Unit 1

* Sequence : In sequence we list the number together.

* Arithmetic sequence

- A seq. is arithmetic if the diff^n of consecutive terms are the same.

$$a_{k+1} - a_k = d$$ where, d = common diff^n

- 1st term = a
- 2nd term = a + d
- 3rd term = a + 2d
- nth term = $a + (n-1)d$
- Sum of term = $S_n = \frac{n}{2}(a_1 + a_n)$

Ex: a = 1, d = 3, Calculate 78th term

$a + (n-1)d$

$1 + 3 \times 77$

232

Ex: 3, 11, 19 .... find S_6

$a_n = a + (n-1)d$

$a_6 = 3 + 5 \times 8$

$a_6 = 43$

$S_6 = \frac{n}{2}(a_1 + a_n)$

$= \frac{6}{2}(3 + 43)$

$S_6 = 138$

* Geometric sequence and series.

A seq. is geometric if ratio of consecutive term are same.

2, 8, 32, 128, 512 ....

Common Ratio = 4

- 1st term = a
- 2nd term = ar
- 3rd term = ar^2
- nth term = $ar^{(n-1)}$
- $S_n = a(1 - r^n) / 1 - r$

* Sum of finite geometric sequence

$$S_n = \sum_{i=1}^{n} a_1 r^{i-1} = a_1 \frac{1 - r^n}{(1 - r)}$$

Ex: 5 + 10 + 20 + 40 + 80 + 160 + 320 + 640 = ?

$n = 8$, $r = 2$

$a_1 = 5$

$S_n = a_1 \frac{1 - r^n}{(1 - r)}$

$= 5\left(\frac{1 - 2^8}{-1}\right)$

$= 5 \times 255$

$\therefore S_n = 1275$

Scanned with OKEN Scanner

* Algorithm

i/p → Algorithm → o/p
↑
Problm

* Specification of algo
- i/p, o/p, definiteness, finiteness, effective i/p.

* Performance analysis of algorithm
It is the process of calculating the space req. by an algorithm

1. Space Complexity: for a program / algorithm the space is required to
i. store program
ii. store constant variables
iii. store variables.

```
int square (int a)
{
    return a*a;
}
```

- 2 bytes are req. to store variable 'a' & 2 bytes to return the value (result). It takes 4 bytes of memory.

Suppose a = 2 , It takes 4 bytes
" a = 8 —— " ——
" a = 28 —— " ——

If it takes same space all the time then it is called constant space complexity.

Code

```
int sum (int A[], int n) {
    int sum = 0, i ;  → 2 → 4
    for (i=0; i<n; i++) {
        sum = sum + A[i];
    }
    return sum;  → 2 bytes
}
```

In above code, int A[] takes 2*n space
2 cuz integer datatype
n cuz of size of array
2 byte of memory for each sum & i
—— " —— for array A
—— " —— for integer n

→ 2n+8

In above code, the space complexity required to an algo. will inc. or dec. according to the value of 'n'.
Therefore, it is called as linear space complexity.

* Time Complexity
Processor depend on TC. It also depends on bit 64 bytes/bit req. less TC than 32 bytes/bits.

* Parameter of TC
1. Uniprocessor
2. 32 bit
3. 1 unit Read/write
4. 1 unit for assignment & return (ALU)

Scanned with OKEN Scanner

Ex: int sum (int a, int b) {
    return (a+b);
}
1 unit, 1 unit

Requires 2 unit of time for above code.

→ Constant TC, Irrespective of i/p for 'a' & 'b'

Ex:
```
int sum (int A[], n) {
    int sum = 0, i ;          1 unit
    for (i=0; i<n; i++) {     2n + 2 unit
        sum = sum + A[i]; }   2n unit
    return sum;               1 unit
}
```

- 4n+4

Time complexity will change based on value of 'n'. Hence it is called Linear TC.

* For 'for Loop'

i=0, Initialization happens at onces, then i<=n cond^n check and n+1 & i++, nth time.

sum = sum + A[i]

sum take 1 unit, 1 unit take A[i]. Therefore, it take 2 unit & also execute n time. therefore it take 2n

1 + 1 = 2 } 2n
n = n }

Above code take total 4n+4 time for execution and it will get change accordingly to value of 'n'.

If amount of time required by an algo. is increasing with inc. of i/p value then that TC is called linear TC

Ex:
```
for (      ) {  —— n times
    for (      ) { ——> n times
    }
}
```
O(n²)

Ex:
```
for (      ) {  —— n times
}
for (      ) {  —— n times
}
```
O(2N).

* Asymptotic Notation

A asymptotic notation is a language that allows us to analyse an algorithm own time/ performance. It is mathematical representation of algo. complexity.

1. Best Case : Min time req.
2. Avg " : Avg " "
3. Worst " : Max " "

Scanned with OKEN Scanner

* Part of Notation

1. Big O : Upper bound, worst
2. Omega : Lower bound, Best
3. Theta : Tight bound, avg

1. Big O notation

- Represent max time required by an algorithm for all i/p value.

Mathematical func.

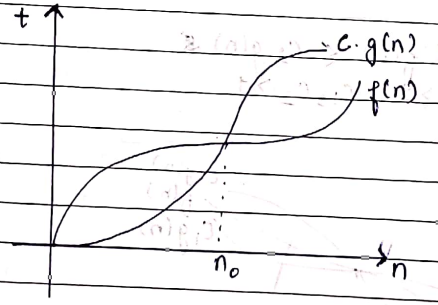
$$0 \leq f(n) \leq c.g(n)$$

for given func. $g(n)$

$O(g(n)) = f(n)$ where

$$0 \leq f(n) \leq c.g(n)$$

there exists a +ve const 'c' and 'no' for all $n \geq n_0$, $n_0 \geq 1$, $c > 0$



Here, the function $g(n)$ is a asymptotic upper bound for $f(n)$

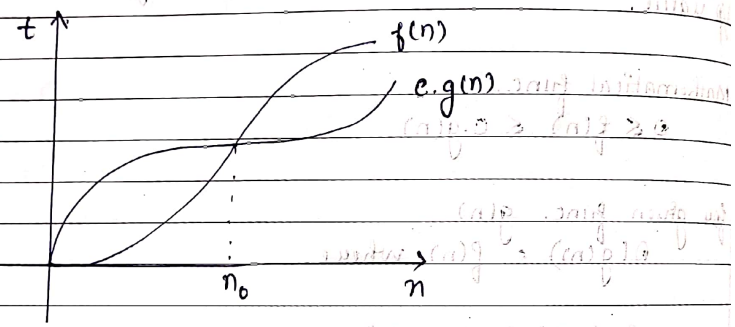
$$\boxed{f(n) = O.g(n)}$$

2. Big - omega notation (Ω)

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

$$0 \leq c.g(n) \leq f(n)$$

for $n \geq n_0$, $c > 1$, $n_0 \geq 1$



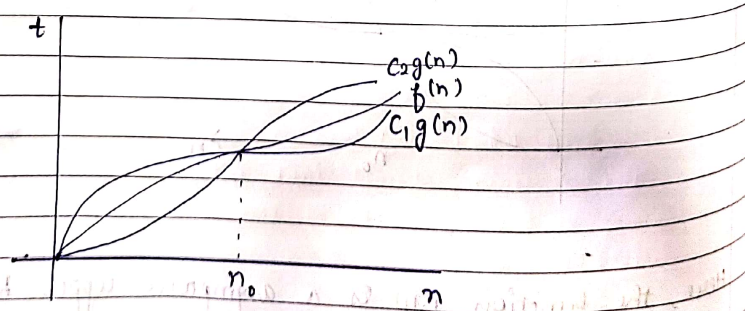
* Theta Notation

$\Theta\, g(n) = f(n)$

there exist a +ve const. c_1 and c_2

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

for $n \geq n_0$, $n_0 \geq 1$, $c_1, c_2 > 1$



Lower bound + UB = TB.

Scanned with OKEN Scanner

$g(n)$: is greater value of n
$c.g(n)$: Is const value of n
$f(n)$: Complete func.

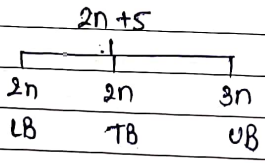
1. $f(n) \leq cg(n)$ then $f(n) = O\,g(n)$] UB

2. $cg(n) \leq f(n)$ then $f(n) = \Omega\, g(n)$] LB

3. $c_1 g(n) \leq f(n) \leq c_2 g(n)$ then $f(n) = \theta\, g(n)$] TB

Q. Find out UB, LB, TB range for following function
$2n+5$

consider $f(n) = 2n+5$
$g(n) = n$
$c.g(n) = 2$



* To calculate 1st range of UB

$f(n) \leq c.g(n)$
$2n+5 \leq 3n$

start from $n=1$

$2(1)+5 \leq 3(1)$
$7 \leq 3$ false

value of n is not satisfied cond^n ∴ false

$n=2$
$9 \leq 6$ — false

$n=3$
$11 \leq 9$ — false

$n=4$
$13 \leq 12$ — false

$n=5$
$15 \leq 15$ — True

therefore, $f(n) \leq cg(n)$
where, n

Scanned with OKEN Scanner

$n=3$ $\quad 2*3+5 \leq 3*3$

$11 \leq 9$ ⟶ False

$n=4$ $\quad 2*4+5 \leq 3*4$

$13 \leq 12$ ⟶ False

$n=5$ $\quad 2*5+5 \leq 3*5$

$15 \leq 15$ ⟶ True

Therefore, $f(n) \leq c\, g(n)$ where $n \geq 5$ & $c = 3$

$\therefore f(n) = O(g(n))$ — Imp

Here the function $g(n)$ is asymptotic upper for $f(n)$

$f(n) = O(g(n))$

+ To calculate the 1st range of LB

Lower bound range

$f(n) \geq c\, g(n)$

where $f(n) = 2n+5$

$g(n) = n$

$c = 2$

start from $n=1$

$f(n) \geq c \cdot g(n)$

$2(1)+5 \geq 2(1)$

$8 \geq 2$ ⟶ True

Here $f(n) \geq c\, g(n)$

where $n \geq 1$ & $c = 2$

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

+ To calculate the 1st range.

Right bound range

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

where $f(n) = 2n+5$

$g(n) = n$

$c_1 = 2n$, $c_2 = 3n$

start from $n=1$

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

$2(1) \leq 2(1)+5 \leq 3(1)$

$2 \leq 8 \leq 3$

start from $n=2$

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

$2n \leq 2n+5 \leq 3n$

$2(2) \leq 2(2)+5 \leq 3(2)$

$4 \leq 8 \leq 6$

$\therefore c_1 g(n) \leq f(n) \leq c_2 g(n)$

$2n \leq 2n+5$ $\quad 2n+5 \leq 3n$

where $c_1 = 2$

$c_2 = 3$

$2n \leq 2n+5 \leq 3n$

Scanned with OKEN Scanner

$n = 1$

$2(1) \leq 2(1) + 5 \leq 3(1)$

$2 \leq 7 \leq 3$ — False

$n = 5$

$2(5) \leq 2(5) + 5 \leq 3(5)$

$10 \leq 15 \leq 15$ — True

Here $c_1 g(n) \leq f(n) \leq c_2 g(n)$

where $n \geq 5$, $c_1 = 2$, $c_2 = 3$

$f(n) = \Theta\, g(n)$

ex2:- Solve :- $10n^2 + 4n + 2$

Soln:- Consider $f(n) = 10n^2 + 4n + 2$

$g(n) = n^2$

$c = 10$

$10n^2 + 4n + 2$

LB $10n^2$ | TB $10n^2$ | UB $11n^2$ $(10+1)$

To calculate the 1st range of UB

Upper bound

$f(n) \leq c\,g(n)$

$10n^2 + 4n + 2 \leq 10n^2$

Start from $n = 1$

$10(1)^2 + 4(1) + 2 \leq 10(1)$

$10 + 4 + 2 \leq 10$

$16 \leq 10$

Start from $n = 2$

$10(2)^2 + 4(10) + 2 \leq 10(22)$

$10 \times 4 + 40 + 2 \leq 20$

$40 + 42 \leq 20$

$82 \leq 20$

Start from $n = 5$

$10(5)^2 + 4(55) \leq 10(11)^2$

$10 \times 25 + 220 + 2 \leq 50$ 275

$250 + 22 \leq 50$

$292 \leq 50$

$272 \leq 275$ — True

Here $f(n) \leq c\,g(n)$

where $n \leq 5$

& $c = 10$

Here fun^n $g(n)$ is asymptotic upper for $f(n)$

$f(n) = O\, g(n)$

Lower bound

$f(n) \geq c\,g(n)$

where $f(n) = 10n^2 + 4n + 2$

$g(n) = n^2$

$c = 10$

Start from $n = 1$

$f(n) \geq c\,g(n)$

$10n^2 + 4n + 2 \geq 10n^2$

$10(1) + 4(1) + 2 \geq 10(1)$

$16 \geq 10$

Here $f(n) \geq c\,g(n)$

where $n \geq 1$, $c = 10$

$\boxed{f(n) = \Omega(g(n))}$

— To calculate tight bound

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

where $f(n) = 10n^2 + 4n + 2$

$g(n) = n^2$

$c_1 = 10$

$c_2 = 11$

$\therefore c_1 g(n) \leq f(n) \leq c_2 g(n)$

$10n \leq 10n^2 + 4n + 2 \leq 11n$

where $n = 5$

— True

Here $c_1 g(n) \leq f(n) \leq c_2 g(n)$

where $n \geq 5$, $c_1 = 10$, $c_2 = 11$

$\boxed{f(n) = \Theta(g(n))}$

Insertion sort code.

```
for (i=1; i<n; i++) {
    key = A[i];
    j = i-1;
    while (j>=0 && A[j]>key) {
        A[j+1] = A[j];
        j--;
    }
    A[j+1] = key;
}
```

* Sort in ascending order

Compare (1 unit), Swap (1 unit)

12 13 18 19 23

Compare	Swap
1	0
1	0
1	0
1	0
4 unit	

12 13 18 19 23

1 1 1 1

Comparisons : $n-1$

Array size : n.

TC : $O(n)$

$O(n)$: 1 2 3 4 5 → Only compare (n)

$O(n^2)$: 5 4 3 2 1 → compare + swap $n + n$

1. Best case : $O(n)$

When?

The i/p array is already sorted like :

1, 2, 3, 4 . . . . n.

Just req. comparison & not swapping. Hence comparison time = TC.

Why $O(n)$?

For each element $A[i]$, the cond^n $A[j] > key$ fails on 1st check and so no shifting occurs.

The inner while loop runs 0 times for each i.

Only one comparison per element is made (to check if it's correct position.

Sin already its

Total operation

Outer loop runs $n-1$ times. Inner loop : 0 comparison + 0 swap

Time = $O(n)$

	Compare	Swap
6 5 | 4 3 2	1	1
5 6 4 | 3 2	2	2
4 5 6 | 3 2	3	3
3 4 5 6 | 2	4	4
2 3 4 5 6	⋮	⋮
sorted	$S_n = \frac{n}{2}(n+1)$	
	$= \frac{n^2+1}{2}$	

$TC = O(n^2)$.

$1+2+3+4 \ldots . n \longrightarrow \Sigma = \frac{n}{2}(n+1)$

$1+2+3+4 \ldots . n-1 \longrightarrow \Sigma = \frac{n}{2}(n-1)$

$\Rightarrow TC : O(n^2)$

- Best case TC for Insertion sort $O(n)$
- Worst —"— $O(n^2)$

Avg case complexity. : $O(n^2)$.

1 | 3 2 4 6 5 : Some are sorted, some not.

	Compare	Swap
1 | 3 2 4 6 5	1	0
1 3 | 2 4 6 5	2	1
1 3 2 | 4 6 5	3	0
1 2 3 | 4 6 5	4	0
1 2 3 4 | 6 5	5	1
1 2 3 4 6 | 5		
1 2 3 4 5 6		

1	2	3	4	5	6

* Selection Sort

Works repeatedly by selecting smallest (or largest) element from unsorted part and moving it to the sorted part of the array.

working

- Size n
- Start at index 0
- Find min. in unsorted part.
- Swap it with the element at current idx.
- Move boundary of sorted and unsorted part by 1.
- Repeat steps 2-4 until array is sorted.

Scanned with OKEN Scanner

* Syllabus for MSE

Unit 1

Q1. Geometric / Arithmetic series

2. Asymptotic notations : Theory + numerical
   - Func. → U.B, L.B, T.B
   - Graph

3. Sorting.

4. Complexity of code → Space, Time

Unit 2

Q4. All recursive method
- Tree method
- Substitute "
- Change of variable
- Characteristic eq. → Homogeneous, Non-Homo

Unit 3

Q5. Greedy algorithm

Q6. knapsack problem : Justify ques.

Selection Sort

Total comparison :- $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$

$= O(n^2)$

Note :- In Best case, array is already in sorted order :- $O(n^2)$

In Average case, array is in sorted as well as in unsorted order.

For worst case, array is in unsorted order.

Time complexity Analysis

+ Best case : $O(n^2)$

Even if array is already sorted, selection sort still scans unsorted part every time.

+ Worst case : $O(n^2)$

- If array is in reverse order, it still makes same number of comparisons.
- If array is in ~~sorted~~ reverse order, selection sort still :-
  - Make same number of comparisons
  - Just perform different swaps

  so, worst case = $O(n^2)$

Scanned with OKEN Scanner

+ Average case :- $O(n^2)$

Total no. of comparisons is always
$(n-1) + (n-2) + \cdots 1 = \frac{n(n-1)}{2}$

$= O(n^2)$

Note:
Comparison :- $O(n^2)$
Min^m selection :- $O(n^2)$
Swaps :- $O(n)$
Overall :- $O(n^2)$

Q Comparison for insertion, selection such as best case, worst case, average.

Case	Insertion	Selection
Best	$O(n)$	$O(n^2)$
worst	$O(n^2)$	$O(n^2)$
Avg	$O(n^2)$	$O(n^2)$

Type of array
+ U array is in reverse order U = [10 9 8 7
+ V array is in sorted order V = [1 2 3 4 5
+ W array is in unsorted order W = [10 3 9
+ X array in which element is same.
X = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Interview:- tree is a graphy but graphy is not a tree

Bubble sort

Time Complexity Analysis

+ Best case : $O(n)$
Occurs when array is already sorted. Only one pass is needed with no swaps.

+ Worst Case : $O(n^2)$

Occurs when array is in reverse order. Every pair of element needs to be compared & swapped

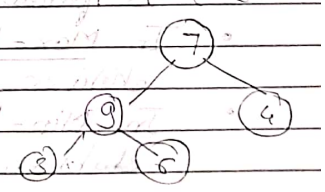
+ Average Case :- $O(n^2)$
Each element is compared with other element.

Heap Sort

Heap sort is called balance binary tree.

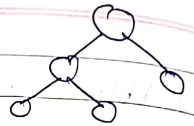
7 9 6 3 6

- Binary tree
  - binary tree
  - Full binary tree
  - Complete binary tree.



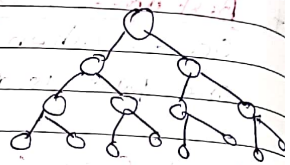
* binary tree can have 0, 1, 2 child (i.e almost 2 child)

* Full binary tree



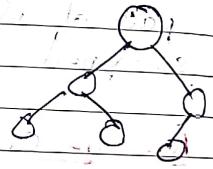
The tree that have 0 or 2 child is called full binary tree.

* Complete binary tree



It should be completely filled.

* almost complete binary tree



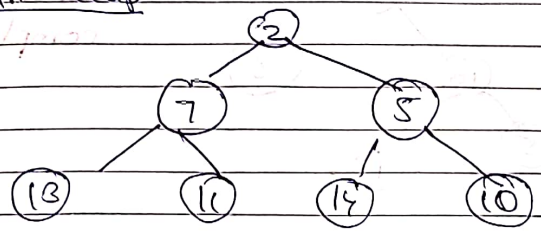
Heap Sort

Note:- Heap Sort based on almost complete binary tree.

- ~~It follows~~
- In Max-heap, every parent is greater than its children
- In Min-heap, every parent is less than its children.

In heap sort, there is no any condition. (i.e greater is always in right or less is in left side)

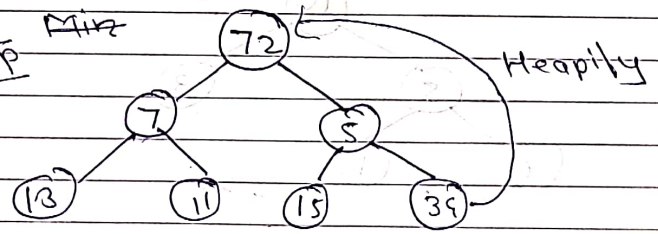
Min heap



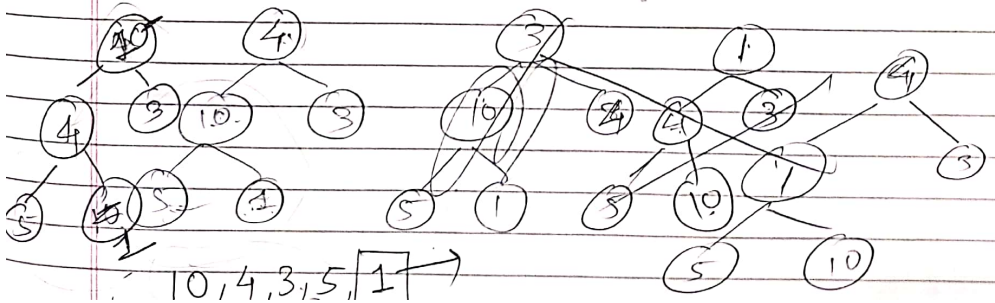
steps of heap sort (Using Max heap)

- build a max heap from unsorted array
- Swap the root (~~element~~ max.m element) with last element.
- Reduce the heap size by 1.
- Heapify the root again to maintain max-heap property.
- Repeat step 2-4 until heap size becomes 1.

Max heap ~~Min~~

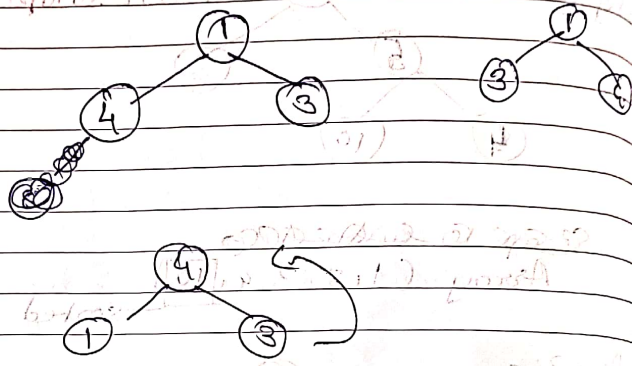


eg] Sort [4, 10, 3, 5, 1] using heap sort.



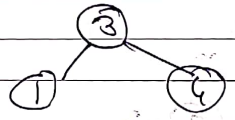
Scanned with OKEN Scanner

Pass 3



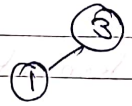
Max heap

Replace last element with 1st

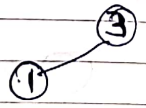


Array [3 1 4]
sorted

Pass 4



Max heap



already in max-heap.

Array [3, 1] ∴ [10, 5, 4, 3, 1]

∴ [1, 3, 4, 5, 10]

Q Comparison of time complexity will be a>b
i.e n, O(1), O(logn), O(n), O(n logn), O(n²), O(2ⁿ)

Pass 4

Swap 4 with 3

Pass 4

- Swap 3 with 1 [1, 3, 4, 5, 10]
- Done

exhq step.

Time Complexity of heap sort

Case	Time complexity
Best	O(n(logn))
worst	O(n logn)
Average	O(n logn)

Scanned with OKEN Scanner

17/07/25

Unit: 2

A recurrence relation is an eqn. which defines term of itself

$$n! = n*(n-1)!$$

$1! = 1$

$0! = 1$

$$n! = \begin{cases} 1 & \text{when } n = 0, 1 \\ n*(n-1)! & \text{otherwise.} \end{cases}$$

```
int fact (int num) {
    if (num == 0) {
        return 1;
    }
    return (num * fact (num - 1));
}
```

Calculate the complexity of recursive function we have following methods

1. Substitution Method
2. Tree Method
3. Master Method
4. Characteristics eqn. Method.
5. Change of variables.

Q.1 Solve the following Recursive function by using Substitution Method.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 1 + T(n-1) & \text{otherwise} \end{cases}$$

Soln: $T(n) = 1 + T(n-1)$ —— (1)

find $T(n-1)$

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

$= 1 + T(n-2)$ —— (2)

find $T(n-2)$

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

$= 1 + T(n-3)$ —— (3)

Put eqn (2) in eqn (1)

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

$T(n) = 2 + T(n-2)$ —— (4)

Put eqn (3) in eqn (4)

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

$T(n) = 3 + T(n-3)$

⋮

k times

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

Consider, $n - k = 1$ $\Rightarrow$ $k = n-1$

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

$= n - 1 + 1$

$\therefore$ $T(n) = n$

$T(n) = O(n)$

18/07/25

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{otherwise} \end{cases}$$

Q2. Find time complexity of Recursive function.

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

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

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

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

$$= T(n-3) + n - 2 \quad \text{---} \quad (3)$$

Put eqn (2) in eqn (1)

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

Put eqn (3) in eqn (4)

$$T(n) = T(n-3) + n - 2 + n - 1 + n.$$

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

$$\vdots$$

$$= T(n-k) + \underbrace{n-(k-1)}_{(n-k)+1} + \underbrace{n-(k-2)}_{(n-k)+2} + \dots n$$

Consider, $n - k = 1$

$$T(n) = T(1) + (n-k) + 1 + (n-k) + 2 + \dots + n.$$

$$= T(1) + 1 + 1 + 1 + 2 + \dots + n$$

$$= 1 + 1 + 1 + 1 + 2 + \dots + n$$

$$= 1 + 2 + 3 + \dots + n.$$

$$= \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

$\therefore$ $\boxed{T(n) = O(n^2)}$

Q3.

$$T(n) = \begin{cases} 4 & \text{for } n = 1 \\ 2T\left(\frac{n}{2}\right) + 4n & \text{otherwise.} \end{cases}$$

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

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n/2}{2}\right) + 4 \cdot \frac{n}{2}$$

$$= 2T\left(\frac{n}{4}\right) + 4\frac{n}{2} \quad \text{---} \quad (2)$$

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

$$= 2T\left(\frac{n}{8}\right) + 4\frac{n}{4} \quad \text{---} \quad (3)$$

Put eqn (2) in eq (1)

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

$$= 4T\left(\frac{n}{4}\right) + 8\frac{n}{2} + 4n \quad \text{---} \quad (4)$$

Put eqn (3) in eqn (4)

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + 4\frac{n}{4}\right] + 8\frac{n}{2} + 4n.$$

$$T(n) = 8T\left(\frac{n}{8}\right) + 16\frac{n}{4} + 8\frac{n}{2} + 4n.$$

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

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

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

Consider, $\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n.$

$T(n) = 2^{\log_2 n} T(1) + \log_2 n \ (4n)$

$= n^{\log_2 2} T(1) + \log_2 n \ (4n)$

$\log_2 2 = 1$

$= n * 4 + 4n \log_2 n$

$= 4n + 4n \log_2 n$

$= 4(n + n \log_2 n)$

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

---

* Recursion Tree Method.

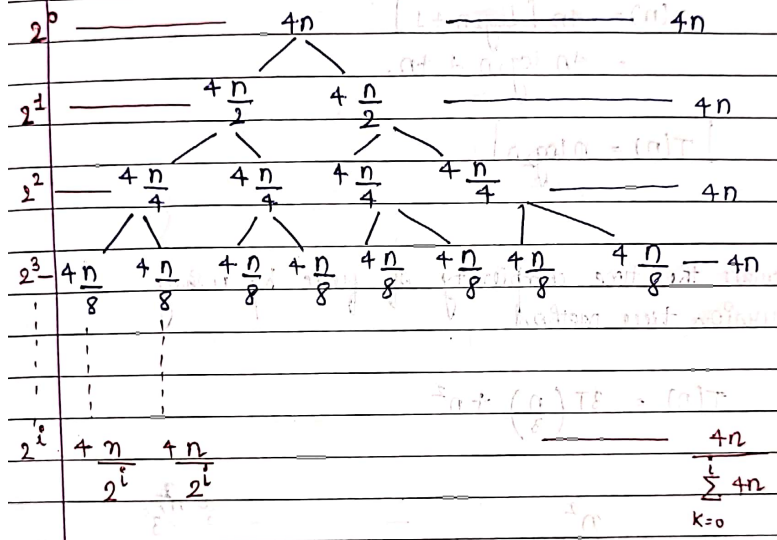
$$\sum_{K=N_1}^{N_2} \alpha^K = \frac{\alpha^{N_1} - \alpha^{N_2+1}}{1-\alpha} \quad ; \ \alpha \neq 1$$

$= N_2 - N_1 + 1 \quad ; \ \alpha = 1$

---

2. Solve the following recurrence rel^n by using recursion tree method.

$$T(n) = \begin{cases} 4 & ; \ n = 1 \\ 2T(n/2) + 4n & ; \ \text{otherwise} \end{cases}$$



$n - k = 1$

$\frac{n}{2^k} = 1$

Consider

$\frac{n}{2^i} = 1$

$n = 2^i$

$\therefore$ $\boxed{i = \log_2 n}$

Scanned with OKEN Scanner

$$T(n) = \sum_{i=0}^{\log_2 n} 4n$$

$$T(n) = 4n \sum_{i=0}^{\log_2 n} 1$$

$$T(n) = 4n \left[\log_2 n - 0 + 1\right]$$

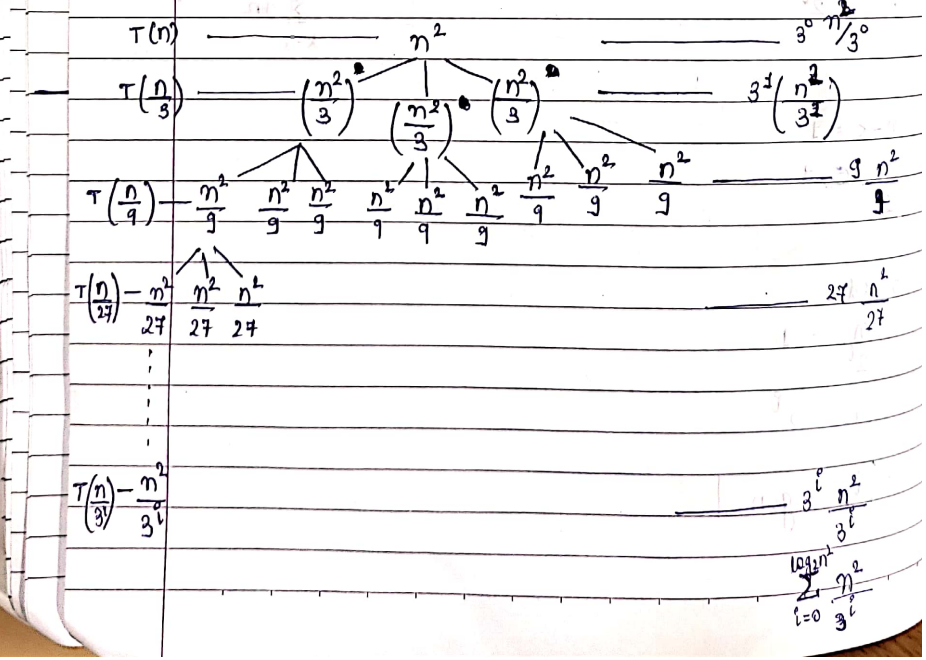
$$T(n) = 4n \left[\log_2 n + 1\right]$$

$$= 4n \log_2 n + 4n.$$

$$\boxed{T(n) = n \log_2 n}$$

Q2. Calculate the time complexity of func. by using recursion tree method

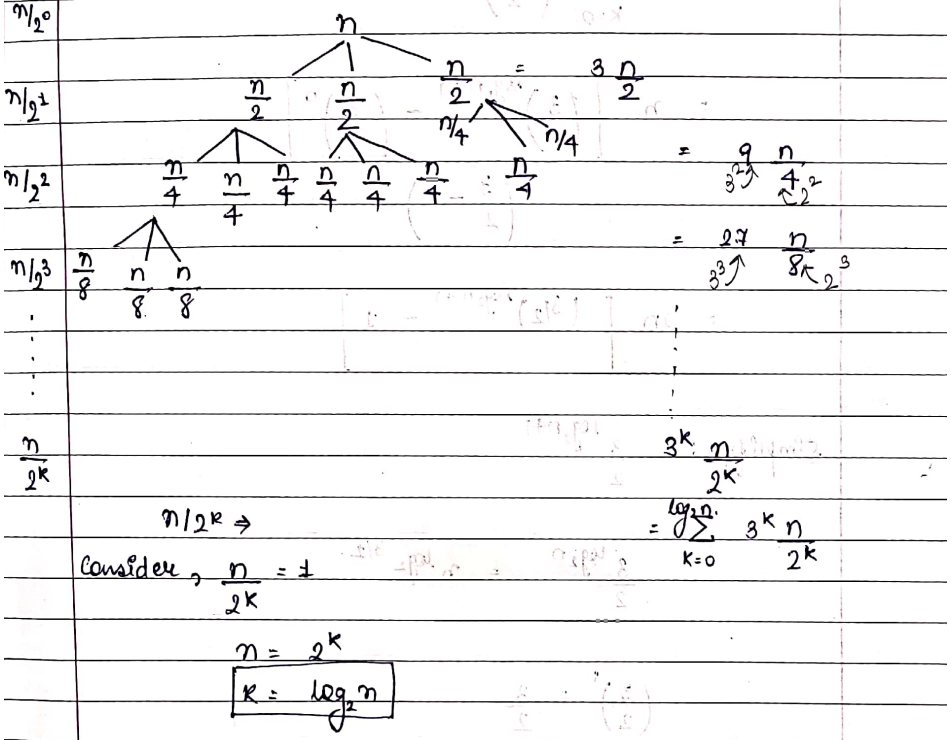
$$T(n) = 3T\left(\frac{n}{3}\right) + n^2$$



Q. $T(n) = 3T\left(\frac{n}{2}\right) + n$

In this problem, the function split, the input of size 'n' into three sub problem of size n/2

The work done outside of recursive call at each level is n.



$n/2^k \Rightarrow$

Consider, $\frac{n}{2^k} = 1$

$n = 2^k$

$\boxed{k = \log_2 n}$

Scanned with OKEN Scanner

$$T(n) = \sum_{k=0}^{\log_2 n} \frac{3^k}{2^k} n$$

$$= \sum_{k=0}^{\log_2 n} \left(\frac{3}{2}\right)^k \cdot n$$

$$= n \sum_{k=0}^{\log_2 n} \left(\frac{3}{2}\right)^k$$

$$= n \cdot \frac{\left[\left(\frac{3}{2}\right)^{\log_2 n + 1} - \left(\frac{3}{2}\right)^0\right]}{\left(\frac{3}{2} - 1\right)}$$

$$= 2n \left[\left(3/2\right)^{\log_2 n + 1} - 1\right]$$

Simplify : $\frac{3}{2}^{\log_2 n + 1}$

$$\frac{3}{2}^{\log_2 n} = n^{\log_2 3/2}$$

$$\left(\frac{3}{2}\right)^1 = \frac{3}{2}$$

$$\frac{3}{2}^{\log_2 n + 1} = \frac{3}{2} \cdot n^{\log_2 3/2}$$

$$= 2n \left[\frac{3}{2} n^{\log_2 3/2} - 1\right]$$

$$= 2n \frac{3}{2} n^{\log_2 3/2} - 2n$$

$$= \frac{2n}{2} 3 n^{\log_2 3/2}$$

$$= 3n \cdot n^{\log_2 3/2}$$

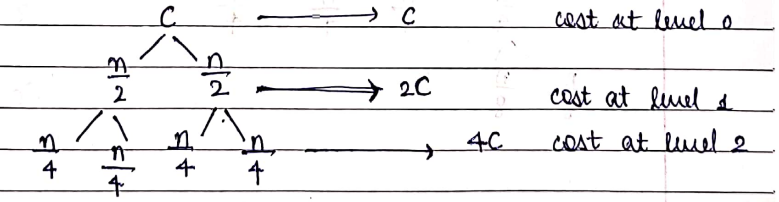
$$= n \cdot n^{\log_2 3/2}$$

Q. $T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$

here $f(n)$ is the cost of work done at each node, Not for whole level.

$$T(n) = 2\,T\left(\frac{n}{2}\right) + C$$

here C is constant, → i.e each node does constant work.



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

At level 0, we solve the problem of size n. The cost is n^2

when split into 3 subproblem of size $\frac{n}{3}$

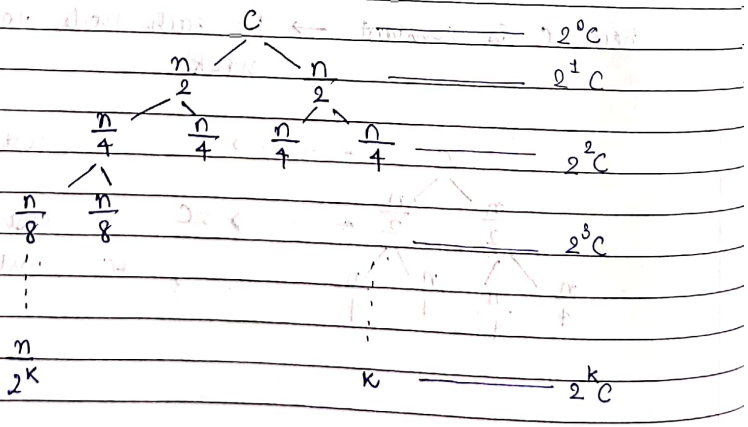
So in the next level we have 3 nested of each size and at each node the cost is $\left(\frac{n}{3}\right)^2$.

Q. Solve the following recursive function By using recursive tree method

$$T(n) = 2T\left(\frac{n}{2}\right) + c \qquad n > 1$$
$$n = 1$$

Soln: $f(n) = c$

here c, is constant i.e each node does constant work



Consider, $\frac{n}{2^k} = 1$ i.e $n = 2^k$

$k = \log_2 n$

$$= c \sum_{k=0}^{\log_2 n} 2^k$$

$$= [2^0 + 2^1 + 2^2 + \cdots + 2^{\log_2 n}]$$

$$= [1 + 2 + 4 + \cdots + 2^{\log_2 n}] \quad \text{—— GP}$$

$$\left[\text{formula of GP} = \frac{a[r^n r^n - 1]}{r - 1}\right]$$

By putting this $= 1\,[2^{\log_2 n} - 1]$

$$= 1\,[n^{\log_2 2} - 1]$$

$T(n) = 1\,[n^1 - 1]$ [here degree n & 1 ∴ we consider n because $n > 1$]

$T(n) = O(n)$

* Master Method to solve Recurrence Relation

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

let $a \geq 1$, $b > 1$ be constant.

let $f(n)$ be a function & $T(n)$ be define non-negative integers.

then $T(n)$ can be bounded as follows.

Note: solve all recursion problem by using master method (as much as possible)

case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ $\epsilon > 0$ for some constant
then
$T(n) = \theta(n^{\log_b a})$ $\boxed{f(n) \leq n^{\log_b a}}$

[ϵ is negligible value, sometime use consider, sometime not]

case 2: If $f(n) = \theta(n^{\log_b a})$ then ——— $\boxed{f(n) = n^{\log_b a}}$
$T(n) = \theta(n^{\log_b a} \cdot \log n)$

case 3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$
& If $a f(\frac{n}{b}) \leq c f(n)$ ——— $\{c < 1\}$

then $\boxed{T(n) = \theta f(n)}$ $\boxed{f(n) \geq n^{\log_b a}}$

* Every time we do comparison bet^n $f(n)$ & $n^{\log_b a}$

Note:

- $f(n) = O(n^{\log_b a - \epsilon})$
  $f(n) \leq n^{\log_b a}$

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

- $f(n) = \Omega(n^{\log_b a + \epsilon})$
  $f(n) \geq n^{\log_b a}$

Q. Solve following recurrsion relⁿ by using master method.

$T(n) = 9T(\frac{n}{3}) + n$

Sol: $T(n) = 9T(\frac{n}{3}) + n$ ——— (1)

Compare eqn (1) with standard eqn.

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

where $a = 9$, $b = 3$, $f(n) = n$

∴ $f(n)$
$n^{\log_b a}$ by putting value of a & b
$n^{\log_b a} = n^{\log_3 9}$

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

here, $f(n) \leq n^{\log_b a}$ – standard

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

∴ by case 1 of master theorem

$T(n) = O(n^{\log_b a})$

$\boxed{T(n) = O(n^2)}$

Note: If $f(n) = O(n^{\log_b a - \epsilon})$ where ϵ is negligible
where $\epsilon = 1$
∴ $f(n) = O(n^{\log_b a - 1})$ where $n^{\log_b a} = n^2$
$f(n) = O(n^{2-1})$

$f(n) = n$ ——— then it belong to case 2

Q. $T(n) = T\left(\frac{2n}{3}\right) + 1$ ——— ① ∴ $T(n) = T\left(\frac{n}{3/2}\right)$

Compare with ① with standard form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a = 1$, $b = \frac{3}{2}$, $f(n) = 1$

$f(n)$

$n^{\log_b a}$ by putting value of a & b

$n^{\log_b a} = n^{\log_{3/2} 1}$

$= n^0$

$n^{\log_b a} = 1$

$f(n) = n^{\log_b a}$ where $f(n) = 1$

$1 = 1$

As per case 2 of master method

$T(n) = \theta\left(n^{\log_b a} \cdot \log n\right)$

$T(n) = \theta(1 \cdot \log n)$ Here $n^{\log_b a} = 1$

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

∴ $T(n) = T\left(\frac{n}{3/2}\right) + 1$

& b

$) = 1$

$n^{\log_b a} = 1$

Q. $T(n) = 2T\left(\frac{n}{2}\right) + n$ ——— ①

Comparing with ① with standard form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a = 2$, $b = 2$, $f(n) = n$

$n^{\log_b a}$ by putting value of a & b

$n^{\log_b a} = n^{\log_{1/2} 2}$

$= n$

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

$n = n$ where $f(n) = n$

As per case 2 of master method

$T(n) = \theta\left(n^{\log_b a} \cdot \log n\right)$

$T(n) = \theta(n \cdot \log n)$ Here $n^{\log_b a} = n$

$\boxed{T(n) = \theta(n \cdot \log n)}$ → this is eqn. of merge sort

Scanned with OKEN Scanner

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

Solve using Master Method.

Comp. with

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$a = 2,\ b = 2\quad f(n) = n^4$

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

Compare $f(n)$ & $n^{\log_b a}$

$f(n) > n^{\log_b a}$

$n^4 > n^1$

∴ This satisfy the Master Method

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$

$n^4 = \Omega\left(n^{1+\epsilon}\right)$ where $\epsilon < 3$

as its under case 3

so check Regularity condn

$af\left(\frac{n}{b}\right) \leq c\,f(n)$

$2f\left(\frac{n}{2}\right) \leq cn^4$

$2\left(\frac{n}{2}\right)^4 \leq cn^4$

$\frac{n^4}{8} \leq cn^4$

$\frac{1}{8} \leq c$

∴ with this Regularity condn is satisfied

$\boxed{T(n) = \theta(n^4)}$

Q. Solve the Recurrence Reln by using change of variable

$T(n) = 2T(\sqrt{n}) + \log n$

Consider,

$n = 2^m$

$\boxed{m = \log_2 n}$

$T(2^m) + 2T(2^{m/2}) + m$

$\sqrt{n} = n^{1/2}$

$= 2^{m \cdot 1/2}$

Consider $T(2^m) = S(m)$

$S(m) = 2S\left(\frac{m}{2}\right) + m \Rightarrow T(n) = aT\left(\frac{n}{b}\right) + f(n)$

~~$S(m) = 2$~~

$a = 2,\ b = 2,\ f(m) = m$

$m^{\log_2 2} = m^1 \quad f(m) = m$

Here $f(m)$ and $m^{\log_b a}$ is equal

∴ by case II

$S(m) = \theta(m \log m)$

$T(2^m) = \theta(m \log m)$

$\boxed{T(n) = \theta(\log_2 n \log \log_2 n)}$

Scanned with OKEN Scanner

Ex: Change of variable
$T(n) = 3T(\sqrt{n}) + \log n$.

Q. Solve the Recurrence Relation by using Characteristic eqn.

case 1. Roots are Real and distinct i.e $x_1 \neq x_2 \neq x_3 \neq \dots$
$t(n) = c_1(x_1)^n + c_2(x_2)^n + c_3(x_3)^n + \dots$

case 2. Roots are Real and same i.e $x_1 = x_2 = x_3 = \dots$
$t(n) = c_1(x_1)^n + c_2 n(x_2)^n + c_3 n^2(x_3)^n + \dots$

case 3. Roots are real distinct and same i.e $x_1 \neq x_2$ ~~$\neq \dots$~~ $x_3 = x_4$
$t(n) = c_1(x_1)^n + c_2(x_2)^n + c_3(x_3)^n + c_4 n(x_4)^n + c_5 n^4 \dots$

$t(n) = t(n-1) + t(n-2)$

2 types question
1. Homogeneous eqn.
2. Non " eqn.

P.T.O

1. Homogeneous recurrence eqn.

Q1 Solve the following homogeneous recurrence relation by using characteristics eqn.

$$t(n) = \begin{cases} n & ; \text{if } n = 0, n = 1 \\ t(n-1) + t(n-2) & ; \text{otherwise} \end{cases}$$

Soln:

$t(n) = t(n-1) + t(n-2)$

$t(n) - t(n-1) - t(n-2) = 0 \Longrightarrow$ Homogeneous eqn.

Select largest value after $t(n-$ and ~~take~~ start that value as the power.

~~x^0~~

$x^2 - x^1 - x^0 = 0$

$x^2 - x - 1 = 0$ ——— (1)

$x_1 = 1.6180$

$x_2 = -0.6180$

As roots are real and distinct

∴ By case 1

$t(n) = c_1(1.6180)^n + c_2(-0.6180)^n$ ——— (2)

Put initial condn in eq (2)

i.e $n = 0$

$t(0) = c_1(1.6180)^0 + c_2(-0.6180)^0$

$t(n) = n$

∴ $t(0) = 0$

$t(1) = 1$

$0 = c_1 + c_2$ ——— (3)

Put $n = 1$

Put n = 1 in eqn (2)

$$t(1) = C_1 (1.618)^1 + C_2 (-0.618)^1$$
$$1 = 1.618 C_1 - 0.618 C_2 \quad \text{---- (4)}$$

Now solve eqn (3) & (4), find out C_1 & C_2

$$C_1 = 0.447$$
$$C_2 = -0.447$$

Put the value of c_1 and c_2 in eqn (2)

Final Ans $\boxed{t(n) = (0.447)(1.6180)^n - 0.447(-0.618)^n}$

Q2. $t(n) = 0$; if $n = 0$, $t(n) = 5$, if $n = 1$
$t(n) = 3t(n-1) + 4t(n-2)$; otherwise

Soln : $t(n) = 3t(n-1) + 4t(n-2)$

$$t(n) - 3t(n-1) - 4t(n-2) = 0$$

$$x^2 - 3x^1 - 4x^0 = 0$$
$$x^2 - 3x - 4 = 0 \quad \text{---- (1)}$$

Roots are
$$u_1 = 4$$
$$u_2 = -1$$

As the roots are real & distinct
So by case 1
$$C_1 = 1 \text{ and } C_2 = -1$$

⇒ Same as Q1

Q3. $t(n) = n$; if $n = 0, 1, 2$
$5t(n-1) - 8t(n-2) + 4t(n-3)$

$$t(n) = 5t(n-1) - 8t(n-2) + 4t(n-3)$$
$$t(n) - 5t(n-1) + 8t(n-2) - 4t(n-3) = 0$$

$$x^3 - 5x^2 + 8x - 4 = 0$$
$$u_1 = 1, \; u_2 = 2, \; u_3 = 2$$

$$t(n) = C_1 (u_1)^n + C_2 n (u_2)^n + C_3 n^2 (u_3)^n$$
⇒ $u_1 \neq u_2$; $u_2 = u_3$; $u_1 = u_2$ $u_3 \neq$
$$t(n) = C_1 (1)^n + C_2 n (2)^n + C_3 n^2 (2)^n$$

Divide & conquer

* Greedy Method.

- Knapsack Problem

Fractional K.P (Greedy) | 0/1 K.P (Dynamic)

- wt, profit,

•

Q. Find the optimal soln of knapsack problem where $n = 7$; $m = 15$

	Item	Profit	weight	P_i/w_i
✓	1	10	2	5
	2	5	3	1.33
✓	3	15	5	3
✓	4	7	7	1
	5	6	1	6
✓	6	18	4	4.5
	7	3	1	3

* 3 methods
1. Max. Profit
2. Min. weight
3. Profit/weight

w p
7 → 7
4 → ?
Ans = 4

Fractional K.P

1. Max Profit

Item	Profit	weight	Remaining wt. (15)
6	18	4	15 - 4 = 11
3	15	5	11 - 5 = 6
1	10	2	6 - 2 = 4
4	7	4	4 - 4 = 0

$\Sigma = 47$

2 Method

Item	Profit	weight	Rem. (15)
5	6	1	15 - 1 = 14
7	3	1	14 - 1 = 13
1	10	2	13 - 2 = 11
2	5	3	11 - 3 = 8
6	18	4	4
3	12	4	4 - 4 = 0

$\Sigma = 54$

3rd Method P_i/w_i Maximum

Item	Profit	weight	P_i/w_i (15)
5	6	1	15 - 1 = 14
1	10	2	14 - 2 = 12
6	18	4	12 - 4 = 8
3	15	5	8 - 5 = 3
7	3	1	3 - 1 = 2
2	8.8	2	2 - 2 = 0

$\Sigma = 55.3$

Scanned with OKEN Scanner

$t(n) = 2t(n-1) + 2^n + 5$

$t(n) = 2t(n-1) = 2^n + 5$ $\begin{cases} 2^n n^0 \\ -5 \end{cases}$

Page No.: Date: YOUVA

Q. $t(n) = \begin{cases} 1 & \text{if } n = 1 \\ t(n-1) + n & \text{Otherwise} \end{cases}$

Soln :-

$t(n) = t(n-1) + n$ ——— Basic eqn.

$t(n) - t(n-1) = n$

$x^1 - 1 = 0$ # Solve LHS & RHS differently

$x = 1$

$u_1 = 1$

Solve R.H.S $= n \longrightarrow 1^n n^1$

$b = 1$

$d = 1 + 1 = 2$

$u_2 = 1$; $u_3 = 1$

Here, u_1, u_2, u_3 are same

∴ By case III

$t(n) = c_1 (u_1)^n + c_2 n (u_2)^n + c_3 n^2 (u_3)^n$

$= c_1 1^n + c_2 n 1^n + c_3 n^2 1^n$

$t(n) = c_1 + nc_2 + n^2 c_3$ ——— (1)

Initial condition

$n = 1$

$t(1) = c_1 + c_2 + c_3$

$c_1 + c_2 + c_3 = 1$ ——— (2)

Put $n = 2$ in basic eqn.

$\Rightarrow$ $t(n) = t(n-1) + 2$

$t(2) = t(1) + 2$

2. Non-Homogeneous Eqn.

Recurrence Relation.

The general form is

$t(n) + a_0 t(n-1) + a_1 t(n-2) + \ldots = b^n P(n)$

where

b^n = exponential

$P(n)$ = Polynomial form

To find Roots

RHS $= b^n P(n)$

$P(n)$ – Repeat the root value till (degree + 1) times of degree of n

where, b is Root value.

1. RHS : $2^n n^2 \Rightarrow b = 2$; $d+1 = 3$; $u_1 = 2, u_2 = 2, u_3 = 2$
2. $3^n (n^2 + n + 1) \Rightarrow b = 3$; $2+1 = 3$; $u_1 = 3, u_2 = 3, u_3 = 3$
3. $3^n (n+5) \Rightarrow b = 3$; $1+1 = 2$; $u_1 = 3, u_2 = 3$
4. $2^n \Rightarrow b = 2$; $0+1 = 1$; $u_1 = 2$
5. $n^2 \Rightarrow 1^n n^2 \Rightarrow b = 1$; $2+1 = 3$; $u_1 = 1, u_2 = 1, u_3 = 1$
6. $2^n n^2 + n \Rightarrow$

$2^n n^2$ $\quad$ $n \longrightarrow 1^n n^1$

$u_1 = 2$ $\quad$ $u_4 = 1$

$u_2 = 2$ $\quad$ $u_5 = 1$

$u_3 = 2$

$t(n) = 2t(n-1) + 2^n + 5$

$t(n) = 2t(n-1) = 2^n + 5$ $\begin{cases} 2^n n^0 \\ -5 \end{cases}$

Q. $t(n) = \begin{cases} 1 & \text{if } n = 1 \\ t(n-1) + n & \text{Otherwise} \end{cases}$

Soln :-

$t(n) = t(n-1) + n$ ——— Basic eqn.

$t(n) - t(n-1) = n$

$x^1 - 1 = 0$

$x = 1$

$u_1 = 1$

Solve R.H.S $= n \longrightarrow 1^n n^1$

$b = 1$

$d = 1 + 1 = 2$

$u_2 = 1$; $u_3 = 1$

Here, u_1, u_2, u_3 are same

∴ By case III

$t(n) = c_1 (u_1)^n + c_2$

$= c_1 1^n + c_2 n 1^n$

$t(n) = c_1 + nc_2 + \ldots$

Initial condition

$n = 1$

$t(1) = c_1 + c_2 + c_3$

$c_1 + c_2 + c_3 = 1$ ——

Put $n = 2$ in basic eq

$\Rightarrow$ $t(n) = t(n-1) + 2$

$t(2) = t(1) + 2$

$= 1 + 2$

$t(2) = 3$

$t(2) = C_1(1)^2 + C_2(2)(1)^2 + C_3(2)^2(1)^2$

$3 = C_1 + 2C_2 + 4C_3$ ——— (3)

$\Rightarrow$ $t(3) = t(n-1) + 3$

$= t(2) + 3$

$= 3 + 3$

$t(3) = 6$

$6 = C_1 + 3C_2 + 9C_3$ ——— (4)

From eqn. (2), (3) & (4)

$C_1 = 0$

$C_2 = 0.5$

$C_3 = 0.5$