HomeWork 3

Ques. 1. We have N ropes having lengths $L_1, L_2, \ldots, L_N$. We can connect two ropes at a time: Connecting ropes of length L and L' gives a single rope of length L+L' and doing so has a cost of L+L'. We want to repeatedly perform such connections to finally obtain one single rope from the given N ropes. Develop an algorithm to do so, while minimizing the total cost of connecting. No proof is required.

Ans. Algorithm :-

Construct a min-heap containing length of all the ropes as key value.

While no. of elements in the min-heap > 1:
    Extract two smallest elements from the min-heap
    Add length of the two extracted elements (now the key of resultant segment is this sum)
    Put the added length back in the min-heap
end while.

Ques. 2. There are N tasks that need to be completed using 2 computers A and B. Each task i has 2 parts that take time : a_i (first part) and b_i (second part) to be completed. The first part must be completed before starting the second part. Computer A does the first part of all the tasks while Computer B does the second part of all the tasks. Computer A can only do one task at a time, while computer B can do any amount of tasks at the same time. Find an $O(n \log n)$ algorithm that minimizes the time to complete all the tasks, and give a proof of why the solution obtained by the algorithm is optimal.

Ans 2. Algorithm :

Let Matrix $[[a_1, b_1], [a_2, b_2], \ldots [a_n, b_n]]$ where $[a_i, b_i]$ represent task N_i.
Sort Matrix in decreasing order of last b_i.

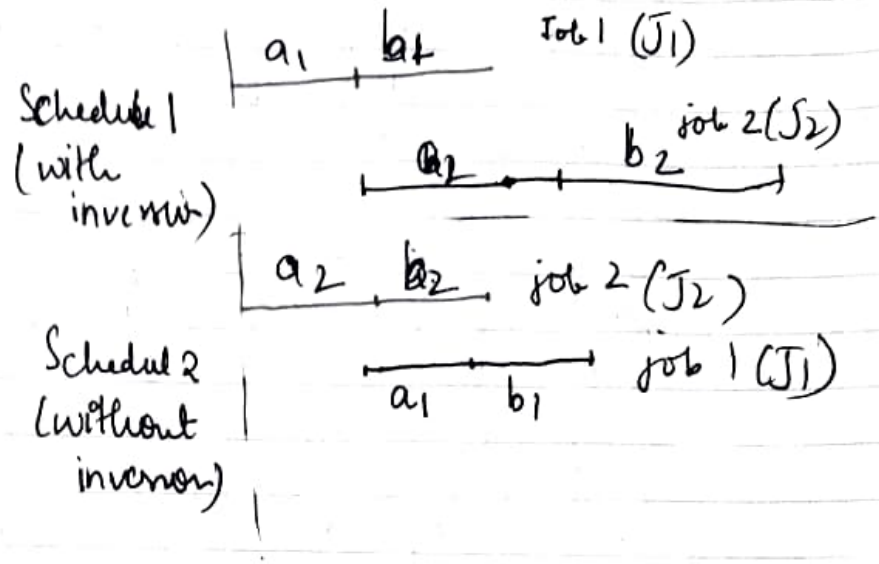
For each task Matrix$_i$ in Matrix :

Start task a_i and complete it
Start task b_i
Go to next task if present

End for

Proof of correctness:

Inversion



Let in a schedule & we swap order of J_2 and J_1. Finish time of all jobs except J_1 and J_2 do not change.
Now J_2 is scheduled earlier and will ~~finish~~ earlier.
J_1 is scheduled later but computer A hands off J_1 to computer B in schedule 2 at the same time it would have handed off J_2 in schedule 1. Since finish time of J_1 < J_2, J_1 will finish earlier in original one.
Hence swapped schedule do not have greater Completion time

→ Our solution will not contain any inversion since the sorting of b_i happens and then tasks are picked up accordingly

Ques 3. Suppose you want to drive from USC to Santa Monica. Your gas tank, when full, holds enough gas to go p miles. Suppose there are n gas stations along the route at distances $d_1 \leq d_2 \leq \ldots \leq d_n$ from USC. Assume that the distance between any neighbouring gas stations, and the distance between USC and the first gas station, as well as the distance between the last gas station and Santa Monica, are all at most p miles. Assume you start from USC with the tank full. Your goal is to make as few gas stops as possible along the way. Give the most efficient algorithm to determine which gas station you should stop at and prove that your algorithm yields an optimal solution (i.e, the minimum number of gas stops). Give the time complexity of your algorithm as a function of n.

Ans. Aim is to select the gas station which is farthest in the range.

- Let us create a array distance [n] which contains distances in increasing order, where the distance of ith gas station is stored in the ith index.
- algorithm -
  1. go as fas as possible you can with full tank of gas.
  2. Say, you have reached ith gas station if you have gas to go to i+1 gas station skip ith gas station and proceed.

3. Otherwise stop and fill the gas tank at ith gas station.

→ say x is current distance travelled,

while x + p < n :

select the farthest gas station distance [i] within range x + p.

update x to x + di

end while

- Proof using induction :
  let $g_1, g_2, \ldots, g_m$ be the gas stations where our algo. refills.
  Let $o_1, o_2, \ldots, o_n$ be the set of gas stations in some optimal soln.
  We need to prove $m < n$ and $o_i \geq g_i$.
- g_1 is the station where the vehicle can go without refuel.
  Hence it is not possible to go g_1 + ith gas station without refuel. Hence, $o_1 \leq g_1$ — (1)
- Let k be some constant, using (1) $\Rightarrow$ $o_k \leq g_k$.
- We start from h_k and have say fuel of x quantity. When we go to g_k ($\because o_k \leq g_k$) we do not refill as we have fuel as much we would have if we had refuelled at g_k.
- It is not possible to go to g_{k+1} without stopping, so we would stop at g_{k+1} or any gas station before g_{k+1}.
  Hence $o_{k+1} \leq g_{k+1}$.
- Now, let us assume our algorithm selects (k+1) gas station, which means $g_m + p <$ final distance.
  Since $o_k \leq g_k$, the optimal solution will not have k gas station, it will also select k+1 gas station.
- Time complexity - at worst case - $O(n)$

Ques 4. (a) Consider the problem of making change for n cents using the fewest number of coins. Describe a greedy algorithm to make change consisting of quarters (25 cents), dimes (10 cents), nickels (5 cents) and pennies (1 cents). Prove that your algorithm yields an optimal solution. (Hints: consider how many pennies, nickles, dimes and dime plus nickels are taken by an optimal solution at most.).

(b) For the previous problem, give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Assume that each coin's value is an integer. Your set should include a penny so that there is a solution for every value of n.

Ans. (a) Select the coin of largest denomination first which is less than n.

Algorithm:

Let C [0,0,0,0] be a array which would contain the no. of 1 cent, 5 cents, 10 cents and 25 cents coins.

→

Ans

while n > 0 :

    Find the largest denomination d ≤ n.
    n = n - d ; // decrement the value of n by d
    C[d] ++ ; // increment the index of denomination
                // d in array c.

end while.

Proof of correctness

Any optimal solution should take largest c_k such that $c_k \leq n$.
At most it can have 4 pennies, since 5 pennies can be replaced with a nickle.
At most it can have 1 nickle, since 2 nickle can be replaced by dime.
it can have at most 2 dimes
∴ 3 dimes can be replaced by a quarter and a nickle.

consider k	c_k	condition to be satisfied	max value
1	1	pennies < 5	—
2	5	nickle < 2	4
3	10	nickle + dime < 3	5 + 4 = 9
4	25	unlimited	20 + 4 = 25

Say, there exists a optimal solution to make change of $c_k \leq n \leq c_{k+1}$, our algorithm would contain c_k.

Any optimal soln would also contain c_k, Otherwise it will select from $c_1, c_2, c_3 \ldots c_{k-1}$ and be finally equal to n, but it is not possible from above. Hence our algorithm solves problem optimally. → (b) part

Ans 4. (b) $D = \{1, 15, 20\}$. $n = 30$

Greedy algo^{n}, $= 30 = 1 \times 20 + 10 \times 1 = 11$ coin

Optimal, $2 \times 15 = 2$ coins.

Ques 5. Suppose you are given two sets A and B, each containing n positive integers. You can choose to order the numbers in each set however you like. After you order them, let a_i be the ith number in set A, and let b_i be the ith element of set B. You then receive a payoff of $\prod_{i=0}^{n} a_i^{b_i}$. Give an algorithm to decide the ordering of the numbers so as to maximize your resultant payoff. Prove that your algorithm maximizes the payoff and state its running time.

Ans. The aim is to select the maximum element from set A and set B.

- Make two max heaps H_A and H_B respectively from sets A and B.
- Payoff = 1.

While H_A and H_B are not empty:

Extract max. item from H_A and H_B and, let us assume them a_k and b_k.

$$\text{Payoff} = \text{payoff} * \max(a_k^{b_k}, b_k^{a_k})$$

end while

* Proof of correctness :

- Base case - Since our algorithm selects the max. element from each set, so $\max(a_0^{b_0}, b_0^{a_0})$ would be more than any solution in any other optimal solution.

- Say for any kth element, $\max(a_k^{b_k}, b_k^{a_k})$ is greater than correspondig ~~max~~ kth element in any optimal solution.

- Now, our algorithm selects (k+1)th element, which will be greatest amongst remaining elements.
  ∴ $\max(a_{k+1}^{b_{k+1}}, b_{k+1}^{a_{k+1}})$ would be greater than the one in optimal solution.
- Hence proved.

* Run time complexity :

$O(n)$ - if ~~not~~ sorted

$O(n \log n)$ - if not sorted.

Ques. The United States Commisson of Southern California Universities (USC - SCU) is researching the impact of class rank on student performance. For this research, they want to find a list of students ordered by GPA containing every student in California. However, each school only has an ordered list of its own students by GPA and the commission needs an algorithm to combine all the lists. Find the fastest algorithm for yielding the combined list and give its runtime in terms of m, the total number of students across all colleges, and n, the number of colleges.

Ans. ~~Aim is to merge pair of~~

- Run time complexity - O(m log n)

Algorithm: A minheap H of size n is used.

- We will insert first element of each of the sorted list by each university into the heap.
- A pointer is set to all n arrays for the second element of the array. for all universities (n).
- array[m] // m is the no. of students.

for all i=1 to m:

extract the min. element of heap

array[i] = GPA of extracted element.

next = college_id of the extracted element

Insert the next element from the list l (l(next) from which min. element was picked.

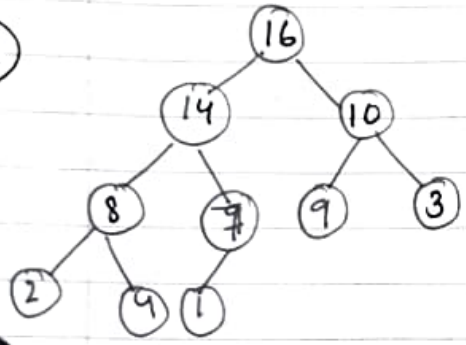
Increment the pointer in list l.

End loop.

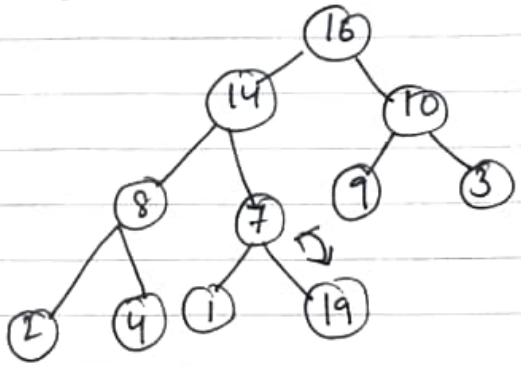
Ques 7. The array A below holds a max-heap. What will be the order of elements in array A after a new entry with value 19 is inserted into this heap? Show all your work. A = {16, 14, 10, 8, 7, 9, 3, 2, 4, 1}

Ans. A = {16, 14, 10, 8, 7, 9, 3, 2, 4, 1}

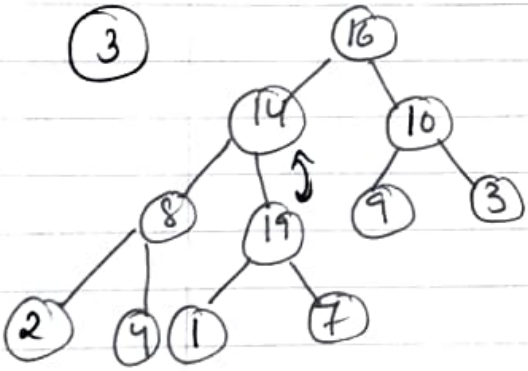
(1)



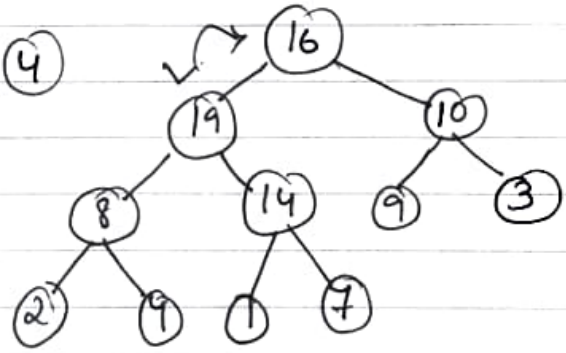
(2) Insert 19



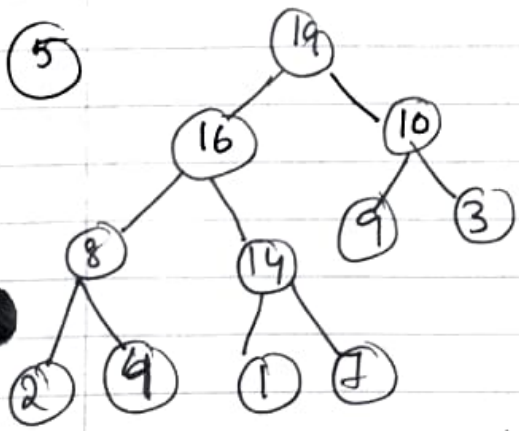
(3)



(4)



(5)



In array → A = {16, 14, 10, 8, 7, 9, 3, 2, 4, 1, 19}

Index = 10/2 = 5 (swap 7, 9)

A = {16, 14, 10, 8, 19, 9, 3, 2, 4, 1, 7}

Index 5/2 = 2.5 (swap, 14, 19)

A = {16, 19, 10, 8, 14, 9, 3, 2, 4, 1, 7}

Index 2/2 = 1 (swap 16, 19)

A = {19, 16, 10, 8, 14, 9, 3, 2, 4, 1, 7}