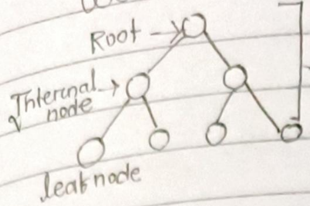
Binary Tree and it's Variations

we know that



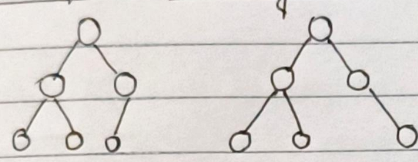
1. That's a [Binary tree] → At most 2 children of each node

2. Precisely that's a [Full Binary tree] [strict or] → (Either 0 or 2 children)

Now,
3. [Complete Binary tree] (CBT) → [No Holes] → (internal nodes of same level must have two children)

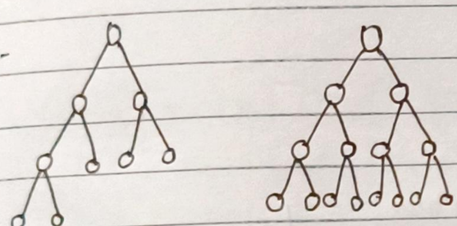
4. Almost complete Binary tree :- [Fill from left to right] Imp

Ex:-



(✓) (X)

Exe-



(not CBT) (CBT)

INTRODUCTION TO "HEAP" TREE :-

heap tree (It is a Almost complete binary tree)

If a tree is heap-tree it must follow

I. Structural property

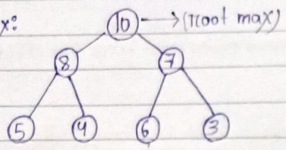
II. Ordering property

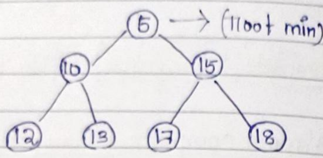
heap
(two type)

Maxheap
(parent > child)
* (we will use this most of the time)

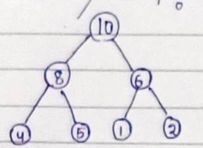
Min heap
(parent < child)

Ex:





Q1. Which one is max-heap?
Ans-(B)



Insertion in heap tree

Construction in heap tree

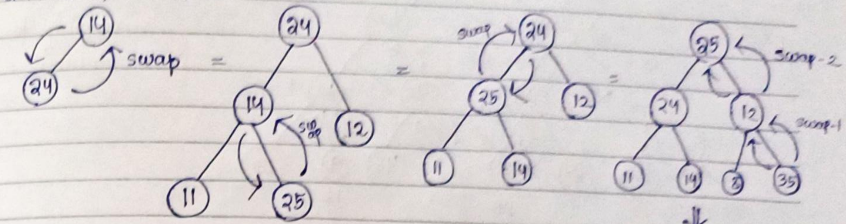
Insert key [one by one] in given order → O(n log n)

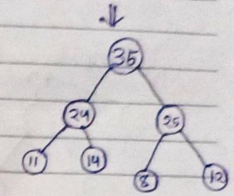
Heapify method O(n)

S	M	T	W	T	F	S
						1
		4	5	6	7	8
		11	12	13	14	15
		18	19	20	21	22
		25	26	27	28	29

Deletion in heap tree →

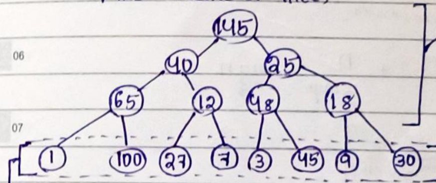
Example: [max heap let's see]
14, 24, 12, 11, 25, 8, 35
So,





(a) This is one-by-one construction
To,
[insert 1 element in heap tree : T = O(log n)
n elements in heap tree: T = O(n log n)]

(b) Heapify method O(n) → (make min heap)
Ex: 145, 40, 25, 65, 12, 48, 18, 1, 100, 27, 7, 3, 45, 9, 30
First make a tree.

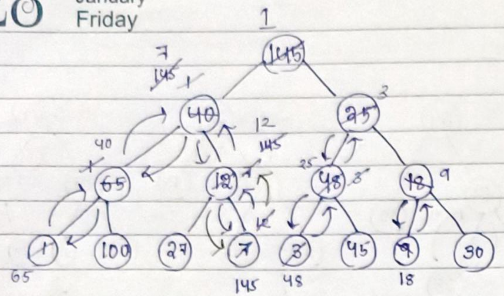


→ swapping will happen here,
so in next page.

Don't do swaping in leaf node
number of elements = n
leaf = n/2 → ignore them
by [n = mi + 1] and [n = l + i]
[m = 2] → here

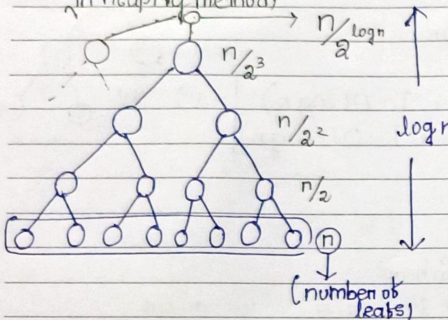
FEBRUARY '22

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28					



Maximum number of swapping for n elements = log n

In heapify method,



here,

Total swap = S

$$S = \frac{n}{2^0} * 0 + \frac{n}{2^1} * 1 + \frac{n}{2^2} * \overbrace{2}^{\text{swapping possible}} + \cdots\cdots + \frac{n}{2^{\log n}} * \log n$$

(number of nodes in a level)

$$S = n\left[\frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \cdots\cdots + \frac{\log n}{2^{\log n}}\right] \text{——— (i)}$$

here,

multiply eqⁿ I by ½,

$$\frac{S}{2} = n\left[\frac{1}{2} + \frac{2}{2^3} + \frac{3}{2^4} + \frac{4}{2^5} + \cdots + \frac{\log(n-1)}{2^{\log n}} + \frac{\log n}{2^{\log(n+1)}}\right] \text{——— (ii)}$$

Substract (ii) from (i)

$$S - \frac{S}{2} = \frac{S}{2} = n\left[\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots + \frac{1}{2^{\log n}} - \frac{\log n}{2^{\log(n+1)}}\right]$$

$$= n\left[\left[\frac{\frac{1}{2}\left(1 - \frac{1}{2^{\log n}}\right)}{\frac{1}{2}}\right] - \frac{\log n}{2^{\log(n+1)}}\right]$$

$$\left[S = \frac{a \cdot (1 - r^n)}{1 - r}\right]$$

Sum of G.P values

$$\left[\begin{aligned} a &= 1/2 \\ r &= 1/2 \\ n &= \log n \end{aligned}\right]$$

$$= n\left[\frac{2^{\log n} - 1}{2^{\log n}} - \frac{\log n}{2^{\log(n+1)}}\right]$$

$$\left[\because 2^{\log n} = n^{\log 2}\right]$$

$$\Rightarrow \frac{S}{2} = n\left[\left[\frac{n-1}{n}\right] - \frac{\log n}{2n}\right]$$

$$\frac{S}{2} = (n-1) - \frac{\log n}{2}$$

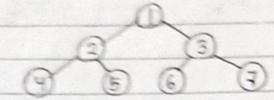
$$\Rightarrow S = (2n-2) - \log n = \overbrace{2n}^{\text{bigger term}} - \log n - 2$$

So, Time complexity, O(n)

Deletion in Heap tree

(always follows Almost complete binary tree)

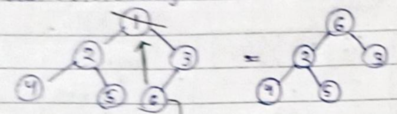
Let's take a Example of min heap.



(Best case)

(a) → here we can delete Right most leaf node here, ex: 7 ⇒ $T = O(n)$

(b) → To Remove Root-



(Replacement of root is rightmost leaf)

But it is not min heap

⇓

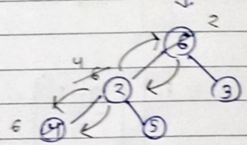
(do swapping)

here time complexity

$\boxed{T = O(\log n)}$

↓

(Worst case)



log n

→ $T(n) = (n \log n)$

Heap sort:- [uses queue data structure]

Rearrange the heap tree from the array

↓ max/min

Build ~~min~~ head using heapify method

↓

Delete root element by directly swapping with last node of the min heap.

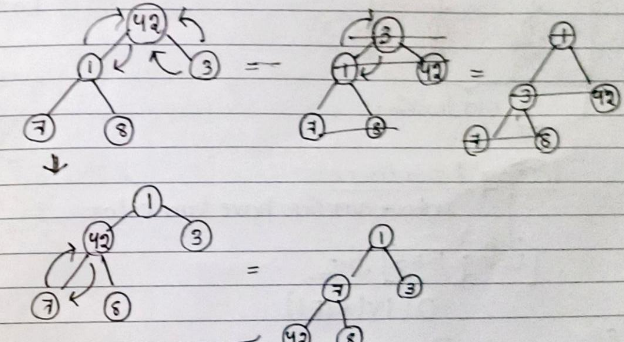
↓

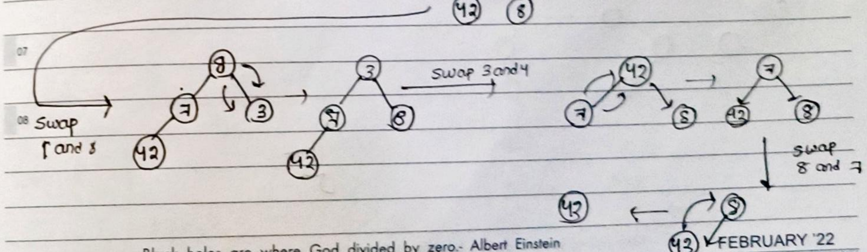
Store the deleted element in an array from left to right manner.

ex:

42	1	3	7	8

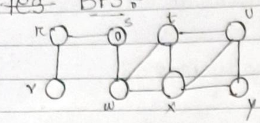
1	3	7	8	42





graph

Tes BFS:



[S]

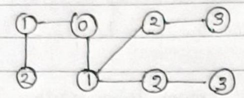
[r | w]

[v | t | x]

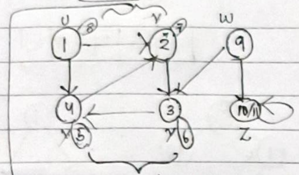
[v | u | y]

[u | y]

[y]



DFS



Backtracking

→ No Backtracking in BFS But Back tracking in DFS

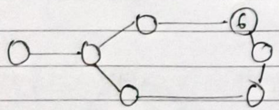
Testing Bipartiteness

nabour vertices have same color

[Using BFS] Imp

O(|V|+|E|)

Ex:



Topological sorting or Topological Ordering

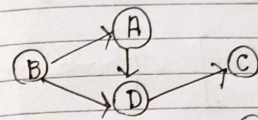
Linear ordering of graph Vertices

for every directed edge UV from U to V

(U come comes Before V)

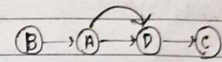
→ applicable on [DAG] → Directed Acyclic graph

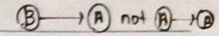
Ex:



Topological order = [BADC] ✓

ABCD (X) → cause

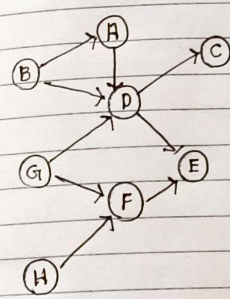




→ we can use DFS (mainly) to write this

also can use BFS

Ex:



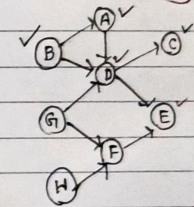
→ Let's first use DFS of example

take the node B

so

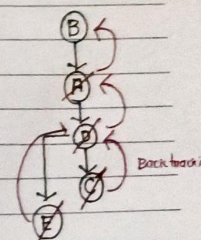
B
A
D
E
C

stack.



(DFS-I)

=



Now,

H
G
F
B
A
D
E
C

→ (Stack)

(DFS-II)

H

DFS-III

Now,

Ans: HGFBADEC

here $T(n) = O(V+E)$

Space complexity = $O(V)$

Note→

Data structure [BFS use :- Queue
DFS use :- Stack]

{For Topological sort always start from the node with indegree = 0} → (Important)

[can see the (Cormen) algorithm on node with outdegree = 0] → (check notes)

Spanning tree→

at least one or unique spanning Tree in a connected graph.

Minimum spanning tree

→ Spanning tree with minimum total cost (For weighted)

→ [Edge = |V| - 1]

Prim's Algorithm

a greedy algorithm

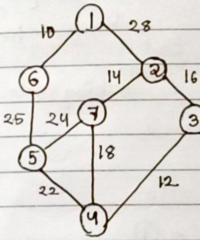
starts with empty spanning tree

→ Idea is to maintain two sets of vertices.

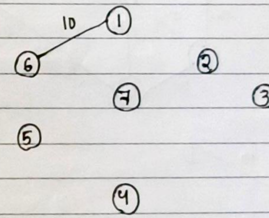
→ contains V included in MST

→ V not yet included.

Ex:-



Start with minimum weight. then take minimum connected to to 1st one

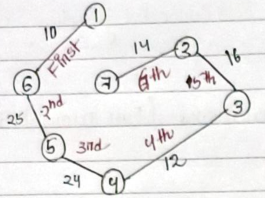


Sunday 06

(Note - always maintain a tree)

→ [Don't form cycle]

next page



cost = 10 + 25 + 24 + 12 + 14 + 16 = 99

Imp { always choose the connected smallest edge
not disconnected smallest edge }

→ Prim's algorithm is applicable only for connected graph.

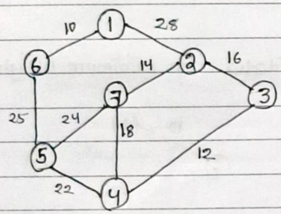
Kruskal's Algorithm

↳ A greedy approach

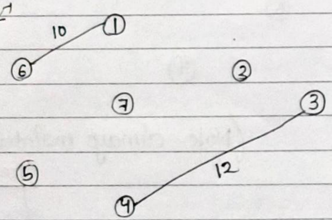
↳ "Always select a minimum cost edge" [The don't need to be connected at first]

But avoid making cycle

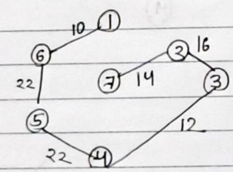
Ex:-



here



Step-1



[step-2, 3, 4, 5]

The capacity for friendship is God's way of apologizing for our families. -Jay McInerney

FEBRUARY '22

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28					

Important points

Time complexity.

$O(|V||E|) = O(n \cdot e) = O(n^2)$ → using adjacency matrix

cause $[n(n-1) = n^2 + n]$

For In the following case we are using min heap

So,

[Time complexity = $\Theta(E \log E)$
$= O(n \log n)$ or $O(V \log V)$]

(Remember)

Using Binary heap ~~and~~ adjacency list

→ Same goes for prim's algorithm

Union-find data structure

I am not a product of my circumstances. I am a product of my decisions. Stephen Covey

MARCH '22

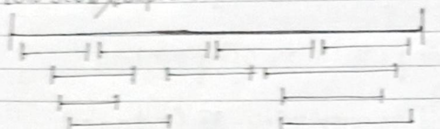
S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

greedy method

~~R~~

Interval scheduling:-

here, lets see Example→



An optimal greedy rule→

we should accept first the request/Interval that $\boxed{\text{finishes first.}}$ → (i.e) The request(i) for which f(i) is as small as possible

(Imp)

then

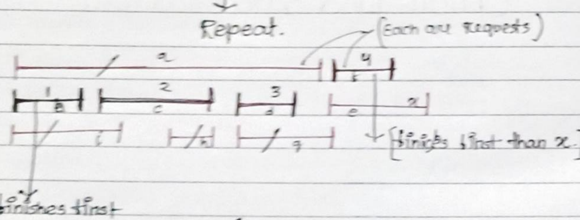
↓

cancel the processes that are not compatible with it

↓

Repeat.

Example→



here Ans→ [b, c, d, e]

Step by step:-

✓ Selecting Interval in each step (in notes)

→ ~~Time complexity O(n)~~

Algorithm → (pseudocode)

here

→ Initially let R be the $\boxed{\text{set of all requests,}}$ and let $\boxed{\text{A be empty}}$ $\boxed{\text{while}}$ R is not yet empty

→ choose a request, $\boxed{i \in R}$ that has the smallest finishing time → (while loop)

→ ~~Add~~ $\boxed{\text{request } i}$ to A (i is added to A)

→ Delete all request from R that are not Compatible with request i

$\boxed{\text{End while}}$

Return the set A as the set of accepted Request

~~Proof of that~~ → greedy approach

→ Prove that this approach is optimal?

Ans: Let O be an optimal set of Intervals

From the psudocode on the algorithm we have set A

A = our derived optimal Intervals.

We need to prove that

O = A or A is a subset of O

let, $A = [i_1, i_2, i_3, \ldots, i_k]$

and $|A| = K$ —— (i)

Similarly: $O = [j_1, j_2, j_3, \ldots, j_m] \Rightarrow |O| = m$ —— (ii)

Assume that,

intervals in O are also ordered in left-to-right direction in order of the start and finish points.

Every interval in O is compatible

↓

the start point has the same order as the end point.

In greedy method, $\boxed{\text{The interval finished first}}$ is the first element and so on.

So, by Greedy method, $f(i_1) = f(j_1)$

So, The approach is optimal.

time complexity of greedy method in Interval scheduling

1. $T(n) = O(n\log n)$ —— (i)

sorting n requests to finishing time and labeling

so, $f(i) \leq f(j)$ when $i<j$

This take $O(n\log n)$

2. additional time $O(n)$ —— (ii)

to construct array the stores compatible Intervals

Scheduling to Minimize Lateness with proof of optimality using An Exchange Argument.

Scheduling to minimize Latterness / Job sequencing with deadline

Suppose there are

→ $\boxed{\text{Single resource}}$ and a set of $\boxed{\text{n requests}}$

→ need to use requests to use resource for an Interval of time.

→ The Resource is available $\boxed{\text{at time s (starting)}}$

→ There is no start point and end point in Interval/requests.
The request i has deadline $\boxed{d_i}$ } (imp)

→ It requires → Time interval length $\boxed{t_i}$

→ can be scheduled at any time before deadline

Time complexity $O(n\log n)$

Example:

→ n=5 requests

here,

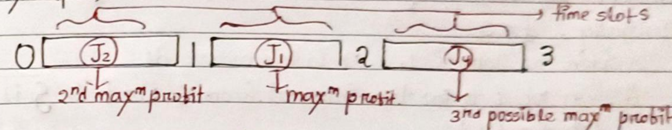
request :	J_1	J_2	J_3	J_4	J_5	
profit :	20	15	10	5	1	
deadlines :	2	2	1	3	3	(minutes/hours)

Rejected

[In minimize Latterness we need to maximize the profit or results] → Note

As it is a optimization issue we need greedy approach

In the example,



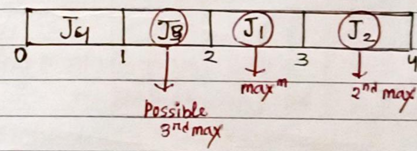
So the maximized Job to do = $\{J_2, J_1, J_4\}$ or $\{J_1, J_2, J_4\}$

∴ Total profit = 20 + 15 + 5 = 40

Example-2

request :-	J_1	J_2	J_3	J_4	J_5	J_6	J_7
profit :-	35	30	25	20	15	12	5
deadline :-	3	4	4	2	3	1	2

here



profit = ~~30 + 35 + 20 + 12~~
= ~~97~~

30 + 35 + 25 + 20
= 110 Ans

Optimally → We simply sort the Jobs in increasing order of their dead line.

[Earlieast deadline First]

Pseudocode→

Order the Jobs in order of their deadlines

Assume for simplicity of notation $d_1 \leqslant d_2 \leqslant d_3 \leqslant \ldots \leqslant d_n$

Initially, $f = s$

Consider the Jobs $i = 1, \ldots n$ in this order d_1 ... d_n

Assign Job i to the time interval from $s(i) = f$ to $f(i) = f + t_i$

Let $f = f + t_i$

end

Return the of sheduled intervals $[s(i), f(i)]$ for $i = 1, \ldots, n$

Optimal catching

Before that understand

Exchange Argument :-

Are powerful and Versatile

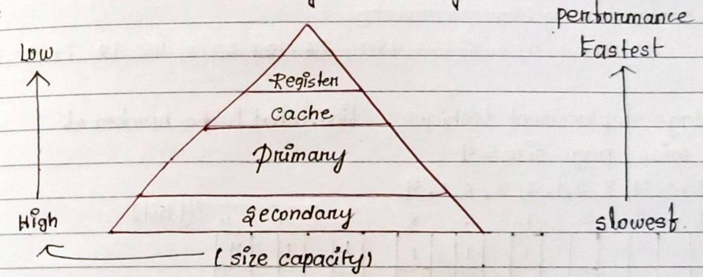
can Iteratively transform any optimal solⁿ into $\boxed{\text{sol}^n \text{ produced by greedy algorithm}}$

(without changing the cost of optimal solⁿ)

That's the reason greedy solⁿ is optimal.

for, Now optimal catching

Need to know memory hierarchy :-



Terms→

Caching :- storing small amount of data in a fast memory so as to reduce the amount of time spent interacting with slow memory

→ cache with capacity to store k items

→ sequence of m item requests $d_1, d_2, d_3, \ldots d_m$

Cache hit :- Item already in cache when requested.

Cache miss :- Item not in cache where requested

Imp

Our gole is to minimize the number of misses. → For that do cache replacement

[Optimal offline caching is :- Farthest in future (~~FIFO~~)]

→ replacement of data in cache.

Cache replacement ALGORITHMS

- FIFO — (First in first out)
- LRU — (Least recently used)
- MRU — (most recently used)
- OCF → (Farthest in future)

Statement: Consider a 4 way set associative cache with total 16 cache blocks.
main memory block requests are:
0, 225, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155

~~Ans:~~

Q. Using page replacement techiques, find what is the number of hit and miss, page size = 4

Data:- 4, 3, 2, 1, 7, 8, 6, 1, 4

(a) FIFO

4	3	2	1	7	8	6	1	4
			1	1	1	1	1	4
		2	2	2	2	6	6	6
	3	3	3	3	8	8	8	8
4	4	4	4	7	7	7	7	7

(1 hit)

(b) LRU

			1
		2	2
	3	3	3
4	4	4	4