UNIT – I

Performance Analysis:

1. Space Complexity:

The space complexity of an algorithm is the amount of money it needs to run to compilation.

2. Time Complexity:

The time complexity of an algorithm is the amount of computer time it needs to run to compilation.

Space Complexity:

Space Complexity Example:

```
Algorithm abc(a,b,c)
{
return a+b++*c+(a+b-c)/(a+b) +4.0;
}
```

→ The Space needed by each of these algorithms is seen to be the sum of the following component.

1.A fixed part that is independent of the characteristics (eg:number,size)of the inputs and outputs.

The part typically includes the instruction space (ie. Space for the code), space for simple variable and fixed-size component variables (also called aggregate) space for constants, and so on.

1. A variable part that consists of the space needed by component variables whose size is dependent on the particular problem instance being solved, the space needed by referenced variables (to the extent that is depends on instance characteristics), and the recursion stack space.

   - The space requirement s(p) of any algorithm p may therefore be written as,
     S(P) = c+ Sp(Instance characteristics)

   Where ‘c’ is a constant.

   Example 2:

```
Algorithm sum(a,n)
{
    s=0.0;
    for I=1 to n do
    s= s+a[I];
    return s;
```

}

- The problem instances for this algorithm are characterized by n,the number of elements to be summed. The space needed d by ‘n’ is one word, since it is of type integer.
- The space needed by ‘a’a is the space needed by variables of tyepe array of floating point numbers.
- This is atleast ‘n’ words, since ‘a’ must be large enough to hold the ‘n’ elements to be summed.
- So,we obtain Ssum(n)>=(n+s)

[n for a[],one each for n,I a& s]

Time Complexity:

The time T(p) taken by a program P is the sum of the compile time and the run time(execution time)

→The compile time does not depend on the instance characteristics. Also we may assume that a compiled program will be run several times without recompilation .This rum time is denoted by tp(instance characteristics).

→ The number of steps any problem statemn t is assigned depends on the kind of statement.

For example, comments → 0 steps.
Assignment statements → 1 steps.
[Which does not involve any calls to other algorithms]

Interactive statement such as for, while & repeat-until→ Control part of the statement.

1. We introduce a variable, count into the program statement to increment count with initial value 0.Statement to increment count by the appropriate amount are introduced into the program.

   This is done so that each time a statement in the original program is executes count is incremented by the step count of that statement.

Algorithm:

```
Algorithm sum(a,n)
{
        s= 0.0;
        count = count+1;
        for I=1 to n do
        {
         count =count+1;
        s=s+a[I];
```

```
count=count+1;
}
count=count+1;
count=count+1;
return s;
}
```

→ If the count is zero to start with, then it will be 2n+3 on termination. So each invocation of sum execute a total of 2n+3 steps.

2. The second method to determine the step count of an algorithm is to build a table in which we list the total number of steps contributes by each statement.

→First determine the number of steps per execution (s/e) of the statement and the total number of times (ie., frequency) each statement is executed.
→By combining these two quantities, the total contribution of all statements, the step count for the entire algorithm is obtained.

Statement	*S/e*	*Frequency*	*Total*
1. Algorithm Sum(a,n)	0	-	0
2.{	0	-	0
3. S=0.0;	1	1	1
4. for I=1 to n do	1	n+1	n+1
5. s=s+a[I];	1	n	n
6. return s;	1	1	1
7. }	0	-	0
Total			2n+3

AVERAGE –CASE ANALYSIS

- Most of the time, average-case analysis are performed under the more or less realistic assumption that all instances of any given size are equally likely.
- For sorting problems, it is simple to assume also that all the elements to be sorted are distinct.
- Suppose we have ‘n’ distinct elements to sort by insertion and all n! permutation of these elements are equally likely.
- To determine the time taken on a average by the algorithm ,we could add the times required to sort each of the possible permutations ,and then divide by n! the answer thus obtained.
- An alternative approach, easier in this case is to analyze directly the time required by the algorithm, reasoning probabilistically as we proceed.

- For any I,2 ≤ I ≤ n, consider the sub array, T[1….i].
- The partial rank of T[I] is defined as the position it would occupy if the sub array were sorted.
- For Example, the partial rank of T[4] in [3,6,2,5,1,7,4] in 3 because T[1….4] once sorted is [2,3,5,6].
- Clearly the partial rank of T[I] does not depend on the order of the element in
- Sub array T[1…I-1].

Analysis

Best case:
This analysis constrains on the input, other than size. Resulting in the fasters possible run time

Worst case:
This analysis constrains on the input, other than size. Resulting in the fasters possible run time

Average case:
This type of analysis results in average running time over every type of input.

Complexity:
Complexity refers to the rate at which the storage time grows as a function of the problem size

Asymptotic analysis:
Expressing the complexity in term of its relationship to know function. This type analysis is called asymptotic analysis.

Asymptotic notation:

Big 'oh': the function f(n)=O(g(n)) iff there exist positive constants c and no such that f(n)≤c*g(n) for all n, n ≥ no.

Omega: the function f(n)=Ω(g(n)) iff there exist positive constants c and no such that f(n) ≥ c*g(n) for all n, n ≥ no.

Theta: the function f(n)=ɵ(g(n)) iff there exist positive constants c1,c2 and no such that c1 g(n) ≤ f(n) ≤ c2 g(n) for all n, n ≥ no.

Recursion:

Recursion may have the following definitions:
-The nested repetition of identical algorithm is recursion.
-It is a technique of defining an object/process by itself.
-Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.

When to use recursion:

Recursion can be used for repetitive computations in which each action is stated in terms of previous result. There are two conditions that must be satisfied by any recursive procedure.

1. Each time a function calls itself it should get nearer to the solution.
2. There must be a decision criterion for stopping the process.

In making the decision about whether to write an algorithm in recursive or non-recursive form, it is always advisable to consider a tree structure for the problem. If the structure is simple then use non-recursive form. If the tree appears quite bushy, with little duplication of tasks, then recursion is suitable.

The recursion algorithm for finding the factorial of a number is given below,

Algorithm : factorial-recursion
Input : n, the number whose factorial is to be found.
Output : f, the factorial of n
Method : if(n=0)
f=1
else
f=factorial(n-1) * n
if end
algorithm ends.

The general procedure for any recursive algorithm is as follows,

1. Save the parameters, local variables and return addresses.
2. If the termination criterion is reached perform final computation and goto step 3 otherwise perform final computations and goto step 1

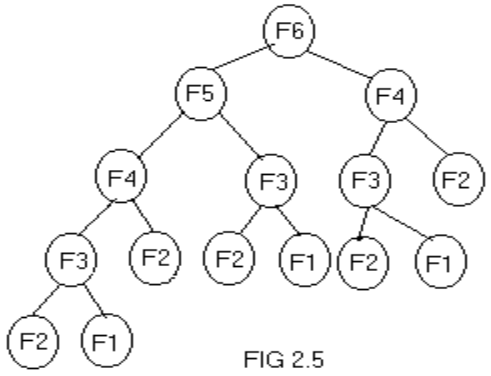

FIG 2.5

3. Restore the most recently saved parameters, local variable and return address and goto the latest return address.

Iteration v/s Recursion:

Demerits of recursive algorithms:

1. Many programming languages do not support recursion; hence, recursive mathematical function is implemented using iterative methods.

2. Even though mathematical functions can be easily implemented using recursion it is always at the cost of execution time and memory space. For example, the recursion tree for generating 6 numbers in a Fibonacci series generation is given in fig 2.5. A Fibonacci series is of the form 0,1,1,2,3,5,8,13,…etc, where the third number is the sum of preceding two numbers and so on. It can be noticed from the fig 2.5 that, f(n-2) is computed twice, f(n-3) is computed thrice, f(n-4) is computed 5 times.

3. A recursive procedure can be called from within or outside itself and to ensure its proper functioning it has to save in some order the return addresses so that, a return to the proper location will result when the return to a calling statement is made.

4. The recursive programs needs considerably more storage and will take more time.

Demerits of iterative methods :

- Mathematical functions such as factorial and Fibonacci series generation can be easily implemented using recursion than iteration.

- In iterative techniques looping of statement is very much necessary.

Recursion is a top down approach to problem solving. It divides the problem into pieces or selects out one key step, postponing the rest.
Iteration is more of a bottom up approach. It begins with what is known and from this constructs the solution step by step. The iterative function obviously uses time that is O(n) where as recursive function has an exponential time complexity.
It is always true that recursion can be replaced by iteration and stacks. It is also true that stack can be replaced by a recursive program with no stack.

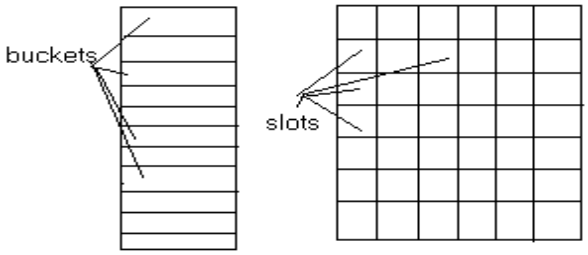

Fig 2.6

SOLVING RECURRENCES :-(Happen again (or) repeatedly)

- The indispensable last step when analyzing an algorithm is often to solve a recurrence equation.
- With a little experience and intention, most recurrence can be solved by intelligent guesswork.
- However, there exists a powerful technique that can be used to solve certain classes of recurrence almost automatically.
- This is a main topic of this section the technique of the characteristic equation.

1. Intelligent guess work:

This approach generally proceeds in 4 stages.

1. Calculate the first few values of the recurrence
2. Look for regularity.
3. Guess a suitable general form.
4. And finally prove by mathematical induction(perhaps constructive induction).

Then this form is correct.

Consider the following recurrence,

$$T(n) = \begin{cases} 0 & \text{if } n=0 \\ 3T(n \div 2)+n & \text{otherwise} \end{cases}$$

- First step is to replace n ÷ 2 by n/2
- It is tempting to restrict 'n' to being ever since in that case n÷2 = n/2, but recursively dividing an even no. by 2, may produce an odd no. larger than 1.
- Therefore, it is a better idea to restrict 'n' to being an exact power of 2.
- First, we tabulate the value of the recurrence on the first few powers of 2.

n	1	2	4	8	16	32
T(n)	1	5	19	65	211	665

* For instance, T(16) = 3 * T(8) +16
= 3 * 65 +16
= 211.

* Instead of writing T(2) = 5, it is more useful to write T(2) = 3 * 1 +2.

Then,

T(A) = 3 * T(2) +4
= 3 * (3 * 1 +2) +4

$= (3^2 * 1) + (3 * 2) + 4$

* We continue in this way, writing 'n' as an explicit power of 2.

n	T(n)
1	1
2	$3 * 1 + 2$
2^2	$3^2 * 1 + 3 * 2 + 2^2$
2^3	$3^3 * 1 + 3^2 * 2 + 3 * 2^2 + 2^3$
2^4	$3^4 * 1 + 3^3 * 2 + 3^2 * 2^2 + 3 * 2^3 + 2^4$
2^5	$3^5 * 1 + 3^4 * 2 + 3^3 * 2^2 + 3^2 * 2^3 + 3 * 2^4 + 2^5$

- The pattern is now obvious.

$T(2^k) = 3^k 2^0 + 3^{k-1} 2^1 + 3^{k-2} 2^2 + \ldots + 3^1 2^{k-1} + 3^0 2^k.$

$= \sum 3^{k-i} 2^i$

$= 3^k \sum (2/3)^i$

$= 3^k * [(1 - (2/3)^{k+1}) / (1 - (2/3)]$
$= 3^{k+1} - 2^{k+1}$

Proposition: (Geometric Series)

Let S_n be the sum of the first n terms of the geometric series a, ar, ar^2….Then $S_n = a(1-r^n)/(1-r)$, except in the special case when r = 1; when $S_n = a_n$.

$= 3^k * [(1 - (2/3)^{k+1}) / (1 - (2/3))]$

$= 3^k * [((3^{k+1} - 2^{k+1}) / 3^{k+1}) / ((3 - 2) / 3)]$

$$= 3^k * \frac{3^{k+1} - 2^{k+1}}{3^{k+1}} * \frac{3}{1}$$

$$= 3^k * \frac{3^{k+1} - 2^{k+1}}{3^{k+1-1}}$$

$= 3^{k+1} - 2^{k+1}$

* It is easy to check this formula against our earlier tabulation.

EG : 2

$$t_n = \begin{cases} 0 & n=0 \\ 5 & n=1 \\ 3t_{n-1} + 4t_{n-2}, \text{ otherwise} \end{cases}$$

$t_n = 3t_{n-1} - 4t_{n-2} = 0$ → General function

Characteristics Polynomial, $x^2 - 3x - 4 = 0$

$(x - 4)(x + 1) = 0$

Roots $r_1 = 4$, $r_2 = -1$

General Solution, $f_n = C_1 r_1^n + C_2 r_2^n$ → (A)

n=0 ➔ $C_1 + C_2 = 0$ → (1)

n=1 ➔ $C_1 r_1 + C_2 r_2 = 5$ → (2)

Eqn 1 ➔ $C_1 = -C_2$

sub C_1 value in Eqn (2)

$-C_2 r_1 + C_2 r_2 = 5$

$C_2(r_2 - r_1) = 5$

$$C_2 = \frac{5}{r_2 - r_1}$$

$$= \frac{5}{-1 + 4}$$

$= 5 / (-5) = -1$

$C_2 = -1$, $C_1 = 1$

Sub C_1, C_2, r_1 & r_2 value in equation → (A)

$f_n = 1. 4^n + (-1) . (-1)^n$

$f_n = 4^n + 1^n$

2. Homogenous Recurrences :

* We begin our study of the technique of the characteristic equation with the resolution of homogenous linear recurrences with constant co-efficient, i.e the recurrences of the form,

$a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} = 0$

where the t_i are the values we are looking for.

* The values of t_i on 'K' values of i (Usually $0 \le i \le k-1$ (or) $0 \le i \le k$) are needed to determine the sequence.

* The initial condition will be considered later.

* The equation typically has infinitely many solution.

* The recurrence is,
  - → linear because it does not contain terms of the form t_{n-i}, t_{n-j}, t^2_{n-i}, and soon.
  - → homogeneous because the linear combination of the t_{n-i} is equal to zero.
  - → With constant co-efficient because the a_i are constants

* Consider for instance our non familiar recurrence for the Fibonacci sequence,
  $f_n = f_{n-1} + f_{n-2}$

* This recurrence easily fits the mould of equation after obvious rewriting.
  $f_n - f_{n-1} - f_{n-2} = 0$

* Therefore, the fibonacci sequence corresponds to a homogenous linear recurrence with constant co-efficient with $k=2, a_0=1 \& a_1=a_2 = -1$.

* In other words, if f_n & g_n satisfy equation.

$$\text{So } \sum_{i=0}^{k} a_i f_{n-i} = 0 \text{ \& similarly for } g_n \text{ \& } f_n$$

We set $t_n = C f_n + d g_n$ for arbitrary constants C & d, then t_n is also a solution to equation.

* This is true because,

$$\begin{aligned} & a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} \\ &= a_0(C f_n + d g_n) + a_1(C f_{n-1} + d g_{n-1}) + \ldots + a_k(C f_{n-k} + d g_{n-k}) \\ &= C(a_0 f_n + a_1 f_{n-1} + \ldots + a_k f_{n-k}) + \\ &\quad d(a_0 g_n + a_1 g_{n-1} + \ldots + a_k g_{n-k}) \\ &= C * 0 + d * 0 \\ &= 0. \end{aligned}$$

1) (Fibonacci) Consider the recurrence.

$$f_n = \begin{cases} n & \text{if } n=0 \text{ or } n=1 \\ \end{cases}$$

$f_{n-1} + f_{n-2}$ otherwise

We rewrite the recurrence as,

$f_n - f_{n-1} - f_{n-2} = 0$.

The characteristic polynomial is,

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

The roots are,

$$x = \frac{-(-1) \pm \sqrt{((-1)2 + 4)}}{2}$$

$$= \frac{1 \pm \sqrt{(1 + 4)}}{2}$$

$$= \frac{1 \pm \sqrt{5}}{2}$$

$$r_1 = \frac{1 + \sqrt{5}}{2} \quad \text{and} \quad r_2 = \frac{1 - \sqrt{5}}{2}$$

The general solution is,

$f_n = C_1 r_1^n + C_2 r_2^n$

when n=0, $f_0 = C_1 + C_2 = 0$

when n=1, $f_1 = C_1 r_1 + C_2 r_2 = 1$

$C_1 + C_2 = 0 \quad \rightarrow (1)$

$C_1 r_1 + C_2 r_2 = 1 \quad \rightarrow (2)$

From equation (1)

$C_1 = -C_2$

Substitute C_1 in equation(2)

$-C_2 r_1 + C_2 r_2 = 1$

$C_2[r_2 - r_1] = 1$

Substitute r_1 and r_2 values

$$C_2 \left[\frac{1 - \sqrt{5}}{2} - \frac{1 - \sqrt{5}}{2}\right] = 1$$

$$C_2 \frac{2 - \sqrt{5} - 1 - \sqrt{5}}{2}^2 = 1$$

$$\frac{-C_2 * 2\sqrt{5}}{2} = 1$$

$$-\sqrt{5}C_2 = 1$$

$C_1 = 1/\sqrt{5}$ $C_2 = -1/\sqrt{5}$

Thus,

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2}\right)^n + \frac{-1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2}\right)^n$$

$$= \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2}\right)^n - \left(\frac{1 - \sqrt{5}}{2}\right)^n$$

3. Inhomogeneous recurrence :

* The solution of a linear recurrence with constant co-efficient becomes more difficult when the recurrence is not homogeneous, that is when the linear combination is not equal to zero.
* Consider the following recurrence
  $a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} = b^n p(n)$
* The left hand side is the same as before,(homogeneous) but on the right-hand side we have $b^n p(n)$, where,
  → b is a constant
  → p(n) is a polynomial in 'n' of degree 'd'.

Example(1) :

Consider the recurrence,

$t_n - 2t_{n-1} = 3^n$ →(A)

In this case, b=3, p(n) = 1, degree = 0.

The characteristic polynomial is,

$(x - 2)(x - 3) = 0$

The roots are, $r_1 = 2$, $r_2 = 3$

The general solution,

$t_n = C_1 r_1^n + C_2 r_2^n$

$t_n = C_1 2^n + C_2 3^n \rightarrow (1)$

when n=0, $C_1 + C_2 = t0 \rightarrow (2)$

when n=1, $2C_1 + 3C_2 = t1 \rightarrow (3)$

sub n=1 in eqn (A)

$t_1 - 2t_0 = 3$

$t_1 = 3 + 2t_0$

substitute t_1 in eqn(3),

(2) * 2 ➔ $2C_1 + 2C_2 = 2t_0$

$2C_1 + 3C_2 = (3 + 2t_0)$

$-C_2 = -3$

$C_2 = 3$

Sub $C_2 = 3$ in eqn (2)

$C_1 + C_2 = t_0$

$C_1 + 3 = t_0$

$C_1 = t_0 - 3$

Therefore $t_n = (t_0-3)2^n + 3.\ 3^n$

$= \text{Max}[O[(t_0 - 3)\ 2^n], O[3.3^n]]$

$= \text{Max}[O(2^n), O(3^n)]$ constants

$= O[3^n]$

Example :(2)

$t_n - 2t_{n-1} = (n + 5)3^n,\ n \geq 1 \rightarrow (A)$

This is Inhomogeneous

In this case, b=3, p(n) = n+5, degree = 1

So, the characteristic polynomial is,

$(x-2)(x-3)^2 = 0$

The roots are,

$r_1 = 2,\ r_2 = 3,\ r_3 = 3$

The general equation,

$t_n = C_1 r_1^n + C_2 r_2^n + C_3 n r_3^n \rightarrow (1)$

when n=0, $t_0 = C_1 + C_2 \rightarrow (2)$

when n=1, $t_1 = 2C_1 + 3C_2 + 3C_3$ →(3)

substituting n=1 in eqn(A),

$t_1 - 2t_0 = 6 . 3$

$t_1 - 2t_0 = 18$

$t_1 = 18 + 2t_0$

substituting t_1 value in eqn(3)

$2C_1 + 3C_2 + 3C_3 = 18 + 2t_0$ →(4)

$C_1 + C_2 + = t_0$ →(2)

Sub. n=2 in eqn(1)

$4C_1 + 9C_2 + 18C_3 = t_2$ → (5)

sub n=2 in eqn (A)

$t_2 - 2t_1 = 7. 9$

$t_2 = 63 + 2t_1$

$= 63 + 2[18 + 2t_0]$

$t_2 = 63 + 36 + 4t_0$

$t_2 = 99 + 4t_0$

sub. t_2 value in eqn(3),

$4C_1 + 9C_2 + 18C_3 = 99 + 4t_0$ →(5)

solve eqn (2),(4) & (5)

n=0, $C_1 + C_2 = t_0$ →(2)

n=1, $2C_1 + 3C_2 + 3C_3 = 18 + 2t_0$ →(4)

n=2, $4C_1 + 9C_2 + 18C_3 = 99 + 4t_0$ →(5)

(4) * 6 ➔ $12C_1 + 18C_2 + 18C_3 = 108 + 2t_0$ →(4)

(5) ➔ $4C_1 + 9C_2 + 18C_3 = 99 + 4t0$ →(5)

$8C_1 + 9C_2 = 9 + 8t_0$ →(6)

(2) * 8 ➔ $8C_1 + 8C_2 = 8t_0$ →(2)

(6) ➔ $8C_1 + 9C_2 = 9 + 8t_0$ →(6)

$-C_2 = -9$

$C_2 = 9$

Sub, $C_2 = 9$ in eqn(2)

$C_1 + C_2 = t_0$

$C_1 + 9 = t_0$

$C_1 = t_0 - 9$

Sub C_1 & C_2 in eqn (4)

$2C_1 + 3C_2 + 3C_3 = 18 + 2t_0$

$2(t_0-9) + 3(9) + 3C_3 = 18 + 2t_0$

$2t_0 – 18 + 27 + 3C_3 = 18 + 2t_0$

$2t_0 + 9 + 3C_3 = 18 + 2t_0$

$3C_3 = 18 – 9 + 2t_0 – 2t_0$

$3C_3 = 9$

$C_3 = 9/3$

$C_3 = 3$

Sub. $C_1, C_2, C_3, r_1, r_2, r_3$ values in eqn (1)

$t_n = C_1 2^n + C_2 3^n + C_3.n.3^n$

$= (t_0 – 9)2^n + 9.3^n + 3.n.3^n$

$= Max[O[(t_0-9),2^n], O[9.3^n], O[3.n.3^n]]$

$= Max[O(2^n), O(3^n), O(n3^n)]$

$tn = O[n3^n]$

Example: (3)

Consider the recurrence,

$$t_n = \begin{cases} 1 & \text{if } n=0 \\ 4t_{n-1} – 2^n & \text{otherwise} \end{cases}$$

$t_n – 4t_{n-1} = -2_n \rightarrow (A)$

In this case , c=2, p(n) = -1, degree =0

$(x-4)(x-2) = 0$

The roots are, $r_1 = 4, r_2 = 2$

The general solution ,

$t_n = C_1 r_1^n + C_2 r_2^n$

$t_n = C_1 4^n + C_2 2^n \rightarrow (1)$

when n=0, in (1) ➔ $C_1 + C_2 = 1 \rightarrow (2)$

when n=1, in (1) ➔ $4C_1 + 2C_2 = t1 \rightarrow (3)$

sub n=1 in (A),

$t_n – 4t_{n-1} = -2^n$

$t_1 – 4t_0 = -2$

$t_1 = 4t_0 – 2$ [since $t_0 = 1$]

$t_1 = 2$

sub t1 value in eqn (3)

$4C_1 + 2C_2 = 4t_0 - 2 \rightarrow (3)$

(2) * 4 ➔ $4C_1 + 4C_2 = 4$

$-2C_2 = 4t_0 - 6$

$= 4(1) - 6$

$= -2$

$C_2 = 1$

$-2C_2 = 4t_0 - 6$

$2C_2 = 6 - 4t_0$

$C_2 = 3 - 2t_0$

$3 - 2(1) = 1$

$C_2 = 1$

Sub. C_2 value in eqn(2),

$C_1 + C_2 = 1$

$C_1 + (3-2t_0) = 1$

$C_1 + 3 - 2t_0 = 1$

$C_1 = 1 - 3 + 2t_0$

$C_1 = 2t_0 - 2$

$= 2(1) - 2 = 0$

$C_1 = 0$

Sub C_1 & C_2 value in eqn (1)

$t_n = C_1 4^n + C_2 2^n$

$= \text{Max}[O(2t_0 - 2).4^n, O(3 - 2t_0).2^n]$

$= \text{Max}[O(2^n)]$

$t_n = O(2^n)$

Example : (4)

$$t_n = \begin{cases} 0 & \text{if } n=0 \\ 2t_{n-1} + n + 2^n & \text{otherwise} \end{cases}$$

$t_n - 2t_{n-1} = n + 2^n \rightarrow (A)$

There are two polynomials.

For n; b=1, p(n), degree = 1

For 2n; b=2, p(n) = 1, degree =0

The characteristic polynomial is,

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

The roots are, $r_1 = 2$, $r_2 = 2$, $r_3 = 1$, $r_4 = 1$.
So, the general solution,

$t_n = C_1 r_1^n + C_2 n r_2^n + C_3 r_3^n + C_4\, n\, r_4^n$

sub r_1, r_2, r_3 in the above eqn

$t_n = 2^n C_1 + 2^n C_2\, n + C_3 \cdot 1^n + C_4 .n.1^n \quad \rightarrow (1)$

sub. n=0 ➔ $C_1 + C_3 = 0 \quad \rightarrow (2)$
sub. n=1 ➔ $2C_1 + 2C_2 + C_3 + C_4 = t_1 \quad \rightarrow (3)$

sub. n=1 in eqn (A)

$t_n - 2t_{n-1} = n + 2^n$
$t_1 - 2t_0 = 1 + 2$
$t_1 - 2t_0 = 3$
$t_1 = 3$ [since $t_0 = 0$]

sub. n=2 in eqn (1)

$2^2 C_1 + 2.\, 2^2.C_2 + C_3 + 2.C_4 = t_2$
$4C_1 + 8C_2 + C_3 + 2C_4 = t_2$

sub n=2 in eqn (A)

$t_2 - 2t_1 = 2 + 2^2$

$t_2 - 2t_1 = 2 + 4$
$t_2 - 2t_1 = 6$
$t_2 = 6 + 2t_1$
$t_2 = 6 + 2.3$
$t_2 = 6 + 6$
$t_2 = 12$

➔ $4C_1 + 8C_2 + C_3 + 2C_4 = 12 \quad \rightarrow (4)$

sub n=3 in eqn (!)

$2^3 C_1 + 3.2^3.C_2 + C_3 + 3C_4 = t_3$
$3C_1 + 24C_2 + C_3 + 3C_4 = t_3$

sub n=3 in eqn (A)

$t_3 - 2t_2 = 3 + 2^3$
$t_3 - 2t_2 = 3 + 8$
$t_3 - 2(12) = 11$
$t_3 - 24 = 11$
$t_3 = 11 + 24$
$t_3 = 35$

➔ $8C_1 + 24C_2 + C_3 + 3C_4 = 35 \quad \rightarrow (5)$

n=0, solve; $C_1 + C_3 = 0$ →(2)
n=1,(2), (3), (4)&(5) $2C_1 + 2C_2 + C_3 + C_4 = 3$ →(3)
n=2, $4C_1 + 8C_2 + C_3 + 2C_4 = 12$ →(4)
n=3, $8C_1 + 24C_2 + C_3 + 3C_4 = 35$ →(5)

$-4C_1 - 16C_2 - C_4 = -23$ →(6)

solve: (2) & (3)
(2) ➔ $C_1 + C_3 = 0$
(3) ➔ $2C_1 + C_3 + 2C_2 + C_4 = 3$

$-C_1 - 2C_2 - C_4 = -3$ →(7)

solve(6) & (7)
(6) ➔ $-4C_1 - 16C_2 - C_4 = -23$
(7) ➔ $-C_1 - 2C_2 - C_4 = -3$

$-3C_1 - 14C_2 = 20$ →(8)

4. Change of variables:

* It is sometimes possible to solve more complicated recurrences by making a change of variable.
* In the following example, we write T(n) for the term of a general recurrences, and t_i for the term of a new recurrence obtained from the first by a change of variable.

Example: (1)
Consider the recurrence,

$$T(n) = \begin{cases} 1 & , \text{ if } n=1 \\ 3T(n/2) + n & , \text{ if 'n' is a power of 2, } n>1 \end{cases}$$

➔ Reconsider the recurrence we solved by intelligent guesswork in the previous section, but only for the case when 'n' is a power of 2

$$T(n) = \begin{cases} 1 \\ 3T(n/2) + n \end{cases}$$

* We replace 'n' by 2^i.
* This is achieved by introducing new recurrence t_i, define by $t_i = T(2^i)$
* This transformation is useful because n/2 becomes $(2^i)/2 = 2^{i-1}$
* In other words, our original recurrence in which T(n) is defined as a function of T(n/2) given way to one in which t_i is defined as a function of t_{i-1}, precisely the type of recurrence we have learned to solve.

$t_i = T(2^i) = 3T(2^{i-1}) + 2^i$
$t_i = 3t_{i-1} + 2^i$
$t_i - 3t_{i-1} = 2^i \rightarrow (A)$

In this case,
b = 2, p(n) = 1, degree = 0

So, the characteristic equation,
$(x - 3)(x - 2) = 0$

The roots are, r1 = 3, r2 = 2.

The general equation,
$t_n = C_1 r_1^i + C_2 r_2^i$
sub. r_1 & r_2: $t_n = 3^n C_1 + C_2 2^n$
$t_n = C_1 3^i + C_2 2^i$

We use the fact that, $T(2^i) = t_i$ & thus T(n) = tlogn when $n = 2^i$ to obtain,
$T(n) = C_1 . 3^{\log_2 n} + C_2 . 2^{\log_2 n}$
$T(n) = C_1 . n^{\log_2 3} + C_2 . n$ [i = logn]
When 'n' is a power of 2, which is sufficient to conclude that,

$T(n) = O(n^{\log 3})$ 'n' is a power of 2

Example: (2)

Consider the recurrence,
$T(n) = 4T(n/2) + n^2 \rightarrow (A)$

Where 'n' is a power of 2, $n \geq 2$.
$t_i = T(2^i) = 4T(2^{i-1}) + (2^i)^2$
$t_i = 4t_{i-1} + 4^i$
$\rightarrow t_i - 4t_{i-1} = 4^i$

In this eqn,
b = 4, P(n) = 1, degree = 0

The characteristic polynomial,
$(x - 4)(x - 4) = 0$

The roots are, $r_1 = 4$, $r_2 = 4$.

So, the general equation,
$t_i = C_1 4^i + C_2 4^i . i$ [since i = logn]
$= C_1 4^{\log n} + C_2 . 4^{\log n} . \log n$ [since $2^i = n$]
$= C_1 . n^{\log 4} + C_2 . n^{\log 4} . n^{\log 1}$

T(n) = $O(n^{\log 4})$ 'n' is the power of 2.

EXAMPLE : 3

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

When 'n' is a power of 2, $n \geq 2$

$t_i = T(2^i) = 2T(2^i/2) + 2^i .i$ [since $2^i = n$; $i = \log n$]

$t_i - 2t_{i-1} = i. 2^i$

In this case,

$b = 2$, $P(n) = i$, degree = 1

$(x - 2)(x - 2)^2 = 0$

The roots are, $r_1 = 2$, $r_2 = 2$, $r_3 = 2$

The general solution is,

$t_n = C_1 2^i + C_2. 2^i . i + C_3. i^2. 2^i$

$= nC_1 + nC_2 + nC_3(\log n^2 2n)$

$t_n = O(n.\log^2 2n)$

Example: 4

$$T(n) = \begin{cases} 2 & ,n=1 \\ 5T(n/4) + Cn^2 & ,n>1 \end{cases}$$

$t_i = T(4^i) = 5T(4^i/4) + C(4^i)^2$

$= 5T\ 4^{i-1} + C. 16^i$

$= 5t_{i-1} + C.16^i$

$t_i - 5t_{i-1} = C. 16^i$

In this case,

$b = 16$, $P(n) = 1$, degree = 0

The characteristic eqn,

$(x - 5)(x - 16) = 0$

The roots are, $r_1 = 5$, $r_2 = 16$

The general solution,

$t_i = C_1.5^i + C_2.16^i$

$= C_1.5^i + C_2.(4^2)^i$

$t_n = O(n^2)$

EXAMPLE: 5

$$T(n) = \begin{cases} 2 & , n = 1 \\ T(n/2) + Cn & , n > 1 \end{cases}$$

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

$= T(2^i/2) + C.\ 2^i$

$= T(2^{\,i-1}) + C.\ 2^i$

$t_i = t_{\,i-1} + C.\ 2^i$

$t_i - t_{\,i-1} = C.\ 2^i$

In this case, b=2, P(n) =1, degree =0

So, the characteristic polynomial,

$(x -1)(x - 2) = 0$

The roots are, $r_1 = 1$, $r_2 = 2$

$t_i = C_1.\ 1^i + c_2.\ 2^i$

$= C_1.\ 1^{\log_2 n} + C_2.n$

$= C_1\ .\ n^{\log_2 1} + C_2.n$

$t_n = O(n)$

EXAMPLE: 6

$$T(n) = \begin{cases} 1 & , n =1 \\ 3T(n/2) + n; \text{ n is a power of 2} \end{cases}$$

$t_i = T(2^i) = 3T(2^i/2) + 2^i$

$= 3T(2^{\,i-1}) + 2^i$

$t_i = 3t_{\,i-1} + 2^i$

So, b = 2, P(n) =1, degree = 0

$(x - 3)(x - 2) = 0$

The roots are, $r_1 = 3$, $r_2 = 2$

$t_i = C_1.\ 3^i + C_2.\ 2^i$

$= C_1.\ 3^{\log_2 n} + C_2.\ 2^{\log_2 n}$

$= C_1.\ n^{\log_2 3} + C_2.\ n^{\log_2 2} = 1$

$= C_1.\ n^{\log_2 3} + C_2.n$

$t_n = O(n^{\log_2 3})$

EXAMPLE: 7

$T(n) = 2T(n/2) + n.\log n$

$t_i = T(2^i) = 2T(2^i/2) + 2^i . i$
$= 2T(2^{i-1}) + i.2^i$
$= 2t_{i-1} + i.2^i$
$t_i - 2t_{i-1} = i.2i$

➔ b=2, P(n) = I, degree = 1
The roots is $(x - 2)(x - 2)2 = 0$
$x = 2,2,2$

General solution,
$t_n = C_1.r_1^i + C_2.i.r_2^i + C_3.i2.r_3^i$
$= C_1.2^i + C_2.i.2^i + C_3.i^2.2^i$
$= C_1.n + C_2.n.^{\log_2 n} + C_3.i^2.n$
$= C_1.n + C_2.n.^{\log_2 n} + C_3(2^{\log_2 n}).n$

$t_n = O(n.2^{\log_2 n})$

5. Range Transformation:

* When we make a change of variable, we transform the domain of the recurrence.
* Instead it may be useful to transform the range to obtain a recurrence in a form that we know how to solve
* Both transformation can be sometimes be used together.

EXAMPLE: 1

Consider the following recurrence , which defines T(n). where 'n' is the power Of 2

$$T(n) = \begin{cases} 1/3 & \text{, if } n=1 \\ n\,T^2(n/2) & \text{, otherwise} \end{cases}$$

The first step is a change of variable,
Let t_i denote $T(2^i)$
$t_i = T(2^i) = 2^i T^2(2^{i-1})$
$= 2^i t^2_{i-1}$

* This recurrence is not clear, furthermore the co-efficient 2^i is not a constant.
* To transform the range, we create another recurrence by using ui to denote

lg t_i

$u_i = lgt_i = i + 2lg\ t_{i-1}$
$= i + 2u_{i-1}$

➔ $u_i - 2u_{i-1} = i$
$(x - 2)(x - 1)^2 = 0$

The roots are, x = 2, 1, 1.
G.S,
$u_i = C_1.2^i + C_2.1^i + C_3.i.1^i$

Sub. This solution into the recurrence,
For u_i yields,
$i = u_i - 2u_{i-1}$
$= C_1 2^i + C_2 + C_3.i - 2(C_1.\ 2^{i-1} + C_2 + C_3\ (i-1))$
$= (2C_3 - C_2) - C_3 i.$
$C_3 = -1$ & $C_2 = 2C_3 = -2$
$u_i = C_1 2^i - i - 2$

This gives us the G.S for t_i & T(n)
$t_i = 2^{ui} = 2^{C_1 2i - i-2}$
$T(n) = t_{lgn} = 2^{C_1 n - logn - 2}$
$= 2^{C_1 n} / 4n$

We use he initial condition T(1) = 1/3
To determine C1: $T(1) = 2^{C_1} / 4 = 1/3$
Implies that C1 = lg(4/3) = 2 – log 3

The final solution is

$T(n) = 2^{2n} / 4n\ 3^n$

1. Newton Raphson method: $x_2 = x_1 - f(x_1)/f^1(x_1)$