Algorithm - SN

- Time and Space analysis –

O → Worst case

Ω → best case

θ → Average case.

→ Any program than can be written using recursion could be written using iteration.

→ if there is no iteration & recursion inside the program = $O(1)$

→ $\boxed{1+2+3+\cdots+n = \frac{n(n+1)}{2}}$ → $\boxed{1^2+2^2+\cdots+n^2 = \frac{n(n+1)(2n+1)}{6}}$

→ for(i=1; i<n; i=i*2) → Ody, $O(\log_2 n)$

3 → $O(\log_3 n)$

→ $\boxed{1+\frac{1}{2}+\cdots\frac{1}{n} = \log n}$

→ Using Back substitution :

```
A(n)
{if(n>1)
    return(A(n-1));}
```

→ $T(n) = 1+T(n-1)$; $n>1$

$= 1$; $n=1$

↳ $T(n) = ?$ —(1)
$T(n-1) = ?$ —(2)
$T(n-2) = ?$ —(3)
→ Put into equation (1)

→ Comparing various function to analysis time complexity –

→ 2^n | n^2

$\log 2^n$ | $\log n^2$

$n\log 2$ | $2\log n$

$\boxed{2^n > n^2}$ put, $n = 2^{128}$

- Masters theorem – (to analyse time complexity)

→ $\boxed{T(n) = aT(n/b) + \theta(n^K \log^P n)}$

$a \geq 1$, $b > 1$, $K \geq 0$ and p is real number.

→ (i) if $a > b^K$; then $T(n) = \theta(n^{\log_b^a})$

→ (ii) if $a = b^K$

→ a) if $P > -1$; then $T(n) = \theta(n^{\log_b^a} \log^{P+1} n)$

b) if $P = -1$; then $T(n) = \theta(n^{\log_b^a} \log\log n)$

c) if $P < -1$; then $T(n) = \theta(n^{\log_b^a})$

→ (iii) if $a < b^K$

a) if $P \geq 0$; then $T(n) = \theta(n^K \log^P n)$

b) if $P < 0$; then $T(n) = O(n^K)$

- Analysis space complexity of iterative & recursive Algo -

iter →

- int i; → O(1)
- int i) int j=10; → O(2)

- int i → (n+1)
  Creat B[n]
- B[n,n] = n^2

rec →

Space complexity is depth of the tree.

** • Sorting Technique :

- Insertion short :

Worst = $O(n^2)$ → reversed array

best TC = $\Omega(n)$ → sorted array.

- Merge short :

T.C to merge = O(n)

each level work done ↑ × no. of level ↑

→ n-element merge short T.C = $O(n \log n)$

→ n string each length n T.C = $O(n^2 \log n)$.

- quick short :

→ element assending or decending T.C = $O(n^2)$.

→ all I/p is same then T.C = $O(n^2)$.

- heaps :

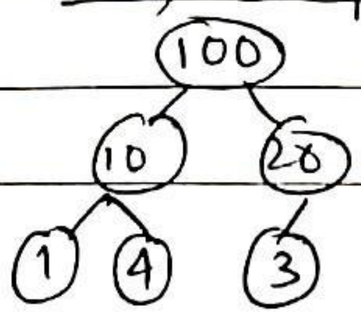
	Insert	Seach	Find min	Delete min
Unsorted array	O(1)	O(n)	O(n)	O(n)
sorted array	O(n)	O(log n)	O(1)	O(n)
unsorted linklist	O(1)	O(n)	O(n)	O(n)
min heap	O(log n)		O(1)	O(log n)

→ used to optimize T.C.

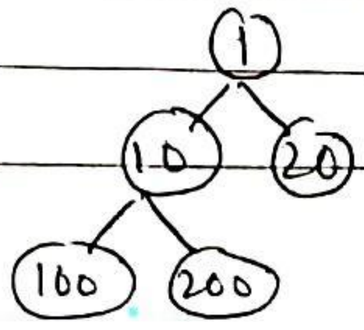
→ heap → min heap
→ max heap.

→ heap can implement as binary tree or 3-ary .. n-ary tree.

Max heap

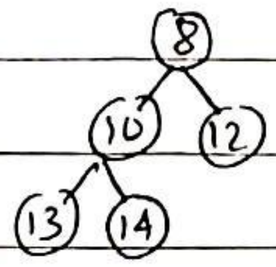


min heap



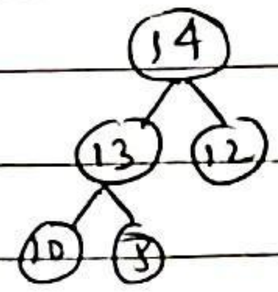
→ array assending order (Min heap)

8	10	12	13	14



array decending order (max heap)

14	13	12	10	8



** → max no. of node in complete binary tree = $(2^{h+1}-1)$

h → tree hight.

→ 'n' nodes inside a complete or almost complete binary tree, then hight of tree = $\lfloor \log n \rfloor$.

- Max-HEAPIFY : (Max heapify algo)

  S.C = O(logn)

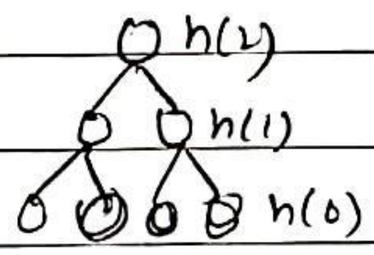
  T.C = O(logn).

- Buil-Max heap algo :

  S.C = O(logn)

  T.C = O(n).

** → Max no. of node present in hight 'h' = $\left\lceil \frac{n}{2^{h+1}} \right\rceil$



n → total node.

- Extract max from MAX heap :

  S.C = O(logn)

  T.C = O(logn)

- Insert key into max heap :

  T.C = O(logn)

* Max heap

→ Find max	Delete max	Insert	Key increase	Key decreare
O(1)	O(logn)	O(logn)	O(logn)	O(logn)

→ Find min	Search random element	delete random element
Θ(n)	O(n)	O(n).

- Heap short :

  Time complexity = O(nlogn)

- Greedy algorithm –

→ spanning tree $\boxed{K_n \rightarrow n^{n-2}}$ → When a complete graph

K_n: no. of vertices; n^{n-2}: no. of ST possible

→ Find min cost spanning tree – (PRIMS Algo)

Step:

S-1 → 1st Draw all the vertices.

S-2 → Connect by min weight edge.

- Primes Algo : T.C → $O(n^2) \rightarrow O(V^2)$

→ cost of spanning tree same both the cases prime & Kruskal's

- DISKSTRA ALGO –

→ diskstra algo not applicable for negative edges.

- Bellman ford Algo –

Time complexity = $O(VE)$ V → vertices; E = Edges

- Shortest path algo – (Directed acyclic graph)

T.C = $(V+E)$

- Dynamic programming –

→ Matrix multiplication –

$[\ ]_{p \times q} \times [\ ]_{q \times r}$

* ↳ total no. of multiplication req = $\boxed{p \times q \times r}$ **

ex: $(A_{2\times1} \times B_{1\times2}) C_{2\times4}$ → MR = 4

$\Rightarrow (D_{2\times2} \times C_{2\times4})$ — MR = 16

$\Rightarrow E_{2\times4}$

→ total multiplication req = 4 + 16 = 20

→ Matrix A B C D E

A(BCDE) — 5 way

(AB)(CDE) — 1 × 2 way

(ABC)(DE) — 2 × 1

(ABCD)(E) — 5 way

no. of way com parenthesis = $\frac{(2n)!}{(n+1)!\,n!}$

↳ if multiply 4-Mtx then put n = 3

↳ if multiply 5-Matrix then put n = 4.

→ Dynamic programming bottom up / Topdown Implementation =

T.C = $O(n^3)$ & S.C = $O(n^2)$