Homework 3

1] Greedy choice: merge two ropes with smaller size first.

Algorithm:

Make a minimum heap containing the sizes of all ropes.

While there is more than one element in the min-heap

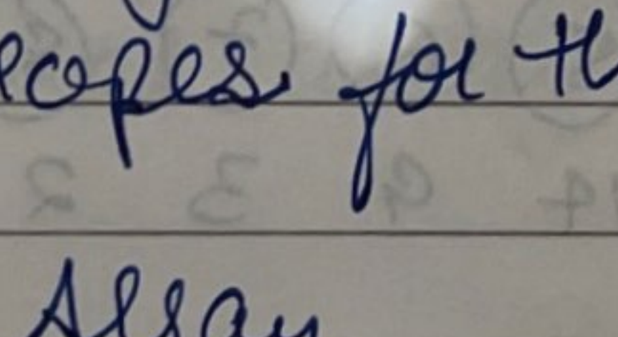
Extract two ropes with smallest sizes from the minheap, add their length and put their sum back into the heap.

End while

Here the key-value would be the length of the ropes for the minheap.

7] Initial Array

Indices	1	2	3	4	5	6	7	8	9	10
Elements	16	14	10	8	7	9	3	2	4	1



16
14 10
8 7 9 3
2 4 1 19

We add 19 in the last and 80 our array looks like.

①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪
16	14	10	8	7	9	3	2	4	1	19

Now we swap 7 and 19 since 19>7
11/2 = 5. 80,

16
14 10
8 19 9 3
2 4 1 7

①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪
16	14	10	8	19	9	3	2	4	1	7

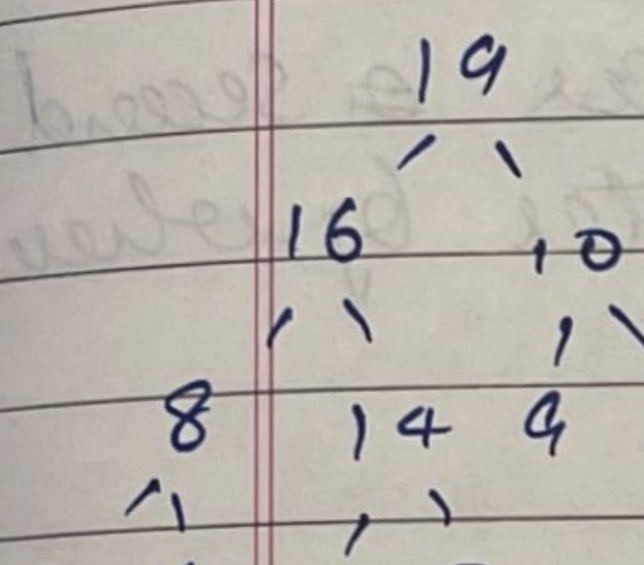
Now we swap 19 and 14 since 19>14
5/2 = 2 80,

①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪
16	19	10	8	14	9	3	2	4	1	7

16
19 10
8 14 9 3
2 4 1 7

Now, we swap 19 & 16 since 19>16 and 2/2 = 1 so,

1	2	3	4	5	6	7	8	9	10	11
19	16	10	8	14	9	3	2	4	1	7



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

2] Greedy choice : Execute that task first in which the second part (b) of the operation takes more time.

Algorithm :-

Let Z = [a_i, b_i] .... [a_n, b_n] be a two dimensional array representing N tasks.

Sorting Z in decreasing order of b_i.

~~For each task if Z, [a_i, b_i] in~~

For each task N_i [a_i, b_i] in Z,
start and finish task a_i and then start task b_i

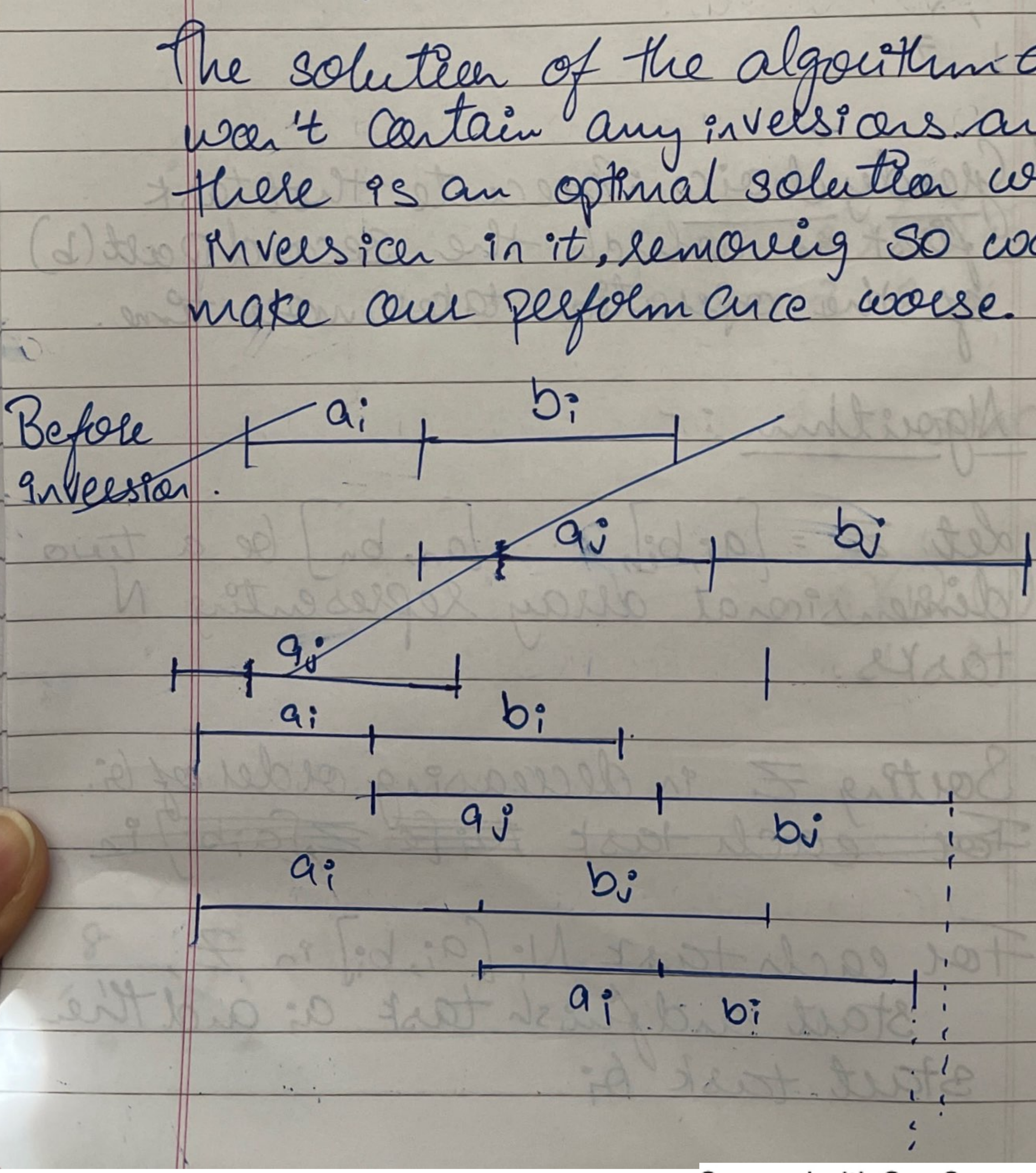
Go to next task N_{i+1} if present
End For

Proof of Correctness :-

An inversion in this situation would occur when a task b_i as a second part is scheduled after b_j when b_i is faster than b_j.

The solution of the algorithm above won't contain any inversions and if there is an optimal solution with inversion in it, removing so won't make our performance worse.

Before inversion.



Therefore our solution is as good as the optimal one.

5] Greedy choice: Selecting max elements from sets A and B.

Algorithm:

Building two max heaps from A' & B' respectively.

Let s be the total payoff.
Initialising s = 1.
While A' and B' are not empty.
    Extract max elements from them (a_i and b_i).
    Update S to be $(S * \{\max(a_i^{b_i}, b_i^{a_i})\}$.
End while.

Proof of correctness:

By showing that our algorithm stays ahead of the corresponding optimal solution always, we can prove optimality.

Proof by Contradiction:
Lets assume that the optimal payoff

is not obtained by above algorithm. Let C be the optimal solution and m be the highest index where $a_m^{b_m}$ doesn't appear in C, and a_m is paired with b_c and a_s is paired with b_m.

$a_m > a_s$ and $b_m > b_s$

We consider C' to be another solution where a_m is paired with b_m and a_s with b_c and the other pairs are same as C.

$$\frac{\text{Payoff}(C)}{\text{Payoff}(C')} = \frac{\prod_{c} a_i^{b_i}}{\prod_{c'} a_i^{b_i}}$$

$$= \frac{a_m^{b_c} \cdot a_s^{b_m}}{a_m^{b_m} \cdot a_s^{b_c}}$$

$$= \left(\frac{a_m}{a_s}\right)^{b_c - b_m}$$

$\because a_m \geq a_s$ & $b_m \geq C_c$

$$\frac{\text{Payoff}(C)}{\text{Payoff}(C')} \leq 1$$

This contradicts our assumption of of C being an optimal solution.
So, am should be paired with bm & this holds true for other elements in A & B as well.

∴ A & B sets ~~the~~ sorted in the same order gives the maximum payoff.

Time Complexity.

If the sets are sorted already then time complexity is O(n) otherwise O(n log n) [as we would have to sort them first and then traverse].

6] Use a minheap H of size n
Insert the first element of each sorted array in H. They will consists of (GPA, college-ID) pair where GPA would be the key value.
~~the~~ Set pointers to all the n arrays to the second element S(j) = 2 for for j = 1 to n.
Loop over all students (i = 1 to n)
z = Extract_min(H)
Combined Sort(i) = z.GPA

j = Z.college-ID
Insert element at S(j) from j college into heap.
Increment S(j)
End loop.

Here the runtime complexity would be O(nlogn).

3] Greedy choice: Selecting the farthest gas station within the range.

Algorithm:

Let D be the array where $[d_1, d_2, \ldots d_n]$ are arranged in increasing order. They are the distances from USC.
Let the tank of the car be full and x is the ~~car~~ current distance travelled from USC and y be the distance of SM from USC.

While $x + p < y$
    Select the farthest gas station d_i within $x + p$ range.
    Update x to $x + d_i$
End while

Proof of Correctness :

By showing that our algorithm would always be ahead of the corresponding optimal solution, we will prove optimality.

We prove this by mathematical induction.

Base Case : The first gas station that our algorithm picks would be the farthest one within the range of the car, so that would never be behind the optimal solution.

Induction step : We assume that the k^{th} gas station selected by our algorithm is ahead of the k^{th} gas station selected by the optimal solution.

Let the distance of k^{th} and $(k+1)^{th}$ gas station selected by our algorithm be d'_k and d'_{k+1} respectively and the one selected by optimal be d^o_k and d^o_{k+1}

Clearly we can see that

$$d'_{k+1} \leq d'_k + p \text{ and } d^o_{k+1} \leq d^o_k + p$$

and $d^o_k \leq d'_k$

$\therefore d^{\circ}_{K+1} \leq d'$ which proves our hypothesis

Using the same proof, we can show that the no. of gas stations selected by our algorithm is same as that of the optimal solution.

Let us suppose that our algorithm selects $(K+1)$ gas stations, which would mean that $(d'_K + P)$ is less than the distance to the final destination.

$\therefore d^{\circ}_K \leq d'_K$ then the ~~optimal~~ optimal solution has to solution $(K+1)$ Gas stations and can't leave it at K gas stations.

Time Complexity (Worst case) is $O(n)$.

4] Greedy choice: Select the coin of largest denomination first such that it is less than n.

Algorithm:-

Let N[0,0,0,0] be an initial array of fix size which contains

number of 1 cents, 5, 10 and 25 cents coins respectively which are required to form ~~to~~ the change of n cents.

while $n \geq 0$

Find the largest denomination d which is $\leq n$.

Update n to be (n-d)

Increment the i^{th} element element present in N pertaining to index corresponding to d.

End while.

Proof of Correctness:

According to the question's given information,

Any optimal solution would have the following properties in order to minimize the no. of coins given in cents :-

(1) No. of pennies ≤ 4
Because 5 pennies would be replaced by nickel.

(2) No. of nickels ≤ 1

Because 5 nickels can be replaced by dime.

(3) No. Of nickels + dimes ≤ 2
Because 2dimes and 1 nickel ca be replaced by a quarter & 3 dimes by 2quarter & 1nickel.

S	C_S	Satisfying conditions by the optimal solution	Max. value of coins for $C_1, C_2, \ldots, C_{S-1}$
1	1	Pennies ≤ 4	—
2	5	Nickels ≤ 1	$4^{*}1 = 4$
3	20	Nickel + Dimes ≤ 2	$5^{*}1 + 4^{*}1 = 9$
4	25	Unlimited	$20^{*}1 + 4^{*}1 = 24$

Let's consider an optimal solution to make $C_S \leq n \leq C_{S+1}$
According to our algorithm it would contain C_S.
We can poove that any optimal solution would also contain C_S because otherwise optimal

solution can't be achieved.

$D = \{25, 20, 5, 1\}$

If we want to make change of 40 cents then our solution would give #

$1 * 25 + 3 * 5 = 40.$

whereas the optimal one would give $2 * 20 = 40$ using 2 coins.