

# Advanced Algorithms (Practice Problems)

August 23, 2024

## 1 Divide and Conquer

1. Given an array  $A$  storing  $n$  distinct numbers. A pair  $(i, j)$  where  $0 \leq i < j \leq n-1$ , is said to be an inversion if  $A[i] > A[j]$ . Design an  $O(n \log n)$  time algorithm to count all inversions in  $A$ .
2. Given an array that represents elements of arithmetic progression in order. One element is missing in the progression, find the missing number in  $O(\log n)$ .
3. There are 2 sorted arrays  $A$  and  $B$  of size  $n$  each. Design an algorithm to find the median of the array obtained after merging the above 2 arrays (i.e. array of length  $2n$ ). The time complexity of the algorithm should be  $O(\log n)$ .
4. Given a list of values  $z_0, \dots, z_{n-1}$  (possibly with repetitions), show how to find the coefficients of the polynomial  $P(x)$  of degree less than  $n$  that has zeros only at  $z_0, \dots, z_{n-1}$  (possibly with repetitions). Your procedure should run in time  $O(n \log^2 n)$ . (Hint: The polynomial  $P(x)$  has a zero at  $z_j$  if and only if  $P(x)$  is a multiple of  $(x - z_j)$ ).
5. A Toeplitz matrix is an  $n \times n$  matrix  $A = (a_{i,j})$  such that  $a_{i,j} = a_{i-1,j-1}$  for  $i = 2, 3, \dots, n$  and  $j = 2, 3, \dots, n$ .
  - (a) Is the sum of two Toeplitz matrix necessarily Toeplitz? What about the product?
  - (b) Describe how to represent a Toeplitz matrix so that two  $n \times n$  matrices can be added in  $O(n)$  time.
  - (c) Give an  $O(n \log n)$  time algorithm for multiplying an  $n \times n$  Toeplitz matrix by a vector of length  $n$ . Use your representation from part (b).
  - (d) Give an efficient algorithm for multiplying two  $n \times n$  Toeplitz matrices. Analyze its running time.
6. Consider an  $n$ -node complete binary tree  $T$ , where  $n = 2^d - 1$  for some  $d$ . Each node  $v$  of  $T$  is labeled with a real number  $x_v$ . You may assume that the real numbers labeling the nodes are all distinct. A node  $v$  of  $T$  is a local minimum if the label of  $x_v$  is less than the label  $x_w$  for all nodes  $w$  that are joined to  $v$  by an edge.

You are given such a complete binary tree  $T$ , but the labeling is only specified in the following implicit way: For each node  $v$ , you can determine the value  $x_v$  by probing the node  $v$ . Show how to find a local minimum of  $T$  using only  $O(\log n)$  probes to the nodes of  $T$ .
7. There is an array  $A[1 \dots n]$  storing the prices of a share on  $n$  consecutive days. If we buy a share on  $i$ th day and sell it on  $j$ th day, we incur a profit of  $A[j] - A[i]$ . Note that it may be loss as well if  $A[j] < A[i]$ . Our aim is to compute the maximum possible profit we can obtain by buying a share on some day and selling it on some other (same/after) day. Note that we are allowed to buy on exactly one day and sell on exactly one day. Give a divide and conquer algorithm to compute this maximum profit for a period  $[i, j]$  where  $1 \leq i \leq j \leq n$ .

## 2 Augmented BSTs

1. In class, we discussed the operation  $\text{FINDRANK}(T, x)$  on a red-black tree. We found that we can perform this operation by augmenting each node  $v$  of the tree with a field  $\text{SIZE}(v)$  that stores the number of nodes in the subtree rooted at  $v$ . Can we keep any other field instead of size field to solve this problem? Find out which of the following fields we can keep and still achieve  $O(\log n)$  time for each operation on red-black tree.

- $\text{LEFT-SIZE}(v)$ : the size of the left subtree.
- $\text{RANK}(v)$ : the rank of the element  $v$  in tree  $T$ .
- $\text{SUBTREE-RANK}(v)$ : the rank of the element  $v$  in the subtree  $T(v)$ .

2. Recall the XOR operation that you might have studied in Boolean arithmetic or electric circuits or elsewhere. Maintain a data structure for storing a sequence of  $n$  bits  $\langle b_1, b_2, \dots, b_n \rangle$  under the following operations.

- (a)  $\text{INSERT}(i, b)$ : Insert a bit at  $i$ th position in the sequence and set its value to  $b$ .
- (b)  $\text{DELETE}(i)$ : Delete the bit at  $i$ th position in the sequence.
- (c)  $\text{REPORT-XOR}(j, k)$ : Return the XOR of bits  $b_j, b_{j+1}, \dots, b_k$  in the sequence.

Each operation must take  $O(\log n)$  worst case time.

3. Given  $n$  chords in a circle, design an  $O(n \log n)$  time algorithm to count their number of intersections.
4. In class we discussed a line sweep method to solve the following problem. Given a set of  $n$  axis-parallel rectangles, determine whether there is any pair of them that intersect. The following is another interesting application of line sweep method.

There are  $n$  horizontal line segments and each of them is colored red. There are  $n$  vertical line segments and each of them is colored blue. Design an  $O(n \log n)$  time algorithm to count all the intersections between red and blue segments.

5. The Josephus problem is defined as follows. Suppose that  $n$  people are arranged in a circle and that we are given a positive integer  $m \leq n$ . Beginning with a designated first person, we proceed around the circle, removing every  $m$ th person. After each person is removed, counting continues around the circle that remains. This process continues until all  $n$  people have been removed. The order in which the people are removed from the circle defines the  $(n, m)$ -Josephus permutation of the integers  $1, 2, \dots, n$ . For example, the  $(7, 3)$ -Josephus permutation is  $\langle 3, 6, 2, 7, 5, 1, 4 \rangle$ .

- (a) Suppose that  $m$  is a constant. Describe an  $O(n)$  time algorithm that, given an integer  $n$ , outputs the  $(n, m)$ -Josephus permutation.
- (b) Suppose that  $m$  is not a constant. Describe an  $O(n \log n)$  time algorithm that, given integers  $n$  and  $m$ , outputs the  $(n, m)$ -Josephus permutation.

6. Maintain a data structure for storing a sequence  $S$  of numbers  $\langle a_1, a_2, \dots, a_n \rangle$  under the following operations.

- (a)  $\text{INSERT}(S, i, x)$ : Insert a number  $x$  at position  $i$  in the sequence  $S$ .
- (b)  $\text{REPORT}(S, i)$ : Return the value of  $i$ th number from the sequence  $S$ .
- (c)  $\text{DELETE}(S, i)$ : Delete  $i$ th number from the sequence  $S$ .
- (d)  $\text{ROTATE}(S, i, j)$ : Rotate the sequence from  $i$ th element to  $j$ th element. For example, if sequence is  $S = \langle 3, 1, 66, 5, 9, 12, 34, 76 \rangle$ , then after  $\text{ROTATE}(S, 2, 5)$ , it becomes:  $\langle 3, 12, 9, 5, 66, 1, 34, 76 \rangle$ .

Each operation must take  $O(\log n)$  time.

7. This problem will test all your knowledge of binary search trees.

Maintain a data structure for storing a sequence  $S$  of numbers  $\langle a_1, a_2, \dots, a_n \rangle$  under the following operations. Each operation must take  $O(\log n)$  time.

- (a)  $\text{INSERT}(S, i, x)$ : Insert a number  $x$  at position  $i$  in the sequence  $S$ .
- (b)  $\text{DELETE}(S, i)$ : Delete  $i$ th number from the sequence  $S$ .
- (c)  $\text{ROTATE}(S, i, j)$ : Rotate the sequence from  $i$ th element to  $j$ th element. For example, if sequence is  $S = \langle 3, 1, 66, 5, 9, 12, 34, 76 \rangle$ , then after  $\text{ROTATE}(S, 2, 5)$ , it becomes:  $\langle 3, 12, 9, 5, 66, 1, 34, 76 \rangle$ .

- (d)  $\text{MULTI-ADD}(S, i, j, x)$ : Add  $x$  to all numbers starting from the  $i$ th number and ending at  $j$ th number in the sequence  $S$ .
- (e)  $\text{REPORTMIN}(S, i, j)$ : Return the smallest number from the  $i$ th number to  $j$ th number of the sequence  $S$ .

### 3 Greedy Strategy

1. There is a long and straight road. There are  $n$  houses located along the road. You need to place mobile towers so that each house has at least one tower within distance  $c$  from it. Design an efficient algorithm to compute the minimum number of towers to be placed to achieve this objective. You must give a formal proof of correctness for the algorithm you design.
2. There are  $n$  jobs. There are  $n$  PCs and one supercomputer. Each job consists of two stages: first it needs to be preprocessed on the supercomputer, and then it needs to be finished on one of the PCs. Let us say that job  $J_i$  needs  $p_i$  seconds of time on the supercomputer, followed by  $f_i$  seconds of time on a PC. Since there are  $n$  PCs available, the finishing of the jobs can be performed in parallel - all jobs can be processed at the same time on their respective PCs. However, the supercomputer can only work on a single job at a time. So we need to find out the order in which to feed the jobs to the supercomputer. Our aim is to minimize the completion time of the schedule which is defined as the earliest time at which all jobs will have finished processing on the PCs. Design a greedy strategy/algorithm that finds the order in which the jobs should be scheduled on the supercomputer so that completion time achieved is as small as possible.
3. A photocopy shop has a single large machine. Each morning the shop receives a set of jobs from customers. The shopkeeper wants to do the jobs on the single photocopying machine in an order that keeps their customers happiest. Customer  $i$ 's job will take  $t_i$  time to complete. Given a schedule (ordering of the jobs), let  $C_i$  denote the finishing time of job  $i$ . For example, if job  $i$  is the first to be done, we would have  $C_i = t_i$ ; and if job  $j$  is done right after job  $i$ , we would have  $C_j = C_i + t_j$ . Each customer has a given weight  $w_i$  that represents his or her importance to the business. The happiness of customer  $i$  is expected to be dependent on the finishing time of  $i$ 's job. So the company decides that they want to order the jobs to minimize the weighted sum of the completion time,  $\sum_{i=1}^n w_i C_i$ .

Design an efficient algorithm to solve this problem. That is, you are given a set of  $n$  jobs: job  $i$  has a processing time  $t_i$  and a weight  $w_i$ . You want to order the jobs so as to minimize the weighted sum of the completion time,  $\sum_{i=1}^n w_i C_i$ .

4. There is a set of  $n$  jobs. Each job has a start time and finish time. There is a single processor and it can execute only one job at a time. Design an algorithm to compute the largest subset  $R \subseteq S$  of jobs that can be executed on the processor. It can be observed that no two jobs in  $R$  are allowed to overlap. You may assume for simplicity that no two jobs have same start or end times.
5. You have a processor that can operate 24 hours a day, every day. People submit requests to run daily jobs on the processor. Each such job comes with a start time and an end time; if the job is accepted to run on the processor, it must run continuously, every day, for the period between its start and end times. Note that certain jobs can begin before midnight and end after midnight; this makes for a type of situation different from what we see in the interval scheduling problem. Given a list of  $n$  such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to the constraint that the processor can run at most one job at any given point of time. Provide an algorithm to do this with a running time polynomial in  $n$ . You may assume for simplicity that no two jobs have same start or end times.

Example: Consider four jobs : (6PM, 6AM), (9PM, 4AM), (3AM, 2PM), (1PM, 7 PM).

The optimal solution would be to pick the two jobs (9PM, 4AM) and (1PM, 7 PM), which can be scheduled without overlapping.

6. Some security consultants working in the financial domain are currently advising a client who is investigating a potential money-laundering scam. The investigation thus far has indicated that  $n$  suspicious transactions took place in recent days, each involving money transferred into a single account. Unfortunately, the sketchy nature of the evidence to date means that they don't know the identity of the account, the amounts of the transactions, or the exact times at which the transactions took place. What they do have is an approximate time-stamp for each transaction; the evidence indicates that transaction  $i$  took place at time during the interval  $(t_i - e_i, t_i + e_i)$  for some margin of error  $e_i$ . Note that different transactions may have different margins of errors.

In the last day or so, they have come across a bank account that (for other reasons we don't need to go into here) they suspect might be the one involved on the crime. There are  $n$  recent events involving the account, which took place at times  $x_1, x_2, \dots, x_n$ . To see whether it is plausible that this really is the account they are looking for, they are wondering whether it is possible to associate each of the account's  $n$  events with a

distinct one of the  $n$  suspicious transactions in such a way that, if the account event at time  $x_i$  is associated with the suspicious transaction that occurred approximately at time  $t_j$ , then  $|t_j - x_j| \leq e_j$ . In other words, they want to know if the activity on the account lines up with the suspicious transactions to within the margin of error; the tricky part here is that they don't know which account event to associate with which suspicious transactions.

Given an efficient algorithm that takes the given data and decided whether such an association exists. If possible, you should ensure that the running time is  $O(n^2)$ .

7. Given an undirected weighted graph  $G = (V, E)$ , where each edge  $e \in E$  has two parameters  $a_e$  and  $b_e$ . The weight of edge  $e$  at time  $t$  is equal to  $a_e t + b_e$ . Your aim is to monitor the minimum spanning tree of the graph during a time interval  $[t_1, t_2]$ . So this purpose, you wish to compute the minimum spanning tree which has the least weight in the interval  $[t_1, t_2]$ . Design an  $O(m \log n)$  time algorithm for this problem.
8. Given a directed graph  $G = (V, E)$  where each edge  $e$  has a capacity  $c(e) > 0$ . There is a source vertex  $s \in V$  and a sink vertex  $t \in V$ . For any path  $P$  from  $s$  to  $t$ , we define its capacity as the capacity of the least capacity edge lying on this path. Design an efficient algorithm to compute the path of maximum capacity from  $s$  to  $t$  in  $G$ .  
Hint: Get inspiration from the Dijkstra's algorithm and start from scratch.
9. There is an undirected graph  $G = (V, E)$  on  $n$  vertices and  $m$  edges. Each edge is either red or blue. You are also given a parameter  $k$ . Design an algorithm that computes, if exists, a spanning tree which has exactly  $k$  red edges and  $n - k$  blue edges.
10. Given a directed graph on  $n$  vertices and  $m$  edges, where each edge has been assigned a positive weight. There are two parameters that characterize lightness of a path  $P$ : the first parameter is its length  $l(P)$  which is defined as the number of edges of  $P$ . The second parameter is its weight  $w(P)$  which is defined as the sum of weights of all its edges. A path  $P_1$  is said to be lighter than path  $P_2$  if either  $l(P_1) < l(P_2)$  or  $l(P_1) = l(P_2)$  and  $w(P_1) < w(P_2)$ . A path  $P$  from  $u$  to  $v$  is said to be the lightest path from  $u$  to  $v$  if no other path from  $u$  to  $v$  is lighter than  $P$ . Design an  $O(m + n)$  time algorithm that for a source  $s$  and a destination  $d$ , compute the lightest path from  $s$  to  $d$ .
11. There is a directed graph  $G = (V, E)$  with positive weights on its edges. Let  $\delta(u, v)$  denote the distance from vertex  $u$  to vertex  $v$  in the graph  $G$ . Each vertex is assigned a unique color from the set  $\{red, blue, yellow\}$ . As a result  $V$  has been divided among 3 sets  $R$ ,  $B$ , and  $Y$  based on their colors:  $R$  is the set of vertices with red color,  $B$  is the set of vertices with blue color, and  $Y$  is the set of vertices with yellow color. It is also given that  $|R| = |B| = |Y|$ . Design an  $O(m \log n)$  time algorithm to compute the following quantity.  
 $\min_{u \in R, v \in B} \delta(u, v)$
12. Your friend has recently purchased a motorcycle. He is very excited and wants to drive it from some town  $s$  to another town  $d$  in Thar desert. He is provided with the complete road map of the desert - the roads, their lengths and junctions (where two or more roads meet). The motorcycle has a very natural limitation - it can drive for  $c$  kilometers with full fuel tank. Since the destination is very far, your friend must plan his route so that he can refill the fuel tank along the way. Note that the shortest route may not necessarily be the feasible route. You friend is bit confused and scared - what if he gets lost in Thar desert due to improper planning.

He approaches you with full faith that you will help him find the shortest feasible route from  $s$  to  $d$  if it exists. Model the problem in terms of a weighted, undirected graph in which roads are edges, junctions are vertices, and some junctions have fuel stations, and design an efficient algorithm to find shortest feasible route from  $s$  to  $d$  and inform if no route is feasible. Assume that both  $s$  and  $d$  are also at junctions and the source  $s$  has a fuel station. Your algorithm should take time of the order of  $O(mn \log n)$  where  $m$  is the number of edges and  $n$  is the number of vertices. You should not exploit the planarity of the input graph while designing your algorithm. In other words, your algorithm should work for any arbitrary undirected graph with positive edge weights and under the fuel constraint mentioned above.

Also note that the shortest feasible route need not be a simple path. For example, consider the following situation : There are five towns :  $s, a, b, e, d$  where fuel stations are available at  $s, b, e$  only. The connecting roads are :  $(s, a)$  of length 200 km,  $(s, d)$  of length 300 km,  $(a, b)$  of length 40 km,  $(a, d)$  of length 150 km,  $(b, e)$  of length 200 km,  $(e, d)$  of length 200 km; and motor bike can travel 250 km with full fuel tank. For this road network, shortest feasible route from  $s$  to  $d$  is  $s \rightarrow a \rightarrow b \rightarrow e \rightarrow d$ , and has length 430 km.