Date _____ Page _____

Day - 18 & 4

Heap - 3

* Height of a Heap:

⇒

1	3	6	5	9	8

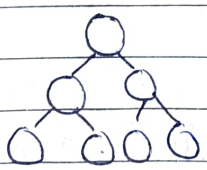
⇒ We have given a array that is a Heap.

⇒ So, for finding height, we can simply create a tree from array that will take O(n) Time and then find the longest path that also takes O(n) Time.

⇒ So, this approach in total will take O(n) Time.

⇒ So, can we further optimized this approach.

⇒ we can observe a pattern —



Node	Height
2^0 ← 1	0
2^1 ← 2	1
3	1
2^2 ← 4	2
5	2
6	2
7	2
2^3 ← 8	3
⋮	⋮
2^4 ← 16	4

Date _____ Page _____

⇒ So, our height is the power of 2.

⇒ $\log_2 N$ = Height

$2^{\text{power}} = N$

⇒ So, for doing this, we will divide the N by 2 until we get 1. Then count the number of divisions.

⇒ So, time complexity will be $O(\log_2 n)$.

⇒ There is a edge case here that if ~~height of~~ no. of nodes is 1 then height is also 1.

Code

```
if (N == 1)
    return 1;
int height = 0;
while( N > 1){
    height ++;
    N /= 2;
}
return height;
```

* Minimum cost of Ropes:

4	2	7	6	9

Date _____ Page _____

⇒ We have given a array of length of ropes.

⇒ So, we have give minimum cost after connecting all the ropes.

⇒ Connecting ropes will have a cost.

⇒ Ex: Suppose we connect 2 & 6 then cost will be 2+6 = 8.

⇒ So, remaining ropes will be —

⇒ 4 7 9 8 — 8

⇒ 4 16 8 — 16

⇒ 12 16 — 12

28 — 28

64

⇒ There will be different cost for different strategies.

⇒ We can do that we will find 2 min ropes everytime.

⇒ In this way, we will always get min. cost.

⇒ So, we can sort the array then select starting 2 elements after that adding again put the sum ~~in~~ at the correct position in the array.

⇒ Then shift all the elements in the starting of the array.

⇒ So, sorting will take → O(nlogn) T.C.

⇒ shifting will take → $O(n^2)$.

Date _____ Page _____

⇒ We know that, we have ~~s~~ to select two min. everytime. So, we can use min heap or priority queue.

⇒ And then do the sum and push this sum again into queue.

⇒ Time complexity will be O(nlogn).

* Magician and Chocolates:

A = 5, | 2 | 4 | 8 | 6 | 10 |

⇒ We have to eat max chocates in a given time.

⇒ Also, in 1 sec, you can eat any bag.

⇒ Every ~~index~~ item in the array is a bag that contains that no. of chocolates.

⇒ Also, there is a twist in the question that, when you select any bag and eat that chocates then half the no. of chocolates of that bag will again add into the array.

⇒ First, we sort the array.

⇒ So, we will select max element everytime.

⇒ And then again add half no. of chocolates into the array.

⇒ Then, find its correct position.

Date Page

⇒ This approach will take $O(n^2)$ Time.

⇒ we can also do that, we will find max element every time.

⇒ This approach will take $O(A * n)$ Time.

⇒ In the third approach, we can use max heap.

⇒ First, we select the top element then do half and again add into the ~~array of~~ queue.

Note:

⇒ If in the question, we only doing deletion then we can prefer → array.

⇒ If we have to do deletion & insertion then we can prefer → heap.

* Last stone Weight:

2	7	4	1	8	1

⇒ We have 'n' no. of stones.

⇒ Now, we select the two ~~bi~~ heaviest ~~stop~~ stone.

⇒ Then we collide them and less heaviest will destroy and also the weight of other stone reduce by the weight of destroyed stone.

Date _____ Page _____

⇒ Then, push this again into the array.
⇒ ~~Now~~ So, we have to return the weight of last stone.

⇒ So, we will use max heap for this.

* Take gift from Richest Pile:

25	64	9	4	100

⇒ Every second we have to do —
⇒ Choose the pile with the max. no. of gifts.
⇒ Leave behind the floor of the square root of the no. of gifts in the pile. Take the rest of the pile.

* Profit Maximization:

6	4	2	3

⇒ Suppose, you are a theatre owner & in every row (index), you have some seats left i.e. in 1st row → 6 seats left and so on.
⇒ And you have only 5 tickets left to sell & you have to maximize your profit.
⇒ And the price will be i.e. the no. of seats left. For ex: If 6 seats left then the price is 6 then here

Date _____ Page _____

one seat booked, so remaining seat will be 5. Now the price is 5.

⇒ So, here we use max heap.