AOA week 3 Notes

Goal : To Minimize the Maximum Lateness

Req can be scheduled at any time.
Each request has a deadline.

$$L_i = f(i) - d_i$$

deadline

Lateness   Finish Time

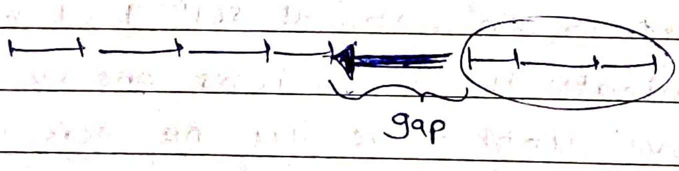
Sol 1   job1 late by 5 hrs
          job2 late by 4 hrs ✓

Sol 2   job1 late by 7 hrs
          job2 late by 0 hrs

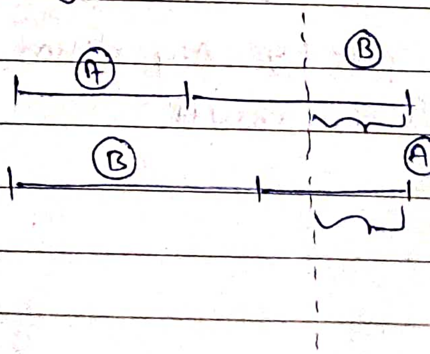
Solution:
Schedule jobs in order of their deadline without any gaps between jobs.

Proof of correctness

① There is an optimal soln with no gaps

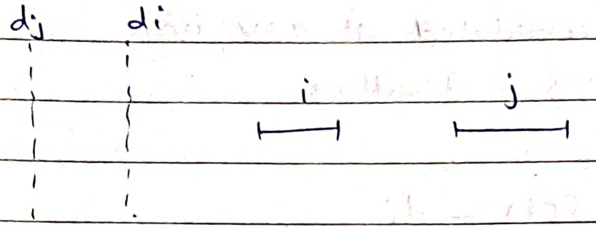


② Jobs with identical deadlines can be scheduled in any order without affecting maximum lateness



Lateness is same.

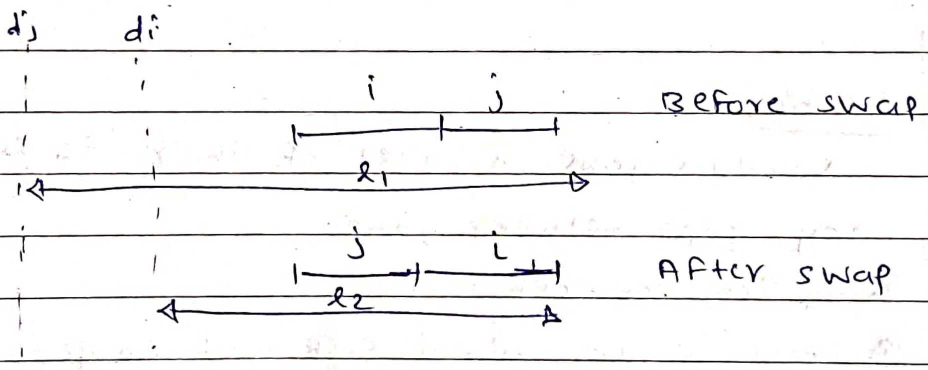
③ Schedule A' has an inversion if a job i with deadline d_i is scheduled before job j with earlier deadline d_j



Our soln has no inversions as tasks are scheduled acc to their deadline

④ All schedules with no inversions and no idle time have the same maximum lateness

⑤ There is an optimal ~~soln~~ schedule that has no inversions and no idle time.



So if there is an optimal soln that has inversions, we can eliminate the inversions one by one as shown above until there are no more inversions. This soln will also be optimal.

∴ our greedy Algorithm produces one of the optimal result.

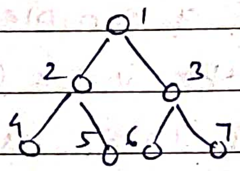
Priority Queues

A priority Queue has to perform these two operations

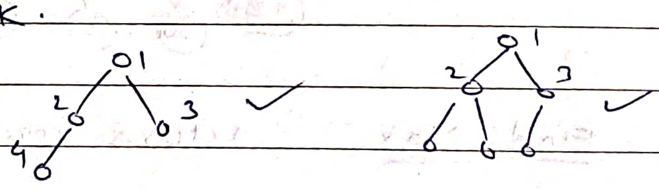
1. Insert an element into the set
2. Find the smallest element in the set.

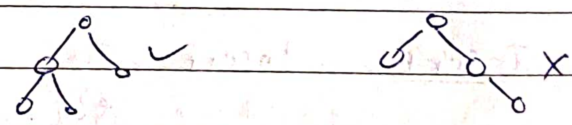
	insert	find smallest
Array implementation	$O(1)$	$O(n)$
Sorted "	$O(n)$	$O(1)$
Linked list	$O(1)$	$O(n)$
Sorted "	$O(n)$	$O(1)$

Binary Tree of depth K which has exactly $2^K - 1$ nodes is called Full Binary Tree.



A Binary Tree with n nodes and of depth K is complete if its nodes correspond to the nodes which are numbered 1 to n in Full Binary Tree of depth K.





∴ A complete binary Tree is a binary Tree in which every level, except possibly last, is completely filled, and all nodes are as far left as possible.

Traversing a complete binary stored as an array

Parent (i) is at $\lfloor i/2 \rfloor$ if $i \neq 1$

if $i=1$, i is root

L child (i) is at $2i$ if $2i \leq n$

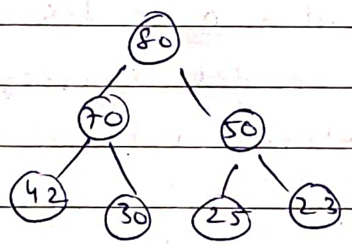
otherwise no left child

Rchild (i) is at $2i+1$ if $2i+1 \leq n$

Otherwise no right child.

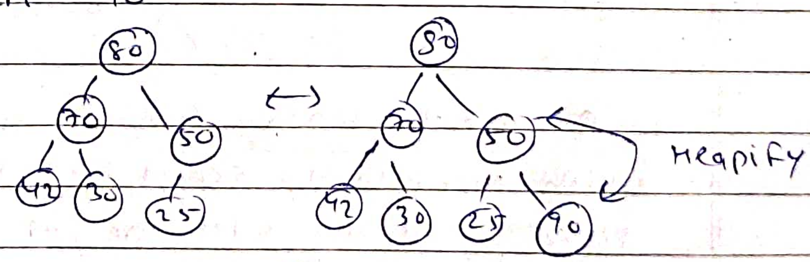
(*) Binary Heap

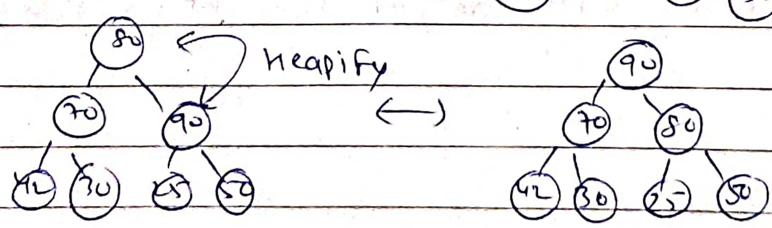
Binary Heap is a complete binary Tree with the property that the value (of the key) at each node is at least as large as the values at its children (Max Heap)



Find Max : return root O(1)

Insert Insert 90



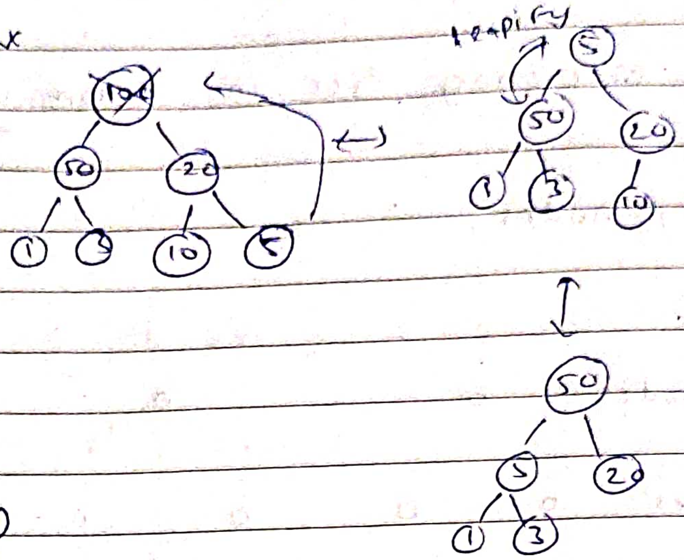


there are logn levels

Hence maximum of logn operations

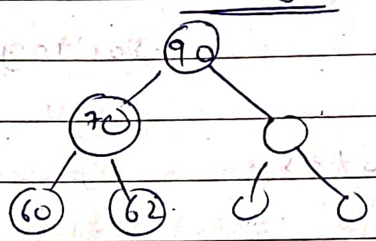
O(logn)

Extract Max

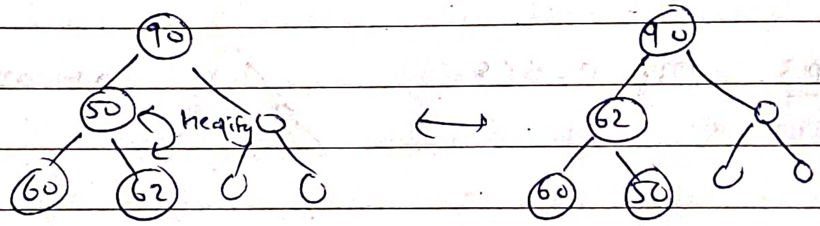


$O(\log n)$

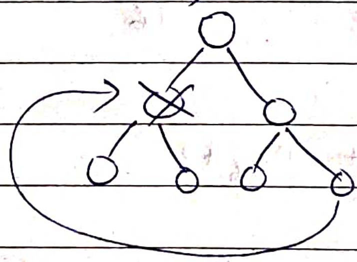
Decrease Key $O(\log n)$



Dec $70 \to 50$



Delete $O(\log n)$



Then heapify

Construction of Binary Heap

$O(n \log n)$

$n * O(\text{insertion})$

$\therefore O(n \log n)$

How to reduce it?

Start preparing Tree in Bottom up Fashion

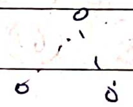
$n/8$ nodes

$n/4$ nodes

$n/2$ nodes

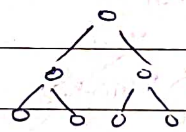
Step1 $n/2$ nodes All heaps of size 1 so no sorting

Step2 $n/4$ nodes



At max 1 operations $\leftarrow \log_2(3)$

Step3 $n/8$ nodes



At max 2 operations. $\leftarrow \log_2(7)$ $\cancel{\log_2(4)} = 2$

Total cost

$$T = \frac{n}{4} * 1 + \frac{n}{8} * 2 + \frac{n}{16} * 3 + \dots$$

$$T/2 = n/8 * 1 + n/16 * 2 + \dots$$

$$T - T/2 = T/2 = n/4 + n/8 + n/16 + \dots$$

$$T/2 = n/2$$

$$T = O(n)$$

Merge of 2 binary Heaps of size n
Takes linear time using linear time construction of the heap.

(*) Problem

i/p : unsorted array

o/p : top 'k' values

Constraints:

- No extra memory
- Time $O(n \log k)$ or less

Sol^n

min Heap

put First k elements in min Heap

~~[illegible]~~ $O(K)$

n-k elements

For every new element compare root $O(1)$

if element > root # not in smallest k

else

insert in min Heap → $O(\log k)$

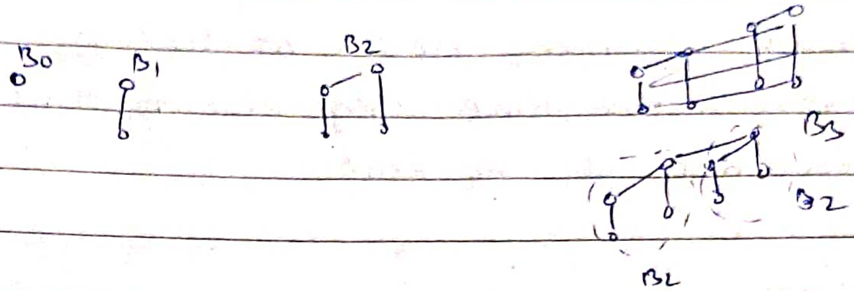
$O(\cancel{\log k}\ k) + (n-k)\,O(\log k)$

$O(n \log k)$

(*) Binomial ~~Heap~~ Tree.

Binomial Tree B_k is an ordered tree defined recursively

- Binomial Tree B_0 consists of one node
- B_k consists of $2 * B_{k-1}$ linked together such that root of one is the leftmost child of the root of the other



To construct heap of size $\underline{\underline{n}}$

$$\overbrace{0110\ 11\ 0}^{\log n}$$

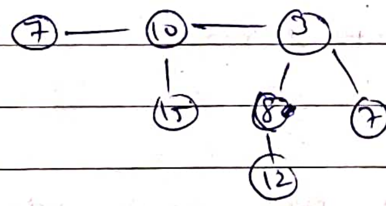
11 0 → B_2 B_1 B_0

(*) Binomial Heap:

A Binomial Heap H is a set of binomial trees that satisfies the foll:

1. Each Binomial tree in H obeys min Heap property.
2. For any non-negative integer K, there is at-most one binomial tree in H whose root has degree K.

Head →



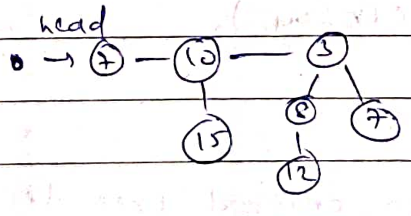
Find min

O(log(n))

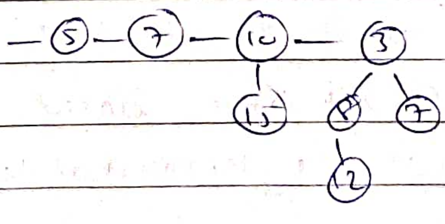
min (7, 10, 3)

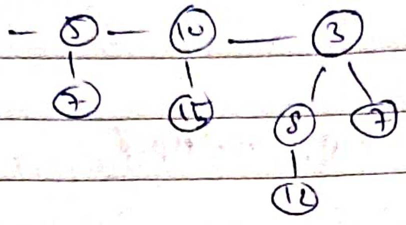
3

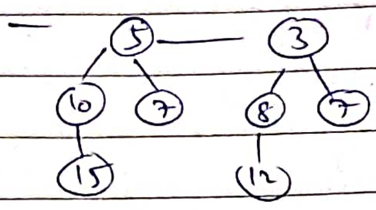
Insert O(log n)

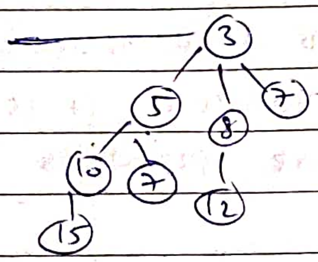


Insert (5)









Amortized Cost Analysis.

For i=1 to n

push or pop or multipop.

push → $O(1)$, pop → $O(1)$, multipop → $O(n)$

∴ worst case $O(n^2)$

Aggregate Analysis

We show that a sequence of n operations (for all n) takes worst-case Time $T(n)$ Total.

So in worst case, the amortized cost (average cost) per operation will be $T(n)/n$

multipop → $O(n)$

seq of n pushes $O(n)$

multipop $O(n)$

$T(n) = O(n)$

∴ amortized cost for n operations $O(n)/n = O(1)$

avg cost per operation = O(1)

for i=1 to n

push, pop, multipop

push O(1)

pop O(1)

multipop O(1)

$\therefore$ O(n)

Accounting method.

- Assign diff charges to diff operation
- If charge exceeds actual cost, it is stored as credit and used in future operations

Total credit any time = Total amortized cost − Total actual cost

Total credit can never be −ve

push − 2 O(1)

pop − 0 O(1)

multipop − 0 O(1)

op	charge	Actual cost	credit
push	2	1	1
push	2	1	2
multipop	0	2	0

for i=1 to n

push, pop, multipop

$\hookrightarrow$ O(1) $\hookrightarrow$ O(1) $\hookrightarrow$ O(1)

O(n)

PAGE No.
DATE / /

Fibonacci Heaps-

Fibonacci Heap is a collection of min-Heap trees similar to Binomial Heaps. However, trees in a Fibonacci heap are not constrained to be binomial trees. Also, unlike binomial Heaps, trees in Fibonacci Heaps are not ordered.