HOMEWORK-3
CSCI 570

Name: SRI MANVITH VADDEBOYINA | USC ID: 1231409457

1. You have N ropes each with length $L_1, L_2, \ldots L_n$ and we want to connect the ropes into one rope. Each time, we can connect 2 ropes, and the cost is the sum of lengths of the 2 ropes. Develop an algorithm such that we minimize the cost of connecting all the ropes.

Using greedy strategy we could initially pick two shortest ones.

Approach: Insert all of the rope segments into a min-heap containing length as the value of key.

Pop 2 segments each time and add their lengths and push it back to the min-heap again. Do this until we are left with only 1 rope.

Algorithm:

Construct a min-heap with all the rope segments

```
WHILE no.of elements in HEAP >1:

    POP 2 elements from HEAP
    ADD THEIR lengths and
    PUSH back to the HEAP (MIN-HEAP)

ENDWHILE
```

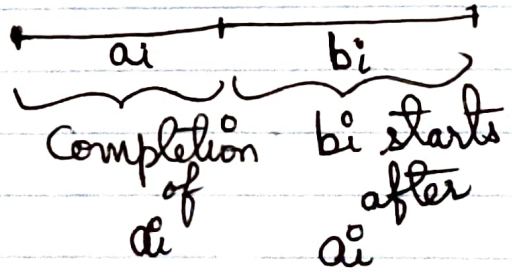
return the COST (Length of rope segment) of the connecting all ropes.

2. There are N tasks that need to be completed by 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 minimises the time to complete all the tasks, and give a proof of why the solution is optimal.

Approach:

Initially let us sort all the tasks in the descending order of $b_i \in B$.

Now as the first part must be completed before the second part starts, Computer A does the first part in the sorted order and after it is completed, Computer B starts its task.



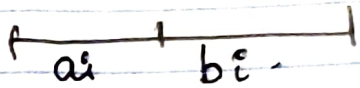
ALGORITHM:

Let there be 'N' tasks each containing a pair (a_i, b_i)
Let the tasks be represented as an array, tasks.

$$tasks = \begin{bmatrix} [a_1, b_1] \\ [a_2, b_2] \\ [a_n, b_n] \end{bmatrix}$$

Sort the tasks array in the descending order of b_i

```
FOR (ai, bi) in tasks[]
    START ai, FINISH ai
    THEN START bi, FINISH bi
    CONTINUE with next iteration and track time
ENDFOR
```



return minimum time to complete all tasks along with schedule.

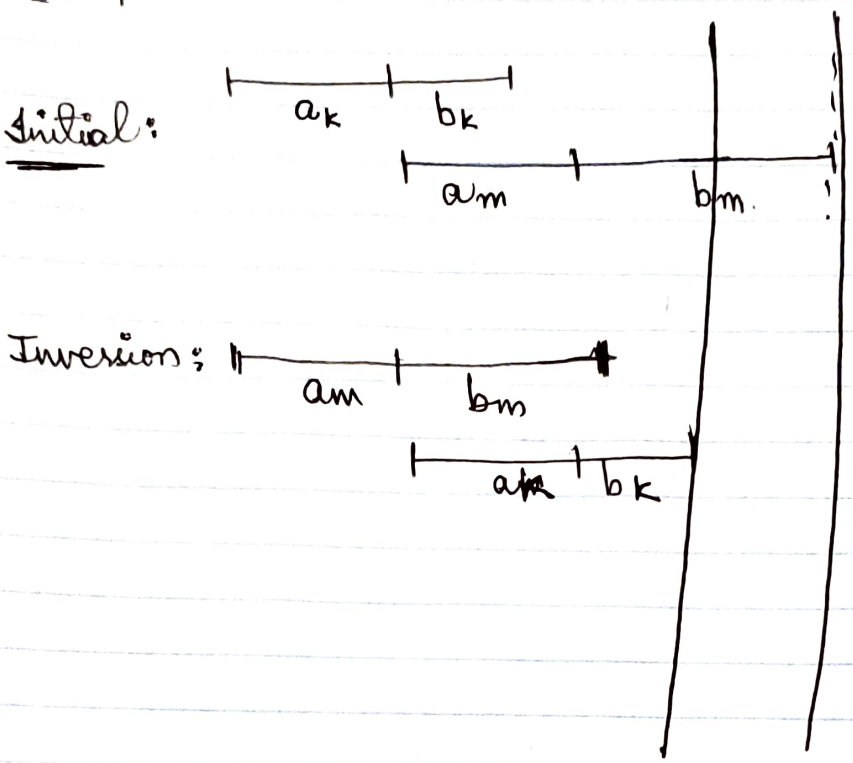
PROOF:

Let us prove that the solution we arrive at is an optimal solution. For this let us use the ~~inversion~~ exchange argument.
An inversion occurs when the order of the completion / finishing times of a task is scheduled before the other when the previous task takes lesser time to finish. i.e the task b_i is faster than b_j but b_i is scheduled before b_j.

For any optimal solution if we show that if we

swap adjacent tasks, using inversion and the finish / completion time is not increased.

Example:



If a task completes before some other task then we swap them and the remaining ones are not changed. i.e only the two tasks that are being swapped are changed and their finishing times are changed. If a task that takes higher time is swapped then it completes before the original time that it took earlier. The task that has lower finishing time would now finish earlier than the ~~other task~~ time taken by higher task in its original schedule. Hence we can say that if we swap the tasks, the finishing time is not greater than the earlier schedule. And even if there are inversions in the schedule we can remove them to get ~~the~~ optimal solution without affecting the optimality. The completion time is not greater than any other schedule and Hence the solution is optimal.

3. Suppose you were to drive from USC to Santa Monica along I-10. Your gas tank, when full, holds enough gas to go p miles, and you have a map that contains the information on the distances between gas stations along the route. Let $d_1 < d_2 < \cdots < d_n$ be the locations of all the gas stations along the route where d_i is the distance from USC to gas station

Approach:
Given the information on distances and gas stations we need to check if we can reach the station S_{i+1} stopping at S_i and checking this if we can reach S_{i+1} without filling gas at S_i, then we will proceed to reach S_{i+1} skipping the tank filling at S_i. If not we can stop and fill the tank at S_i. Continue the same until we reach the destination.

PROOF:
By using mathematical induction, we can prove the optimality.

Let the gas stations be $g_1, g_2, \cdots g_n$
Let the optimal solution be $o_1, o_2, \cdots o_n$
BASE CASE: We need to reach a gas station g_i before g_{i+1}. We stop at a gas station g_i or before g_i for checking the refill condition.

$$o_1 \leq g_1$$

INDUCTION HYPOTHESIS: If there is an optimal solution that has lesser stops than our solution, then we can let the last station g_n be removed. Then the car cannot reach the destination Santa Monica

without filling gas at g_{n-1}. Hence the number of stops in both our solution and optimal solution should be same. Then from base case we can say,

$$O_k \leq g_k$$

INDUCTIVE STEP: We need to prove $O_{k+1} \leq g_{k+1}$. If we are starting from O_k and reach g_k we know that we cannot reach g_{k+1} without stopping and refilling. It should stop at or before g_k to reach its next station.

$$O_{k+1} \leq g_{k+1}$$

Hence the ~~optimal~~ solution we get is an optimal solution.

The T.C = $O(n)$

↳ 1 computation at a gas station g_i

'n' stations : $n * 1 = n$

4.

a) Quarters – 25 cents
Dimes – 10 cents
Nickels – 5 cents
Pennies – 1 cent

Make change for n cents using fewest no. of coins.

Approach: Choose the coin of the highest value i.e denomination which is less than or equal value of given 'n'.

Subtract the difference if any and continue the process.

Algorithm:

D[0,0,0,0] be initial no. of coins of quarters, dimes, nickels and pennies. Goal is to have coins of less number which would sum up to the given value of change 'n'

WHILE n>0:
Get the largest value denomination less than (or) equal to value of 'n' (d) D[i]

n: n-d ⇒ Update value
D: D+1 → D[i] to D[i+1]
ENDWHILE

return the number of coins required to represent a given change 'n'

Proof of optimality:

Let us find the maximum value that each of the denomination of coins can take which cannot be replaced with higher denomination.

1. Pennies: $P < 5$, if $P \geq 5$ a penny can be replaced by a nickel.

2. Nickel: $N < 2$, if $N \geq 2$ a nickel can be replaced by a dime.

3. Dime: $D + N < 3$, if $D + N \geq 3$ a dime and a nickel can be replaced by a quarter.
   and
   Nickel

1. For $P < 5$, we can have value of 4 cents
2. For $N < 2$, $P < 5$, we can have value of $1 * 5 = 9$ cents
   $4 * 1$
3. For $D + N < 3$, $P < 5$, we can have value of $1 * 10 + 1 * 5 = 24$ cents
   $4 * 1$

Let an optimal solution: $C_k \leq n \leq C_{k+1}$

From ①, ②, ③ there could not be any solution $C_1 \to C_{k-1}$ that would sum up to n.

Hence our solution C_k is optimal solution.

b) Let the
Change value, n = 50
denominations, D = [20, 15, 1]

According to our greedy algorithm, we would get a change of 50 using:

$$50 = 20*2 + 1*10 = 12 \text{ coins}.$$

But the optimal solution would be picking

$$50 = 20*1 + 15*2 = 3 \text{ coins}.$$

5. $a_i \in A$
$b_i \in B$

$$\text{Pay off} = \prod_{i=0}^{n} a_i^{b_i}$$

Approach:

Pay off will be maximum if the values of a_i & b_i are maximum in their respective sets.

Algorithm:

Construct max-heap using elements from A, B
Let heaps be A' and B'

Payoff = 1

WHILE A', B' are not empty:

EXTRACT_MAX() from both A' & B'.
payoff = payoff * maximum ($a_i^{b_i}$, $b_i^{a_i}$)

ENDWHILE

return payoff.

Time Complexity : O(n log n)

If the a_i, b_i set is sorted: O(n)

Proof:

Mathematical induction

BASIS: As we have used max-heap, algorithm chooses the maximum element from each set. Hence $a_0^{b_0}$ or $b_0^{a_0}$ would not be less than optimal solution.

HYPOTHESIS: Now if we assume that our algorithm choses m^{th} element from set A & set B, in a way that max $(a_m^{b_m}, b_m^{a_m})$ is greater than that of m^{th} element in optimal solution.

INDUCTIVE STEP: Now algorithm would choose $(m+1)^{th}$ element from set A & set B that would greater than rest of elements Hence, max $(b_{m+1}^{a_{m+1}}, b_{m+1}^{a_{m+1}})$ would be greater than the same in case of optimal solution.

Hence, maximum payoff is given by our algorithm.

Time Complexity : $O(n \log n)$

If set is sorted, T.C = $O(n)$

6. Total no. of students = m
Number of Colleges = n

Goal:
Merge 'n' sorted lists.

Algorithm:

Construct a min-heap of size = n.

Push first element of each list into min-heap.
Apply min-heapify algorithm. Root = minimum element

```
WHILE size of HEAP != EMPTY:
  K: EXTRACT-MIN()  POP()
    ADD K to list result.
  PUSH the next element of the popped element
     using the pointer address if it is not NULL
    MIN-HEAPIFY()
ENDWHILE
```

return the result list.

Result list will have all the elements from all Colleges.

Time Complexity:

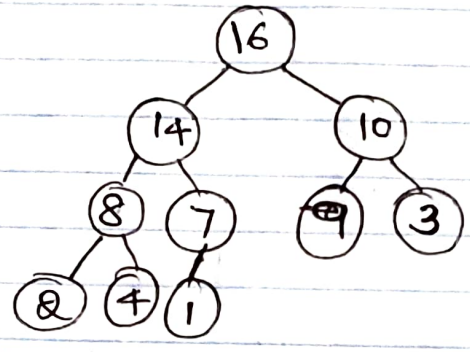
1. BUILD HEAP : $O(n)$
2. PUSH/POP : $O(\log n)$
3. Total operations: m

Total T.C = $m * \log(n) = m \log n$

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}

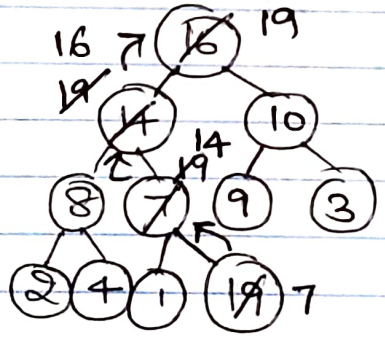
size = 10



⇒ Initial Array

1. Initial : {16, 14, 10, 8, 7, 9, 3, 2, 4, 1}

2. After the insertion of value - 19,



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

size = 11

19 > 7, swap them

arr[11/2] = 19

arr[10] = 7

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

14 < 19, swap them, 19 > 14

arr[5/2] = 19

arr[4] = 14

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

16 < 19, swap them

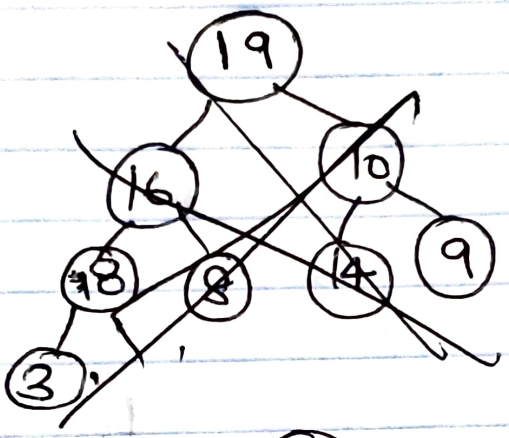
19 > 16.

arr[2/2] = 16

arr[0] = 19

5. {19, 16, 10, 8, 14, 9, 3, 2, 4, 1, 7}

Array (Final) : {19, 16, 10, 8, 14, 9, 3, 2, 4, 1, 7}



Final heap :

max-heap