Greedy Algorithm

1. Interval Scheduling Problem

(a) Problem Description

Input: Set of request {1,2....n}

ith request starts at s(i) and ends at f(i)

Objective: To find the **largest compatible subset** of these requests

(b) Several Failed Attempt

1. Earliest Start Time
2. Smallest Request First
3. Smallest Number of Overlap First
4. Earliest Finish Time

(c) Solution

Initially R is the complete set of request and A is empty

While R is not empty

Choose a request i∈R that has the **smallest finish time**

Add request i to A

Delete all request from R that are **not compatible with i**

End while

Return A

(d) Proof of Correctness

(i) Show that A is a compatible Set

Delete all requests that are not compatible with A could proof it

(ii) Show that A is an optimal set

First we need to prove that for all indices r ≤ k, we have $f(i_r) \leq f(j_r)$

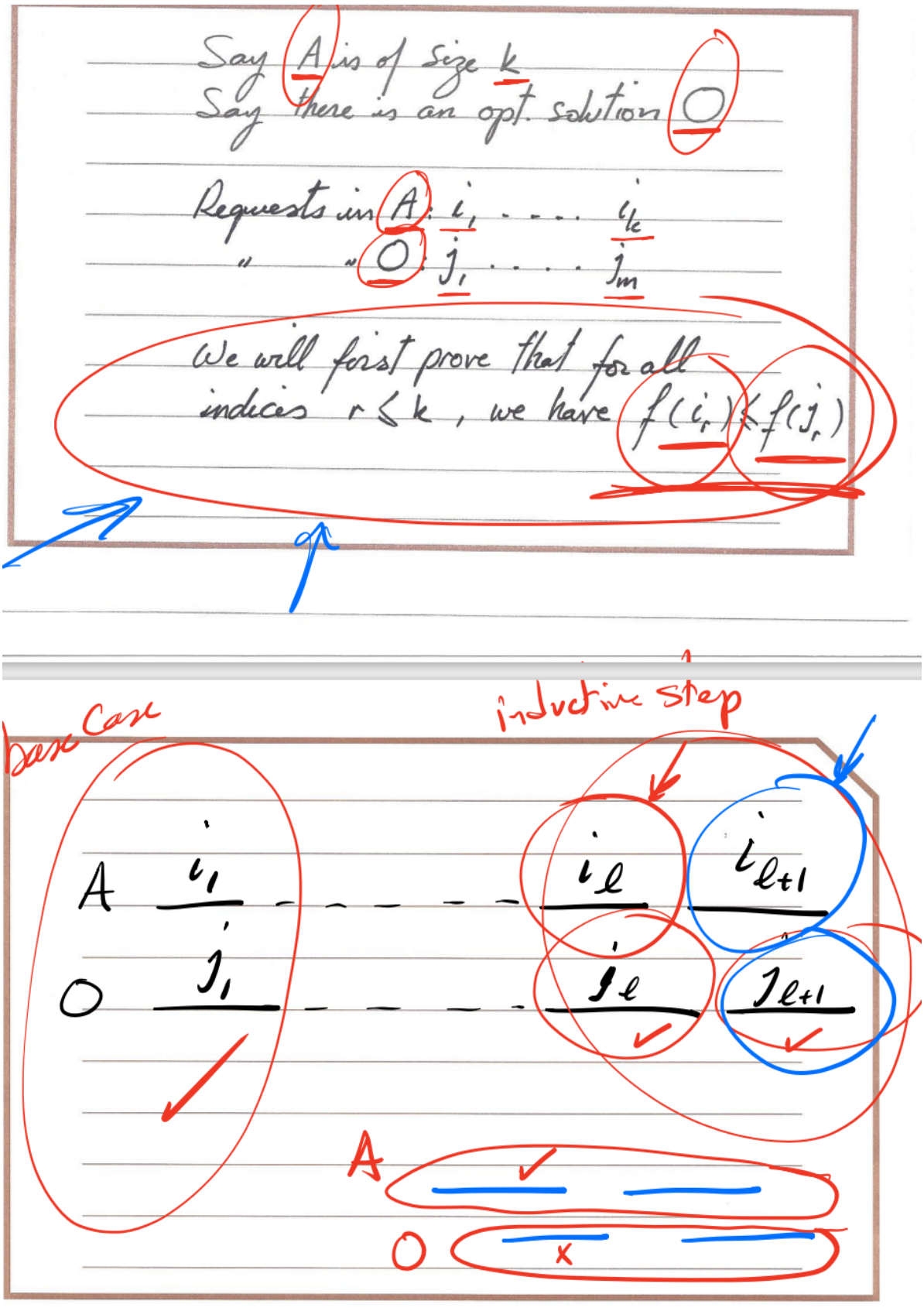
Mathematical Induction: 数学归纳法

Base Case: $f(i_1) \leq f(j_1)$

if j_{l+1} is compatible with j_l , then j_{l+1} must compatible with i_l

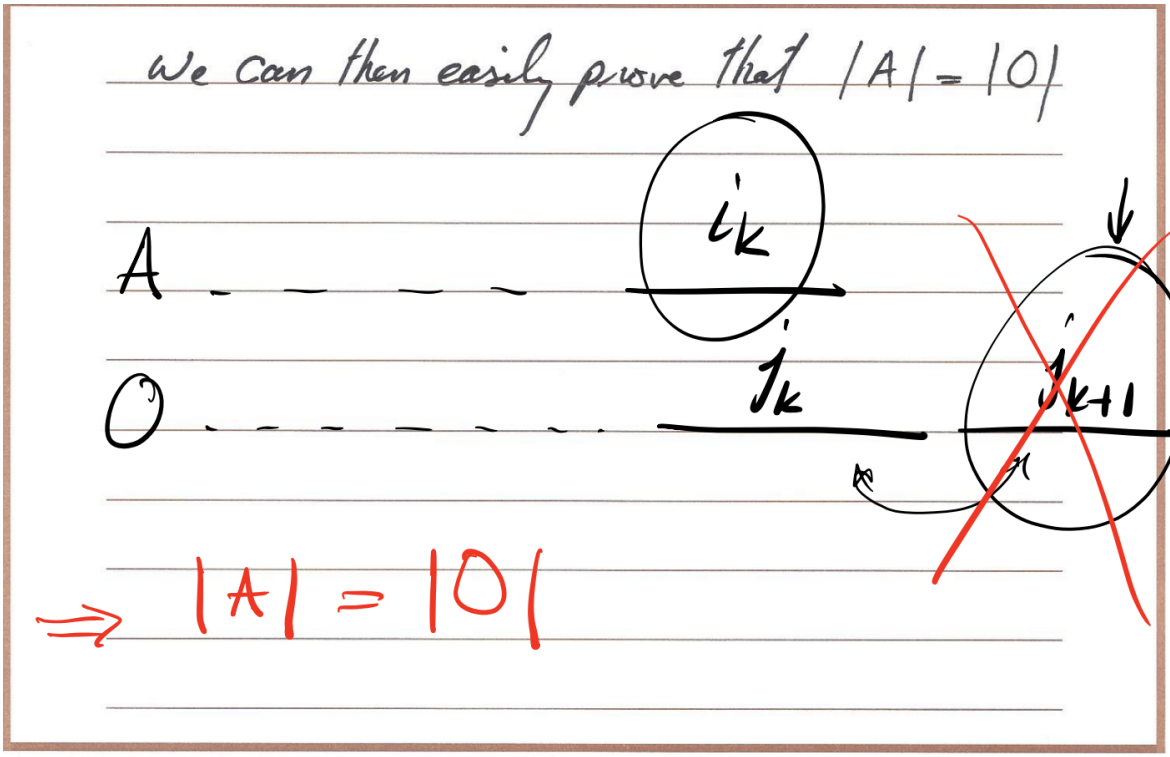
i_{l+1} is compatible with i_l and have the earliest time

so we have $f(i_{l+1}) \leq f(j_{l+1})$



Second we need to proof |A| = |O|

If there was a j_{k+1} out there, then it is must compatible with i_l , then it will be picked by A



(e) Implementation & Complexity

Sort request in **order of finish time** and label in their order **---- O(nlogn)**

Select request in order of increasing $f(i_j)$, always selecting the first. Then iterate through the intervals in their order **---- O(n)**

Overall Complexity = O(nlogn)

2. Scheduling to Minimize Lateness

(a) Problem Description

Request can be scheduled at any time

Each request has a deadline

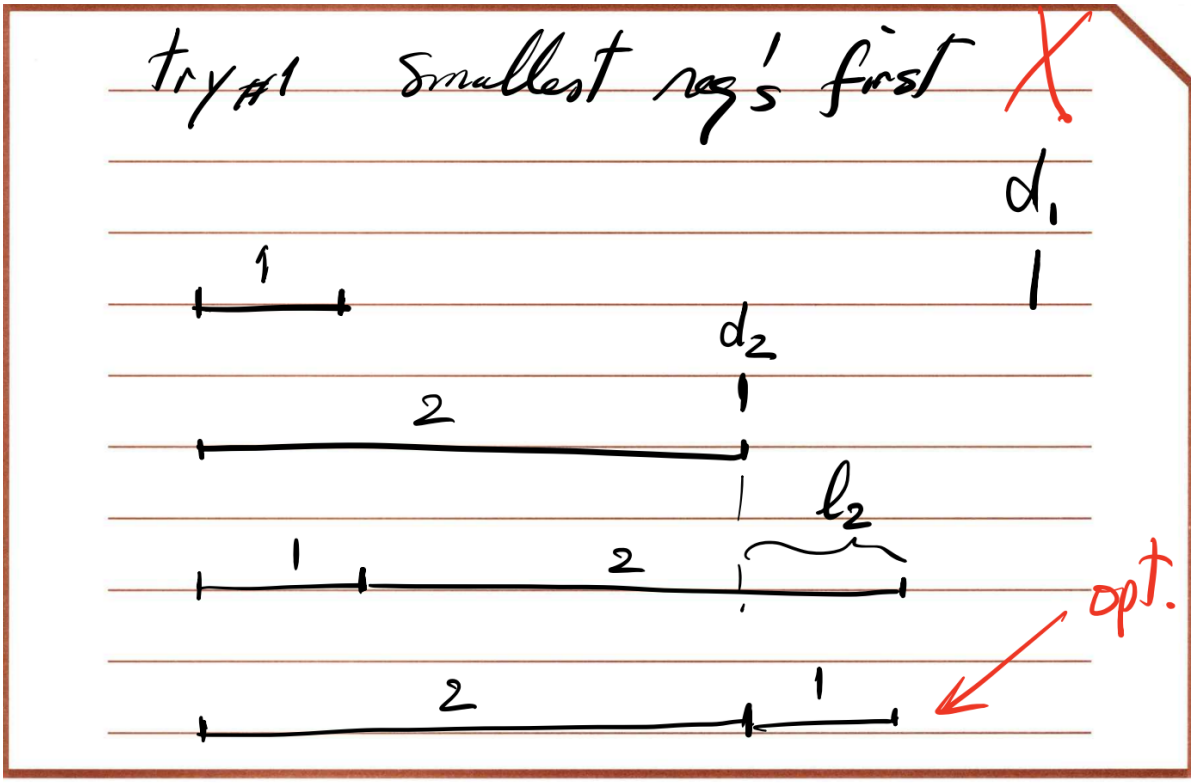
Notation: $L_i = f(i) - d(i)$

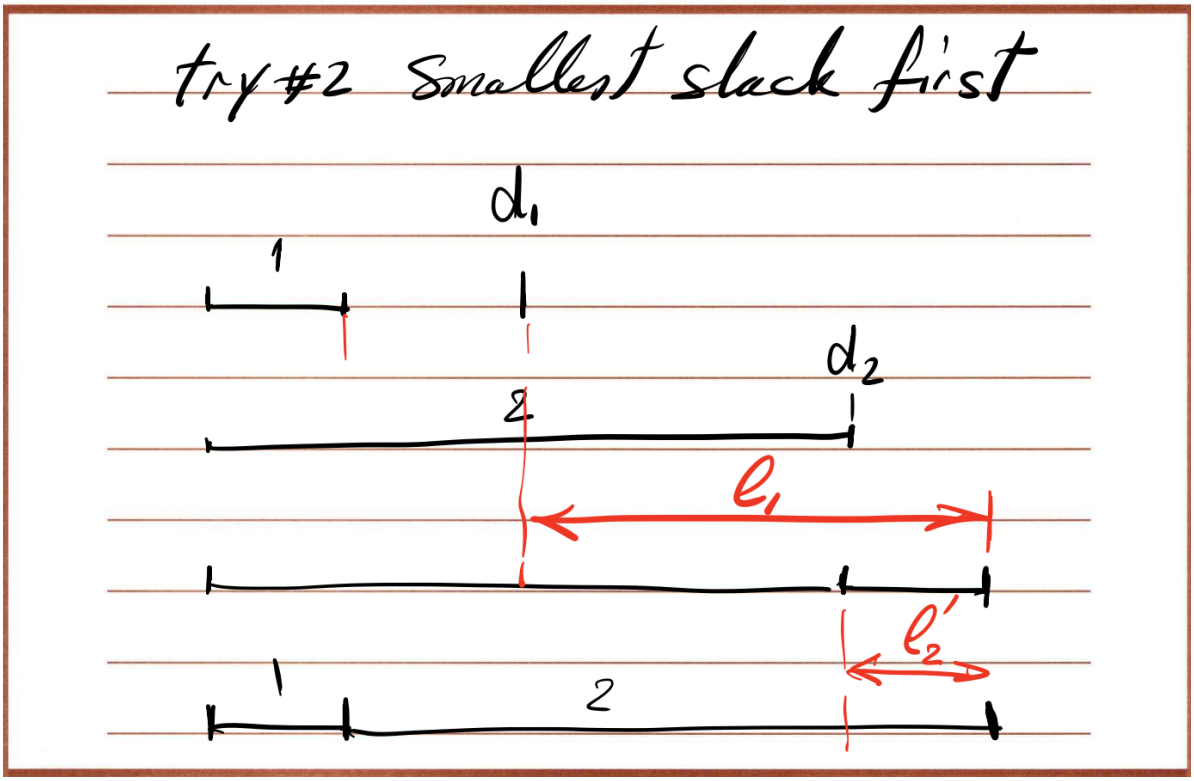
L_i is called lateness for request i

Goal: Minimize the Maximum Lateness L

(b) Several Failed Attempt

1. Smallest msg First
2. Smallest Slack First





(c) Solution

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

(d) Proof of Correctness

(i) There is an optimal solution with no gaps

remove the gap is safe

(ii) Jobs with identical deadline can be scheduled in any order without affecting Maximum Lateness

The lateness if fixed

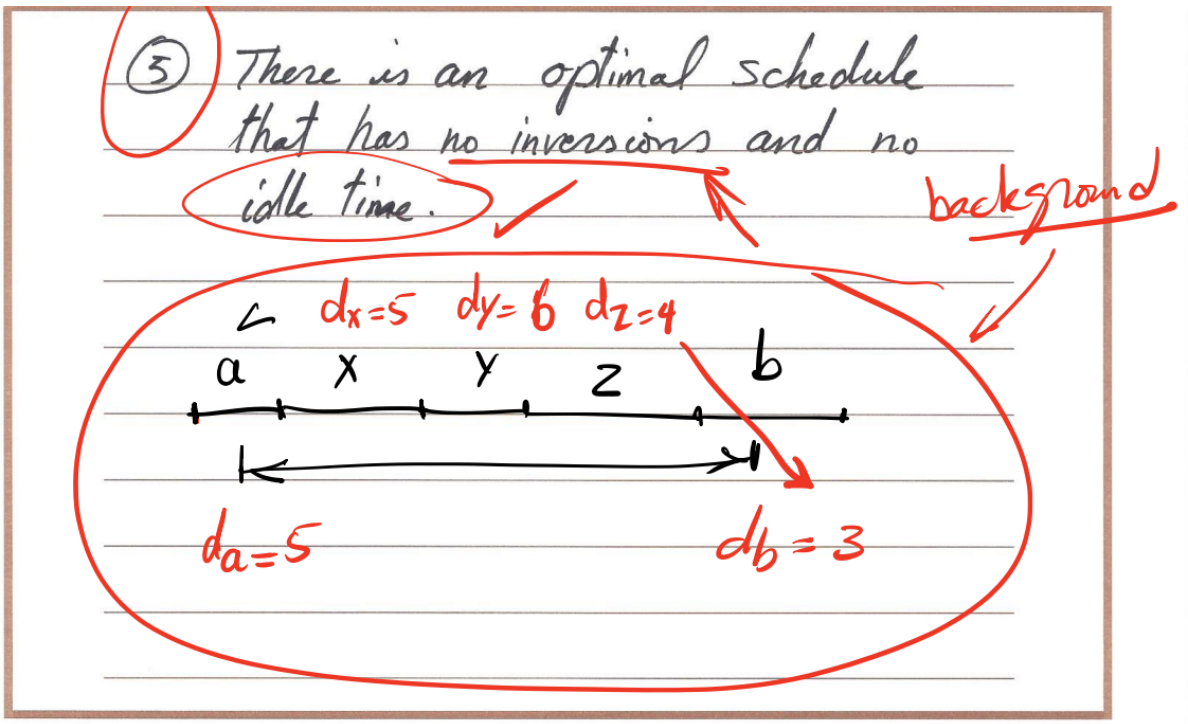
(iii) All schedules with no inversion and no idle time have the same maximum Lateness

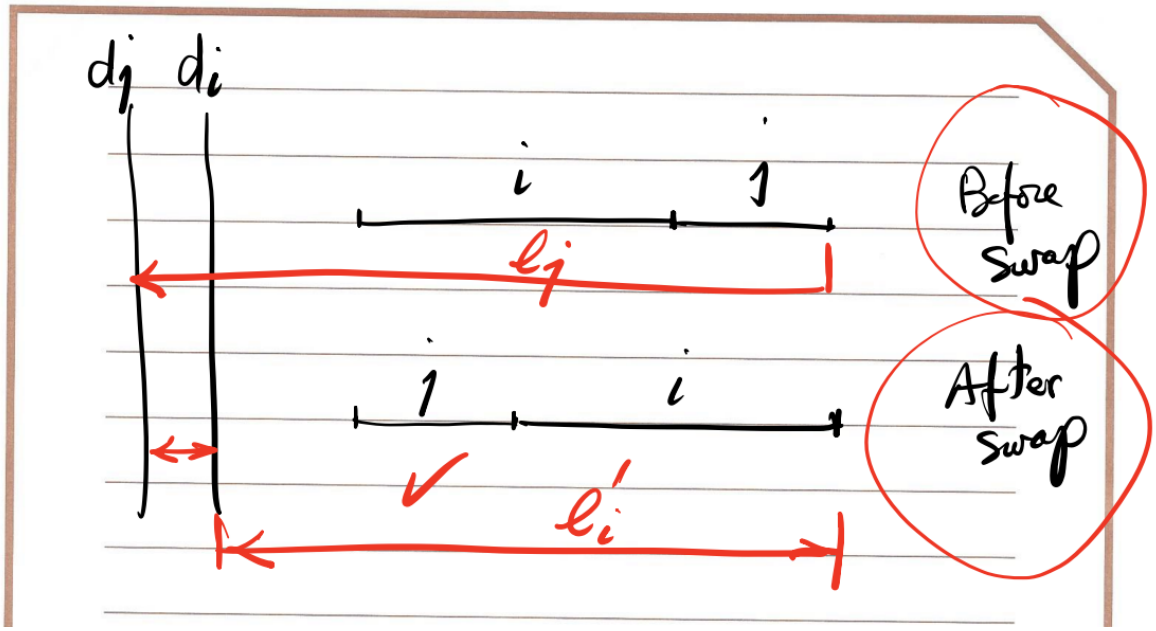
Def:

Schedule A' Has an inversion if job i with deadline di is scheduled before job j with **earliest deadline**

(iV) There is an optimal schedule that has no inversion and no idle time

- If there is an inversion between a & b, then there must be an inversion **between 2 adjacent nodes** among them(in this case, z & b)
- Swap the inversion is safe, Li after the swap will not be larger than the maximum Lateness before Swap





(V) Summary

Proved that there exists an optimal schedule with no inversion and no idle time

Also proved that all schedules with on inversion and no idle time have same maximum lateness

Our greedy algorithm produced on such solution -> It will be optimal

3. Priority Queue

(a) Problem Description

A priority queue has to perform these two operation fast

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

	Insert	Find-Min
Array Implementation	O(1)	O(n)
Sorted Array	O(n)	O(1)
Linked List	O(1)	O(n)
Sorted Linked List	O(n)	O(1)

I**nput: An unsorted array of length n**

Output: Top k values in the array (k<n)

Constraints:

- You cannot use any additional memory
- Find an algorithm that runs in line

(b) Binary Tree & Binary Heap

(i) Binary Tree

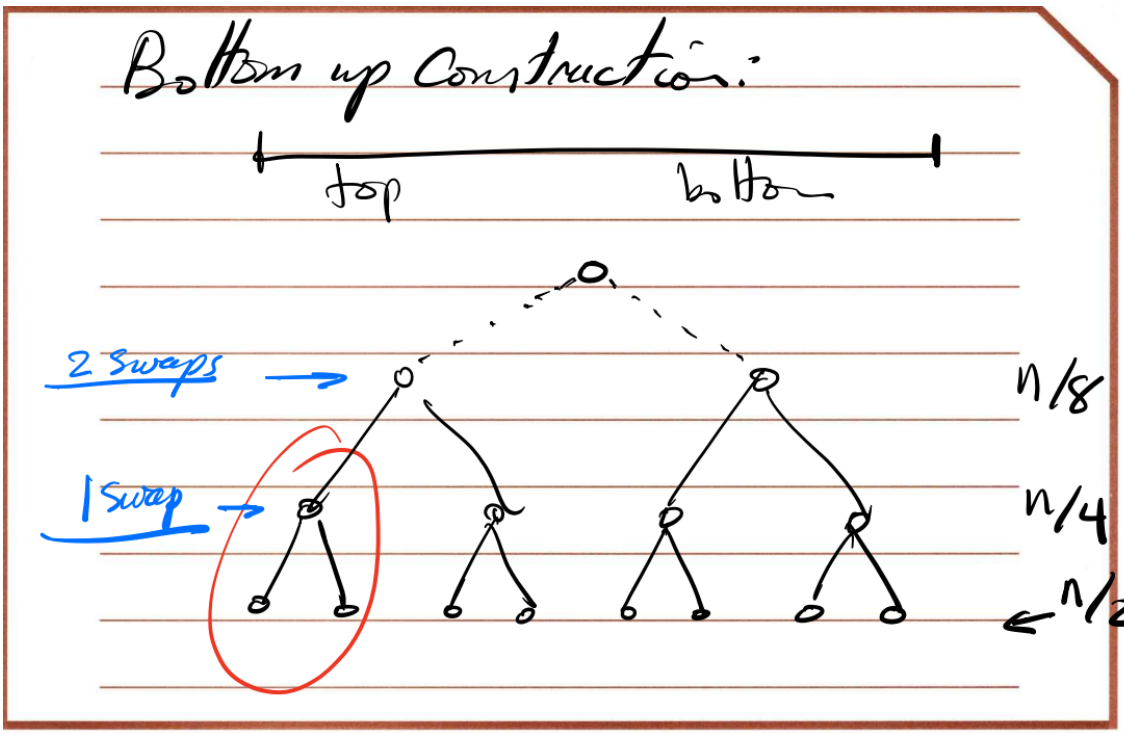
A binary tree is a tree data structure in which each node has at most two children, which are referred to as the *left child* and the *right child*.

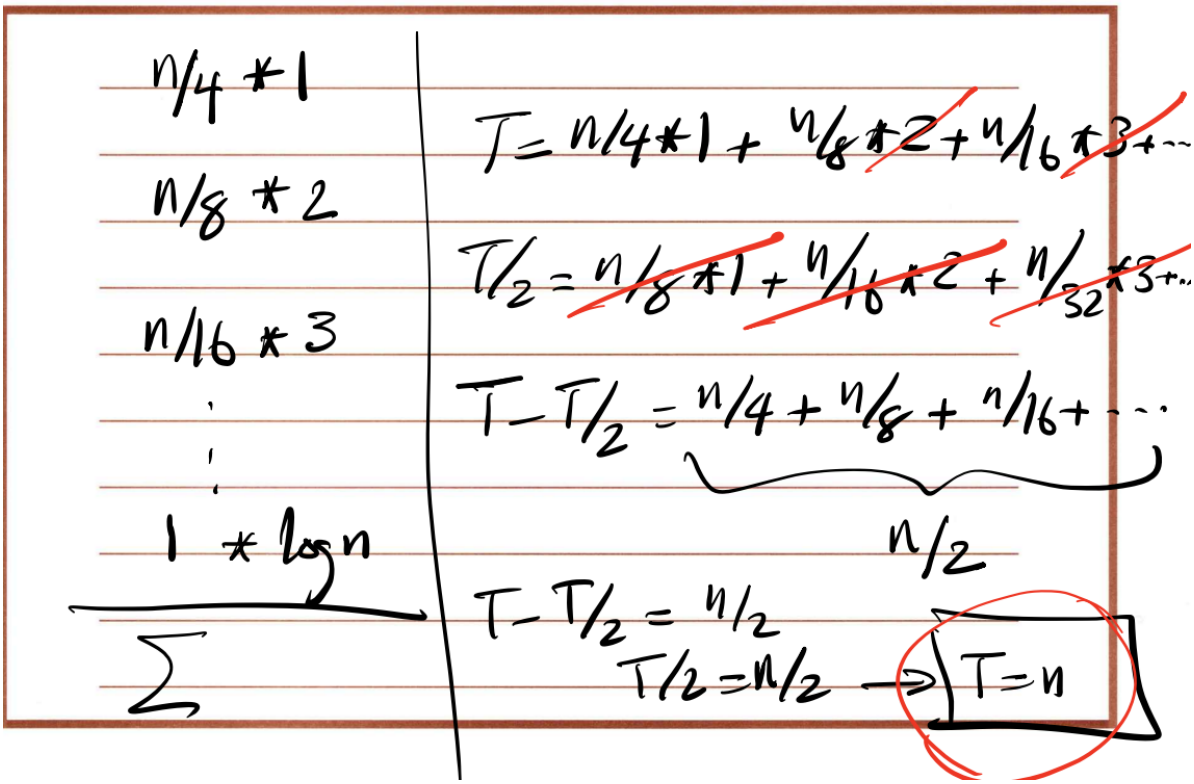
A binary tree of depth k which has exactly 2^k-1 nodes is called a **full binary tree**. A binary tree with n nodes and of depth k is complete if its nodes correspond to the nodes which are numbered 1 to n in the full binary tree of depth k

(ii) Binary Heap

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 value as its children **(Max Heap)**

	Binary Heap	Binominal Heap	Fibonacci Heap
Find-Min	O(1)	O(Log(n))	O(1)
Insert	O(Log(n))	O(Log(n))	O(1)
Extract-Min	O(Log(n))	O(Log(n))	O(Log(n))
Delete	O(Log(n))	O(Log(n))	O(Log(n))
Decerease-Key	O(Log(n))	O(Log(n))	O(1)
Merge	O(Log(n))	O(Log(n))	O(1)
Construct	O(n)	O(n)	O(n)

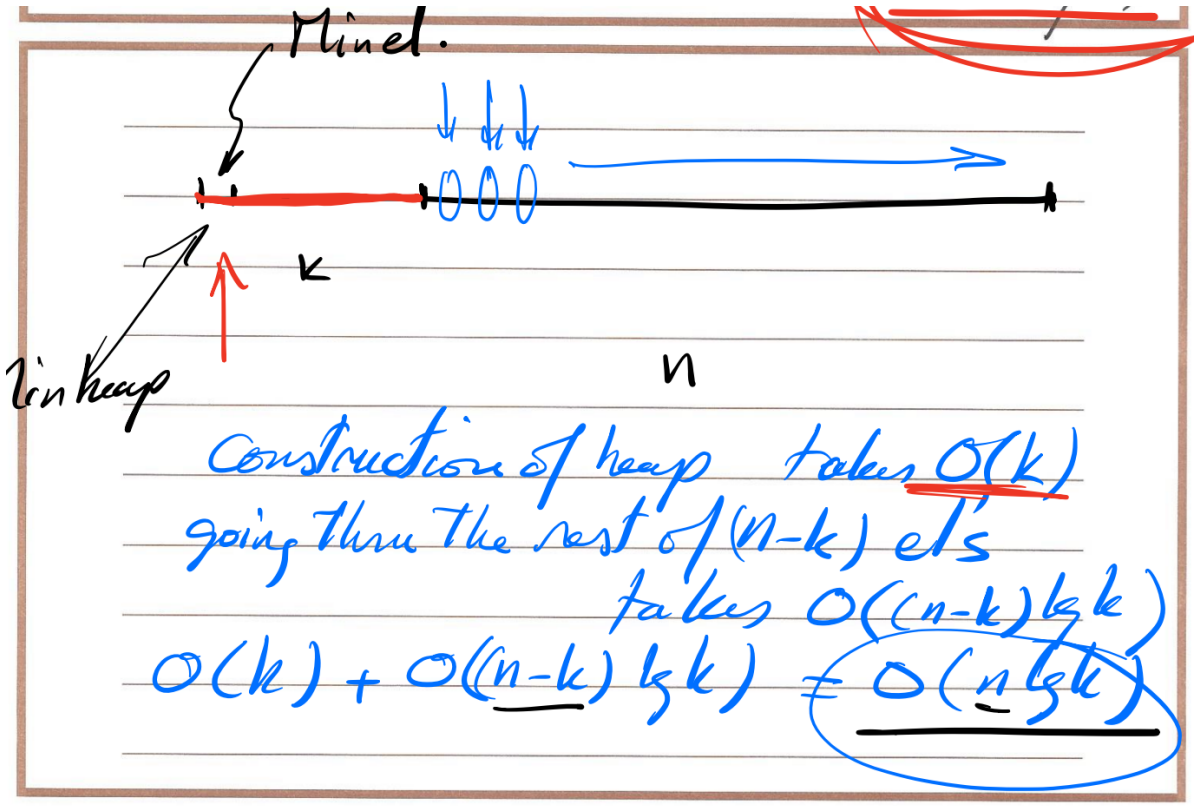




(c) Implementation & Complexity

1. Constructions of heap **---- O(k)**
2. Going through the rest of n-k values **---- O((n-k)logk)**

Overall complexity = **O(nlogk)**



Construction of heap takes $O(k)$

going thru the rest of $(n-k)$ el's takes $O((n-k)\lg k)$

$O(k) + O((n-k)\lg k) = O(n \lg k)$