Unit-1

DAA [1Marks]

1) Algorithm:- It is a step by step process of how the program should be executed or defined. It is used by the end users.

Condition:- If zero inputs will be given then the o/p shd'nt be zero

Algorithm should be simple, effective & time saving

Algorithm
- what [define]
- where [used]
- how [condition]

2) Time complexity:- Amount of time does it take (or) required to run the algorithm

3) Space:- How much space required

4) Uses of Big-O notation, it is used for describing the efficiency or complexity of an algorithm, especially how it's performance changes as the size of input grows.

5) The worst case time complexity of Merge sort is
$O(n \log n)$

6) Divide & conquer strategy used for Quick sort
Partitioning

7) Theta(Θ) notation:- Used in algorithm analysis to describe the tight bound of running time of an algorithm. It is a function is $\Theta(g(n))$ which means

→ The algo grows no faster than $(g(n))$ up to upper bound
" no slower " " lower bound

8) Best case time complexity of binary search is $O(1)$

9) Strassen's matrix multiplication:-

It can be used in solving large systems of linear equations, which appear in scientific & engineering problems.

10) Little-oh:- the little oh is denoted as (o). It is defined as: Let $f(n)$ & $g(n)$ be non-negative functions, then

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

such that $f(n) = o(g(n))$

i.e, $f(n)$ is little oh of $g(n)$

SM

1) Quick sort ascending order.

Given list:- 30, 20, 10, 50, 60, 40

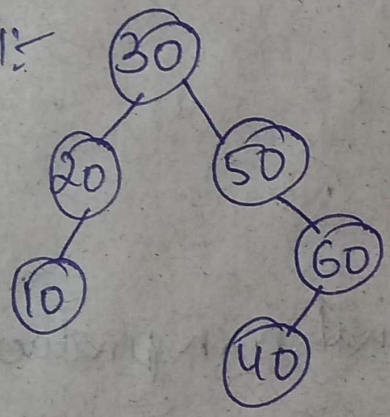
Step 1:- Mark the pivot element

Step 2:- Place the lower elements on the left side of pivot & higher elements on the right side of the pivot element

The first element should be chosen as the pivot element, so; from the given list

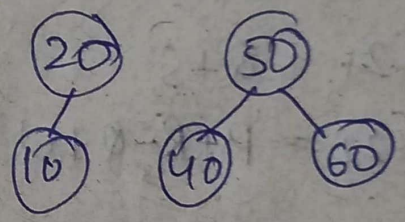
i) 30 will be the pivot element

Step 1:-



20, 10, [30], 50, 60, 40

Step 2:-



10[20][30]40[50]60

Hence the order of list is sorted in ascending order by quicksort.

---

2) process for strassen's matrix multiplication.

Let A & B two n×n matrices (assume n is power of 2)

Partition A & B into four $\frac{n}{2} \times \frac{n}{2}$ submatrices

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Step1 :- Compute the following 4 products each of size $\left[\frac{n}{2} \times \frac{n}{2}\right]$

1) $P = (A_{11} + A_{22})(B_{11} + B_{22})$
2) $Q = (A_{21} + A_{22}) B_{11}$
3) $R = A_{11}(B_{12} - B_{22})$
4) $S = A_{22}(B_{21} - B_{11})$
5) $T = (A_{11} + A_{12}) B_{22}$
6) $U = (A_{21} - A_{11})(B_{11} + B_{12})$
7) $V = (A_{12} - A_{22})(B_{21} + B_{22})$

Step2 :- Compute submatrices $C = A \times B$

$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

Step3 :- Combine $C_{11}, C_{12}, C_{21}, C_{22}$ into final $n \times n$ product matrix C

Time complexity:

$$T(n) = \begin{cases} b, & \text{if } n \leq 2 \\ 7T(n/2) + cn^2, & \text{if } n > 2 \end{cases}$$

3) Merge sort 7,5,2,4,1,6,3,0

If n=even n=odd
$n=\frac{n}{2}$ $n=\frac{n+1}{2}$
low-left, high-right

Divide & conquer rule

1) Divided into smaller subproblems.

2) conquered by solving each subproblem.

3) combined to produce the final solution.

[7,5,2,4][1,6,3,0]

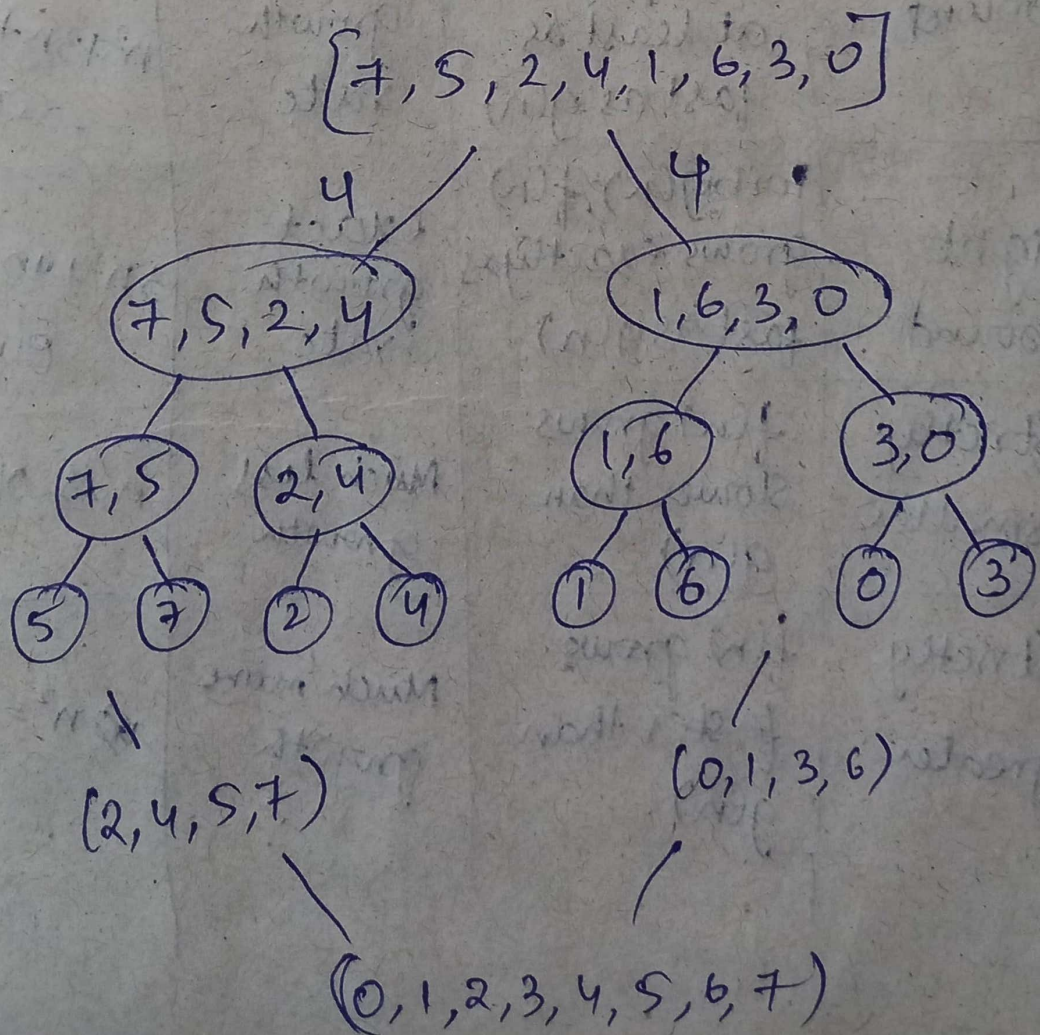
n=4

$\frac{4}{2}$ 2

[7,5] [2,4] [1,6] [3,0]

n=2/2 =1

[7] [5] [2] [4] [1] [6] [3] [0]



Note

less-left
high-right

4] Asymptotic notations - Formal way notation to speak about functions & classify them.

The following notations are commonly use notations in performance analysis & used to characterize the complexity of an algorithm

1) Big-Oh(O) 2) Big-Omega(Ω) 3) Big Theta(Θ) & 4) Little-Oh(o) 5) Little omega(ω)

O(g(n))

Big-Oh :- It is the upper bound and it represents the worst case growth rate. For big n, f(n) grows at most fast as g(n). Example: $n^2+3n+5 \rightarrow O(n)^2$

Notation	Meaning	Definition	Represents	Example
Big Omega $\Omega(g(n))$	Lower bound	For big n(f) grows at least as fast as g(n)	Best case growth rate	$n^2+3n+5 \rightarrow \Omega(n)^2$
$\Theta(g(n))$ Big Theta	Tight bound	For big(n); f(n) grows exactly as fast as g(n)	Exact growth rate	$3n^2+4n \rightarrow O(n^2)$
Little(o) $o(g(n))$	Strictly smaller	f(n) grows slower than g(n)	Much less growth	n is $O(n^2)$
Little omega (ω) $\omega(g(n))$	Strictly greater	f(n) grows faster than g(n)	Much more growth	$n^2 = \omega(n)$

5) Divide & conquer technique & applications

It is a problem solving strategy wher.

1) Divide :- Split the problem into smaller subproblems of same type.

2) Conquer :- Solve them (solve each subproblem)

3) Combine :- Merge the soln's of the subproblems.

Steps :- 1) If the problem is small, solve directly

2) If not, divide, solve & then combine.

Applications :-

1) Merge sort [splits the list, sorts halves & merge results)

2) Quick sort [partitions array around pivot)

3) Binary search [divides search interval in half)

4) Strassen's matrix multiplication [splits into submatrices]

6) Notation	Meaning	Growth bound	Example
Big O $O(g(n))$	Upper bound.	Algorithm grows faster than $g(n)$	Binary search $O(\log n)$
Big Ω $\Omega(g(n))$	Lower bound	at least as fast than $g(n)$	Binary search $\Omega(1)$
Big Θ $\Theta(g(n))$	Tight bound	Exactly as $g(n)$	Merge sort $\Theta(n \log n)$

Example :- $f(n) = 3n^2 + 5n + 2$

$O(n^2) \leq$ growth

$\Omega(n^2) \geq$ growth

$\Theta(n^2)$ = exact growth

7) Binary search as D & Conquer.

Write about D&C.

D:- Compare middle element with target, split search into 2 halves.

Conquer:- Recursively search the correct half

Combine:- NO combinations step needed - result is obtained directly.

Ex:- Sorted array :- [5, 13, 28, 34, 45, 58, 61, 78, 87]

Need to search 45 :-

1) Divide :- $n=9$ so for odd; $n=\frac{n+1}{2}=\frac{9+1}{2}=\frac{10}{2}=5$

So, the 5th element is 45

2) Conquer:- Compare the 5th element with the target element i.e, 45!

3) Combine:- Not needed

If not :- Search 13

1) D :- Mid index = 4 → element 45

target < 45 (search left half)

2) C :- New array [5, 13, 28, 34]

mid = 1 ⇒ element 13 found

3) C :- Return found index.

Complexities : Best case O(1) : Target is at middle

Worst/Avg O(logn) : repeatedly having search space.

8) Recurrence relation : [chat GPT ans]

If $n \leq 1 \rightarrow T(n) = \theta(1)$

else;

$$T(n) = 2T\left[\frac{n}{2}\right] + \theta(n)$$

$2T[n/2]$ → sorting 2 halves ; $\theta(n)$ ⇒ merging steps

Time complexity derivation :

$a=2, b=2, f(n)=\theta(n)$

$$n^{\log_b a} = n^{\log_2 2} = n^1$$

Since $f(n) = \theta(n^{\log_b a})$

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

Final complexities :

Best case = $\theta(n \log n)$

Avg " = $\theta(n \log n)$

Worst " = $\theta(n \log n)$

Space complexity : $O(n)$

[OR]

Material ans:

$$T(n) = a \quad \text{if } n=1;$$
$$2T(n/2) + cn \quad \text{if } n>1$$

c, a → constants

If n is power of 2, $n = 2^k$

$T(n) = 2T(n/2) + cn$

$2[2T(n/4) + cn/2] + cn$

$4T(n/4) + 2cn$

$2^2T[n/4] + 2cn$

$2^3T[n/8] + 3cn$

$2^4T(n/16) + 4cn$

$2^kT(1) + kcn$

$an + cn(\log n)$

So by represting it by asymptotic notation O is

$T(n) = O(n\log n)$

9) Quick sort :-

Best case - Pivot splits ~~randomly~~ evenly - O(nlogn)
Avg - Random pivots - O(nlogn)
worst - pivot is always smaller/largest - $O(n^2)$

Cases	Time	Space
Best	O(nlogn)	O(logn)
Avg	O(nlogn)	O(logn)
worst	$O(n^2)$	O(n)

10) Quick vs Merge

Feature	Quick sort	Merge sort
Design	D&C used partioning	using merging
Best/Avg	O(nlogn)	O(nlogn)
Worst	$O(n^2)$	O(nlogn)
Space	O(logn)	O(n) extra
Stability	Not stable	Stable
Preference	In-place, small memory, avg case fast	Stable sort needed, guranteed peformance

Quick & merge sort gurinchi write konchem