3 - P7

(15) Output C

(16) End If

Multiply the following matrices using the above algorithm.

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix} \qquad B = \begin{pmatrix} 9 & 8 & 7 & 5 \\ 2 & 3 & 1 & 5 \\ 6 & 6 & 3 & 4 \\ 7 & 9 & 1 & 2 \end{pmatrix}$$

124. Let T(n) be the total number of mathematical operations performed by Strassen(A, B), show

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$$

125. What is the running time of Strassen's algorithm for matrix multiplication?

126. Use Strassen's algorithm to multiply the two matrices

$$\begin{pmatrix} 0 & 2 \\ 1 & 5 \end{pmatrix} \cdot \begin{pmatrix} 7 & 2 \\ 0 & 5 \end{pmatrix}$$

127. Use Strassen's algorithm to multiply the two matrices

$$\begin{pmatrix} 0 & 0 & 3 & 1 \\ 3 & 1 & 2 & 1 \\ 1 & 0 & 4 & 1 \\ 5 & 0 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 5 \\ 3 & 0 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 0 & 2 & 4 & 0 \end{pmatrix}$$

the multiplication ends when n = 2, i.e., use the naive algorithm to compute the products of 2×2 matrices.

128. Suppose for a specific crossover point, Strassen's algorithm use the brute force method after matrix sizes become smaller than the crossover point. Implement this version of the Strassen's algorithm and find the appropriate crossover point on your computer system.

129. Explain how Strassen's algorithm can be applied to multiply $n \times n$ matrices in which n is not a power of 2?

130. Give a divide and conquer algorithm to multiply $n \times n$ matrices in time $O(n^{\log 7})$.

131. V. Pan has discovered a way of multiplying 68×68 matrices using 132,464 multiplications, a way of multiplying 70×70 matrices using 143,640 multiplications, and a way of multiplying 72×72 matrices using 155,424 multiplications. Which method yields the best asymptotic running time when used in a divide and conquer matrix multiplication algorithm? How does it compare to Strassen's algorithm? [Ref]

132. Let X and Y are two matrix. Given the program of naive matrix multiplication algorithm.

```
for i=1 to n do
  for j=1 to n do
      Z[i][j]=0;
      for k=1 to n do
       ……..
```

Fill in the blanks with appropriate formula.

a) Z[i][j] = Z[i][j] + X[i][k] * Y[k][j]

b) Z[i][j] = Z[i][j] + X[i][k] + Y[k][j]

c) Z[i][j] = Z[i][j] * X[i][k] *Y [k][j]

d) Z[i][j] = Z[i][j] * X[i][k] + Y[k][j]

133. What is the recurrence relation used in Strassen's algorithm?

a) $7\mathrm{T}(\mathrm{n}/2) + \Theta(n^2)$

b) $8\mathrm{T}(\mathrm{n}/2) + \Theta(n^2)$

c) $7\mathrm{T}(\mathrm{n}/2) + O(n^2)$

d) $8\mathrm{T}(\mathrm{n}/2) + O(n^2)$

134. The square of a matrix A is its product with itself, AA.

a) Show that five multiplications are sufficient to compute the square of a 2×2 matrix.

b) Can you give a more efficient algorithm than the Strassen's algorithm for this problem?

c) True or False, and Justify. Squaring a matrix is easier than matrix multiplication.

135. Suppose we want to evaluate the polynomial $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ at point x.

a) Show that the following simple routine, known as Horner's rule, does the job and leaves the answer in z.

$z = a_n$

for $i = n - 1$ **downto** 0

$z = zx + a_i$

b) How many additions and multiplications does this routine use, as a function of n? Can you find a polynomial for which an alternative method is substantially better?

136. The standard description of Strassen's algorithm assumes that n is a power of 2. Devise an algorithm that runs in time $O(n^{2.81})$ when n is not necessarily a power of 2.

137. Another version of Strassen's algorithm uses the following identities. To compute

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \times \begin{bmatrix} I & J \\ K & L \end{bmatrix}$$

first compute the following values:

$s_1 = G + H$	$m_1 = s_2s_6$	$t_1 = m_1 + m_2$
$s_2 = s_1 - E$	$m_2 = EI$	$t_2 = t_1 + m_4$
$s_3 = E - G$	$m_3 = FK$	
$s_4 = F - s_2$	$m_4 = s_3s_7$	
$s_5 = J - I$	$m_5 = s_1s_5$	
$s_6 = L - s_5$	$m_6 = s_4L$	
$s_7 = L - J$	$m_7 = Hs_8$	
$s_8 = s_6 - K.$		

Then

$A = m_2 + m_3$
$B = t_1 + m_5 + m_6$
$C = t_2 - m_7$
$D = t_2 + m_5.$

a) Prove that this algorithm is correct.

b) Analyze its running time. Is it likely to be faster or slower than the standard version of Strassen's algorithm in practice?

138. Suppose we were to come up with a variant of Strassen's algorithm based on the fact that 3×3 matrices can be multiplied in only m multiplications instead of the normal 27. How small would m have to be for this algorithm to be faster than Strassen's algorithm for large enough n?

139. Using Strassen's algorithm, give a divide and conquer algorithm to multiply a $kn \times n$ matrix by an $n \times kn$ matrix.

1.10 Application

140. Let A be a sorted array of n distinct integers. Describe a divide and conquer algorithm in time O(log n) that finds an index i such that A[i] = i (if one exists).

141. *Finding the Missing Number.* Suppose you are given an unsorted array A of all integers in the range 0 to n except for one integer, denoted the *missing number*. Assume $n = 2^k - 1$. Answer the following questions.

a) Design a Divide and Conquer algorithm to find the missing number.

b) Analyze its running time

c) Show that the recursive relation for this algorithm is: $T(n) = T(\frac{n}{2}) + n$.

142. *Powering a number.* Given a number a and given an integer n, compute a to the power n. The idea to calculate a^n is to express a^n as:

$$a^n = \begin{cases} a^{n/2}.a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2}.a^{(n-1)/2}.a & \text{if } n \text{ is odd;} \end{cases}$$

a) Give a divide and conquer algorithm for this problem.

b) Show that the recursive relation for this algorithm is:

$$T(n) = T(n/2) + 1$$

143. Assume A be an array of size n and a number p is given. We need to find a pair of elements in the array whose sum is exactly p. If there is a such pair, print the answer "yes," and otherwise "no." For example, given the array [2, 5, 3] and p = 5, the answer is "yes," but given p = 4 the answer is "no."

a) Use merge sort and binary search algorithms to solve this problem. Analyse its running time.

b) Give a $\Theta(n \log n)$-time algorithm (different above) to this problem.

144. Let S[1..n] is an array of size n. For some $m \leq n$, a subsequence is called ascending if $A[i_1] \leq A[i_2] \leq ... \leq A[i_m]$. Using divide and conquer algorithm, describe an algorithm to find the longest ascending subsequence (i.e., the largest m) [Ref].

145. What is the time complexity of computing the sum of $\sqrt{n}$ smallest elements in an unsorted array of length n?

146. [Ref] In this problem we consider divide and conquer algorithms for building a heap H on n elements given in an array A. Recall that a heap is an (almost) perfectly balanced binary tree where key(v) $\geq$ key(parent(v)) for all nodes v. We assume $n = 2^h - 1$ for some constant h, such that H is perfectly balanced (leaf level is "full").

First consider the following algorithm SlowHeap$(1, n)$ which constructs (a pointer to) H by finding the minimal element x in A, making x the root in H, and recursively constructing the two sub-heaps below x (each of size approximately $\frac{n-1}{2}$).

SlowHeap(i, j)

If $i = j$ then return pointer to heap consisting of node containing $A[i]$

Find $i \leq l \leq j$ such that $x = A[l]$ is the minimum element in $A[i \ldots j]$

Exchange $A[l]$ and $A[j]$

Ptr_{left} = SlowHeap$(i, \lfloor \frac{i+j-1}{2} \rfloor)$

Ptr_{right} = SlowHeap$(\lfloor \frac{i+j-1}{2} \rfloor + 1, j - 1)$

Return pointer to heap consisting of root r containing x with child pointers Ptr_{left} and Ptr_{right}

End

a) Define and solve a recurrence equation for the running time of Slowheap.

Consider the following algorithm FastHeap$(1, n)$ which constructs (a pointer to) H by placing an arbitrary element x from A (the last one) in the root of H, recursively constructing the two sub-heaps below x, and finally performing a Down-Heapify operation on x to make H a heap.

FastHeap(i, j)

Ptr_{left} = FastHeap$(i, \lfloor \frac{i+j-1}{2} \rfloor)$

Ptr_{right} = FastHeap$(\lfloor \frac{i+j-1}{2} \rfloor + 1, j - 1)$

Let Ptr be pointer to tree consisting of root r containing $x = A[j]$ with child pointers Ptr_{left} and Ptr_{right}

Perform Down-Heapify on Ptr

Return *Ptr*

End

b) Define and solve a recurrence equation for the running time of FASTHEAP.

147. [Ref] *2-D maxima finding problem.* A point (x_1, y_1) dominates (x_2, y_2) if $x_1 > x_2$ and $y_1 > y_2$. A point is called a maximum if no other point dominates it. Straightforward method for this problem is to compare every pair of points. In this case time complexity is $O(n^2)$. Divide-and-conquer for maxima finding can be as follow:

Input: A set S of n planar points.

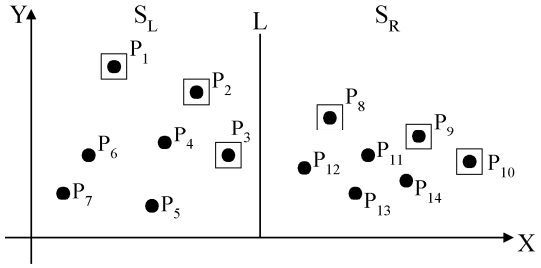
Output: The maximal points of S.

Step 1: If S contains only one point, return it as the maximum. Otherwise, find a line L perpendicular to the X-axis which separates S into S_L and S_R, with equal sizes.

Step 2: Recursively find the maximal points of S_L and S_R.

Step 3: Find the largest y-value of S_R, denoted as y_R. Discard each of the maximal points of S_L if its y-value is less than or equal to y_R.

For example:



The maximal points of S_L and S_R

Compute its recursive relation and time complexity.

148. [Ref] *The closest pair problem.* Given a set S of n points, find a pair of points which are closest together. Divide-and-conquer for the closest pair problem can be as follow:

Input: A set S of n planar points.

Output: The distance between two closest points.

Step 1: Sort points in S according to their y-values.

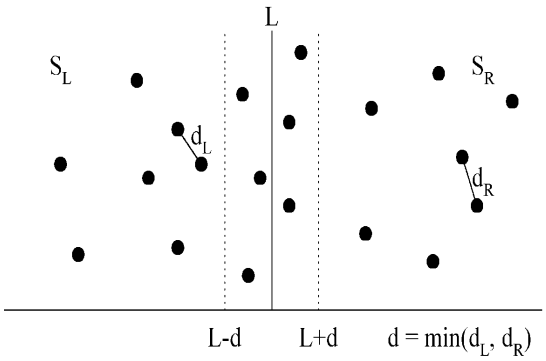
Step 2: If S contains only one point, return infinity as its distance.

Step 3: Find a median line L perpendicular to the X-axis to divide S into S_L and S_R, with equal sizes.

Step 4: Recursively apply Steps 2 and 3 to solve the closest pair problems of S_L and S_R. Let d_L(d_R) denote the distance between the closest pair in S_L (S_R). Let d = min(d_L, d_R).

Step 5: For a point P in the half-slab bounded by L-d and L, let its y-value be denoted as y_P. For each such P, find all points in the half-slab bounded by L and L+d whose y-value fall within y_P+d and y_P-d. If the distance d' between P and a point in the other half-slab is less than d, let d=d'. The final value of d is the answer.

For example:



Compute its recursive relation and time complexity.

149. [Ref] Practice with the Fast Fourier transform (FFT).

a) What is the FFT of (1, 0, 0, 0)? What is the appropriate value of ω in this case? And of which sequence is (1, 0, 0, 0) the FFT?

b) Repeat for (1, 0, 1, -1).

150. Consider Fast Fourier transform (FFT) method:

a) Explain how to multiply two polynomials using the FFT.

b) Multiply the two polynomials $2x + 3$ and $3x^2 + 4$ using the FFT.

c) Multiply the two polynomials $2 + 2x + 3x^2$ and $x^2 + 1$ using the FFT.

151. [Ref] An array A[1..n] is said to have a majority element if more than half of its entries are the same. Given an array, the task is to design an efficient

algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is A[i] > A[j]?". (Think of the array elements as GIF files, say.) However you can answer questions of the form: "is A[i] = A[j]?" in constant time.

a) Show how to solve this problem in O(n log n) time. (Hint: Split the array A into two arrays A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 help you figure out the majority element of A? If so, you can use a divide-and-conquer approach.)

b) Can you give a linear-time algorithm? (Hint: Here's another divide-and-conquer approach:

   - Pair up the elements of A arbitrarily, to get *n/2* pairs
   - Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them

     Show that after this procedure there are at most n/2 elements left, and that they have a majority element if and only if A does.)

152. [Ref] *Closest pair.* In this problem we will develop a divide-and-conquer algorithm for the following geometric task.

Input: A set of points in the plane, $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \dot{;} p_n = (x_n, y_n)$

Output: The closest pair of points: that is, the pair $p_i \neq p_j$ for which the distance between p_i and p_j, that is,

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

is minimized.

For simplicity, assume that n is a power of two, and that all the x-coordinates x_i are distinct, as are the y-coordinates.

Here's a high-level overview of the algorithm:

- Find a value x for which exactly half the points have $x_i < x$, and half have $x_i > x$. On this basis, split the points into two groups, L and R.
- Recursively find the closest pair in L and in R. Say these pairs are $p_L, q_L \in L$ and $p_R, q_R \in R$, with distances d_L and d_R respectively. Let d be the smaller of these two distances.
- It remains to be seen whether there is a point in L and a point in R that are less than distance d apart from each other. To this end, discard all points with $x_i < x - d$ or $x_i > x + d$ and sort the remaining points by y-coordinate.

- Now, go through this sorted list, and for each point, compute its distance to the *seven* subsequent points in the list. Let p_M, q_M be the closest pair found in this way.
- The answer is one of the three pairs $p_L, q_L, p_R, q_R, p_M, q_M$, whichever is closest.

a) In order to prove the correctness of this algorithm, start by showing the following property: any square of size $d \times d$ in the plane contains at most four points of L.

b) Now show that the algorithm is correct. The only case which needs careful consideration is when the closest pair is split between L and R.

c) Write down the pseudocode for the algorithm, and show that its running time is given by the recurrence:

$$T(n) = 2T(n/2) + O(n \log n)$$

Show that the solution to this recurrence is $O(n \log^2 n)$.

d) Can you bring the running time down to $O(n \log n)$?

153. [ref] *Monge arrays.* An $m \times n$ array A of real numbers is a Monge array if for all *i, j, k,* and l such that $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$, we have

$$A[i, j] + A[k, l] \leq A[i, l] + A[k, j].$$

In other words, whenever we pick two rows and two columns of a Monge array and consider the four elements at the intersections of the rows and the columns, the sum of the upper-left and lower-right elements is less or equal to the sum of the lower-left and upper-right elements. For example, the following array is Monge:

10 17 13 28 23
17 22 16 29 23

a) Let f(i) be the index of the column containing the leftmost minimum element of row i. Here is a description of a divide-and-conquer algorithm that computes the left-most minimum element in each row of an $m \times n$ Monge array A:

- Construct a submatrix *A'* of A consisting of the even-numbered rows of A. Recursively determine the leftmost minimum for each row of *A'*. Then compute the leftmost minimum in the odd-numbered rows of A.

Explain how to compute the leftmost minimum in the odd-numbered rows of A (given that the leftmost minimum of the even-numbered rows is known) in O(m + n) time.

b) Write the recurrence describing the running time of the algorithm described in part (d). Show that its solution is O(m + n log m).

154. Can you design a more efficient algorithm than the one based on the bruteforce strategy to solve the closest-pair problem for n points $x_1, x_2, ..., x_n$ on the real line?

155. [Ref] Let $x_1 < x_2 < ... < x_n$ be real numbers representing coordinates of n villages located along a straight road. A post office needs to be built in one of these villages.

a) Design an efficient algorithm to find the post-office location minimizing the average distance between the villages and the post office.

b) Design an efficient algorithm to find the post-office location minimizing the maximum distance from a village to the post office.

156. *Odd pie fight.* There are $n \geq 3$ people positioned on a field (Euclidean plane) so that each has a unique nearest neighbor. Each person has a cream pie. At a signal, everybody hurls his or her pie at the nearest neighbor. Assuming that n is odd and that nobody can miss his or her target, true or false: There always remains at least one person not hit by a pie. [Car79]

157. Find the convex hulls of the following sets and identify their extreme points (if they have any):

a) a line segment

b) a square

c) the boundary of a square

d) a straight line

158. Design a linear-time algorithm to determine two extreme points of the convex hull of a given set of $n > 1$ points in the plane.

159. What modification needs to be made in the brute-force algorithm for the convex-hull problem to handle more than two points on the same straight line?

160. Write a program implementing the brute-force algorithm for the convex-hull problem.

161. Let A denote an array of size n:

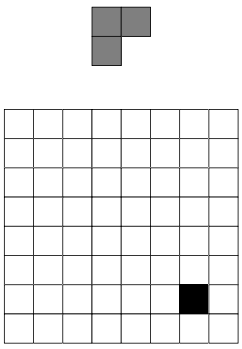
a) Using divide-and-conquer technique, give an algorithm to find the position of the largest element in the array A.

b) Apply the algorithm on arrays with several elements of the largest value and print the output.

c) Give a recurrence relation for the number of key comparisons made by the algorithm.

d) Compare the designed algorithm with the brute-force algorithm for this problem?

162. *Tromino puzzle.* Atromino (more accurately, a right tromino) is an L-shaped tile formed by three 1×1 squares. The problem is to cover any $2^n \times 2^n$ chessboard with a missing square with trominoes. Trominoes can be oriented in an arbitrary way, but they should cover all the squares of the board except the missing one exactly and with no overlaps. [Gol94]



Design a divide-and-conquer algorithm for this problem.

163. [Ref] *Nuts and bolts.* You are given a collection of n bolts of different widths and n corresponding nuts. You are allowed to try a nut and bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. The problem is to match each bolt to its nut. Design an algorithm for this problem with average-case efficiency in $\Theta(n \log n)$. [Raw91]

164. We can compute the number of levels in a binary tree by the dividing the tree into left and right sub-trees and computing the height of each sub trees recursively. Give a divide and conquer algorithm for computing the number of levels in a binary tree. What is the running time of your algorithm?

165. Let A[1..n] denotes an array of size n. An element is referred a majority element of A if it occur in more than n/2 positions. Assume that elements cannot be ordered or sorted, but can be compared for equality. Develop a divide and conquer algorithm to find a majority element in A (or determine that no majority element exists).