RTPNOTES.vercel.app

Algorithm-Analysis-Module-1-Important-Topics

For more notes visit

https://rtpnotes.vercel.app

RTPNOTES.vercel.app

RTPNOTES.vercel.app

---

1. Methods to solve Recurrence Equation

- Function on the left side is occuring in the right side with lesser argument, Then its a recurrence equation.
  - For example
    - $T(n) = 2T(n/2) + cn$
- The methods to solve recurrence equation are
  - **Iteration Method**
  - **Recursion Tree Method**
  - **Substitution method**
  - **Masters theorem**

---

2. Iteration Method

RTPNOTES.vercel.app

Example 1

- Consider the following Recurrence Relation
- $T(n) = 1 + T(n-1)$
- We have to solve this equation using iterative method
- Find $T(n-1)$, $T(n-2)$ etc...
    - $T(n) = 1 + [1 + T(n-2)] = 2 + T(n-2)$
    - $T(n) = 2 + [1 + T(n-3)] = 3 + T(n-3)$
    - Repeat this k times
    - $T(n) = k + T(n-k)$
- Lets assume $n - k$ = 1, k = n - 1
- $T(n) = n - 1 + T(1)$
- Expression in terms of Time Complexity
- $T(n) = n - 1 + T(1)$ = $O(n) + O(1)$
- The time complexity is $O(n)$

Example 2

$$\mathbf{T(n) = 2\ T(n/2) + n} \qquad \mathbf{T(1)=1}$$

$$
\begin{aligned}
T(n) &= n + 2\ T(n/2) \\
&= n + 2\ [(n/2) + 2\ T(n/2^2)] = 2n + 2^2 T(n/2^2) \\
&= 2n + 2^2\ [(n/2^2) + 2T(n/2^3)] = 3n + 2^3 T(n/2^3)
\end{aligned}
$$

- Repeating k times

  $\cdots\cdots\cdots\cdots\cdots$

- $= k\ n + 2^k T(n/2^k)$
- Assume that $n/2^k = 1$
- $2^k = n \rightarrow \quad k = \log_2(n)$

RTPNOTES.vercel.app

- Subbing these values, we get

$$T(n) = n\log_2(n) + nT(1)$$
$$= n\log_2(n) + n$$
$$= \mathbf{O(n\log_2(n))}$$

-

Example 3

$$T(n) = 2 \qquad \text{if } n=1$$
$$T(n) = 2T(n/2) + 2n + 3 \qquad \text{Otherwise}$$
$$T(n) = 3 + 2n + 2T(n/2)$$
$$= 3 + 2n + 2[3+2(n/2)+2T(n/2^2)]$$
$$=3+2x3+2x2n+2^2T(n/2^2)$$
$$=3+2x3+2x2n+2^2[3+2(n/2^2)+2T(n/2^3)]$$
$$= 3+2x3+2^2x3+3x2n+2^3T(n/2^3)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$= [3+2x3+2^2x3+\ldots. +2^{k-1}x3]+[kx2n]+2^kT(n/2^k)$$

Assume that $n/2^k=1 \rightarrow 2^k=n \rightarrow k=\log_2(n)$

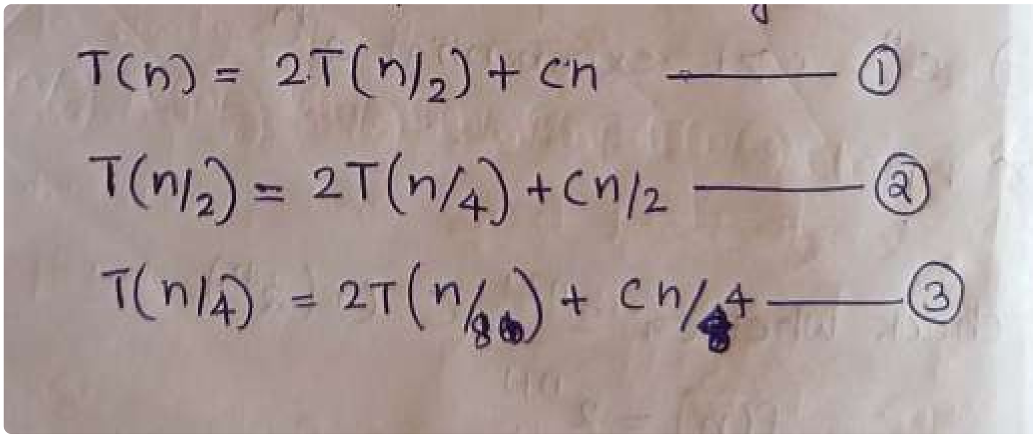
$$T(n) =3[2^0+2^1+2^2+\ldots+2^{k-1}]+2n\log_2(n) + nT(1)$$
$$=3[(2^k-1)/(2-1)] +2n\log_2(n) + 2n = 3(n-1) +2n\log_2(n) + 2n$$
$$= 5n -3 +2n\log_2(n) = \mathbf{O(n\log_2(n))}$$

---

3. Recursion Tree method

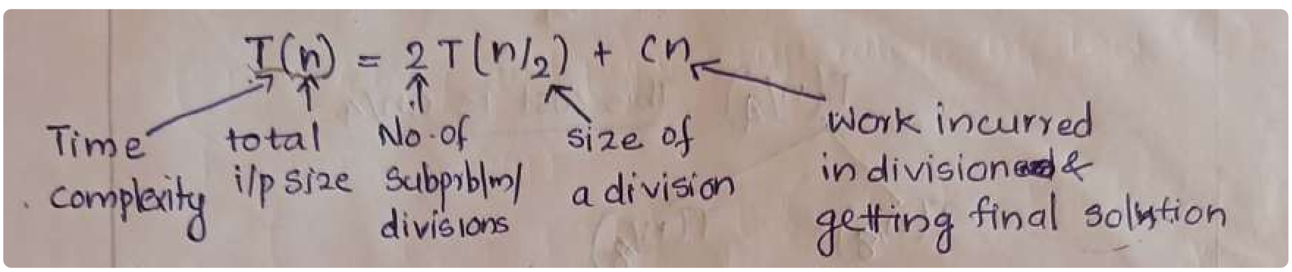
Example Problem

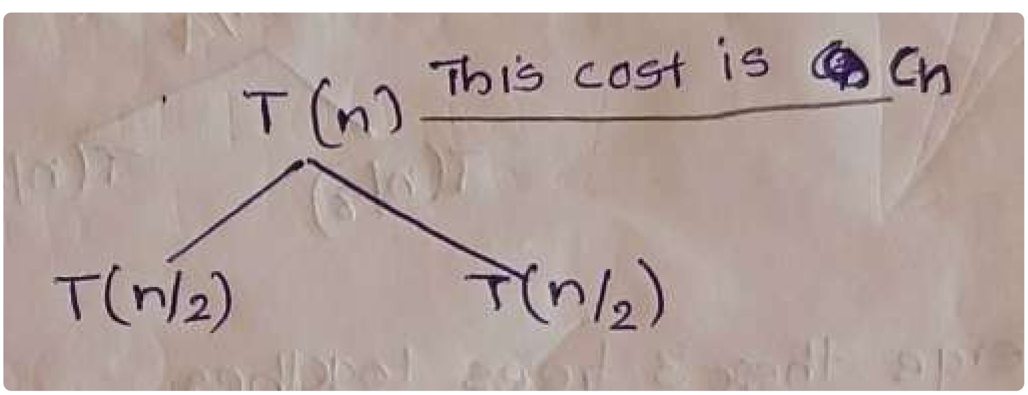
RTPNOTES.vercel.app



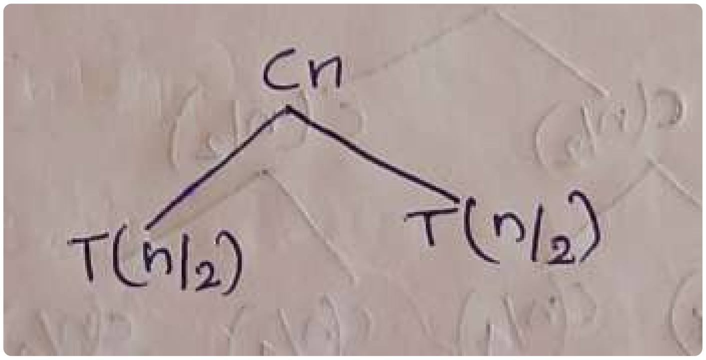
Lets go over the equations one by one

Tree for Equation 1

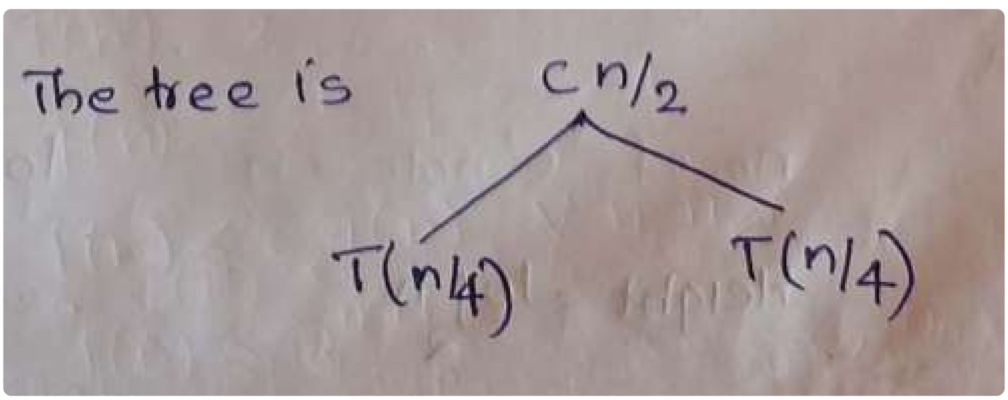
- **Root node is LHS, ie T(n)*
- **From the equation we can understand**
  - Time Complexity
  - Total input size
  - No of subproblems/divisions
  - Size of division
  - Work incurred in division and getting final solution
- 

- No of Child Nodes = No of Problem divisions
  - From the above equation, No of problem divisions = 2
  - So we need to create 2 child nodes
- Lets take 2 child nodes
  - 

RTPNOTES.vercel.app

- The Tree can be also be expressed in terms of cost
    - 

Tree for Equation 2

- The given equation
    - $$T(n/2) = 2T(n/4) + Cn/2$$
    - No of divisions = 2
    - Size of division = n/4
    - Cost = Cn/2
    - Input size = n/2
- 

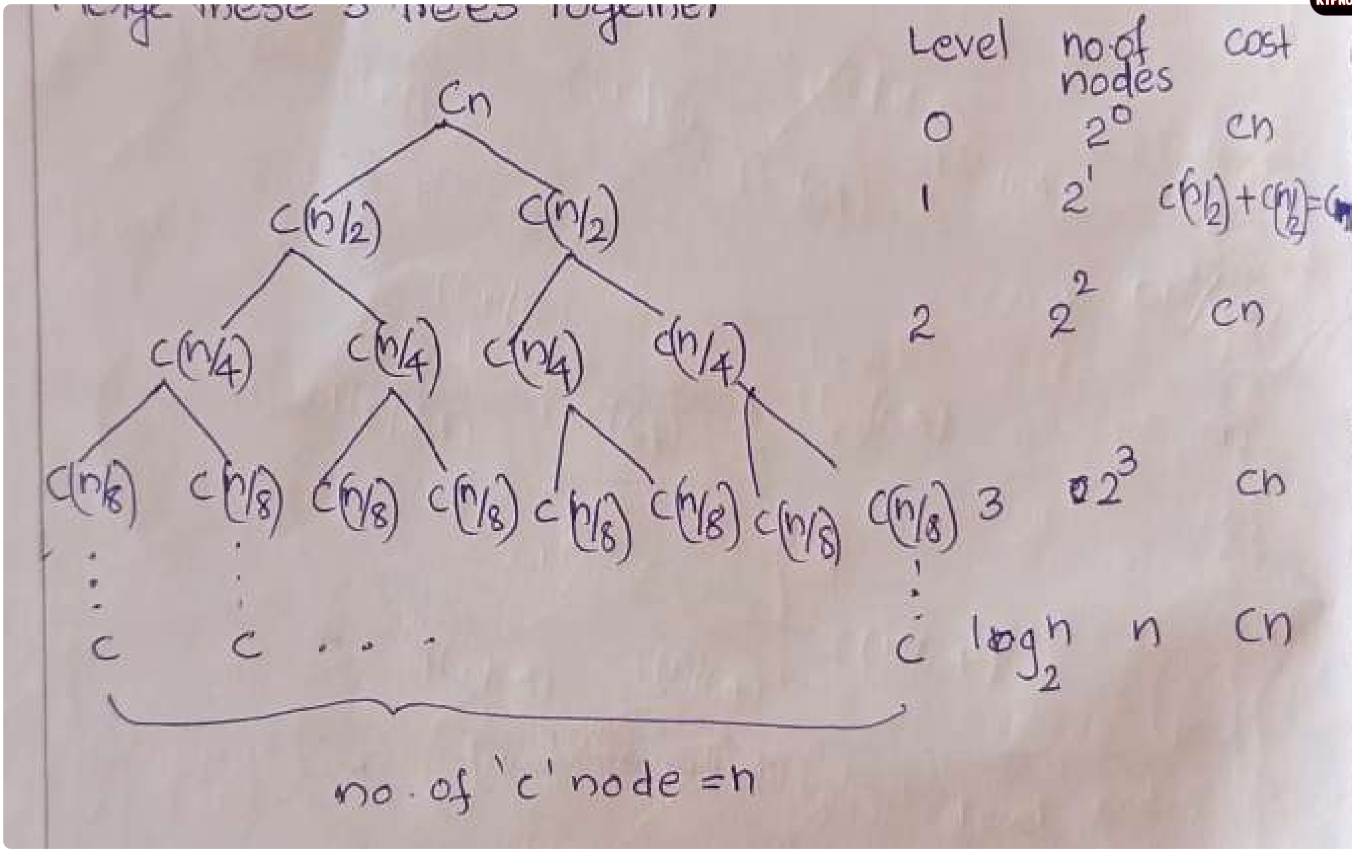
Tree for Equation 3

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

- No of divisions = 2
- Size of division = n/8
- Cost = cn/4

Merging the 3 trees together

RTPNOTES.vercel.app



- **The height of this tree can be calculated by the formula**
  - Height of the tree $= \log_2 n$
  - No of nodes at the last level
    - $2^i = n$
  - $\boxed{\text{level } i = \log_2 n}$
- **We can calculate the total cost by**
  - Total Cost = log(n+1) x Cn
  - Cnlogn + Cn

RTPNOTES.vercel.app

- $$\text{Total cost} = (\log(n+1))\,cn$$
  $$= cn\log n + cn$$
- **When ignoring Cn and c, we get the complexity**
- $$T(n) = n\log n$$
  $$= O(n\log n)$$

---

4. Substitution Method

How to do the substitution method?

1. Guess the form of the solution
2. Prove that is guess is correct by performing a valid substitution
3. To guess a solution we can use Recursion tree method or iteration method
4. To prove the validity of the guess use induction proof method

Example

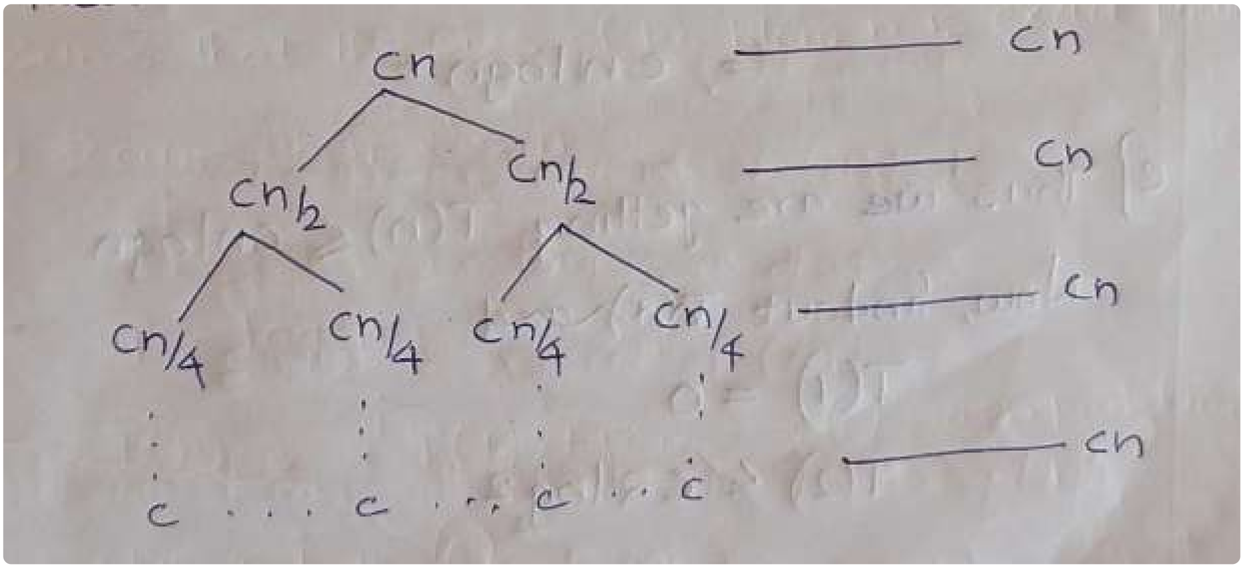
$$T(n) = 2T(n/2) + \theta(n)$$

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

Step 1: Guess the form of the answer

RTPNOTES.vercel.app

Lets the draw the tree



- Now to calculate the complexity, we have to use this formula
  - **T(n) = (Count of levels) * Cn**
  - = (count of 0,1,2,... logn) x cn
    - Logn is taken because we know that
      - Height of the tree is Logn
  - = (logn + 1) x cn
  - = cnlogn + cn
  - = O(n logn)

Step 2

- From the above, our guess for the complexity is
- $T(n) \leq cn\log n$

Step 3

- From the above guess, write the form of equation for T(n/2)
- $T(n/2) \leq cn/2 \log n/2$

Step 4

RTPNOTES.vercel.app

- Substitute the previous equation into equation 1 (Step 1)
- Equation 1
  - $T(n) = 2T(n/2) + cn$ —— (1)
- Equation 2
  - $T(n/2) \leq cn/2 \log n/2$
- Subbing 2 in 1
  - $T(n) \leq 2[cn/2 \log n/2] + cn$
  - $= cn \log n/2 + cn$
    $= cn[\log n - \log 2] + cn$
  - Ignoring 2
  - $= cn\{\log n - cn + cn$
    $= cn \log n$

Step 5

- We are getting T(n)<= cnlogn
- Now we have to find out T(1), T(2), T(3)

RTPNOTES.vercel.app

- $$T(1) = 0$$
  $$T(2) \le c\,2\log_2 2 = c\,2$$
  $$T(3) \le c\,3\log_2 3$$
  Hence $T(n) \le c\,n\log n$, for +ve constant c

---

5. Master theorem

What is Masters Theorem?

- Its a cookbook method
- Its associated with divide and conquer paradigm (DAC)

General form of DAC is

- $$T(n) = aT(n/b) + f(n) \quad , \; b > 1, \; a \ge 1$$
- f(n) is the amount involved in splitting

Statement

Let $a \ge 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on non-negative integers by the recurrence

$$\boxed{T(n) = aT(n/b) + f(n)}$$

RTPNOTES.vercel.app

- Where n / b is either a floor or ceil operation

T(n) can be the following based on the conditons

1. Case 1
    1. $$f(n) = O\left(n^{\log_b a - \epsilon}\right), \quad \epsilon > 0$$
       $$\text{Then } T(n) = \Theta\left(n^{\log_b a}\right)$$
2. Case 2
    1. $$f(n) = O\left(n^{\log_b a}\right).$$
       $$\text{Then } T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$$
3. Case 3
    1. $$f(n) = \Omega\left(\cancel{\log}\, n^{\log_b a + \epsilon}\right), \quad \epsilon > 0$$
    2. and $af(n/b) \leq cf(n)$, $c < 1$ and n is sufficiently large, Then $T(n) = \Theta(f(n))$

Master theorem Example 1

Solve the following recurrence using Master theorem

1. $T(n) = 9T(n/3) + n$

- First lets write the general form

RTPNOTES.vercel.app

- $$T(n) = aT(n/b) + f(n) \quad , b > 1 , a \geq 1$$
  - Lets compare it with the question
    - a = 9
    - b = 3
    - f(n) = n
- Now lets find $n^{\log_b a}$
  - $n^{\log_b a}$ = $n^{\log_3 9}$ = n^2
- Lets compare f(n) and $n^{\log_b a}$
  - f(n) is smaller - Case 1
  - Both are equal - Case 2
  - f(n) is larger - Case 3
- In this case n < n^2
  - Which means its case 1, f(n) is smaller
- Applying case 1 of master theorem
  - $$f(n) = O(n^{\log_b a - \epsilon}), \ \epsilon > 0$$
    $$\text{then } T(n) = \Theta(n^{\log_b a})$$
  - $$\text{So here } T(n) = \Theta(n^2)$$

Master theorem Example 2

$$T(n) = T(2n/3) + 1$$

- First lets write the general form

RTPNOTES.vercel.app

- $$T(n) = aT(n/b) + f(n) \quad , b > 1 , a \geq 1$$
- Lets compare it with the question
  - a = 1
  - b = 3/2
  - f(n) = 1
- Now lets find $n^{\log_b a}$
  - $$n^{\log_b a} = n^{\log_{3/2} 1} = \cancel{1}\, n^0 = 1$$
- F(n) and $n^{\log_b a}$ are equal = 1
- Applying Case 2 of Master Theorem
  - $$T(n) = \theta(n^{\log_b a} \cdot \log n)$$
  - $$= \theta(\log n)$$

Master theorem Example 3

$$T(n) = 3T(n/4) + n\log n$$

First lets write the general form

- $$T(n) = aT(n/b) + f(n) \quad , b > 1 , a \geq 1$$
- Lets compare it with the question
  - a = 3
  - b = 4
  - f(n) = nlogn
- Now lets find $n^{\log_b a}$

RTPNOTES.vercel.app

  - $$n^{\log_b a} = n^{\log_4 3} = n^{0.79}$$
- Here F(n) is larger than $n^{\log_b a}$
- Checking case 3 of Masters theorem
  - $$f(n) = \Omega(\cancel{\log}\, n^{\log_b a + \epsilon}), \quad \epsilon > 0$$
  - By the condition, we need to add epsilon and verify
    - $\varepsilon = 0.21$, then $n^{\log_b a} = n^{0.79+0.21} = n^1$
    - $$n^1 < n\log n$$
  - We Have another condition
    - $af(n/b) \le cf(n)$, $c < 1$ and n is sufficiently large
    - $$3\frac{n}{4}\log\frac{n}{4} \le cn\log n$$
      $$\frac{3n}{4}[\log n - \log 4] \le cn\log n$$
      $$\frac{3}{4}n\log n \le cn\log n$$
    - $$c = \frac{3}{4} < 1$$
    - The C value is less than 1, This satisfies case 3

RTPNOTES.vercel.app

- $T(n) = \Theta(n \log n)$

6. Asymptotic Notations

Theta Notation

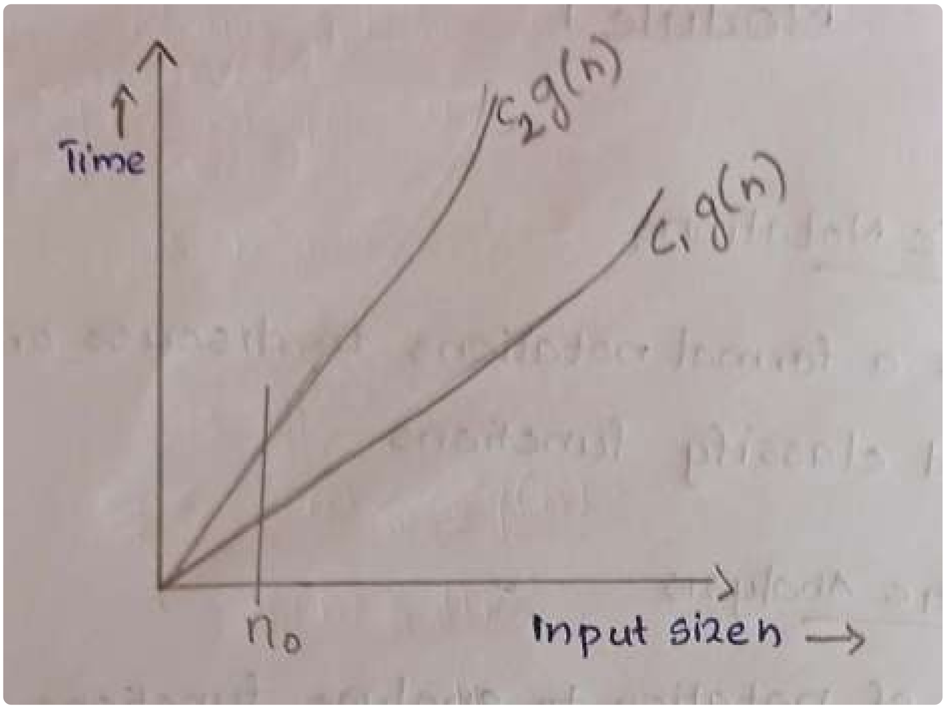
- Lets consider 2 functions
  - f(n) and g(n)
  - The argument n is input size
- n0 -> Input size boundary

Definition

$\Theta(g(n))$: { f | f is a non negative function and there exists positive constants c_1 & c_2 such that $\cancel{c_1 g(n) \le f(n) \le}$ $c_1 g(n) \le f(n) \le c_2 g(n), \forall n \ge n_0$ }

Graph

RTPNOTES.vercel.app



Problems based on Theta Notations

Problem 1

Prove that f(n) =10n^3 + 5n^2+17 ∈ θ(n^3)

1. First write the below statement
    1. C1 g(n) <= f(n) <= C2 g(n)
2. Our task is to find C1,C2 and n0
3. We know that 10n^3 <=f(n)
4. So we can equate it to C1 g(n)
    1. So we get C1= 10
5. Similarly for f(n) <=C2 g(n)
    1. Give the highest degree to all terms
    2. 10n^3+5n^2+17 -> 10n^3+5n^3+17n^3
    3. After adding we get 32n^3
    4. So C2= 32
6. Set n0 = 1

Problem 2

Prove that f(n) =2n^3 + 3n+79 ∈ θ(n^3)

1. First write the below statement

RTPNOTES.vercel.app

    1. C1 g(n) <= f(n) <= C2 g(n)
2. Our task is to find C1,C2 and n0
3. We know that 2n^3 <=f(n)
4. So we can equate it to C1 g(n)
    1. So we get C1= 2
5. Similarly for f(n) <=C2 g(n)
    1. Give the highest degree to all terms
    2. 2n^3+3n+79 -> 2n^3+3n^3+79n^3
    3. After adding we get 84n^3
    4. So C2= 84
6. Set n0 = 1

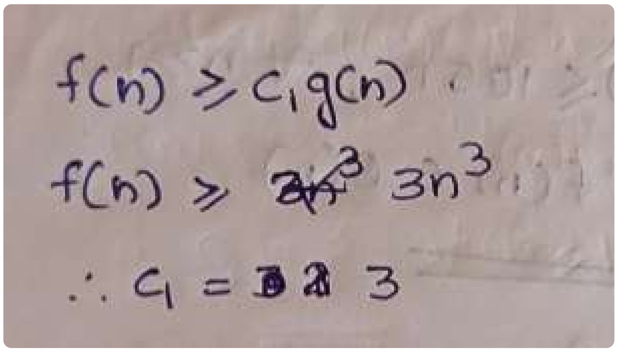
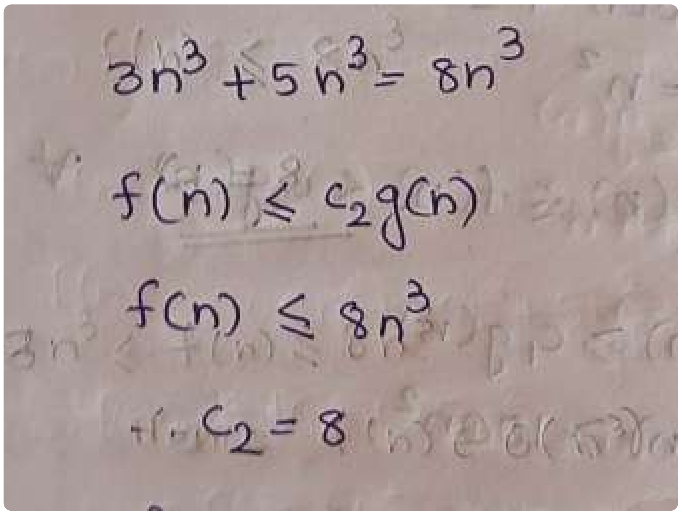
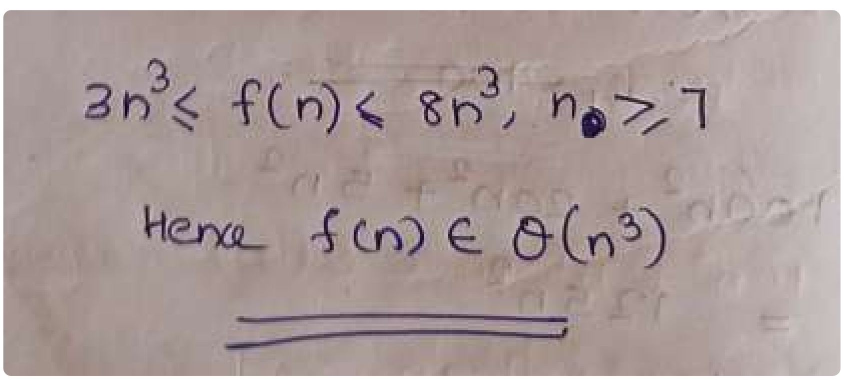
Problem 3

Prove that f(n) =10n^3 + nlogn ∈ θ(n^3)

1. First write the below statement
    1. C1 g(n) <= f(n) <= C2 g(n)
2. Our task is to find C1,C2 and n0
3. We know that 10n^3 <=f(n)
4. So we can equate it to C1 g(n)
    1. So we get C1= 10
5. Similarly for f(n) <=C2 g(n)
    1. Give the highest degree to all terms
        1. nlogn -> n^3
    2. 10n^3 + nlogn -> 10n^3 + n^3
    3. After adding we get 11n^3
    4. So C2= 11
6. Set n0 = 1

Problem 4

$$f(n) = 3n^3 - 20n^2 + 5 \in \theta(n^3)$$

1. Get the C1 value
    1. 
       $f(n) \geq c_1 g(n)$
       $f(n) \geq \cancel{3n^3}\ 3n^3$
       $\therefore c_1 = 3$
2. Get the C2 Value
    1. Dont Consider the 20n^2
    2. 
       $3n^3 + 5n^3 = 8n^3$
       $f(n) \leq c_2 g(n)$
       $f(n) \leq 8n^3$
       $\therefore c_2 = 8$
3. 
   $3n^3 \leq f(n) \leq 8n^3,\ n_0 \geq 1$
   Hence $f(n) \in \Theta(n^3)$

Big Oh Notation

- Lets consider 2 functions
    - f(n) and g(n)
    - The argument n is input size

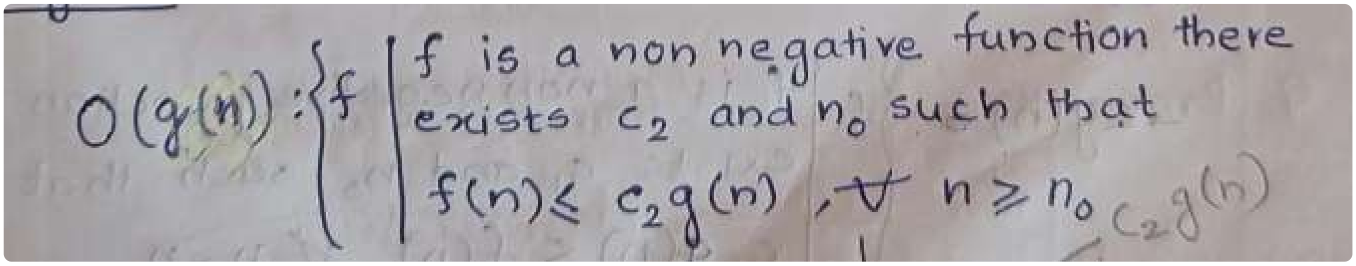
RTPNOTES.vercel.app

O(g(n)) = {f | f is a non negative function and there exist positive constants C2 and n0, such that 0 <= f(n) <= C2 g(n) ∀ n>=n0

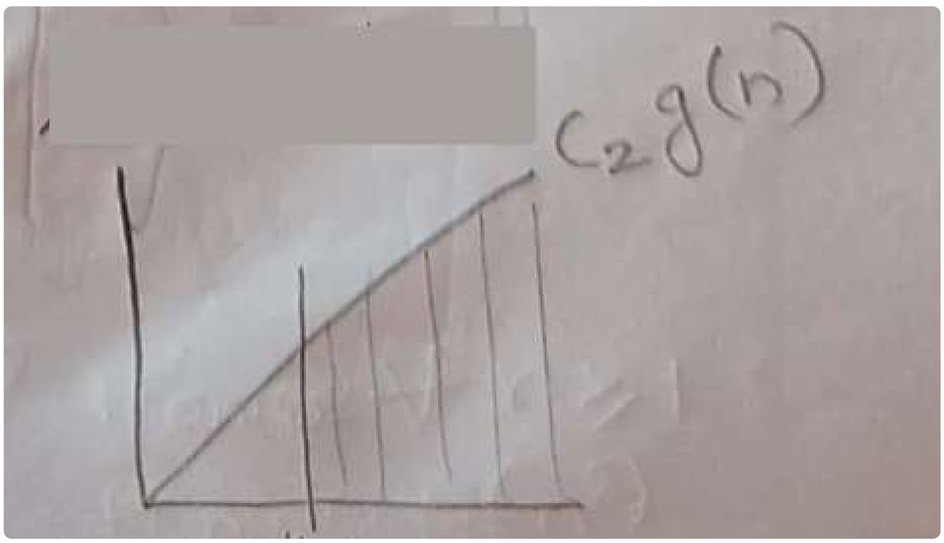
}

n0 -> Input size boundary

Definition



Graph



Problems based on Big Oh Notations

Problem 1

3n^2 ∈ O(n^2)

Use the equation

1. f(n) <=C2 g(n)
2. 3n^2 <=C2 g(n)
3. C2 = 3

Problem 2

100n^2 + 20n+ 5 ∈ O(n^2)

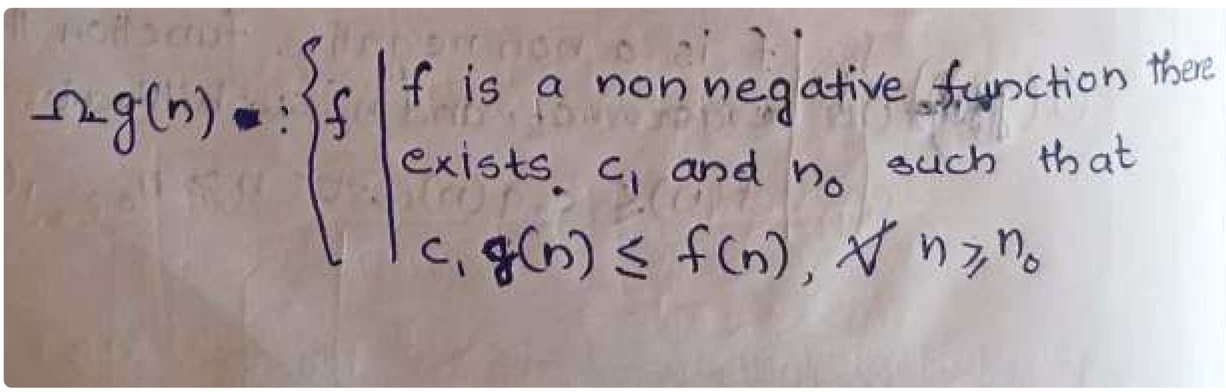
RTPNOTES.vercel.app

1. $f(n) <= C2\ g(n)$
2. $100n^2 + 20\ n^2 + 5\ n^2 <= C2\ g(n)$
    1. Apply highest degree everywhere
3. $125\ n^2 = C2\ g(n)$
4. $C2 = 125$

Big Omega

$\Omega(g) = \{f \mid f$ is non negative function, there exists positive constant c1 and n0, where
$C1g(gn) <= f(n)$
$\}$

Definition



Problems based on Big Omega

Problem 1

$100n^2 + 20n + 5 \in \Omega(n^2)$

Use the equation
1. $C1g(gn) <= f(n)$
1. $100n^2 <= f(n)$
2. $100n^2 = C1g(n)$
3. $C1 = 100$

Little oh notation

RTPNOTES.vercel.app

- Here the upperbound is not tight as in Big Oh
- In Big Oh Notiation the upperbound is
    - $f(n) \leq c\,g(n)$
- Where as in little oh notation
    - $f(n) < c\,g(n)$

Definition

$$o(g(n)) = \begin{cases} f(n) : \text{for any positive constant } c > 0, \text{ there} \\ \text{exists a constant } n_0 > 0 \text{ such that} \\ 0 \leq f(n) < c\,g(n) \ \forall\ n \geq n_0 \end{cases}$$

Problems based on Little oh notation

Problem 1

Prove that the function $5n \in o(n^2)$

- Let f(n) = 5n
- Let g(n) = n^2

We need to use this formula here

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

- When we apply this formula, we get
    - $\lim_{n \to \infty} \frac{5n}{n^2}$

RTPNOTES.vercel.app

- Cutting the n, Removing coefficient
  - $$\lim_{n \to \infty} \frac{n}{n^2}$$
  - $$\lim_{n \to \infty} \frac{1}{n}$$
- Applying the limit
  - $$= \frac{1}{\infty} = 0$$

Problem 2

$f(n) = \log n$ and $g(n) = n$. Prove that $f(n) \in o g(n)$

- We apply the same formula as before
- But We will get infinity/infinity, which is undefined
  - $$\lim_{n \to \infty} \frac{\log n}{n} = \frac{\log \infty}{\infty} = \frac{\infty}{\infty} \text{ [undefined]}$$
- So we have to use L Hospitals Rule
  - $$\lim_{x \to c} \frac{f(x)}{g(x)} = \lim_{x \to c} \frac{f'(x)}{g'(x)}.$$
- When Applying L Hospitals rule

RTPNOTES.vercel.app

- $$\lim_{n_2 \to \infty} \frac{f'(n)}{g'(n)} = \lim_{n_2 \to \infty} \frac{1/n}{1} = \lim_{n_2 \to \infty} \frac{1}{n} = \frac{1}{\infty} = 0$$

Little Omega Notation

Definition

$$\omega(g(n)) : \{ f(n) \mid \text{for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq c\,g(n) < f(n), \ \forall\, n \geq n_0 \}$$

Problems based on Little Omega

For Little Omega problems we need to use this formula

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

Problem 1

$f(n) = n^2$ and $g(n) = n$, prove that $f(n) \in \omega\, g(n)$

RTPNOTES.vercel.app

Apply the formula, And we will get infinity, which is what we need.

$$\lim_{n\to\infty} \frac{n^2}{n} = \lim_{n\to\infty} n = \infty$$

Hence proved $f(n) \in \omega(n)$

7. Time complexity,space complexity

Space Complexity

- The space complexity of an algorithm is the amount of memory it needs to run to completion
- Space complexity = FIxed part + Variable Part
  - Fixed part
    - Its independent of the characteristics oof inputs and outputs
  - Variable part
    - Dependent on the characteristics of inputs and outputs

Example

```
Algorithm mAdd(m,n,a,b,c)
{
        for i = 1 to m do
                for j = 1 to n do
                        c[i,j] := a[i,j] + b[i,j]
}
```

Space Complexity = Space for parameters and space for local variables

- m -> 1
- n -> 1
- a[] -> mn
- b[] -> mn

RTPNOTES.vercel.app

- c[] -> mn
- i -> 1
- j -> 1
- Here the space complexity = 3mn + 4

Time Complexity

- The time complexity of an algorithm is the amount of computer time it needs to run to completion
- Compilation time is excluded

8. Analysis of algorithms

Linear Search

Algorithm

> **Note**
>
> Except do while, all other loops execute n+1 times

1. n<-- length(A) -- 1
2. for i <-- 1 to n do -- n+1
    1. if A[i] = key then -- n
        1. found, break -- 1
3. end for -- 1

Best input

- When Key (The value we need to search) is the first element
- No of comparisons required = 1
- Best case complexity O(1)

Worst case input

- Key is the last element in list, or not found in list

RTPNOTES.vercel.app

- No of comparisons = n
- Worst case complexity O(n)

Average Case Input

- All positions have equal probability
- Involves probability distribution of the underlying input
- To find the complexity of average case
- First packet + Second packet .... = 1/n x 1 + 1/n x 2 + 1/n x3 + ... 1/n x n
- = 1/n(1+2+3+4+...+ n)
- = 1/n (n(n+1)/2) = **1/2n + 1/2**
- In (1/2) x n + 1/2, The largest number is n, so our complexity is O(n)

Insertion sort

Algorithm

Number	Code			Cost	Count
1	n <-length (A)			C1	1
2	for j = 2 to n do			C2	n
3		key = A[j]		C3	$n-1$
4		i = j-1		C4	$n-1$
5		while(i>0 and A[j]>key) then		C5	$\sum_{j=2}^{n} t_j$
6			A[i+1] = A[i]	C6	$\sum_{j=2}^{n} t_j - 1$
7			i = i-1	C7	$\sum_{j=2}^{n} t_j - 1$
8		A[i+1] = key		C8	$n-1$

Number	Code			Cost	Count
9	endof				

Total time taken for execution of insertion sort

- **Sum of Cost x Count**
  - $T(n)$ = C1 + C2n + C3(n-1) + C4(n-1) + C5 $\sum_{j=2}^{n} t_j$ + C6 $\sum_{j=2}^{n} t_j - 1$ + C7 $\sum_{j=2}^{n} t_j - 1$ + C8(n-1)
- Taking n as common
  - = n(C2+C3+C4+C8) + C5 $\sum_{j=2}^{n} t_j$ + (C6+C7) $\sum_{j=2}^{n} t_j - 1$ + (C1-C3-C4-C8)
- On further simplification we get
- C2n^2 + C1n + k
- The largest number = n^2
- = **O(n^2)****
  **

Best case analysis of insertion sort

- Best case occurs when the input list given is sorted in ascended order
- For each iteration of the for loop in step 2 observe that while statement in step 5 does not hold true and statements 6 and 7 wont be executing

Hence the total running time, T(n)

- Taking the statement 5
  - $\sum_{j=2}^{n} t_j$ = $\sum_{j=2}^{n} 1$

T(n) = an+b
This is a linear polynomial

Worst case

- Worst case happens when its sorted in reverse order
- Executes statement 5,6,7
- T(n) = Cn^2 + C'n+C'''

Average case

- This involves distribution of underlying input list.

RTPNOTES.vercel.app

- On an average, half of the elements in the sorted list are greater than the key and they are moved to their correct position
- Statement 5 becomes
  - $\sum_{j=2}^{n} t_j$ = $\sum_{j=2}^{n} t_j/2$

$\sum_{j=2}^{n} t_j/2$ = 1/2 x (n(n+1)/2 - 1)

$\sum_{j=2}^{n} t_j/2 - 1$ = 1/2 x (n(n+1)/2)

- The average case complexity is O(n^2)

9. Best, Worst and Average Case Complexities

- Best case: It is the minimum number of steps that can be executed for a given parameter
- Worst Case: It is the maximum number of steps that can be executed for a given parameter
- Average Case: It is the average number fo steps that can be executed for a given parameter

Example: Linear search

- Best Case: Search data will be in first location of the array
- Worst case: Search data does not exist in the array
- Average Case: Search data is in the middle of the array