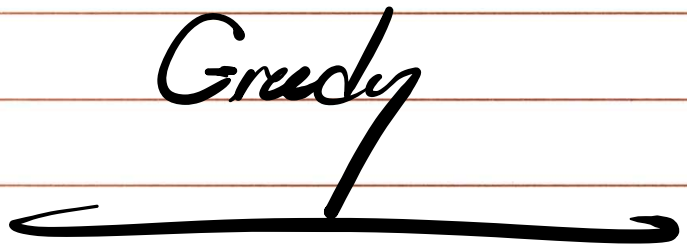
Greedy

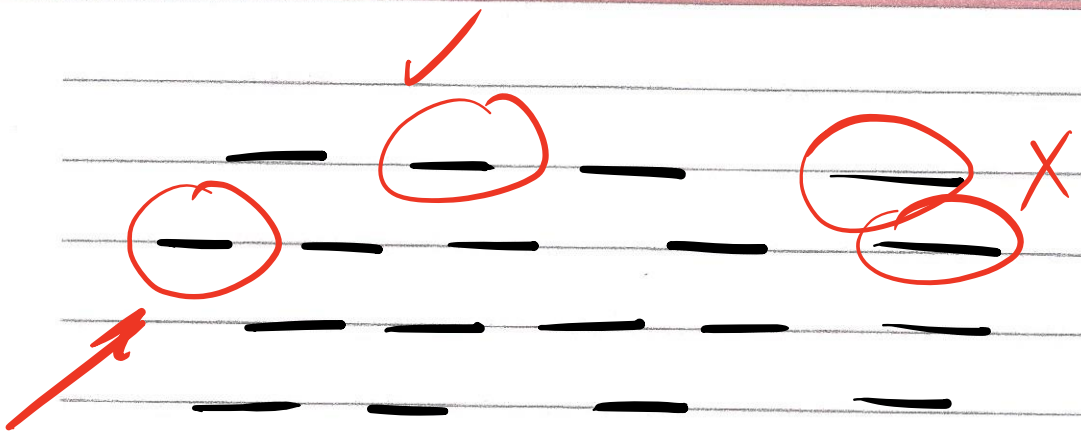


Interval scheduling Problem

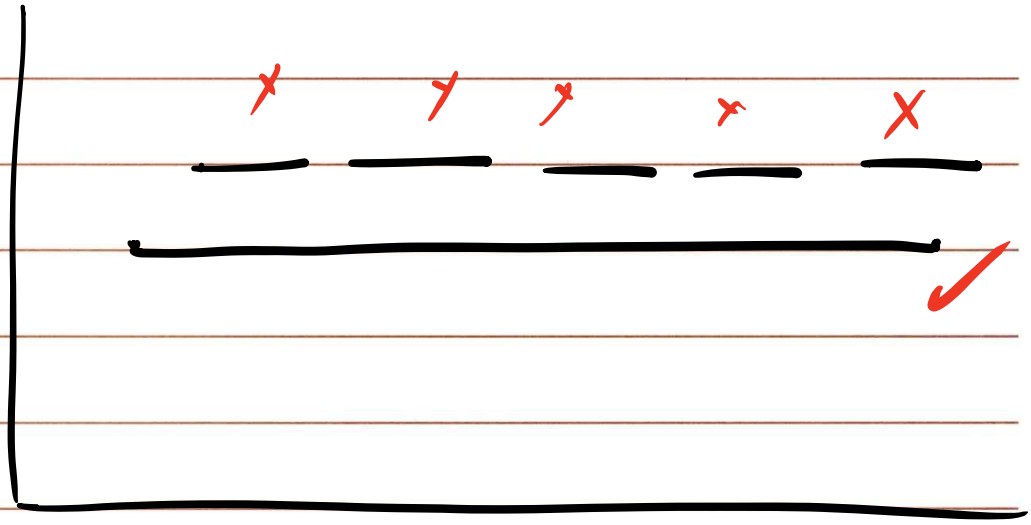
Input: Set of requests {1..n}

i^{th} request starts at s(i) and ends at f(i)

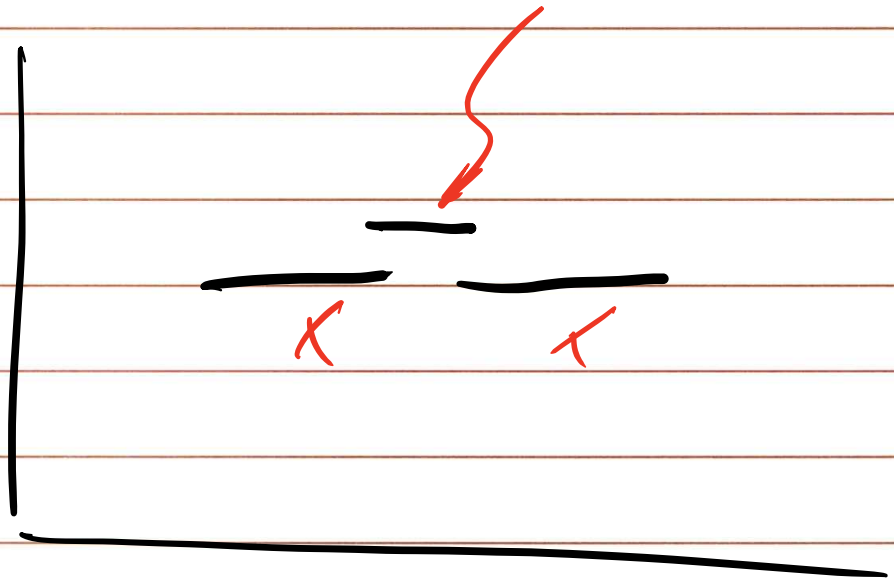
Objective: To find the largest compatible subset of these requests



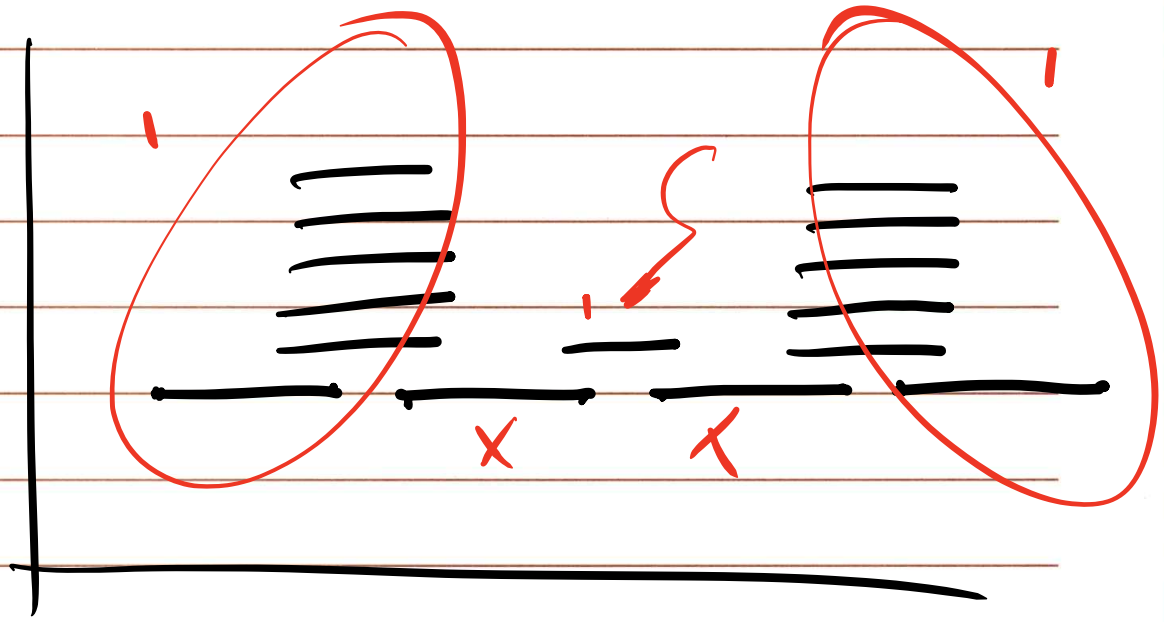
try #1 Earliest start time ✗



try #2 Smallest requests first



try #3 Smallest no. of overlaps first X



try #4 Earliest Finish Time

Solution:

Initially R is the complete set of requests
& A is empty

While R is not empty
  Choose a request $i \in R$ that has the smallest finish time
  Add request i to A
  Delete all requests from R that are not compatible w/ i
endwhile

Return A

Proof of Correctness

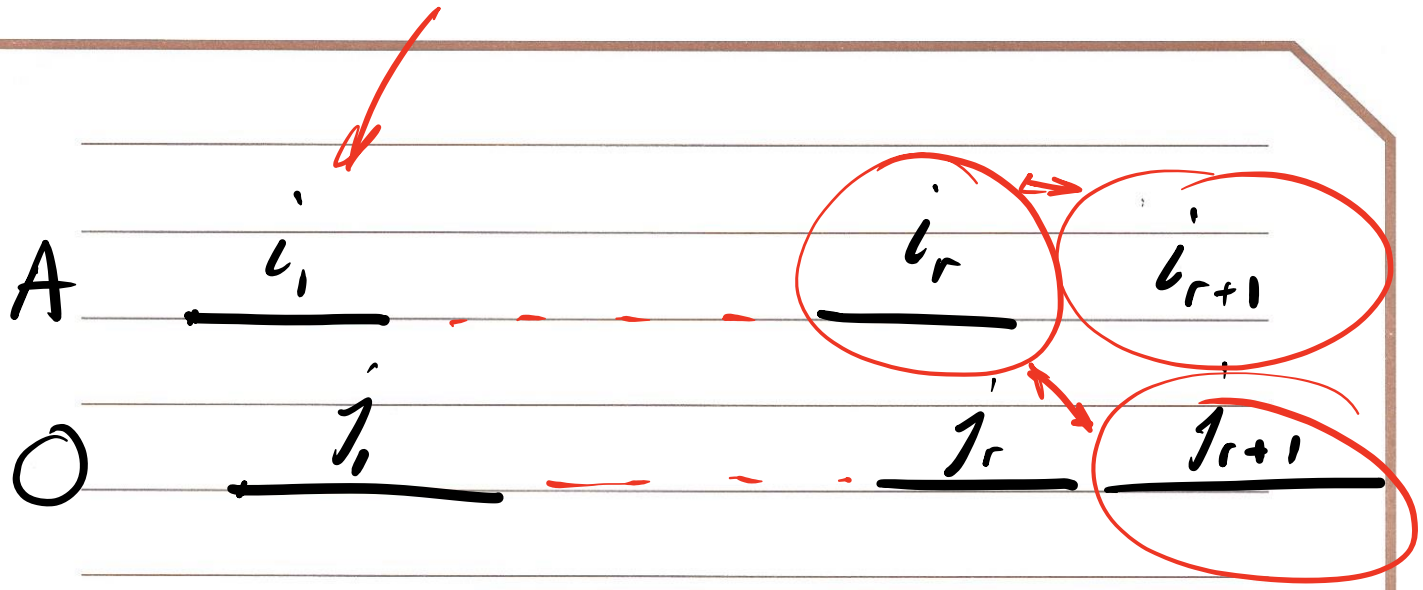
(1) Show that A is a compatible set

(2) Show that A is an optimal set

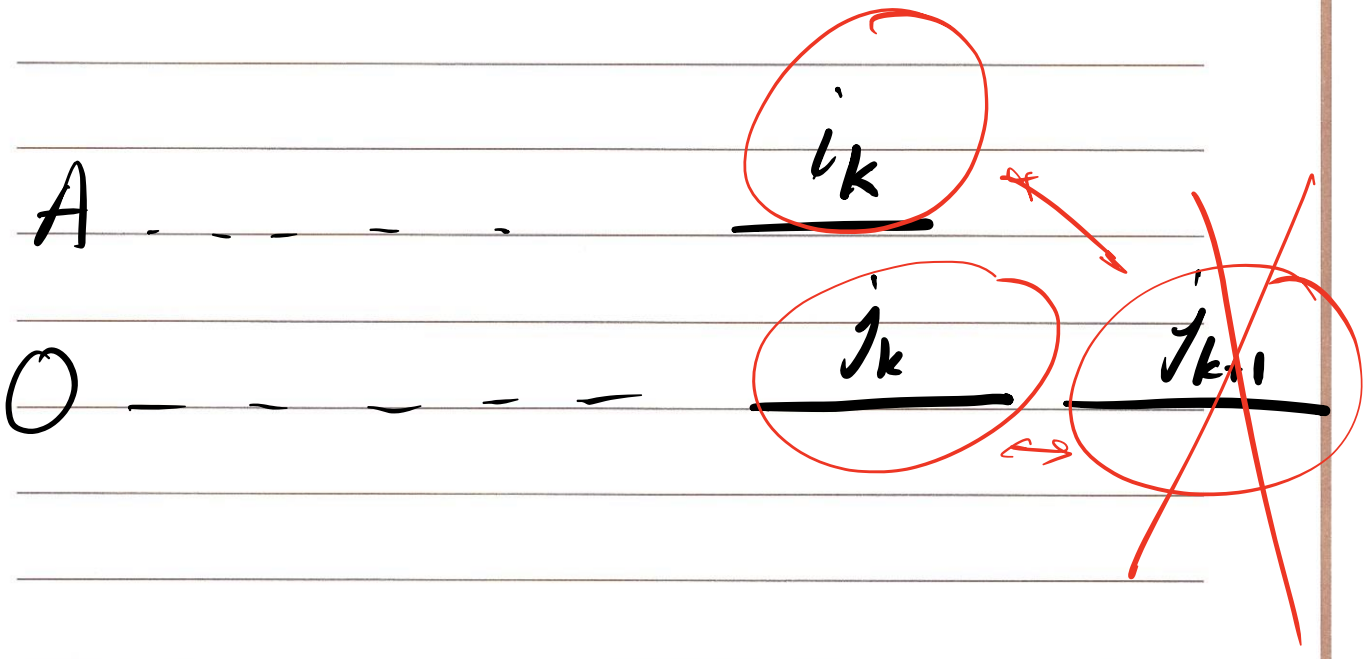
Say A is of size k
Say there is an opt. solution O

Requests in A: $i_1, \ldots, i_k$
" " O: $j_1, \ldots, j_m$

We will first prove that for all indices $r \leq k$, we have $f(i_r) \leq f(j_r)$



We can then easily prove that $|A| = |O|$



Implementation

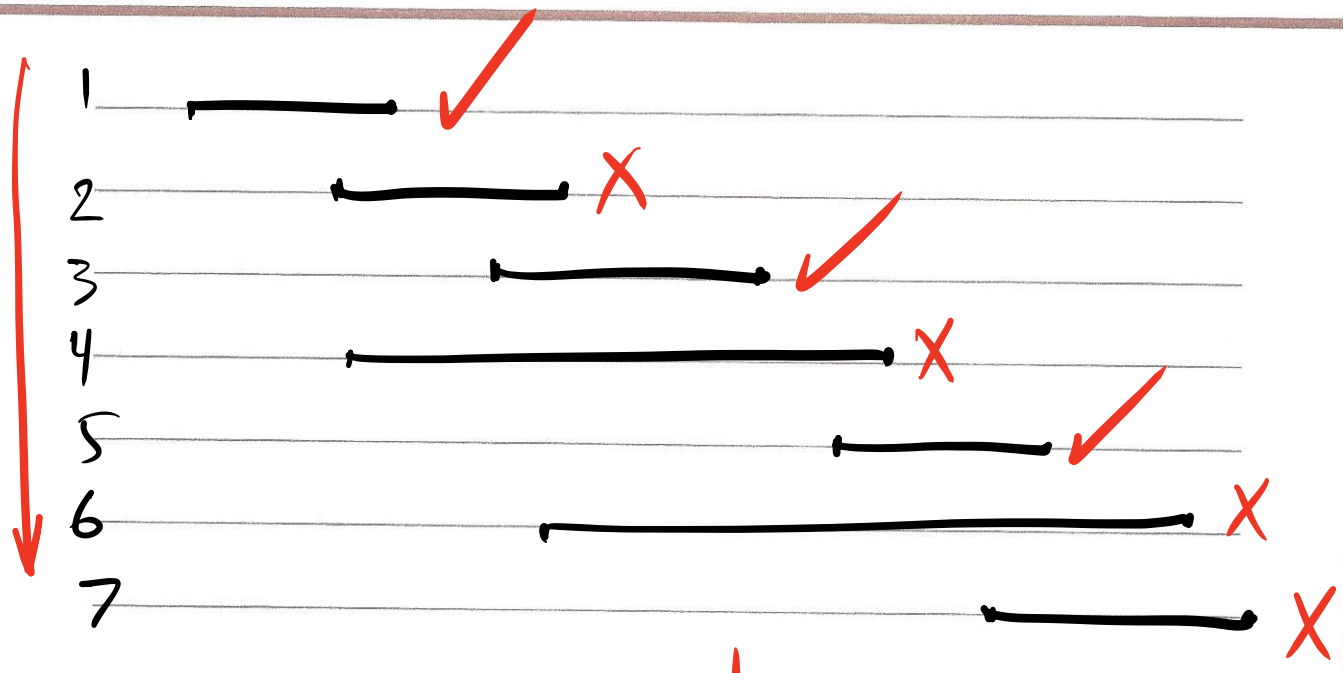
- Sort requests in order of finish time and label in this order:

  $f(i) \leq f(j)$ where $i < j$

- Select requests in order of increasing $f(i)$, always selecting the first. Then iterate through the intervals in this order until reaching the first interval for which $s(j) \geq f(i)$

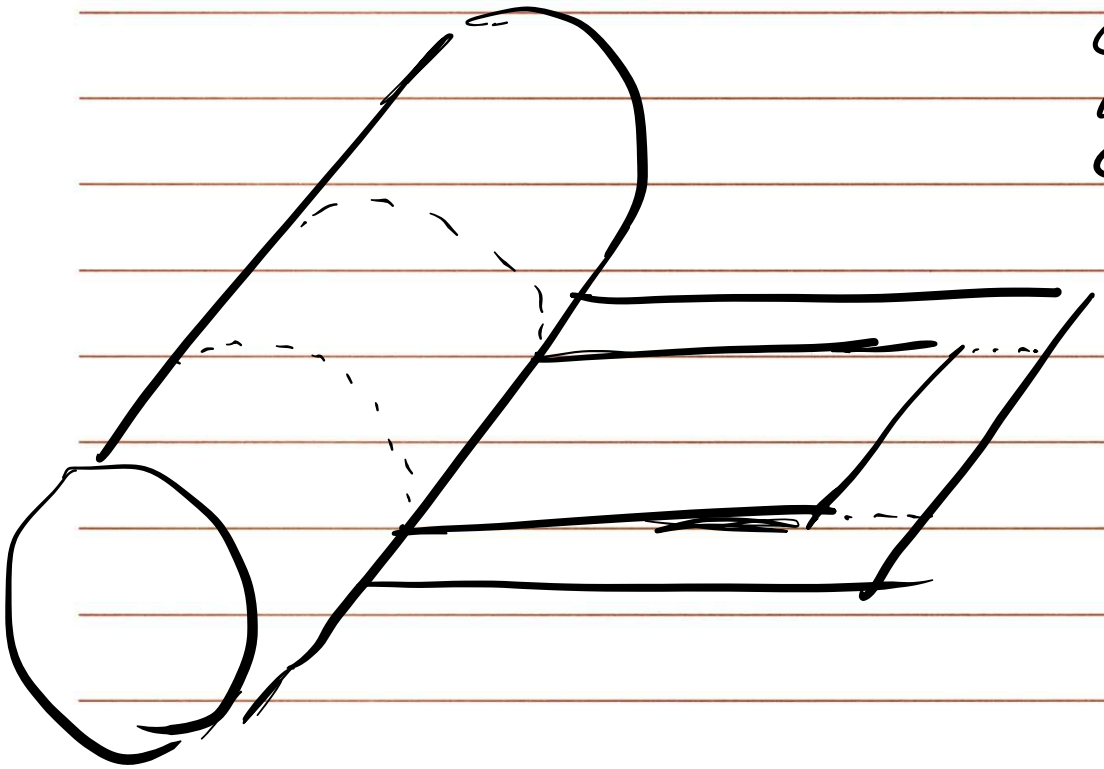
$O(n \log n)$

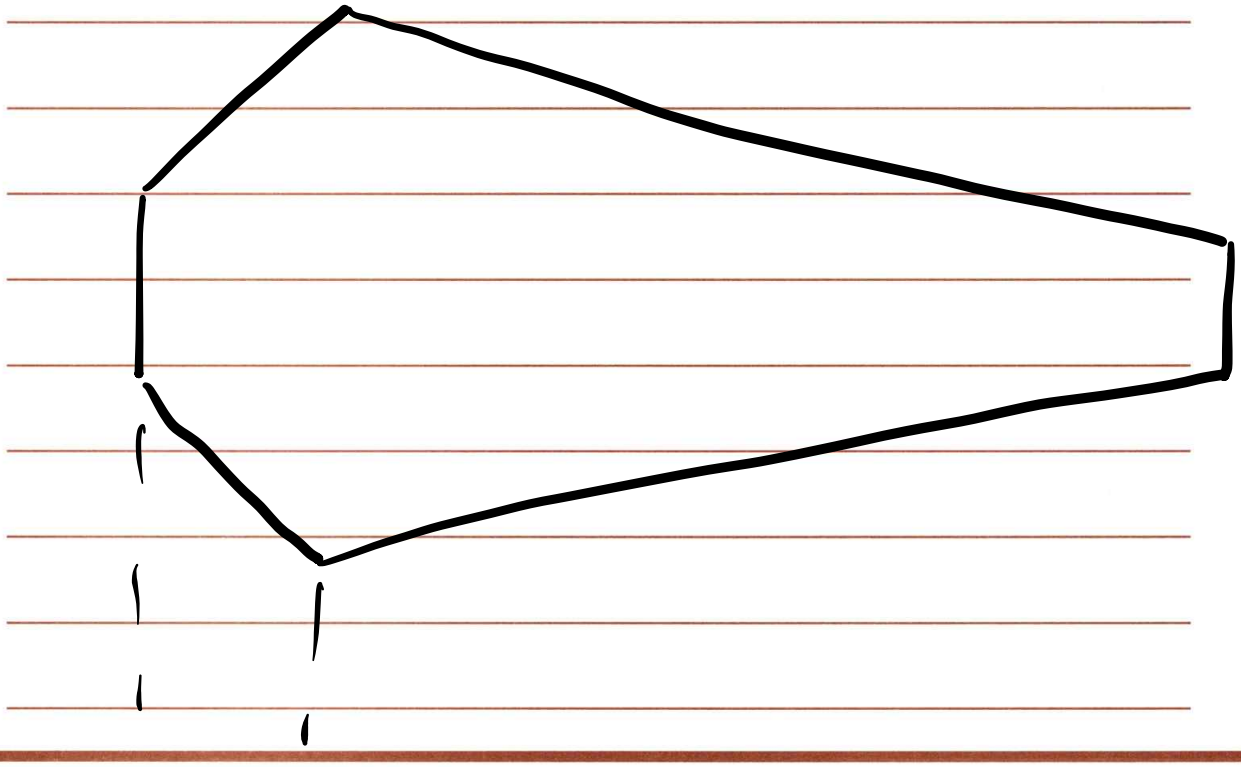
$O(n)$



overall complexity $O(n \log n)$

Cast orders:
qty
width
Grade





warm up stage

Scheduling to Minimize Lateness

Scheduling to Minimize Lateness

- Requests can be scheduled at any time
- Each request has a deadline and a required time to process
- Notation: $L_i = f(i) - d_i$

  L_i is called lateness for request i.

Goal: Minimize the Maximum Lateness $L = \max_i L_i$

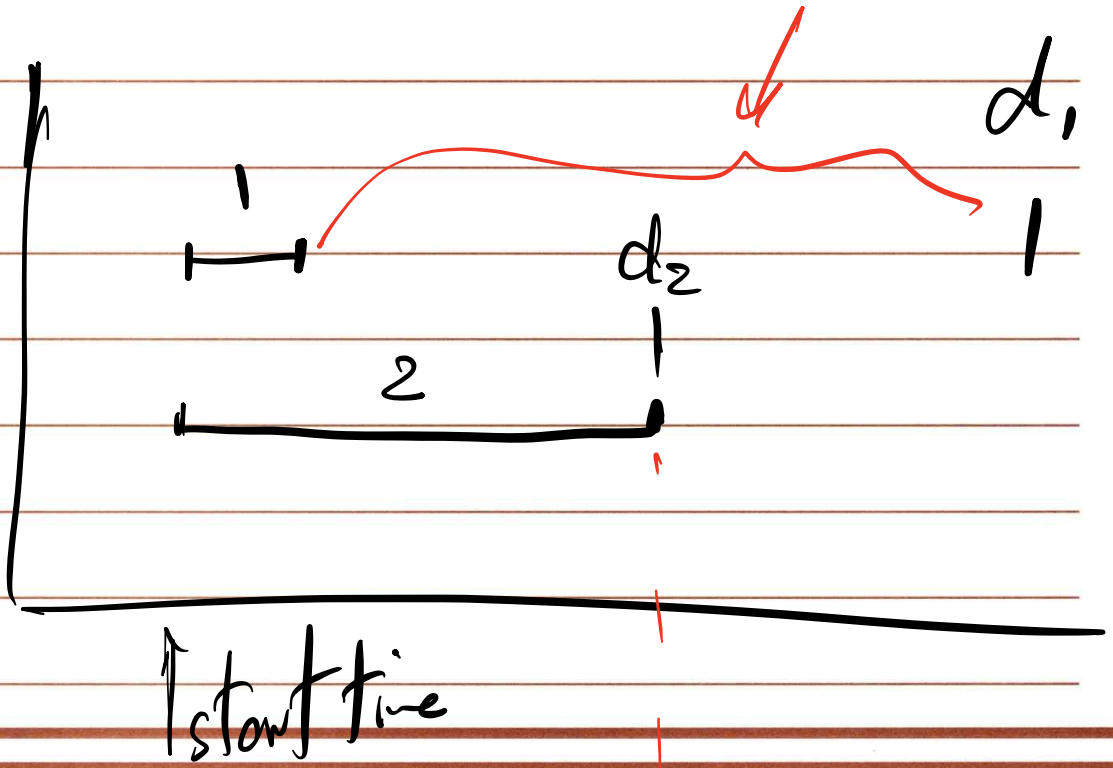
Sol. 1 job 1 late by 5 hrs

job 2 " " 6 "

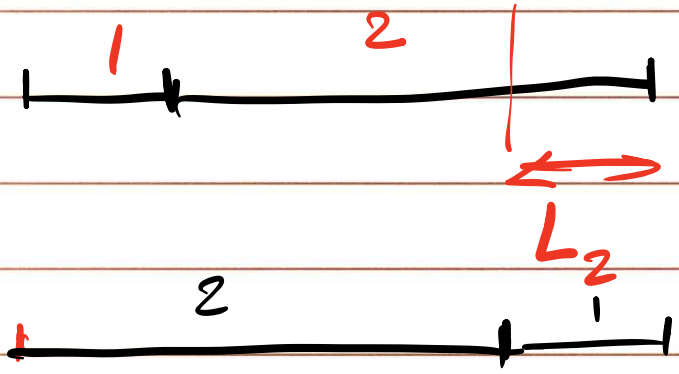
Sol. 2 job 1 late by 0 hrs

job 2 " " 7 hrs

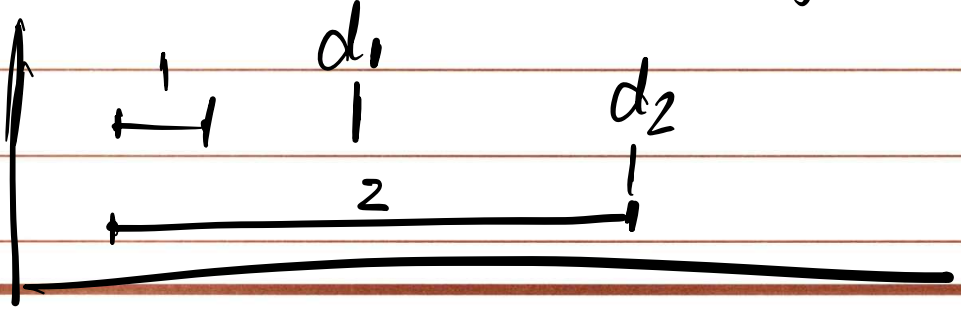
try #1 smallest request first. X



start time



try #2 smallest slack first X

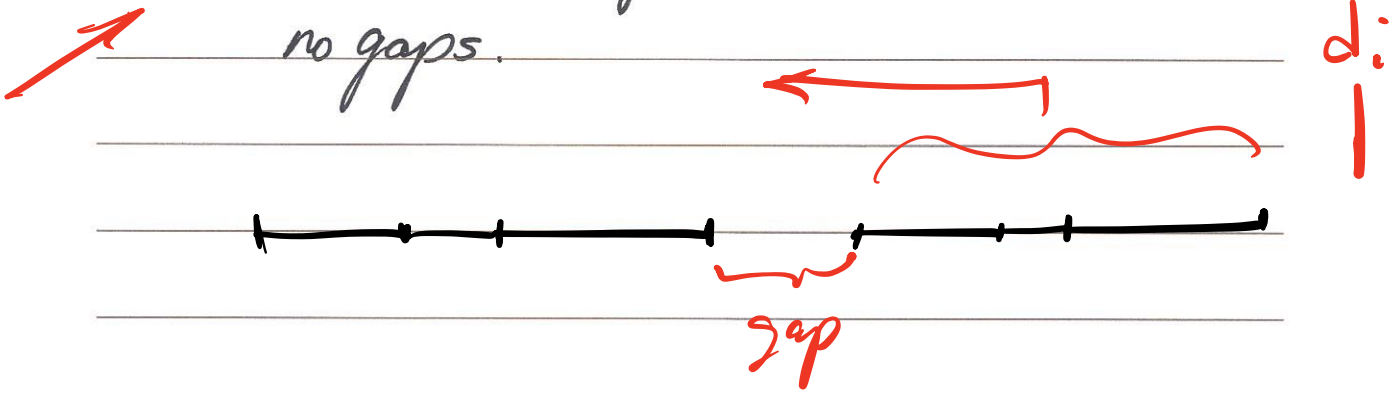


Solution:

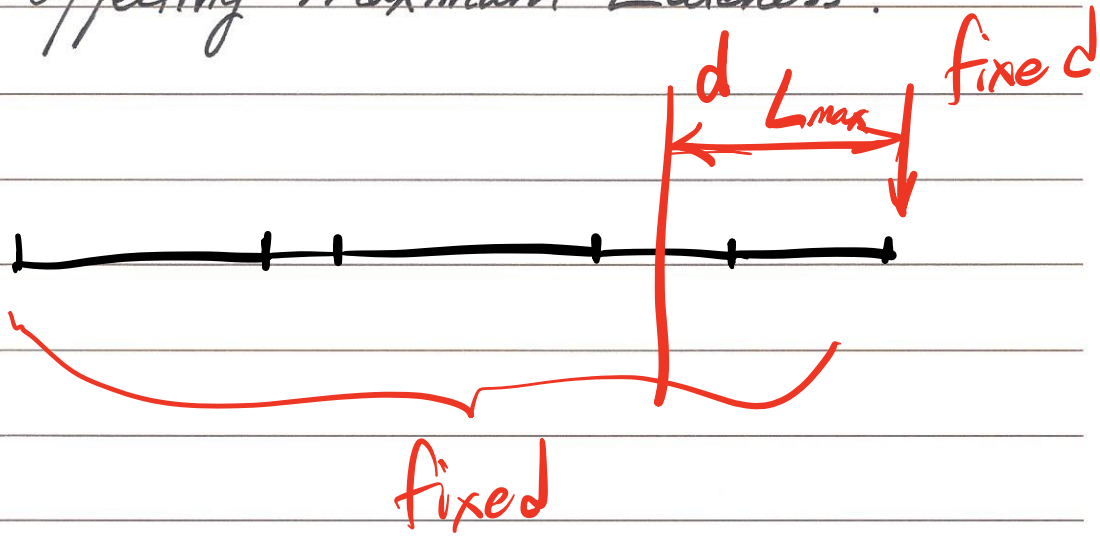
Schedule jobs in order of their deadline without any gaps between jobs.

Proof of Correctness

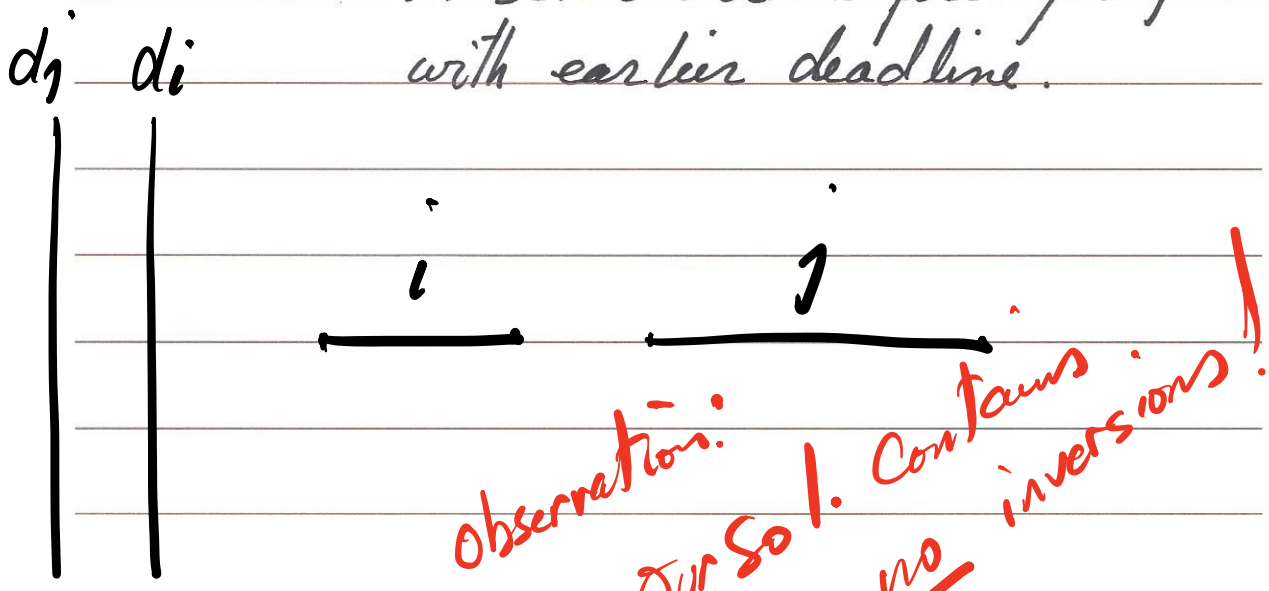
① There is an optimal solution with no gaps.



(2) Jobs with identical deadlines can be scheduled in any order without affecting Maximum Lateness.



(3) Def. Schedule A' has an inversion if a job i with deadline d_i is scheduled before job j with earlier deadline.

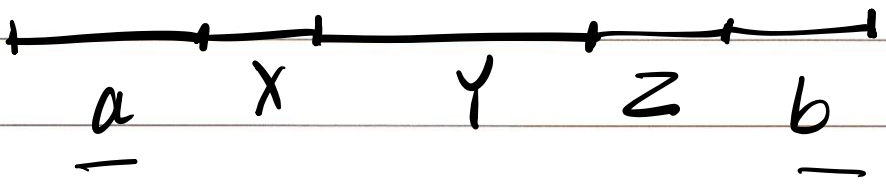


Observation: Our sol. contains no inversions!

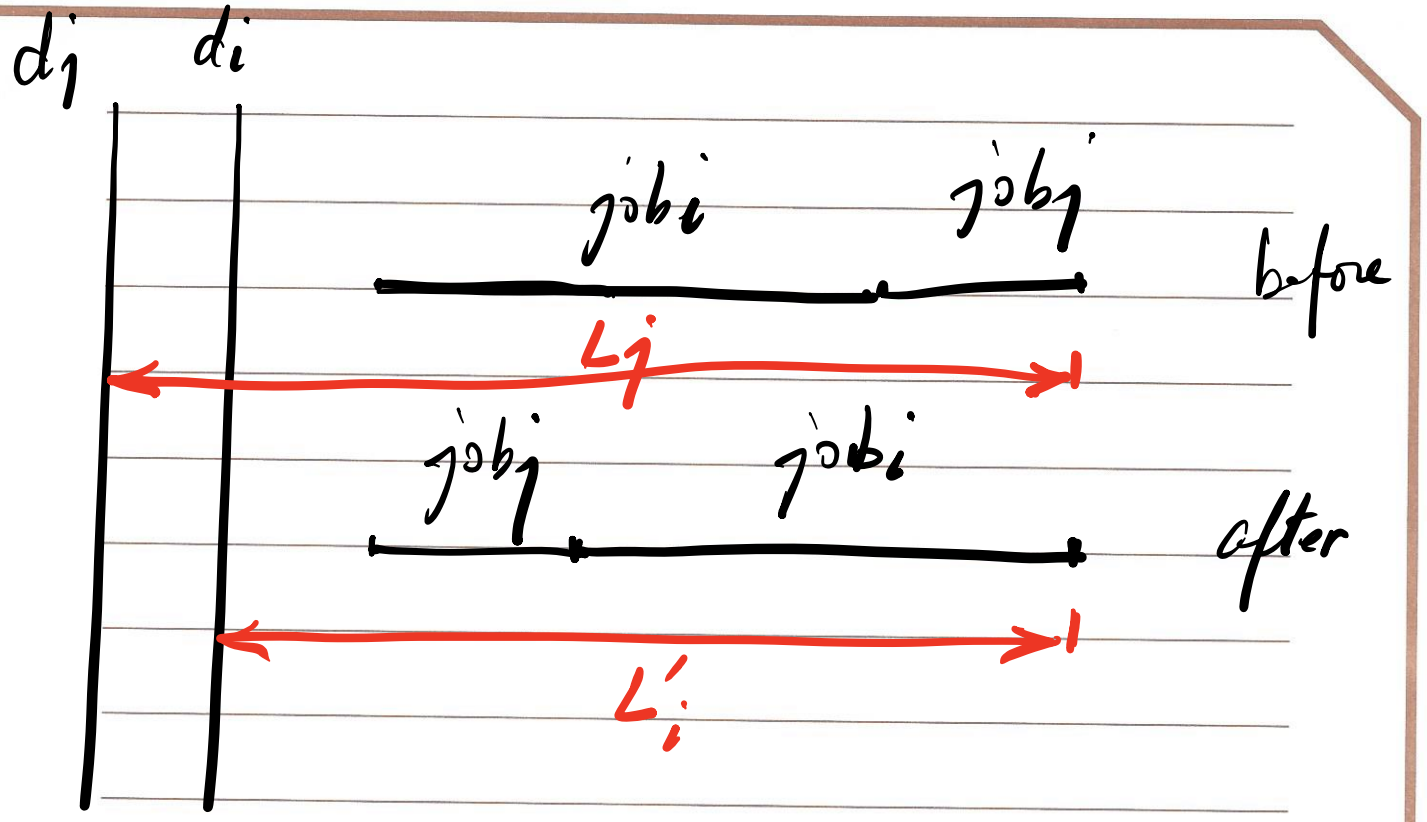
(4) All schedules with no inversions and no idle time have the same Maximum Lateness.

(5) There is an optimal schedule that has no inversions and no idle time.

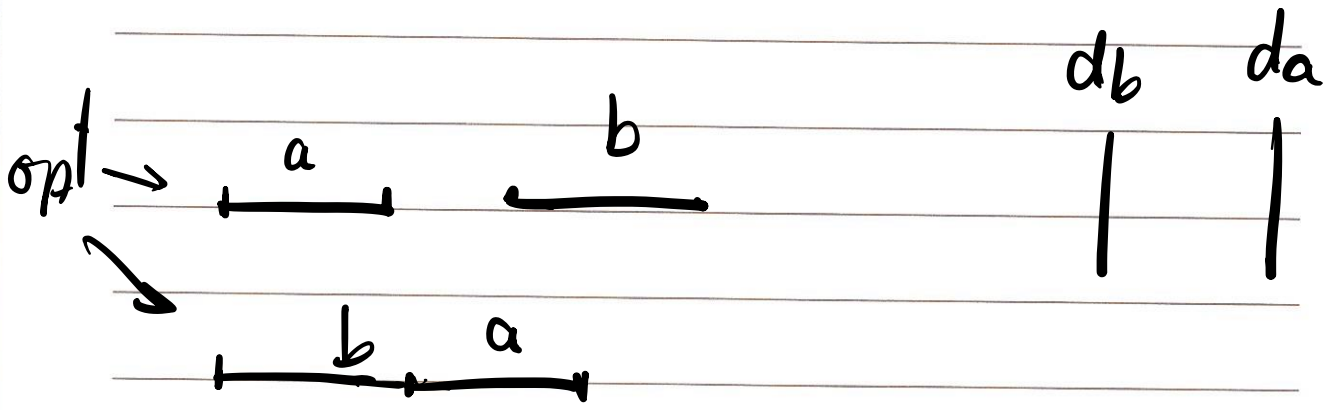
$d_a = 5$ $d_x = 6$ $d_y = 6$ $d_b = 3$



3



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



(6) Proved that there exists an optimal schedule with no inversions and no idle time.

Also proved that all schedules with no inversions and no idle time have the same Maximum Lateness.

Our greedy algorithm produces one such solution ⇒ It will be optimal

Priority Queues

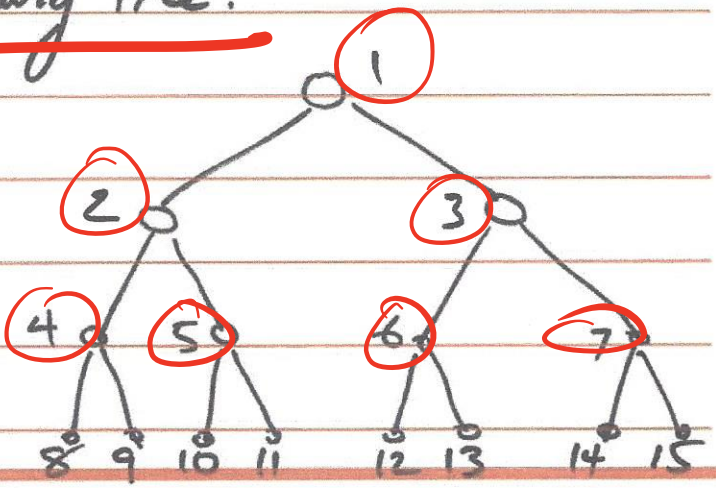
A priority queue has to perform these two operations fast!

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

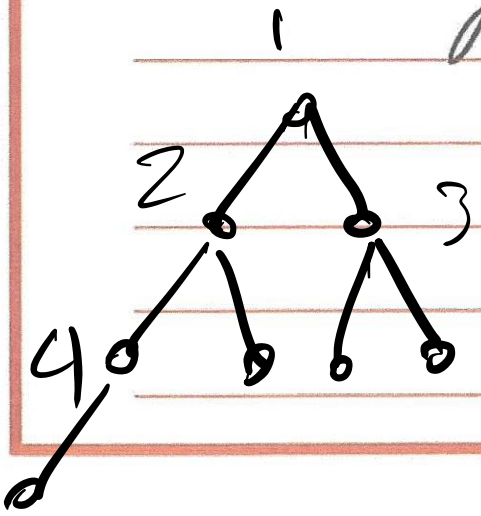
	Insert	Find
Array implementation	$O(1)$	$O(n)$
Sorted " "	$O(n)$	$O(1)$
linked list "	$O(1)$	$O(n)$
Sorted " " "	$O(n)$	$O(1)$

Background

Def. A binary tree of depth k which has exactly $2^k - 1$ nodes is called a full binary tree.



Def. A binary tree with n nodes and of depth k is complete iff its nodes correspond to the nodes which are numbered 1 to n in the full binary tree of depth k.



Traversing a complete binary tree stored as an array

Parent(i) is at $\lfloor i/2 \rfloor$ if $i \neq 1$
if $i = 1$, i is the root

Lchild(i) is at $2i$ if $2i \leq n$
otherwise it has no left child

Rchild(i) is at $2i+1$ if $2i+1 \leq n$
otherwise it has no right child

Def. A 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)

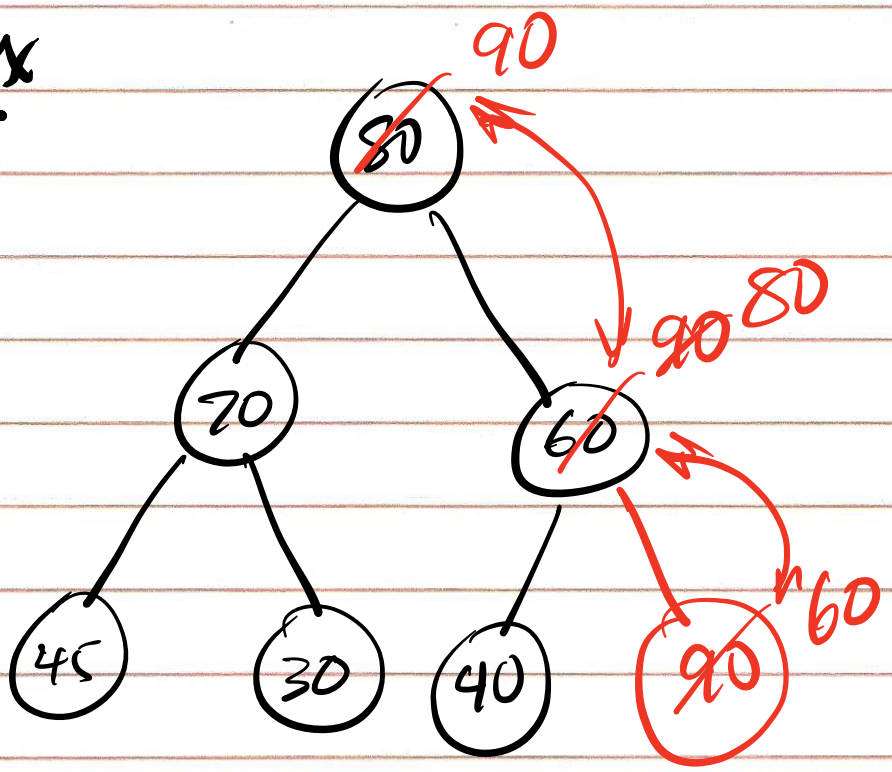
Find_Max

O(1)

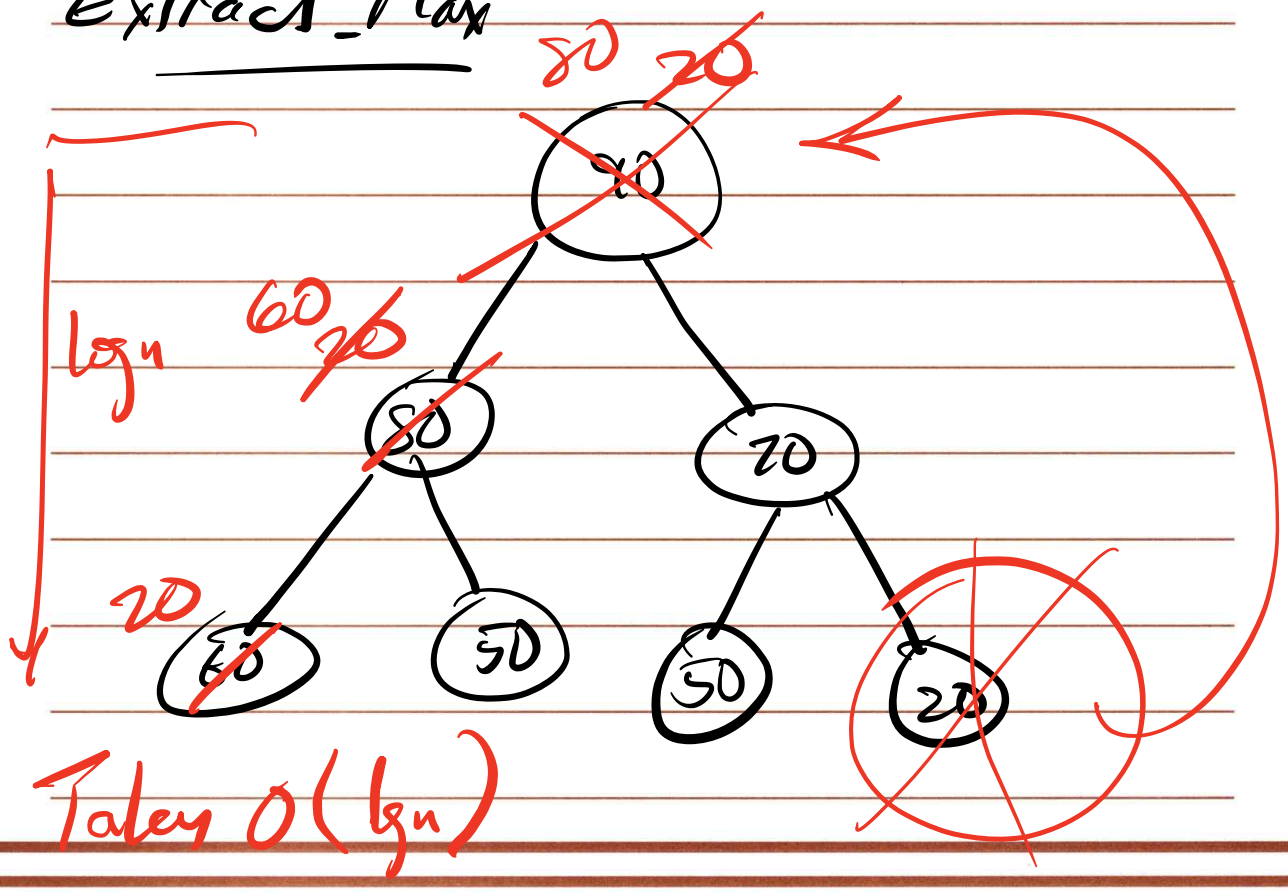
Insert

insert 90

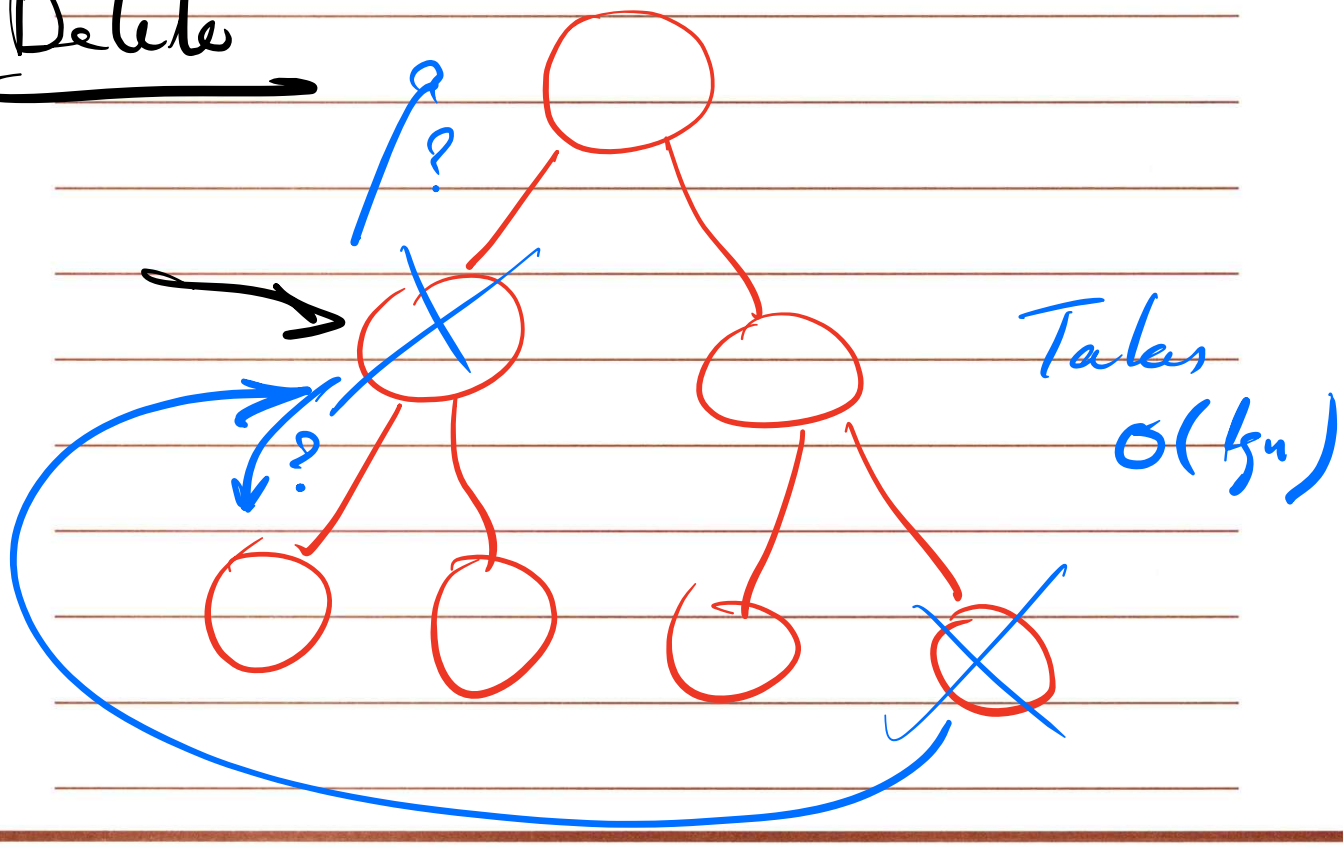
takes O(lg n)



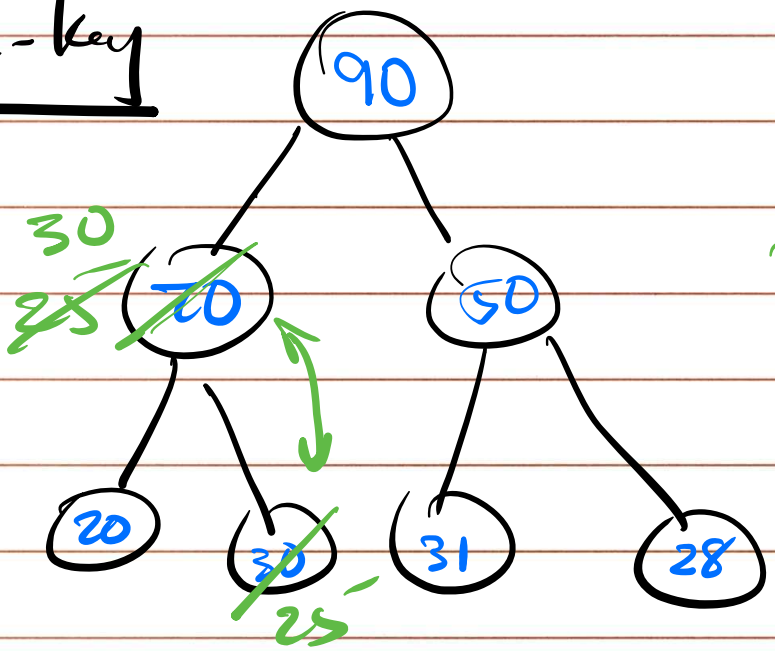
Extract_Max



Delete



Decrease-key

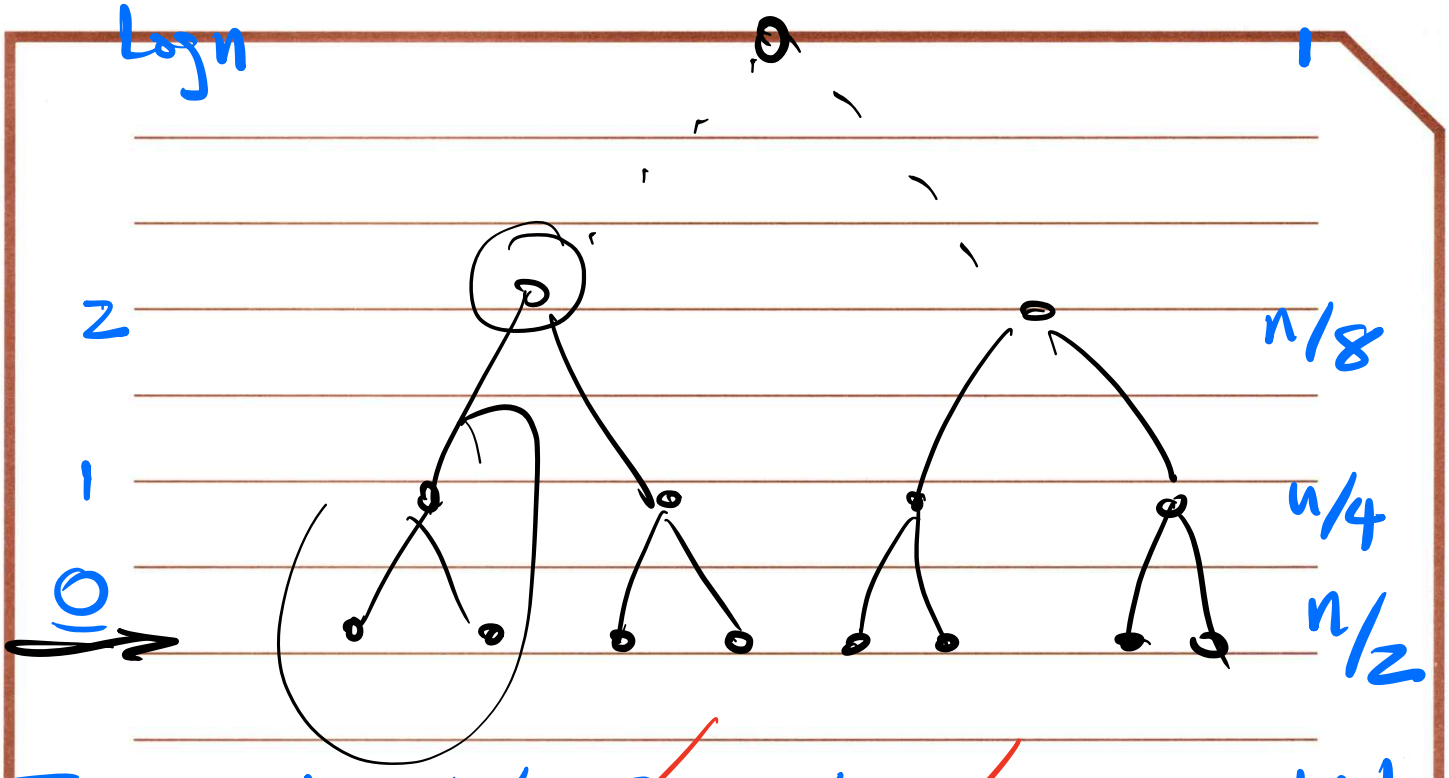


takes $O(\lg n)$

Construction of the binary heap

n insert op's $\longrightarrow O(n \lg n)$

bottom up construction:



$T = n/4 * 1 + n/8 * 2 + n/16 * 3 + \cdots 1 * \log n$

$T/2 = n/8 * 1 + n/16 * 2 + n/32 * 3 + \cdots$

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

$n/2$

$T/2 = n/2$

$T = n$

Merge two binary heaps of size n

takes linear time using

linear time construction

bottom up