Jai Swaminarayan

PAGE NO.
DATE :

L 0024 Rathod Ajinkya

DAA

02 Asymptotic Notation

Asymptotic analysis of an algorithm refers to defining the mathematical boundation of its run-time performance.

Using this, we can conclude the best case, worst case, and average case scenario.

It is input bound
i.e. if there is no input to algorithm, it is concluded to work in constant time.

Other than input all other factors are considered constant.

Following are common used notations

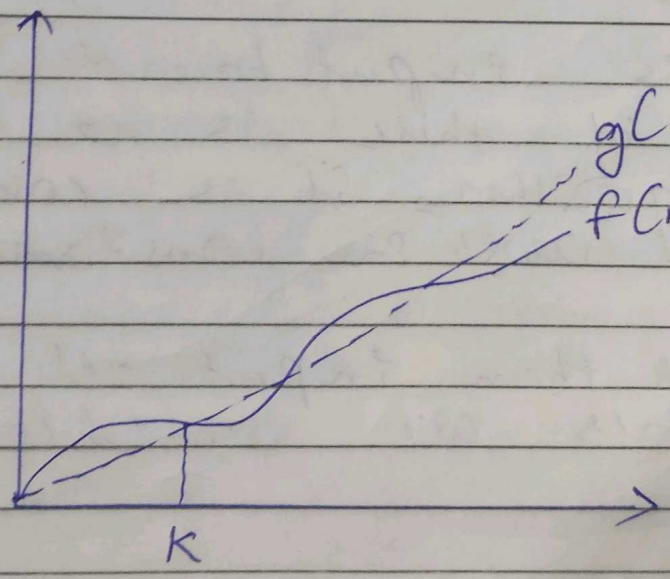
1. O Notation
2. Ω Notation
3. Θ Notation

PAGE NO.
DATE :

Big Oh Notation

The notation $O(n)$ is formal way to express the upper-bound of algorithms running time.

It measures the worst case time complexity or longest amount of time an algorithm can possibly take to complete.

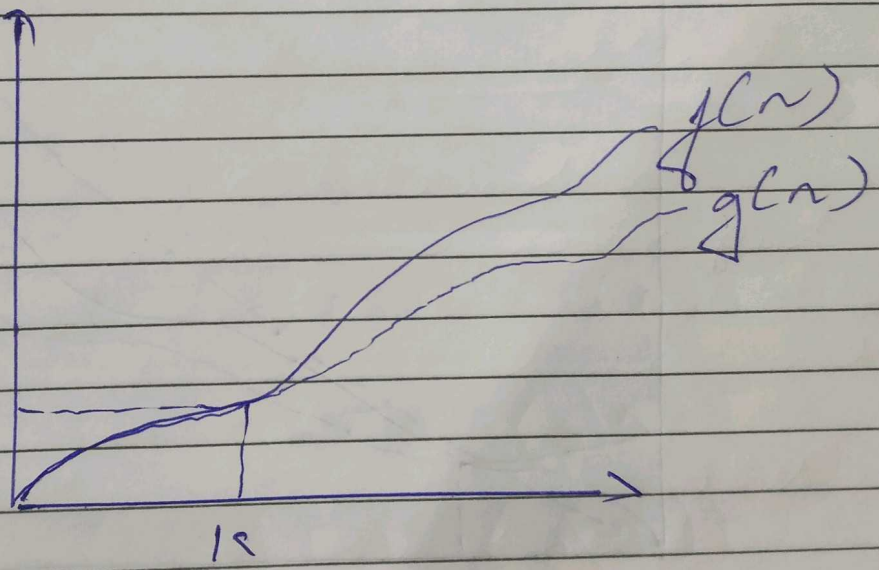


$O(f(n)) = \{ g(n) : \text{there exists } c > 0$ and no such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0 \}$

PAGE NO.
DATE :

Omega Notation

The notation $\Omega(n)$ is formal way to express lower bound of algo running time. It measures best case time complexity or best amount of time an algorithm can possibly take to complete

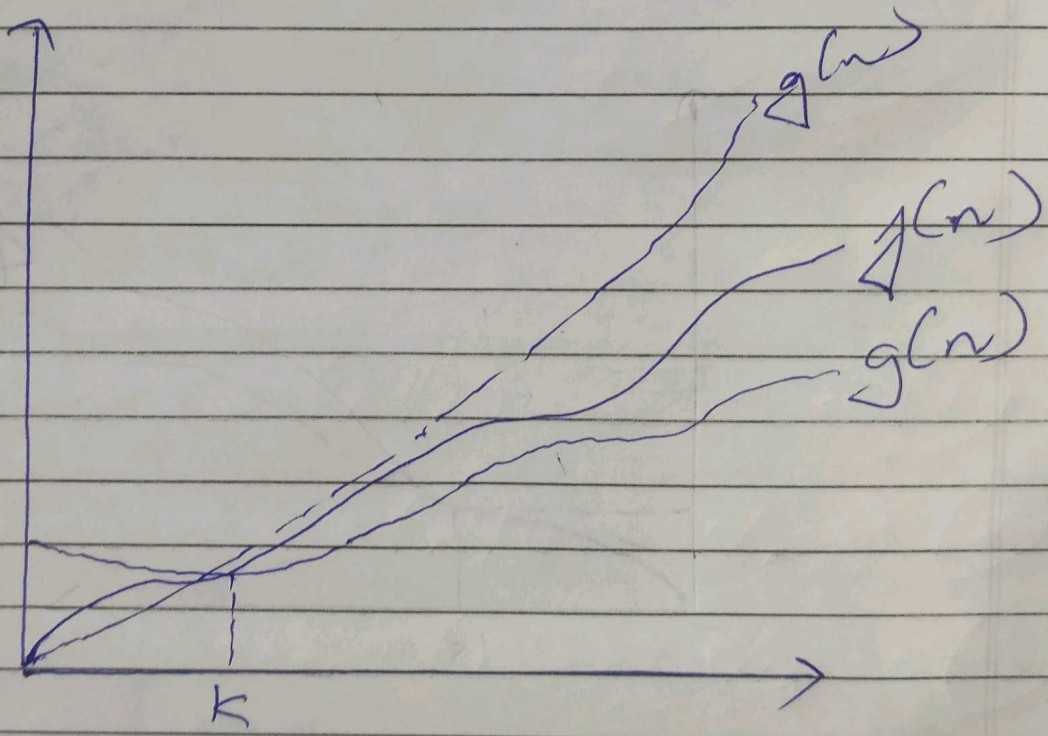


$\Omega(f(n)) \geq g(n)$: there exists $c > 0$ and n_0 such that $g(n) \leq cf(n)$ for all $n > n_0$.

PAGE NO.
DATE :

Theta Notation

The notation $\Theta(n)$ is formal way to express both lower bound and upper bound of an algorithm running time.



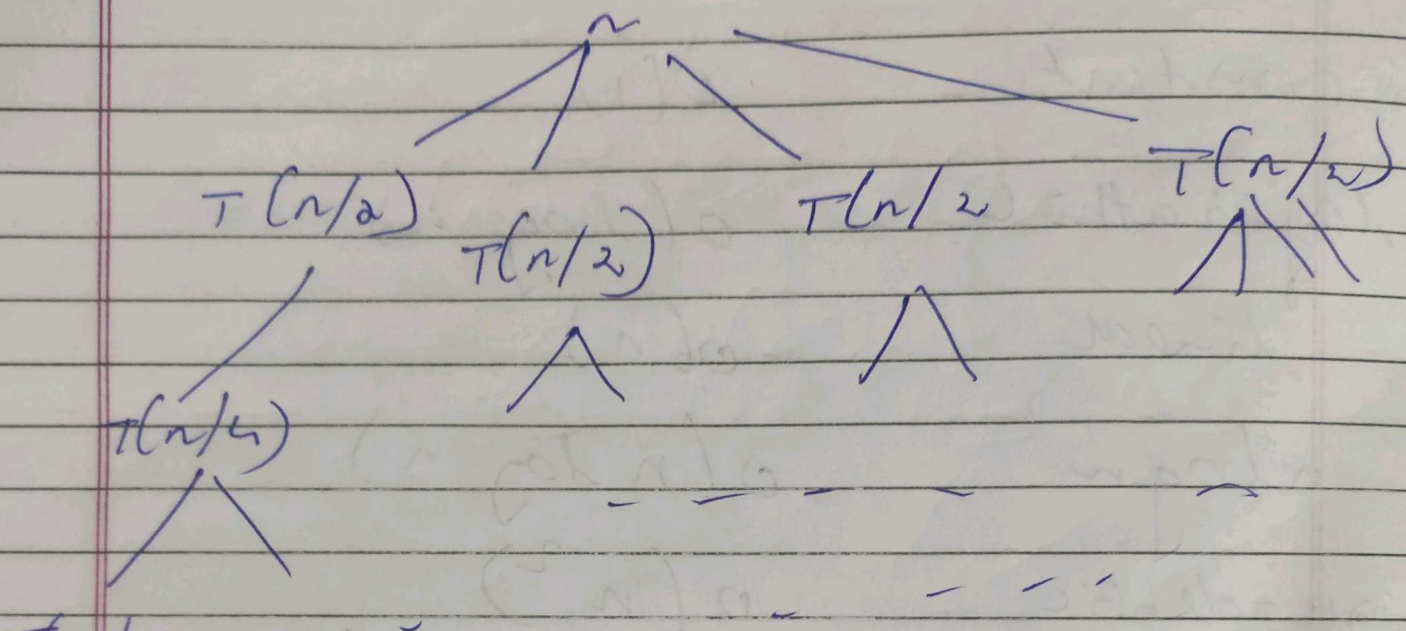
$\Theta(f(n)) = \{g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n \geq n_0\}$

PAGE NO.
DATE :

Common Notations

constant	$O(1)$
logarathic	$o(\log n)$
linear	$o(n)$
nlogn	$o(n \log n)$
quadratic	$o(n^2)$
cubic	$o(n^3)$

Q11



$$T(n) = 4T(n/2) + n \quad \text{--- (1)}$$

$$T(n/2) = 4T(n/4) + n/2 \quad \text{--- (2)}$$

$$T(n/4) = 4T(n/8) + n/4 \quad \text{--- (3)}$$

Substituting (2) in (1)

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

$$\boxed{Tn = 4^2 T(n/2^2) + 2n + n} \quad \text{--- (4)}$$

PAGE NO.
DATE :

Substitution (2) in (1)

$$T_n = 4^2\left\{4T\left(\frac{n}{2^3}\right) + \left[\frac{n}{2^2}\right]\right\} + 2n + n$$

$$= 4^3 T\left(\frac{n}{2^3}\right) + 4n + 2n + n$$

$$= n + 2n + 2^2 n + 4^3 T\left(\frac{n}{2^3}\right)$$

$$T(n) = n + 2n + 4n + \cdots 2^{k-1} n + 4^k T\left(\frac{n}{2^k}\right)$$

Let's assume $\frac{n}{2^k} = 1$

$n = 2^k \quad \therefore k = \log_2 n$

$$T(n) = n\left(2^k - 1\right) \Rightarrow \frac{n\left(2^{k+1} - 1\right)}{2-1} + 4^k T\left(\frac{n}{2^k}\right)$$

$$= n\left(\frac{2^{\log n} - 1}{2-1}\right) + 4^{\log n} T(1)$$

Hule $4^{\log n} = n^2$

Teacher's Signature.....

PAGE NO.
DATE :

$n^2 T(1)$

So, $n\left(2\frac{\log(n)}{\alpha-1} - 1\right) + [illegible]$

$= n(n-1) + n^2 T(1)$

which is $O(n^2)$.

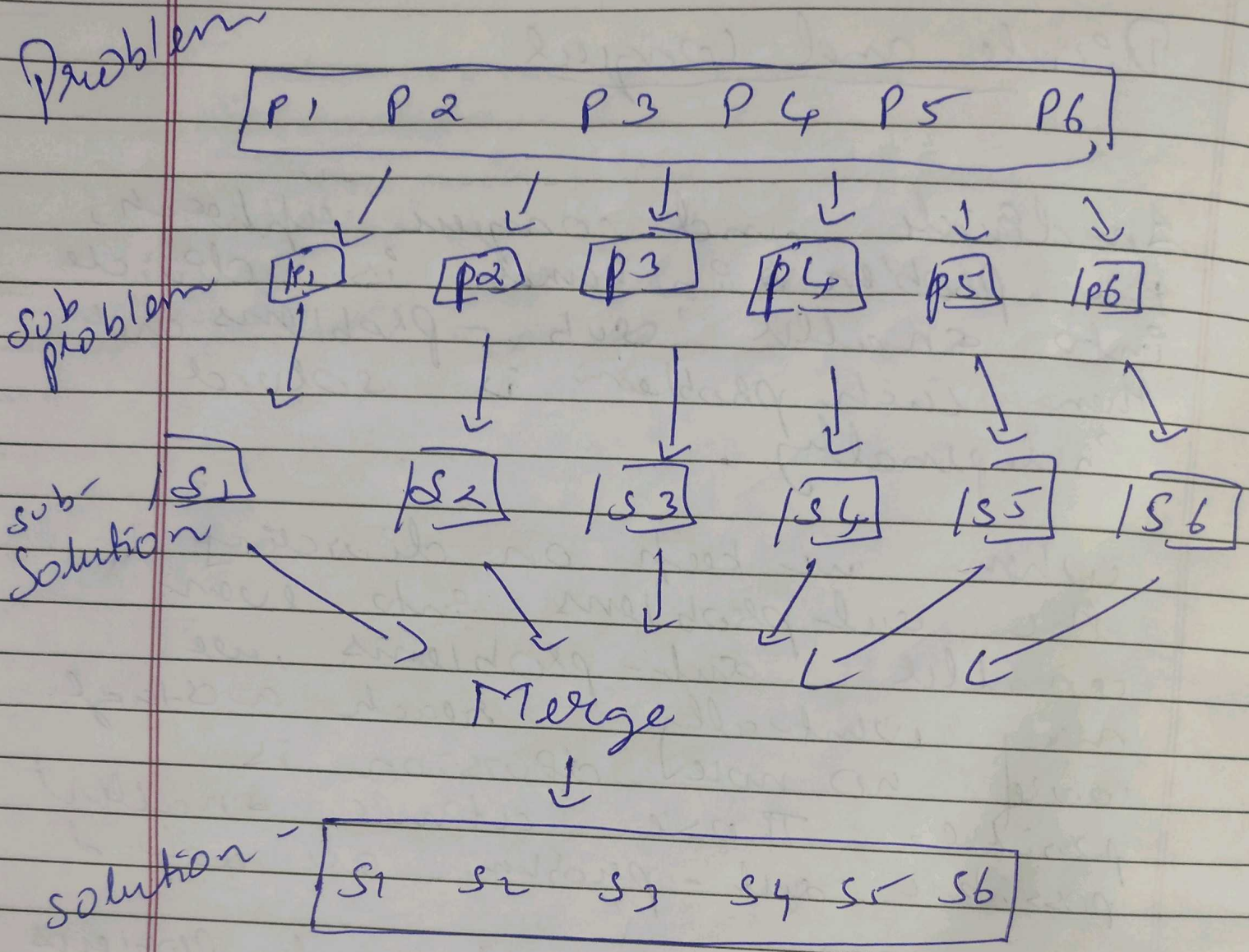
Q6 Divide and conquer

In divide and conquer approach, the problem in hand is divide into smaller sub-problems and then each problem is solved independently.

When we keep on dividing the sub-problems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those atomic smallest possible sub-problem are solved.

The solution of all sub-problems is finally merged in order to obtain the solution of original problem.

Let is understand it diagrammatically at first :-



What is divide?

The steps involves breaking the problem into smaller sub problems.

It should be part of

original solution problem.

→ This generally takes a recursive approach to divide the problem until no sub-problem is further divisible.

→ At this stage, sub-problems become atomic in nature but still represent some part of actual problem.

Conquer

The steps recieves a lot of smaller sub-problem to be solved.

Generally at this level, problems are considered solved on their own.

Merge / combine

when smaller sub-problems are solved this stage recursively combins them until they formulate a solution of original

Teacher's Signature.....

PAGE NO.
DATE :

problem.

→ This algorithmic approach works recursively and conquer and merge steps works so close that they appear as one.

Q12 Different method of recurrence relation

1) Recursive Tree Method

The recursive tree method converts the recurrance into tree verbose node and it represents the cost incurred at various levels of recursion

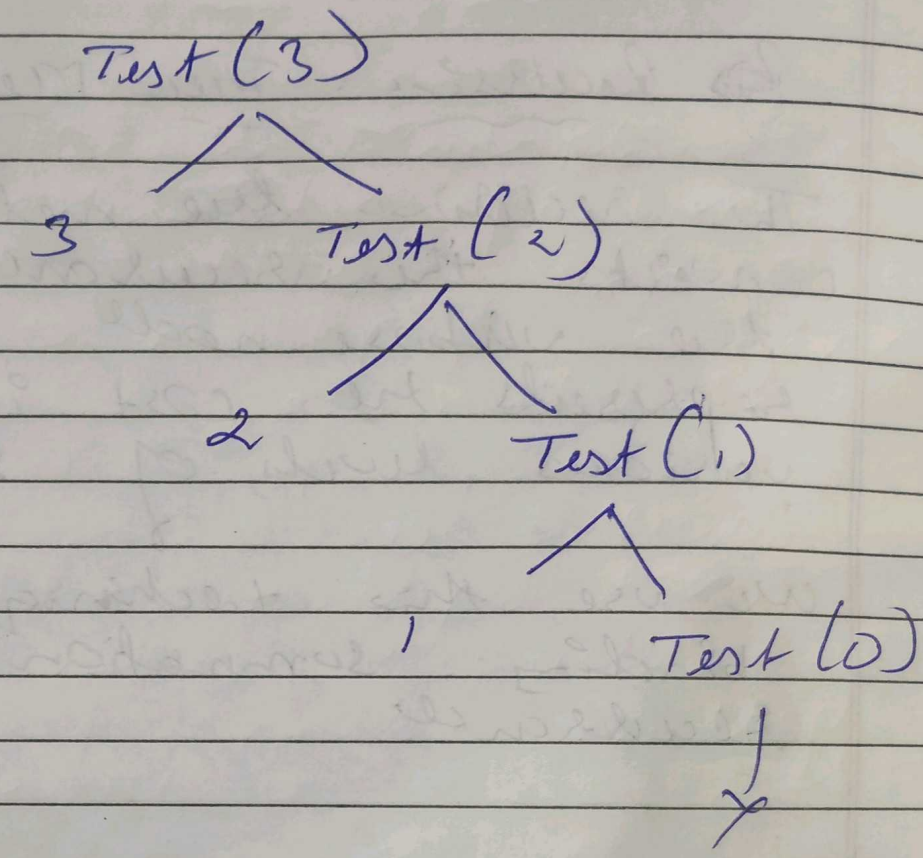
We use this technique for binding summation of the recurrance

Example

```
void test (int n) {                    T(n)
    if (n>0) {
        print("%d", n);
        test(n-1);                     T(n-1)
    }
}
```

PAGE NO.
DATE :

Assume $n = 3$



Here 3 time print value of x

3+1 = 8 times function is calling.

if (n==5) 5 times printing and
5+1=6 times function is call.

$f(n) = n+1$

$f(n) = O(n)$

~~$f(n) = \Theta(n)$~~

$f(n) = O(n)$

$f(n) = \Omega(n)$

Substitution Method

In this method, we guess the bound and then use mathematical induction to prove correct guess

```
void Test (int n) {
    if (n>0) {
        print f ("%d", n)
        Test(n-1)
    }
}
```

T(n) = Total no. of line taken by above function

$$T(n) = T(n-1) + 1$$

So, $$T(n) = \begin{cases} 1 & n=0 \\ T(n+1)+1 & n>0 \end{cases}$$

$$T(n) = T(n-1) + 1$$

09

Matrix-Chain Multiplication

Matrix Chain Multiplication is an optimization problem concerning the most efficient way to multiply a given sequence of matrices.

The problem is not actually to perform multiplications, but merely to decide sequence of matrix multiplications involved.

There are many options because this is associative.

In other words, no matter how product is parenthesized the result will remain same.

For example, A, B, C, D there are 5 possible options

$((AB)C)D = A(BC))D = (AB)(CD) =$

$A((BC)D) = A(B(CD))$.

Although it does not affect the product, the order in which they are pairwise affected.

Ex. A is 10×30

B is 30×5

C is 5×60 then.

$(AB)C$ needs $(10 \times 30 \times 5) + (10 \times 5 \times 60) = 4500$

$A(BC)$ needs $(30 \times 5 \times 60) + (10 \times 30 \times 60) = 27000$

Teacher's Signature.....

PAGE NO.
DATE :

Q7 Dynamic Program Algorithm

We all want is minimum cost, minimum number of arithmetic operations needed to multiply out the matrices.

If we only multiply a matrix, that is one way to multiply them, so minimum cost of 0.

① → Take sequence of matrices and separate into sub sequences

② → Find minimum cost of multiplying out each subsequence

③ → Add these cost together, and add in cost of multiplying the two results matrices

④ → Do this for each possible position

at which sequence of matrices are can be split and take minimum all over of them.

However algorithm has exponential runtime complexity makes it as inefficient as the naive approach of trying all permutations.

The reason is that algorithm does lot of redundant work.

As recursion grows deeper, more repitation occurs.

One solution is "memoization".

Each time we compute the minimum cost needed to multiply out a subsequence, we save it.

It can be shown that this simple trick brings runtime down to $O(n^3)$ from $O(n^2)$, which is

.

Teacher's Signature.....

more efficient for real life applications.

This is top-down programming.

PAGE NO.
DATE :

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T(n/2) + n & \text{if } n = 2^k \end{cases}$$ for $k > 1$

method 1

Master Theorem

$$T(n) = aT(n/b) + \Theta(n^k \log^p n)$$

Constraint, $a \geq 1, b \geq 1, k \geq 0$, p = real number.

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

$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$

$a \geq 1 \quad b \geq 1 \quad k \geq 0$, so, we can apply master theorem.

Case 2 $a = b^k$

$p > -1$

$T(n) = \Theta(n \log_b^a \cdot \log^{p+1} n)$

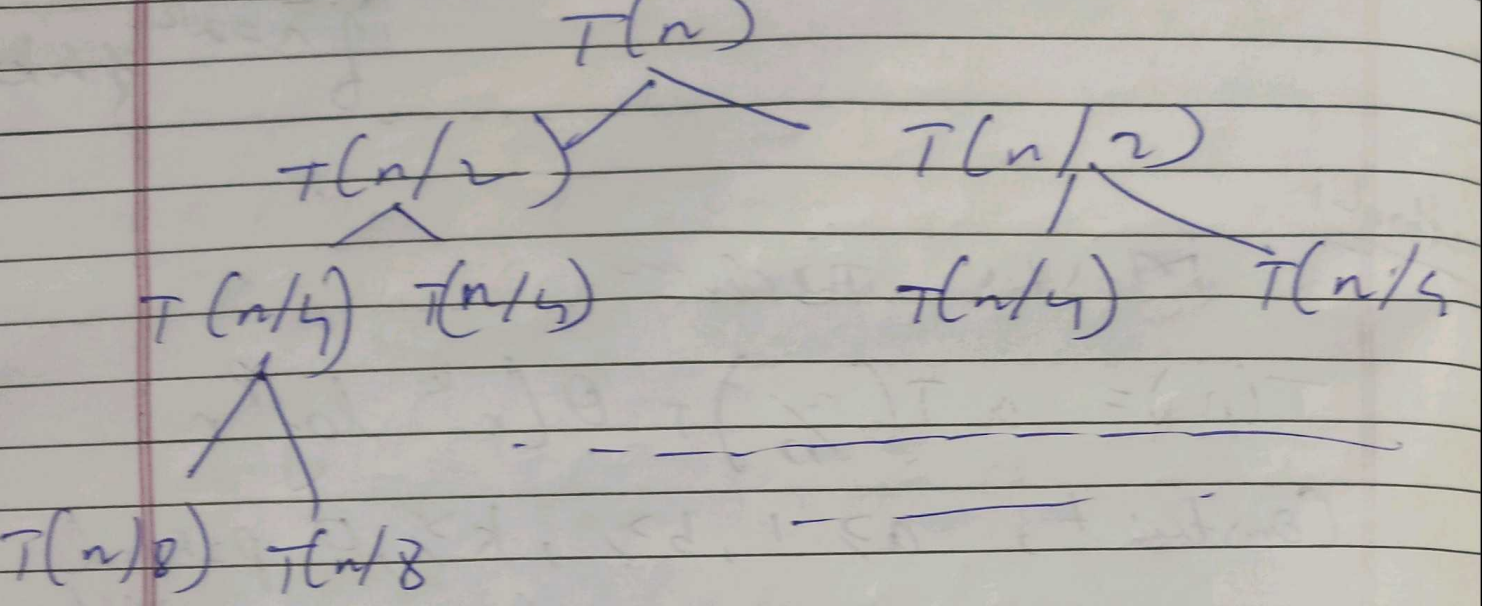
$T(n) = \Theta(n \log_2^1 \log^1 n)$

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

Teacher's Signature.....

Method

(2) Recursive Tree



$T\left(\frac{n}{2^k}\right) \cdots T\left(\frac{n}{2^k}\right)$

So, total $k+$ times it does.

Here, at k^{th} step $= \frac{N}{2^k} = 1$

$2^k = N$

$k = \log_2 N$

Total comparison $= (k+1)\, C \cdot N$

$= (\log_2 N + 1)\, N$

$= N \log_2 N + N$

So, time complexity is $\Theta(N \log_2 N)$

Method

(3) Substitution Method

$T(n) = 2T(n/2) + n$ --- (1)

$T(n/2) = 2T(n/4) + n/2$ --- (2)

$T(n) = 2\{2T(n/4) + n/2\} + n$

$= 2^2 T(n/2^2) + 2n$ --- (3)

$T(n/4) = 2T(n/8) + n/4$

(4) → (3)

$2^3 T(n/2^3) + 3n$ --- (4)

$\boxed{T_n = 2^i T(n/2^i) + in}$ --- (5)

$T(1) = 1$

Let $n/2^i = 1$ so, $\boxed{n = 2^i}$ --- (6)

Teacher's Signature.....

$n = 2^i$

$i = \log_2 n$ --- (7)

Substitute eq (6) and (7) in (5)

$T(n) = 2^i T(n/2^i) + in$

$= n T(1) + n \log_2 n$

$T(n) = n + n \log_2 n$

$T(n) = n \log_2 n$

Time Complexity = $O(n \log_2 n)$

Teacher's Signature.....

8

PAGE NO.
DATE :

Q1) Activity Selection Problem

→ Greedy is an algorithm paradigm that builds up a solution piece by piece, always chosing next piece that offers the most obvious and immediate benefit.

→ They are used for optimization problems.

Note : At every step, we make a choice that looks best at the moment, and we get optimal solution of problem.

1. Kruskals Minimum Sapp; Tree.

We create by picking edge one by one. Smallest weight edge can't cause a cycle in MST.

2. Prim's

We maintain 2 sets.

(1) A sets of vertices included
(2) A set of vertices not include

Smallest edge connects 2 sets.

3. Dijkstra's Shortest Path

This greedy is to pick the edge that connects the 2 sets and is on smallest weight path from source to set that contains not yet included vertices.

4. Huffman Coding

It is ~~loss~~ loss-less compression technique. It uses

variable length bit codes to different characters.

The greedy choice is to assign least bit length code to most frequent character

Q3

To do this perfectly, we use the seven rules of Big O and Θ.

writing e for original expression,

$$= \Theta(\max(n^3/1000 - 100n^2 - 100n + 3))$$

Applying rule (1) to each of its terms we have

$$\Theta(\max(n^3, n^2, n, 1))$$

Now (3) to see that $\Theta(1) < \Theta(n) < \Theta(n^2) < \Theta(n^3)$.

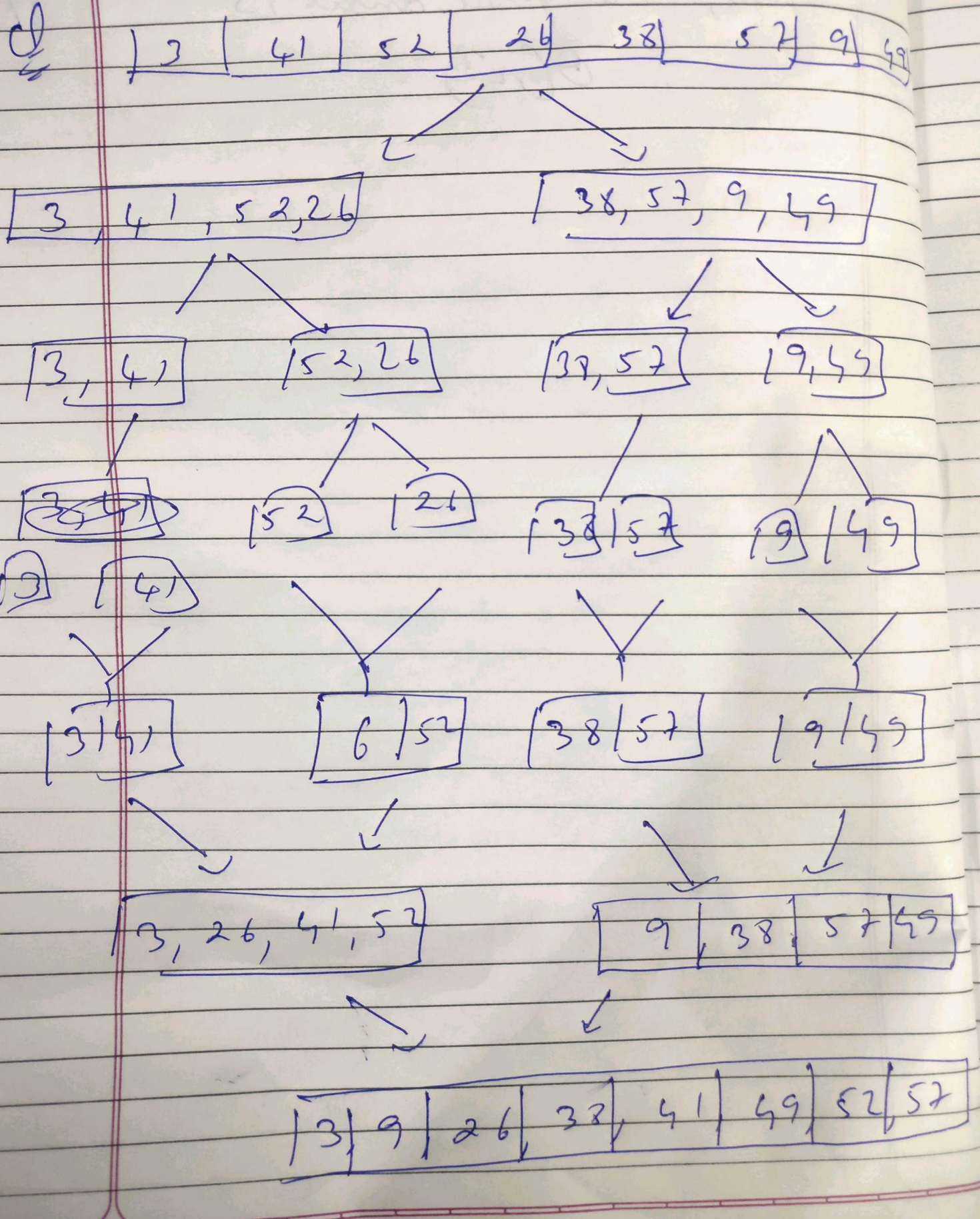
So, $\Theta(\max(n^3, n^2, n, 1)) = \Theta(n^3)$.

Thus, the final answer is

$\Theta(n^3)$

Mam

Q


3 41 52 26 38 57 9 49
3, 41, 52, 26
38, 57, 9, 49
3, 41
52, 26
38, 57
9, 49
3 41
52
26
38 57
9 49
3 41
6 52
38 57
9 49
3, 26, 41, 52
9 38 57 49
3 9 26 38 41 49 52 57