SYLLABUS

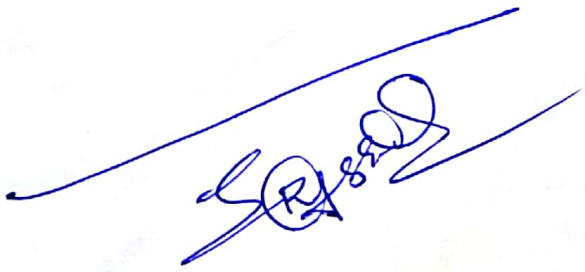
⇒ UNIT – I

⇒ CHAPTER 1 :- Introduction

1) Algorithm
2) pseudo code for expressing an algorithm.
3) performance Analysis
   (i) Space complexity
   (ii) Time complexity
4) Asymptotic Notation.
5) probability Analysis.
6) Amortized.

⇒ CHAPTER 2 :- Divide and conquer.

1) General Method.
2) Application.
   (i) Binary Search.
   (ii) Quick sort.
   (iii) Merge sort.
   (iv) Strassen's matrix Multiplication.



UNIT-I

Chapter 1 :- Introduction

imp ⇒ **Algorithm** :- An algorithm is a finite set of instructions that can be used to perform a particular task. In other words an algorithm is a step by step procedure to solve a particular problem.

Here problem is nothing but an expression of an algorithm and it contains set of instructions.

⇒ The word algorithm comes from persian author "Abdullah Jafar Muhammad ibn musa Al-Khowarizmi".

⇒ An algorithm is a set of rules for carrying output calculations either hand or on a machine.

⇒ An algorithm is a well defined computational process that takes input and produce output.

⇒ **properties of an algorithm (characteristics)** :-

(i) Input

(ii) output

(iii) Definiteness.

(iv) Finiteness.

(v) Effectiveness.

(i) **Input** :- It generally requires finite numbers of inputs

(ii) **output** :- It must produces atleast one output.

(iii) Definiteness :- Each instruction must be clear and unambiguou (no confusion).

(iv) Finiteness :- It must terminate after finite number of step

(v) Effectiveness :- Every step of an algorithm should be feesable In other words every step of algorithm must be carried out by pen or pencil.

⟹ Design steps of an algorithm :-

1) How to device / create an algorithm
2) How to analyse an algorithm.
3) How to validate an algorithm.
4) How to test an algorithm

1) How to device an algorithm :- creating an algorithm is an art which may never be fully automated.

⟹ There are certain algorithmic design strategies. By using these strategies, one can create many useful algorithms.

⟹ Hence such type of design strategies are important in study of design algorithm.

2) How to analyse an algorithm :-

⟹ Analysis of an algorithm is a task of determining how much time will computing and how much space will be required by the algorithm.

⇒ Analysis of an algorithm is also called as performance analysis and these algorithm mainly concentrated on mathematics and logics.

⇒ The behaviours of an algorithm in best case, average case and worest case needs to be obtained.

3) How to validate an algorithm :-

⇒ The next step, after creation of algorithms to validate the algorithms

⇒ The process of checking whether an algorithm computes the correct answer or not for all possible legal inputs is called algorithm validation.

⇒ The purpose of validation of algorithm is to find whether algorithm works properly or not without being dependent on programming languages.

⇒ once validation of algorithm is completed, a programmer can be return by using corresponding algorithms.

4) How to test an algorithm :-

⇒ Testing an algorithm or program can represent.

(i) debugging.

(ii) profiling (performance measurement)

(i) Debugging :- It is the process of executing program on sample data sets and determine whethes faulty results

occur or not and corrects the errors.

⟹ Debugging can only point to the presence of errors but not to their absence.

(ii) profiling :- performance measurement is the process of executing a correct program and data sets and measuring the time and space and it takes to compute the results.

Based on these two debugging and profiling of an algorithm we have to design the perfect algorithm.

⟹ Pseudo code for expressing an algorithm: (algorithm specifications)

algorithm is basically sequence of instruction written in simple english language.

Based on algorithm there are two more representation used by programmer.

They are flow chart and pseudo.

⟹ Flow chart :- It is graphical representation of algorithms.

Similarly, pseudo is a representation of an algorithm in which each instruction can be given with the help of programming

Let us first understand the _____ writting an algorithm using pseudo-code

1) comments begin with "//" and continue until end of the line

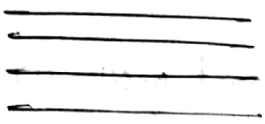
2) Blocks are indicated with matching braces i.e "{" and "}"

3) The compound statement (collection of simple statements) can

represented as a block.

i.e., Algorithm Algorithm_Name (<parameters>)

{

 // collection of statements.

}

Here, Algorithm_Name is the name of the algorithm and <parameters> is the list of parameters and the body has one or more statements enclosed with in the braces i.e., "{" and "}"

4) Statements are end with semicolon ";"

5) The identifier should begin with a letter and we won't use digit (numbers)

6) An identifier is a combination of alphanumeric string and it is not necessary to write data types explicity for identifiers.

7) The basic data types are integers, float, char and boolean and so on.

8) The pointer type is also used to point memory location.

9) The compound data type such as structure (or) record can also be used.

10) Using assignment operator (:, =) we can assign the values to the variables.

i.e., <Variable> := <Expression>.

Ex :- a := 10

a := b + c.

11) There are two boolean values i.e., TRUE and FALSE. Inorder to produce these values AND, OR and NOT are logical operators. and relational operators i.e., <, ≤, =, ≠, >, ≥

12) The array index stored with in [and] braces. The index of array usually start with zero and end with n-1. And the multidimensional array can be used in algorithm.

13) The "while loop" takes the following form while <condition> do

{
<statement 1>
<statement 2>
|
<statement n>
}

As long as condition is true the statements gets executed. when a condition become false the loop is exit and the value of condition is evaluated at the top of the loop.

14) The for loop takes the following form for variable := Value 1 to Value 2 step step do

{
<stmt 1>
<stmt 2>
}

Here, Value 1, Value 2, step are arithmetic expressions.

These step &step is optional and taken as +1 if it doesn't occur.

Step could be positive (or) negative.

14/12/16

15) ⟹ A "repeat - until" statement is constructed as follows:

```
Repeat
    <stat 1>
    <stat 2>
      :
    <stat n>
until <condition>
```

These statements are executed as long as condition is false. The value of condition is computed after executing the statement. Here we can also use "Break" and "return" statements within the loop.

16) ⟹ The "conditional statement" is constructed as follows if

```
if <condition> then <statement>
if <condition> then <statement-1>
else <statement-2>
```

Here condition is boolean expression and statement 1 and statement 2 are arbitary statements. Orbitary statements nothing but

compound statements. Here compound statement nothing but collection of simple statements.

17) The "case" statement is constructed as follows

```
case
{
 :<condition1> : <stmt-1>
 : <condition2> :<stmt -2>
        |
        |
        |
 : <condition n> : <stmt- n>
 : else : <stmt n+1>
}
```

Here statement-1, statement-2, ----- statement-n are the orbitary statements.

18) A "case" statement intrupted constructed as follows:

⇒ if condition-1 is true then statement 1 gets executed and case statement exist.

⇒ If condition -1 is false, then & condition -2 is evaluated. if condition-2 is true then statement 2 gets executed and case statement is exit and so on upto 'n'.

⇒ if none of the above conditions are true i.e., condition-1, condition-2, ------, condition-n then the else statement n+1 is executed. and case statement is exited.

5m

→ performance analysis :-

The efficiency of an algorithm can be decided by measuring the performance of an algorithm. We can measure the performance of an algorithm by computing amount of "time and space" requirement. Here we have to implement performance analysis. we mainly concentrate on :

(i) Space complexity.

(ii) Time complexity.

(i) Space complexity :- Space complexity can be defined as how much space or memory required by an algorithm to run.

To compute the space complexity we have to use 2 factors, i.e "constant" and "instance characteristics".

⇒ The space requirement "S(P)" can be given as $\boxed{S(P) = C + S_p}$

where C = constant i.e., fixed part and denotes space of inputs and outputs. This space is an amount of space taken by inspections, variables and identifiers. where S_p = space dependent upon instance characteristics.

Ex :- Algorithm Add(a,b,c)

```
{
  // a, b, c are floating type
  return a+b+c;
}
```

1 2 3 → 3 space complexity

In the above algorithm a, b and c allocate one word size then the total size is "3"

∴ Space complexity $S(P) = 3$

Ex :- Algorithm Add (x, n)

```
{
  // sum and X[i]
    Sum := 0.0
    for i = 1 to n do
      Sum := sum + X[i];
    return Sum;
}
```

In the above algorithm 'n' space for X[ ], 1 space for 'n', one space for 'i' and 1 space for "Sum".

∴ The space complexity of above algorithm is $S(P) = n+3$.

⟹ Time complexity :- The time complexity can be defined as how much time required to execute an algorithm is called time complexity. There are two types of for evaluating time complexity i.e compile time and run time. The time complexity is generally computed at run time or execution time

Ex :-

```
Algorith Add (x, n)  ——— 0
{                    ——— 0
  // Sum and X[i]
    Sum := 0.0       ——— 1
    for i = 1 to n do ——— n+1
```

```
    sum := sum + x[i];  ——— n
    return sum;         ——— 1
}                       ——— 0
```

Step count = 2n+3

∴ The time complexity is $O(2n+3) = O(n)$.

Ex :- Algorithm add (a, b, c, m, n) ——— 0

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

Step count = $2mn + 2m + 1$

∴ The time complexity is $O(2mn+2m+1) = O(mn)$

Ex :- Algorithm message (n) ——— 0

```
{
  for i:=1 to n do ——— n+1
  {
    write ("Hello"); ——— n
  }
}
```

Step count = $2n+1$

∴ The time complexity $O(2n+1) = O(n)$

⇒ Asymptotic Notations :- To choose the best algorithm, we need to check efficiency of each algorithm, the efficiency can be measured by computing time complexity of each algorithm. In this we have to implement different types of asymptotic notations they are:

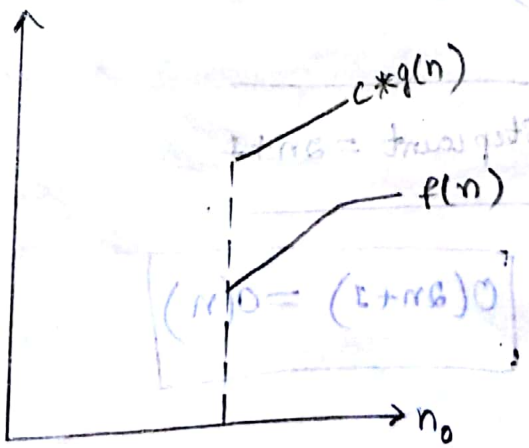
(i) Big oh Notation (O)
(ii) Omega Notation (Ω)
(iii) Theta Notation (Θ)
(iv) Little oh Notation (o)
(v) Little omega Notation (ω).

(i) Big oh Notation :- Let $f(n)$ and $g(n)$ are any non negative function iff their exists positive constants 'c' and 'n_0' then the big oh notation can represent $f(n) \leq cg(n) \quad n \geq n_0$

⇒ In other words $f(n) \leq g(n)$ and $g(n)$ is some multiple of some constant 'c'.

⇒ Big oh Notation is denoted by "O". And it is the method of representing upper bound of algorithm.

⇒ Graph :-



Ex :-(i) $F(n) = 2n^2+3n+1$.

$F(n) \leq cg(n) \quad n \geq n_0$

$2n^2+3n+1 \leq 1 \rightarrow$ False

$2n^2+3n+1 \leq 3n \rightarrow$ False

$2n^2+3n+1 \leq 2n^2 \rightarrow$ False.

$2n^2+3n+1 \leq 3n^2 \rightarrow n \geq n_0$

$45 \leq 48 \rightarrow n_0 = 4$

$\therefore c=3, \; n_0 = 4$

$\therefore \boxed{O(2n^2+3n+1) = O(n^2)}$

(ii) $F(n) = 5n+4$

$F(n) \leq cg(n) \quad n \geq n_0$

$5n+4 \leq 4 \rightarrow$ False

$5n+4 \leq 5n \rightarrow$ False

$5n+4 \leq 6n \rightarrow n \geq n_0$

$24 \leq 24 \rightarrow n_0 = 4$

$\therefore c=6, \; n_0 = 4$

$\therefore \boxed{O(5n+4) = O(n)}$

(iii) $f(n) = 10n^3+6n^2+6n+2$.

$f(n) \leq cg(n) \quad n \geq n_0$

$10n^3+6n^2+6n+2 \leq 2 \rightarrow$ False.

$10n^3+6n^2+6n+2 \leq 6 \rightarrow$ False.

$10n^3+6n^2+6n+2 \leq 6n^2 \longrightarrow$ False.

$10n^3+6n^2+6n+2 \leq 10n^3 \longrightarrow$ False.

$10n^3+6n^2+6n+2 \leq 11n^3 \longrightarrow n \geq n_0$

$3768 \leq 3773 \longrightarrow n_0 = 7$

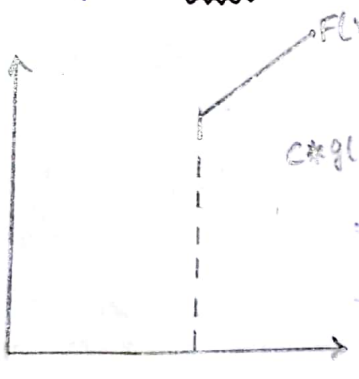
$\therefore c = 11, \; n_0 = 7$

$\therefore \boxed{O(10n^3+6n^2+6n+2) = O(n^3)}$

19/12/16.

(ii) Omega Notation (Ω) :- Let $f(n)$ and $g(n)$ are any non negative functions. if their exists a positive constants 'c' and 'n_0' then the omega notation can be represented as $\boxed{F(n) \geq cg(n)}$ and it is represented by the symbol "Ω".

⇒ Graph :-



Ex :- i) $F(n) = 3n+2$, $g(n) = n$

$F(n) \geq cg(n)$

$3n+2 \geq c(n)$

Sub $n=1 \Rightarrow 5 \geq c(1)$

$c=1. \quad 5 \geq 1$

$\boxed{F(n) \geq cg(n)}$

$3n+2 \geq 3n$

put $n \leq 1 \longrightarrow 5 \geq 3$ (True)

$3n+2 \geq 3n \; \forall \; n \geq 3$

$c=3, \; g(n) = n, \; n_0 = 3$

$\boxed{3n+2 = \Omega(n)}$

(ii) $F(n) = 10n^2 + 4n + 2$, $g(n) = n^2$

$F(n) \geq cg(n)$

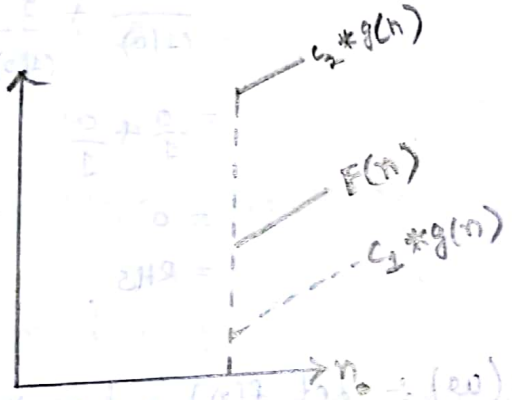
$10n^2 + 4n + 2 \geq cn^2$

Sub $n = 1$, $c = 1$ $\Rightarrow 16 \geq 1$.

(iii) Theta Notation (Θ) :- Let $f(n)$ and $g(n)$ are any non negative functions iff their exists c_1, c_2 and n_0 are the constants such that the theta notation can be represented as

$c_1 * g(n) \leq F(n) \leq c_2 * g(n)$. and it is represented by the Symbol "Θ"

$\Rightarrow$ Graph :-



Ex:-1) $F(n) = 3n + 2$, $g(n) = n$.

$c_1 * g(n) \leq F(n) \leq c_2 * g(n)$

$c_1 * n \leq 3n + 2 \leq c_2 * n$

Sub $c_1 = 1$, $c_2 = 4$, $n = 2$

$1 * n \leq 8 \leq 4 * n$

$2 \leq 8 \leq 8$.

(or)

$3n \leq 3n + 2 \leq 4n$

$n = 1 \Rightarrow 3 \leq 5 \leq 4$

$n = 2 \Rightarrow 6 \leq 8 \leq 8$ (T)

$3n \leq 3n + 2 \leq 4n$, $n \geq 2$

$c_1 = 3$, $c_2 = 4$, $g(n) = n$, $n_0 = 2$

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

$3n + 2 = \Theta(n)$

(iv) Little oh Notation :- (o) Let $f(n)$ and $g(n)$ are any two non negative functions if their exists $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. In other words it can be written as $F(n) = o(g(n))$.

Ex) $F(n) = 3n + 2$, $g(n) = n^2$.

LHS :- $\lim_{n\to\infty} \frac{f(n)}{g(n)}$

$= \lim_{n\to\infty} \frac{3n+2}{n^2}$

$= \lim_{n\to\infty} \frac{3n}{n^2} + \frac{2}{n^2} = \lim_{n\to\infty} \frac{3}{n} + \frac{2}{n^2} = \frac{3}{\infty} + \frac{2}{\infty}$

$= \frac{3}{(1/0)} + \frac{2}{(1/0)}$

$= \frac{0}{1} + \frac{0}{1}$

$= 0$

$=$ RHS.

(v) Little omega Notation (ω) :- Let $f(n)$ and $g(n)$ are any two non negative functions if their exists $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

Ex :- $F(n) = 3n + 2$, $g(n) = n^2$,

LHS :- $\lim_{n\to\infty} \frac{g(n)}{f(n)}$

$= \lim_{n\to\infty} \frac{n^2}{3n+2}$

$= \lim_{n\to\infty} \frac{n^2}{n(3+\frac{2}{n})} = \lim_{n\to\infty} \frac{n}{(3+\frac{2}{n})}$

$= \frac{\infty}{3+\frac{2}{\infty}} = 0 =$ RHS

Imp ⇒ NOTE :-

(i) Best case :- Minimum no. of steps executed by a program or algorithm is called as best case.

(ii) Average case :- Average no. of steps executed by a program or algorithm is called as average case,

(iii) Worest case :- Maximum no. of steps executed by a program or algorithm is called as worest case,

Ex :-

	10	30	20	9	6	8	11	35	60	41
A[	1	2	3	4	5	6	7	8	9	10]

(i) Searching element $x=20$:-

The searching element $x=20$ is occured in best case because minimum no. of steps comparision.

(ii) Searching element $x=6$:-

Here searching element $x=6$ has occured at average case because average no. of steps comparision.

$$Avg = \frac{1+10}{2} = 5$$

$\therefore A[5] = 6.$

(iii) Searching element $x=60$:-

Here searching element is $x=60$ is occured as worest case because of maximum no. of steps comparision.

5m

⇒ **Probabilistic Analysis** :- The probabilistic analysis or probabilistic theory is a goal of characterising the outcomes of natural experiments. i.e

(i) Tossing a coin 'n' times

(ii) Rolling a dye 'n' times.

(iii) playing a lottery.

(iv) Gambling

(v) picking a ball.

(vi) playing cards.

⇒ **Sample point** :- Each possible outcome of an experiment is called as a sample point.

⇒ **Sample Space** :- The set of all possible outcomes is called as Sample Space.

⇒ If 'n' coins are tossed then the no. of possible outcomes of an experiment is 2^n.

⇒ Ex :- Tossing 3 coins, we get $2^n = 2^3 = 8$ possible outcomes.

⇒ Therefore Sample Space = {HHH, HHT, HTH, HTT, THH, THT, TTH, TTT}

⇒ If 'n' dice are rolling then the no. of possible outcomes of an experiment is 6^n

Ex :- If 2 dice are thrown then the possible outcomes of an

experiment is $6^n = 6^2 = 36$ possible outcomes.

∴ Sample space :-

$$\begin{Bmatrix} (1,1)\ (1,2)\ (1,3) \cdots (1,6) \\ (2,1)\ (2,2)\ (2,3) \cdots (2,6) \\ (3,1)\ (3,2)\ (3,3) \cdots (3,6) \\ (4,1)\ (4,2)\ (4,3) \cdots (4,6) \\ (5,1)\ (5,2)\ (5,3) \cdots (5,6) \\ (6,1)\ (6,2)\ (6,3) \cdots (6,6) \end{Bmatrix}$$

⟹ **Mutual exclusive** :- If two events E_1' and E_2' are said to be mutual exclusive if they do not have common sample point.

i.e $\boxed{E_1 \cap E_2 = \phi}$

Ex :- Tossing 3 coins, when we toss 3 coins, Let E_1' be a event that their are 2 heads and E_2' be a event that their are atleast 2 tails.

$E_1 = \{THH, HTH, HHT\}$

$E_2 = \{TTH, THT, HTT, TTT\}$

$\therefore \boxed{E_1 \cap E_2 = \phi}$

21/12/16

5m

⟹ probabilistic Analysis :-

⟹ **Independence** :- Two events E_1' and E_2' are said to be independent if the probability of E_1' and E_2' is $\boxed{P(E_1 \cap E_2) = P(E_1) * P(E_2)}$

⟹ If E_1 and E_2 are any two events then the probability of E_1 and E_2 can represent as

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$$

⟹ **Conditional probability** :- Let E_1 and E_2 are any two events of experiments the conditional probability of E_1 is given by E_2 denoted by.

$$P\left(\frac{E_1}{E_2}\right) = \frac{P(E_1 \cap E_2)}{P(E_2)}$$

here $P(E_2) = \dfrac{P(E_1 \cap E_2)}{P\left(\frac{E_1}{E_2}\right)}$

$$P(E_1 \cap E_2) = P\left(\frac{E_1}{E_2}\right) \cdot P(E_2).$$

similarly if E_2 is given by E_1 then the conditional probability is $P\left(\frac{E_2}{E_1}\right) = \dfrac{P(E_1 \cap E_2)}{P(E_1)}$

$$P(E_1) = \frac{P(E_1 \cap E_2)}{P\left(\frac{E_2}{E_1}\right)}$$

$$P(E_1 \cap E_2) = P\left(\frac{E_2}{E_1}\right) \cdot P(E_1).$$

Ex:- consider the experiment of tossing 4 coins we get $2^4 = 16$ possible outcomes. and let E_1 be the event that the no. of heads is even and E_2 be the event that their are no heads. The probability of no heads is $1/16$ therefore $P(E_1) = 1 - \frac{1}{16} = \frac{15}{16}$

$P(E_2) = 1/16$ then $P(E_1 \cap E_2) = 0$

∴ The conditional probability

$$P(E_1/E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)}$$

$$= \frac{0}{P(E_2)}$$

$$= 0$$

∴ The probability range is $0 \leq p \leq 1$.

⇒ In this probabilistic analysis we have to use random variables and randomized algorithms.

⇒ If we pick 4 balls from bag containing red and white balls then the no of red balls that gets selected is:

$F(RRRW) = 3$

$F(RWWW) = 1$

Ex:- A bag contains 6 red balls and 6 white balls selecting 5 balls it contains atleast 2 red balls. i.e

$6C_2 \times 6C_3 + 6C_3 \times 6C_2 + 6C_4 \times 6C_1 + 6C_5 \times 6C_0$.

⇒ probability distribution :- If 'F' is discrete random variable, for sample space 'S' then the probability distribution can be defined as

$$\sum_{i=1}^{n} P(F = a_i) = 1$$

Ex:- If we pick 4 balls from bag containing red and white balls and F is no. of red balls then F can take 5 values i.e 0, 1, 2, 3, 4 then

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Here '1' means presence of red ball.
and '0' means absence of red ball.
Hence the probability distribution of 'F' is given by :

$P(F=0) = \frac{1}{16}$ i.e no red ball present (0000)

$P(F=1) = \frac{4^{x1}}{16^{84}}$ i.e only 1 red ball present.

$P(F=2) = \frac{6}{16} = \frac{3}{8}$ i.e only 2 red balls present.

$P(F=3) = \frac{4}{16} = \frac{1}{4}$ i.e 3 red balls present.

$P(F=4) = \frac{1}{16}$ i.e 4 red balls present.

⇒ Binomial Distribution :-

A "Bernouli Trail" is an experiment that has two possible outcomes i.e Success and failure. The probability of Success is 'p'. then the binomial distribution constructed as follows

$$P(x=i) = \binom{n}{i} p^i (1-p)^{n-i}$$

$\left(\binom{n}{i} = {}^nC_i\right)$

5m/20

⇒ Amortized complexity (Analysis) :-

↓

(Determine avg runtime for execution of a program).

Amortized analysis defines that to determine the average run time for execution of a program and amortized analysis

does not allow the random selection of all possible inputs.

⇒ Amortized means finalising the average run time for operation over a worst case sequence of operations.

⇒ NOTE :- The only requirement is that the sum of the amortized complexity of all operations in any sequence of operations be greater than or equal to sum of actual complexity i.e

$$\sum_{1 \le i \le n} \text{Amortized}(i) \ge \sum_{1 \le i \le n} \text{Actual}(i)$$

Eg :- (i) Linear Search :-

1	2	3	4	5	6	7	8	9
10	20	15	8	29	30	50	40	25

No. of comparisions required to search 10 is 1 comparision

20 → 2 comparision, 15 → 3 comparisions, 8 → 4 comparisions

29 → 5 comparisions, 30 → 6 comparisions, 50 → 7 comparisions,

40 → 8 comparisons, 25 → 9 comparisions.

Total 45 comparisions $(9+8+7+6+5+4+3+2+1)$

∴ The amortized cost = $\frac{\text{Total no. of comparisions}}{\text{Total no. of elements present in array}}$

$= \frac{45}{9}$

$= 5$

(ii) Binary Search :-

A[1	2	3	4	5	6	7]
10	15	20	21	40	45	70

$$mid = \frac{low+high}{2}$$

$$= \frac{1+7}{2}$$

$$= 8/2 = 4$$

Here a[4] = 21 it takes 1 comparison.

i.e 10 15 20 [21] 40 45 70

left mid right.

The total no of comparisions required to search 10 is 3 comparis

" " " " " " " 15 → 2 compari

" " " " " " " 20 → 3 "

" " " " " " " 21 → 1 "

" " " " " " " 40 → 3 "

" " " " " " " 45 → 2 "

" " " " " " " 70 → 3 "

Total 17 comparisons

∴ The amortized cost = $\frac{\text{Total no. of comparisions}}{\text{Total no. of elements present in array}}$

$$= \frac{17}{7}$$

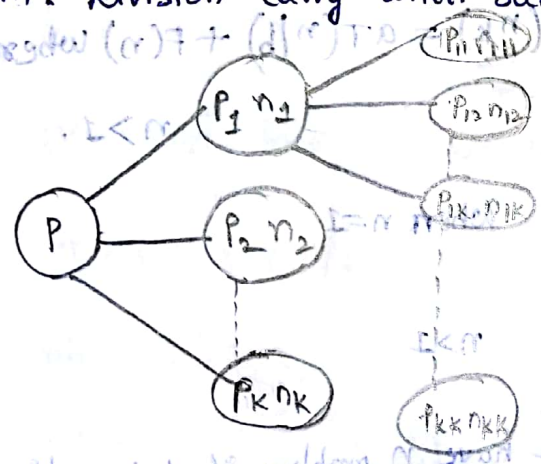
$$= 2.4$$

27/12/16.

CH: 2

Divide and conquer (10 marks important)

1) General Method :- Let 'p' be the problem with 'n' inputs and it is not a small problem then divide problem 'p' into $P_1, P_2, P_3 \ldots P_k$ sub problems with $n_1, n_2, n_3 \ldots n_k$ inputs respectively. If sub problems is not a small problem further we can divide sub-problem into small problems. We solve all individual sub problems and combine all these solutions of sub problems to the main problem. Suppose if the problem is small then return the solution of the problem. Division carry until sub problems become small problems.



2) Control abstraction for divide and conquer :-

```
Algorithm  Divide and Conquer (P)
{
   if (P is small) then
      return S(P);
   else
      Divide problem P into sub problem P1, P2, P3 ---- Pk
      with n1, n2, n3, ---- nk inputs respectively.
```

Apply divide and conquer to the each problem.

return combine (Divide and conquer (P_1), Divide and conquer (P_2), ... Divide and conquer (P_k));

}

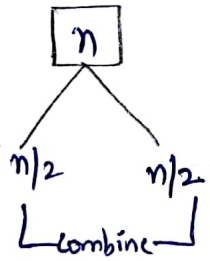
⟹ The time complexity of divide and conquer algorithm is given by recurrance relation

$$T(n) = \begin{cases} T(1) & \text{when } n=1 \\ T(n_1)+T(n_2)+\cdots+T(n_k)+F(n) & \text{otherwise} \end{cases}$$

Here, $T(n_1)+T(n_2)+\cdots+T(n_k) = aT(n/b)+F(n)$ where $n>1$.

$$\therefore T(n) = \begin{cases} T(1) & \text{when } n=1 \\ aT(n/b)+F(n) & n>1 \end{cases}$$

Ex:-



Suppose we have n problem it takes time complexity $T(n)$. we divide 'n' problem into $n/2$, $n/2$ however we get n. $n/2$ time complexity is $T(n/2)$, $T(n/2)$ so total $2T(n/2)$

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

3) Iterative Substitution method :- It is one way to solve divide and conquer recurrance relation by using substitution method. In this method we assume that the problem size 'n' is large.

and substitute alternative recurrance relation. The general recurrance relation of divide and conquer is

$$\boxed{T(n) = 2^i T(n/2^i) + ibn}$$

Here 'n' is power of 'b'. i.e $n = b^k \Rightarrow \boxed{k = \log_b n}$

28/12/16.

⇒ Solve the following recurrance relation

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

Sol:- Given that $T(n) = 2T(n/2)+n$ ——— (1)

put $n=n/2$ in eq (1) we get

$T(n/2) = 2T(n/2^2)+n/2$ ——— (2)

Sub $T(n/2)$ in eq (1)

$T(n) = 2\left(\left(2T(n/2^2)\right)+n/2\right)+n$

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

$= 2^3 T(n/2^3)+3n$ ——

;

;

$= 2^k T(n/2^k)+kn$ ——— (3)

put $n=2^k$ i.e $k = \log_2 n$

Sub n & k values in (3)

$$T(n) = nT\left(\frac{2^k}{2^k}\right) + n\log_2 n$$

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

$$\boxed{T(n) = 2n + n \cdot \log_2 n} \quad \Rightarrow (T(1)) = 2 \text{ from given question.}$$

4) **Master's Method :-** consider the following recurrance relation

$$T(n) = aT(n/b) + F(n) \quad \text{---(1)}$$

cond if $F(n)$ is order $\Theta(n^d)$ where $d \geq 0$ then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b^a}) & \text{if } a > b^d \end{cases}$$

$\Rightarrow$ $T(n) = 9T(n/3) + n$

Sol :- Given that $T(n) = 9T(n/3) + n$

compare with $T(n) = aT(n/b) + F(n)$

Here $a = 9, b = 3$, $d =$ power of 'n' $= 1$,

From all these values we can get

$a > b^d \Rightarrow 9 > 3^1$

$= 9 > 3$ condition is true then

$$\Theta(n^{\log_b^a}) = \Theta(n^{\log_3^9})$$

$$= \Theta(n^{\log_3^{3^2}})$$

$= \Theta(n^{2\log_3^3})$

$= \Theta(n^2)$ $(\log_n^n = 1)$.

$\Rightarrow T(n) = 2T(n/2) + 1$

Compare with general term $T(n) = aT(n/b) + F(n)$

Here $a=2, b=2, d=0$ $(a^0=1)$

$a > b^d = 2 > 2^0 = 2 > 1$ condition is satisfied then

$\Theta(n^{\log_b^a}) = \Theta(n^{\log_2^2})$

$= \Theta(n)$.

$\Rightarrow$ Binary Search :- In binary search to determine given element present in the array or not. Suppose if given element is present in array to determine the position of the array. If given element is not in array it returns zero value.

$\Rightarrow$ Algorithm :-

Algorithm BS (A, n, X) (Binary search)

{

low = 1

high = n

while (low ≤ high)

{

mid = $\frac{low + high}{2}$;

if (x < A[mid]) then

```
        high = mid - 1;
    else
        if (x > A[mid]) then   (x → searching element)
            low = mid + 1;
        else return mid;
    }
    return 0;
}
```

→ Explanation:- First give the name of the algorithm as BS(parameters) where BS → Binary Search and we are passing Variables/parameters as A, n, x.

⇒ We are giving low value as 1 and high value = n. and compare while condition. if while condition (low ≤ high) is true compilers comes down and evaluate mid value as mid = (low+high)/2 and it checks the if condition i.e if (x < A[mid]) where x is searching element if it satisfies it goes to next line i.e high = mid - 1 will be evaluated.

⇒ if it doesn't satisfy it goes to another if condition i.e if (x > A[mid]) if it is true then low = mid + 1 will be executed. if this is also not correct then it returns mid value. If while loop is not true it goes to out of while loop and returns null value i.e return 0 it means there is no such element in the array.

⇒ Ex:- consides the array elements 1, 2, 5, 6, 7, 10, 24,
56, 84, 100, 115, 120, 131, 150.

(1 2 3 4 5 6 7 / 8 9 10 11 12 13 14)

1) Find Searching element X=150

2) Find Searching element X=9.

Sol:- 1) X=150 (Searching element).

Low	High	mid	A[mid]	X ? A[mid]
1	14	7	24	150 > 24 → mid+1.
8	14	11	115	150 > 115
12	14	13 (3rd iteration mid value)	131	150 > 131.
14	14	14	150	150 = 150
				Element is found.

2) X=9 (Searching element).

Low	High	mid	A[mid]	X ? A[mid]
1	14	7	24	9 < 24 → mid-1 = high.
1	6	3	5	9 > 5 → mid+1
4	6	5	7	9 > 7 → mid+1 = low.
6	6	6	10	9 < 10 → high = mid-1.
6	5	—	—	Element not found.

10m

⟹ Consider the elements A[1 2 3 4 5 6 7 8 9 10 11 12 13 14]
−15, −6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151.

(i) Searching element X = 0.

(ii) Searching element x = 9

(iii) Searching element x = −2.

(iv) Draw the binary decission tree for the above elements.

Sol :- Given elements.

−15, −6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151

A[1 2 3 4 5 6 7 8 9 10 11 12 13 14]

(i) Searching element X = 0,

Low	High	mid	A[mid]	X ? A[mid]
1	14	7	54	0 < 54 high = mid − 1
1	6	3	0	0 = 0
				Element is found.

(ii) Searching element X = 9

Low	High	mid	A[mid]	X ? A[mid]
1	14	7	54	9 < 54
1	6	3	0	9 > 0 low = mid + 1
4	6	5	9	9 = 9
				Element found.

(iii) Searching element $x = -2$.

Low	High	mid	A[mid]	x ? A[mid]
1	14	7	54	$-2 < 54$
1	6	3	0	$-2 < 0$
1	2	1	-15	$-2 > -15$
2	2	2	-6	$-2 > -6$.
3	2	—	—	Element not found.

Because while (low ≤ high) i.e

while (3 ≤ 2) condition is false then it returns 0

when compiler returns '0' it means there is no element present in array.

(iv) Decision tree :-

Given

A[1 2 3 4 5 6 7 8 9 10 11 12 13
-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142,
14
151.

$$mid = \frac{low + high}{2} = \frac{1+14}{2} = \frac{15}{2} = 7$$

$A[7] = 54$

1) Here, 7 is a root node i.e

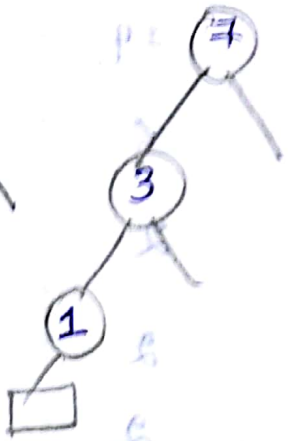
1 2 3 4 5 6 7 8 9 10 11 12 13 14
-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151.

Left side elements mid Right side elements.

2) Now we consider left side elements i.e

* [1 2 3 4 5 6]
-15, -6, 0, 7, 9, 23.

$mid = \frac{1+6}{2} = 3$. i.e.

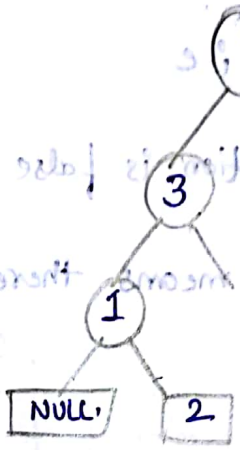


* [1 2 3 4 5 6].

3) -15, -6, 0, 7, 9, 23

left mid right.

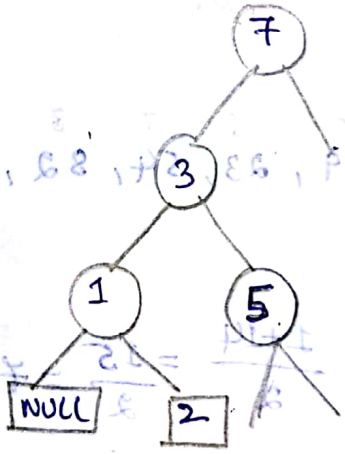
$mid = \frac{1+2}{2} = 1$



1 2 3 4 5 6

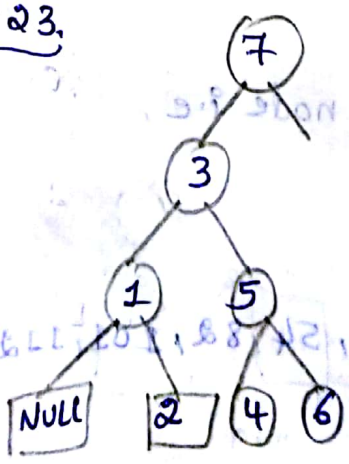
4) -15, -6, 0, 7, 9, 23.

$mid = \frac{4+6}{2} = 5$.

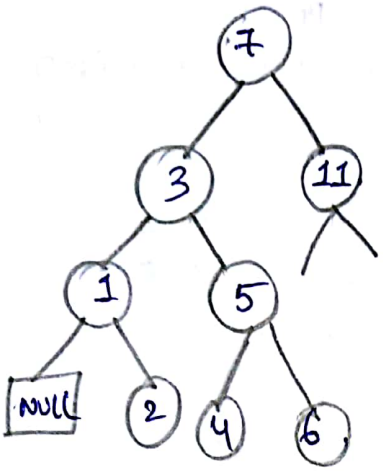


1 2 3 4 5 6

5) -15, -6, 0, 7, 9, 23.



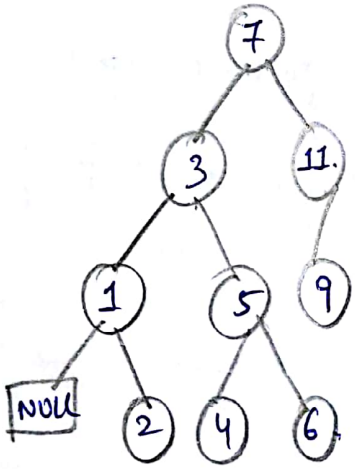
7) Now considers right side elements.



$mid = \frac{8+14}{2} = 11.$

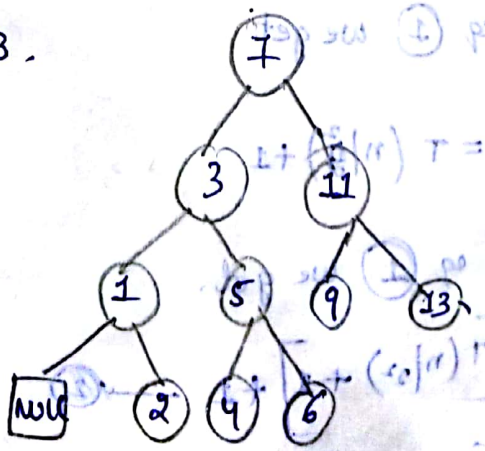
8 9 10 11 12 13 14.
82, 101, 112, 125, 131, 142, 151.

8) $mid = \frac{8+10}{2} = 9$ (Left elements).



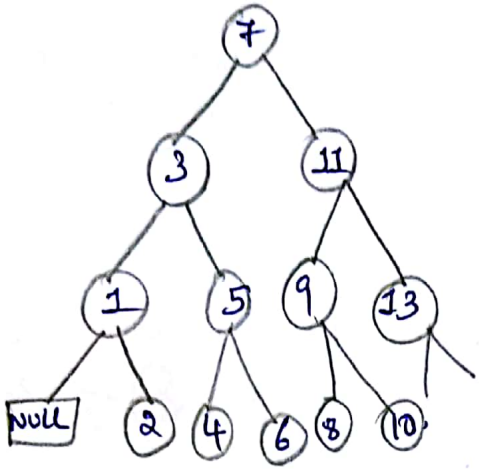
9) 8 9 10 11 12 13 14
82, 101, 112, 125, 131, 142, 151

$mid = \frac{12+14}{2} = 13.$



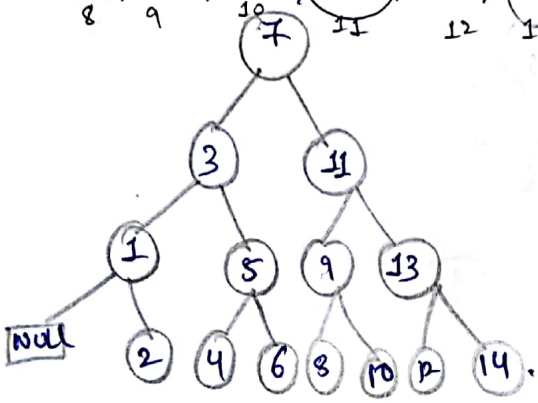
10) 82, 101, 112, 125, 131, 142, 151

8 9 10 11 12 13 14.



11) 82, 101, 112, 125, 131, 142, 151.

8 9 10 11 12 13 14



30/12/16

⇒ Time complexity of Binary Search :-

Let the recurrance relation $T(n) = T(n/2) + 1$ ——— (1) → everytime we should search 1 element

put $n = n/2$ in eq (1) we get

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

Sub $T(n/2)$ in eq (1) we get.

$$T(n) = \left[T(n/2^2) + 1\right] + 1 \quad \text{——— (2)}$$

$$T(n) = \left[T(n/2^2)\right] + 2 \quad \text{——— (2)}$$

Again Substitute $T(n/2)$ in eq (2) we get,

$$T(n) = T\left(n/2^3\right) + 3$$

$$\boxed{T(n) = T\left(n/2^k\right) + k}$$

Put $n = 2^k$ where $k = \log_2 n$

$T(1) = 1$.

$$T(n) = T\left(2^k/2^k\right) + \log_2 n$$

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

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

$$\boxed{T(n) = O(\log n)} \quad \text{(neglect constants)}$$

optional

⇒ Recursive algorithm for binary Search :-

```
Algorithm    RBS (a, low, high, x)
{
  if (high == low) then
  {
    if (x == a[low]) then
      return low;
    else
      return 0;
  else
    mid = (low+high) / 2 ;
```

```
        return mid;
    else
        if (x < A[mid]) then
    return RBS(a, low, mid-1, X)        (high)
    else
        return RBS(a, mid+1, high, X)   (low)
    }
}
```

Merge Sort :- The Merge sort is a sorting algorithm that uses the divide and conquer strategy. In this method division is dynamically carried out. Here we mainly concentrated on 3 steps.

(i) Divide

(ii) conquer

(iii) Combine → Advance tool

(i) Divide :- Divide the array into sub arrays i.e. S_1' & S_2' with $mid = \frac{low + high}{2}$.

(ii) Conquer :- Recursively sort S_1' & S_2'.

(iii) Combine :- Combine the S_1' & S_2'.

Algorithm :-

```
Algorithm  Mergesort (low, high)
{
  if (n=1) then
    return;
  else {
    if (low<high) then
    mid := (low+high)/2 ;
    mergesort (low, mid);
    mergesort (mid+1, high);
    combine (low, mid, high);
  }
}.
```

⟹ Consider the list of elements 65, 70, 60, 75, 85, 80, 55, 50, 45, 57. by using merge sort algorithm. and draw the decision tree.

Sol :- Given elements are

1	2	3	4	5	6	7	8	9	10
65	70	60	75	85	80	55	50	45	57

$$mid = \frac{low + high}{2}$$

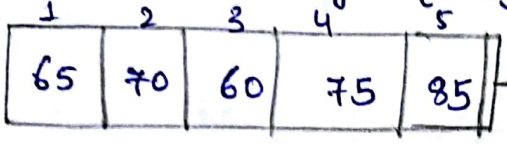
$$mid = \frac{1+10}{2} = 5$$

We will divide the array into sub arrays i.e

A[1-5] and A[6-10]

low, mid mid+1, high

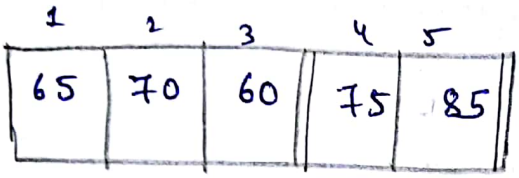
Now consider Sub array A[1-5] elements i.e



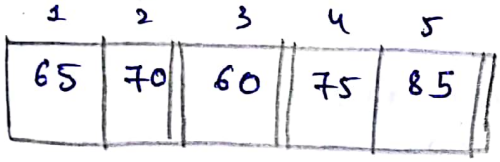
→ double line bec3 array divided

$$mid = \frac{1+5}{2} = 3$$

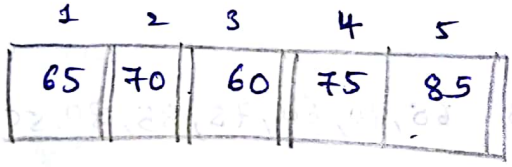
i.e



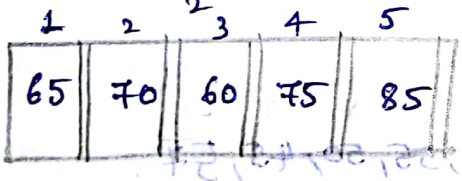
$$mid = \frac{1+3}{2} = 2$$



$$mid = \frac{1+2}{2} = 1$$

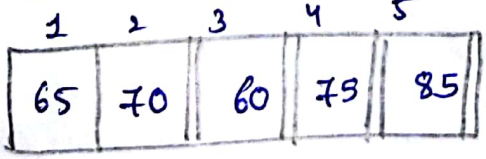


$$mid = \frac{4+5}{2} = 4$$



Now every set contains only one element and combine

A[1] and A[2] are sort them i.e



Now combine A[1-2] and A[3] and sort them i.e

1	2	3	4	5
60	65	70	75	85

Now combine A[1-3] and A[4] and Sort them i.e.

1	2	3	4	5
60	65	70	75	85

Now combine A[1-4] and A[5].

1	2	3	4	5
60	65	70	75	85

Thus the list A[1-5] is sorted and we consides right Sub array. A[6-10] i.e

6	7	8	9	10
80	55	50	45	57

$mid = \frac{6+10}{2} = 8$ i.e

6	7	8	9	10
80	55	50	45	57

$mid = \frac{6+8}{2} = 7$ i.e

6	7	8	9	10
80	55	50	45	57

$mid = \frac{6+7}{2} = 6$. i.e

6	7	8	9	10
80	55	50	45	57

$mid = \frac{9+10}{2} = 9$ i.e

6	7	8	9	10
80	55	50	45	57

Now every set contains only one element. Now combine A[6] and A[7] and sort them i.e

6	7	8	9	10
55	80	50	45	57

Now combine A[6-7] and A[8] and sort them i.e

6	7	8	9	10
50	55	80	45	57

Now combine A[6-8] and A[9] and sort them i.e

6	7	8	9	10
45	50	55	80	57

Now combine A[6-9] and A[10] and sort them i.e

6	7	8	9	10
45	50	55	57	80

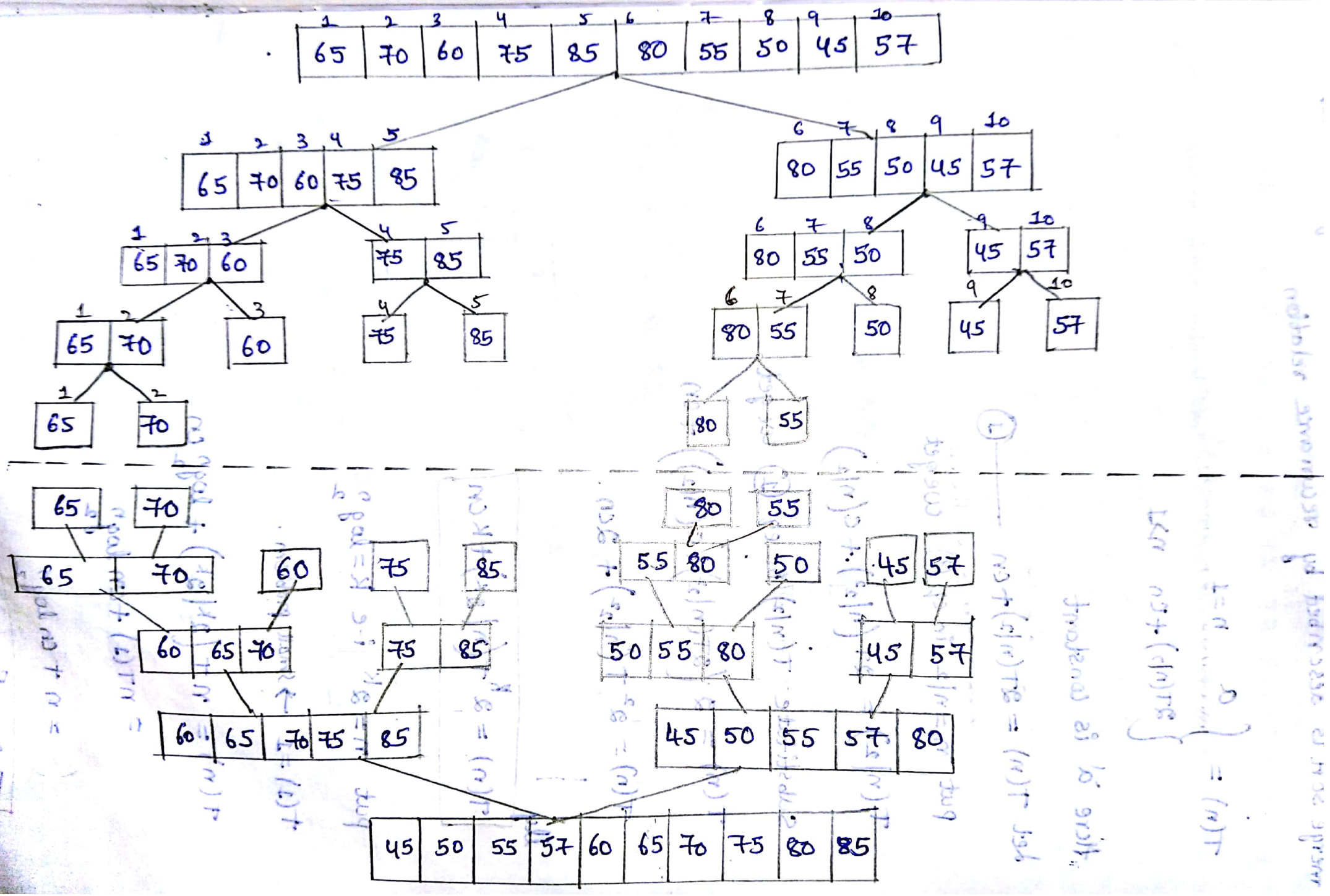
Now combine two sorted sub arrays and sort them i.e

A[1-5] & A[6-10]

1	2	3	4	5	6	7	8	9	10
45	50	55	57	60	65	70	75	80	85

⇒ Decision tree:-

65	70	60	75	85	80	55	50	45	57



⇒ Time complexity of Merge sort :- The time for merging operation is proportional to 'n'. then the computing time for merge sort is described by recurrance relation

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

Here 'a' is constant

Let $T(n) = 2T(n/2)+cn$ ——— (1)

put $n = n/2$ in eq (1) we get

$T(n/2) = 2T(n/2^2) + c(n/2)$

Substitute $T(n/2)$ in eq (1) we get.

$T(n) = 2\cdot\left(2T(n/2^2)+c(n/2)\right)+cn$

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

⋮

Ily

$$\boxed{T(n) = 2^k T(n/2^k) + kcn}$$

put $n = 2^k$ i.e $k = \log_2 n$

$T(1) = 1$ → small problem.

$T(n) = nT(2^k/2^k) + \log_2 n\, cn$

$= nT(1) + cn\log_2 n$

$= n + cn\log_2 n$

$$\boxed{T(n) = n[1+c\log_2 n]}$$

neglect constant values we get efficiency of time complexity of an algorithm.

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

2008 ⇒ Consider the List of elements 62, 71, 72, 80, 82, 60, 52, 51, 42 by using merge sort algorithm, and draw the decision tree.

Given elements are

1	2	3	4	5	6	7	8	9
62	71	72	80	82	60	52	51	42

$$mid = \frac{low + high}{2} = \frac{9+1}{2} = 5$$

We will divide the array into sub-arrays i.e

A[1-5] and A[6-9]

Now consider sub array A[1-5] elements i.e.

1	2	3	4	5
62	71	72	80	82

$$mid = \frac{1+5}{2} = 3 \text{ i.e}$$

1	2	3 2	4	5
62	71	72	80	82

$$mid = \frac{1+3}{2} = 2 \text{ i.e}$$

1	2	2 3	4	5
62	71	72	80	82

$$mid = \frac{1+2}{2} = 1.$$

1	2	3	4	5
62	71	72	80	82

$$mid = \frac{4+5}{2} = 4 \text{ i.e}$$

1	2	3	4	5
62	71	72	80	82

Now every set contains only one element and combine A[1] and A[2] and sort them i.e

1	2	3	4	5
62	71	72	80	82

Now combine A[1-2] and A[3] and sort them i.e

1	2	3	4	5
62	71	72	80	82

Now combine A[1-3] and A[4] and sort them i.e

1	2	3	4	5
62	71	72	80	82

Now combine A[1-4] and A[5] and sort them i.e

1	2	3	4	5
62	71	72	80	82

Thus the list A[1-5] is sorted and now we consider right sub array A[6-9] i.e

6	7	8	9
60	52	51	42

$mid = \frac{6+9}{2} = 7$

6	7	8	9
60	52	51	42

$mid = \frac{6+7}{2} = 6.$

6	7	8	9
60	52	51	42

$mid = \frac{8+9}{2} = 8$ i.e

6	7	8	9
60	52	51	42

Now every set contains only one element. Now combine A[6] and A[7] and sort them i.e

6	7	8	9
52	60	51	42

Now combine A[6-7] and A[8] and sort them i.e

6	7	8	9
51	52	60	42

Now combine A[6-8] and A[9] and sort them i.e

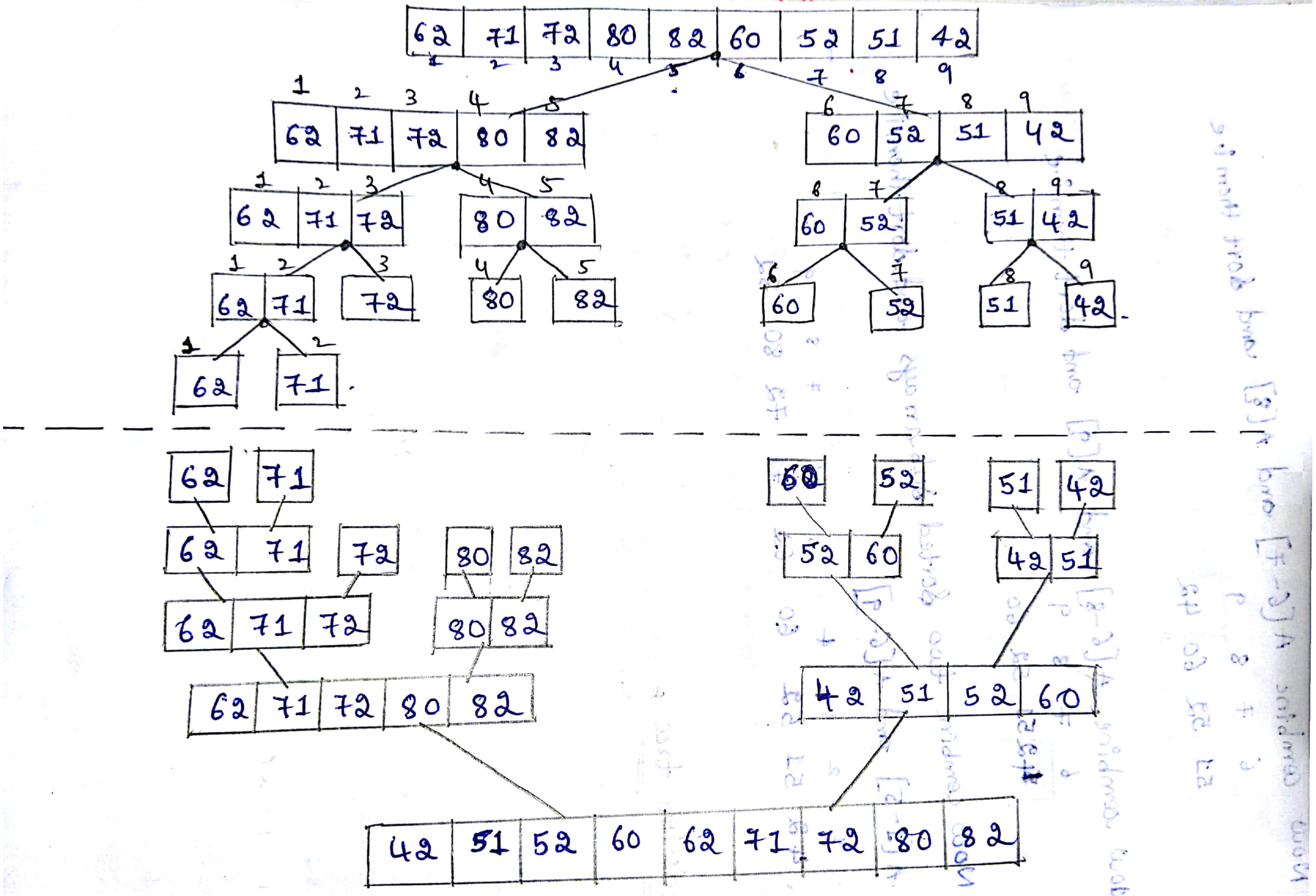
6	7	8	9
42	51	52	60

Now combine two sorted sub-arrays and sort them i.e

A[1-5] and A[6-9]

1	2	3	4	5	6	7	8	9
42	51	52	60	62	71	72	80	82

⇒ Decission tree - (next page)



02/01/2017

Quick sort :- Quick sort is a sorting algorithm that uses the divide and conquer strategy. In this method division is dynamically carried out. So in this we have to mainly concentrate on 3 steps as techniques.

(i) Divide

(ii) conquer

(iii) combine.

(i) Divide :- Rearrange the elements and split the array into two sub arrays and each element in the left sub array is less than or equal to the middle element and each element in right sub array is greater than middle element.

(ii) conquer :- Recursively sort the two sub arrays.

(iii) Combine :- Combine all sorted elements.

Algorithm :-

```
Algorithm Quicksort (a, low, high)
{
    pivot := a[low];
    lb := low;
    ub := high;        (ub → upper bound)
    while (lb ≤ ub) do
    {
        while (a[lb] ≤ pivot)
```

```
lb := lb+1
while (a[ub] ≥ pivot)
    ub := ub-1;
if (lb<ub) then
    swap (a, low, high)
                ↳ swap (a[low], a[high])
{
    a[low] := a[ub];
    a[ub] := pivot;
    return ub;
}
```

→ consider the list of elements 65, 70, 75, 80, 85, 60, 55, 50, 45 by using quick sort algorithm.

Sol :- Given elements are

A[	1 ← lb	2	3	4	5	6	7	8	9 → ub]
	65	70	75	80	85	60	55	50	45
	↑ pivot								

i.e lb=1, ub=9, pivot=65

lb	ub	lb≤ub	A[lb]≤pivot	A[ub]≥pivot	lb<ub
1	9	1≤9 (T)	A[1] 65≤65 (T)	—	—
2	9	2≤9 (T)	70≤65 (F)	45≥65 (F)	2<9 (T) swap

swap A[2] and A[9] i.e

1	2	3	4	5	6	7	8	9
65	45	75	80	85	60	55	50	70

2	9	2≤9 (T)	45≤65 (T)	—	—
3	9	3≤9 (T)	75≤65 (F)	70≥65 (T)	—
3	8 (ub=1)	3≤8 (T)	75≤65 (F)	50≥65 (F)	3<8 (T)

swap A[3] & A[8] i.e

lb ub

65 45 50 80 85 60 55 75 70 .

1 2 3 4 5 6 7 8 9

3	8	3 ≤ 8 (T)	50 ≤ 65 (T)	—	—
4	8	4 ≤ 8 (T)	80 ≤ 65 (F)	75 ≥ 65 (T)	—
4	7	4 ≤ 7 (T)	80 ≤ 65 (F)	55 ≥ 65 (F)	4 < 7 (T)

Swap A[4] & A[7] i.e

lb ub

65 45 50 55 85 60 80 75 70 .

1 2 3 4 5 6 7 8 9

4	7	4 ≤ 7 (T)	55 ≤ 65 (T)	—	—
5	7	5 ≤ 7 (T)	85 ≤ 65 (F)	80 ≥ 65 (T)	—
5	6	5 ≤ 6 (T)	85 ≤ 65 (F)	60 ≥ 65 (F)	5 < 6

Swap A[5] and A[6] i.e

lb ub

65 45 50 55 60 85 80 75 70 .

1 2 3 4 5 6 7 8 9

5	6	5 ≤ 6 (T)	60 ≤ 65 (T)	—	—
6	6	6 ≤ 6 (T)	85 ≤ 65 (F)	85 ≥ 65 (T)	
6	5	6 ≤ 5 (F)			

a[low] = a[ub]

a[low] = a[5] → a[ub] = pivot.

a[5] = 65 (pivot)

return 5 .

∴ The new list elements are :

leftside ≤ middle ; rightside ≥ middle

1 2 3 4 5 6 7 8 9

60 45 50 55 [65] 85 80 75 70 .

List 1 elements — List-2 elements.

Now consides list 1 elements i.e

(lb) 1	2	3	4 (ub)
60	45	50	55 .
↑ pivot			

lb	ub	lb≤ub	A[lb] ≤ pivot	A[ub] ≥ pivot	lb<ub
1	4	1≤4(T)	60≤60(T)	—	—
2	4	2≤4(T)	45≤60(T)	—	—
3	4	3≤4(T)	50≤60(T)	—	—
4	4	4≤4(T)	55≤60(T)	—	—
5	4	5≤4(F)	—	—	—

A[low] = A[ub]

A[low] = A[4] .

A[ub] = pivot.

A[4] = 60

return ub

return 4 i.e

1	2	3	4
55	45	50	60
pivot			

lb	ub	lb≤ub	A[lb] ≤ pivot	A[ub] ≥ pivot	lb<ub
1	4	1≤4(T)	55≤55(T)	60≥55(T)	—
2	43	2≤3(T)	45≤55(T)	—	—
3	43	3≤3(T)	50≤55(T)	—	—
4	43	4≤3(F)	60≤55(F)	60≥55(T)	—
4	32	4≤2(F)	—	—	—

A[low] = A[ub]

A[low] = A[3]

A[ub] = pivot

A[3] = 55

return ub

return 3 i.e

1	2	3	4
50	45	55	60
pivot			

lb	ub	lb ≤ ub	A[lb] ≤ pivot	A[ub] ≥ pivot	lb < ub.
1	4 2	1 ≤ 2 (T)	50 ≤ 50 (T)	—	—
2	4 2	2 ≤ 2 (T)	45 ≤ 50 (T)	—	—
3	4 2	3 ≤ 2 (F)	55 ≤ 50 (F)	60 ≥ 50 (T)	—
4 3	3	3 ≤ 3 (T)	55 ≤ 50 (F)	55 ≥ 50 (T)	—
3	2	3 ≤ 2 (F)	—	—	—

A[low] = A[ub]

A[low] = A[2]

A[ub] = pivot

A[2] = 50

return ub

return 2 i.e.

∴ List 1 elements are sorted

1	2	3	4
45	50	55	60

Now consider list 2 elements i.e

6	7	8	9
85	80	~~75~~	~~70~~

lb = 6, ub = 9, pivot = 85.

6	9	6 ≤ 9 (T)	85 ≤ 85 (T)	—	—
7	9	7 ≤ 9 (T)	80 ≤ 85 (T)	—	—
8	9	8 ≤ 9 (T)	75 ≤ 85 (T)	—	—
9	9	9 ≤ 9 (T)	70 ≤ 85 (T)	—	—
10	9	10 ≤ 9 (F)			

a[low] = a[9]

a[9] = 85

return 9;

6	7	8	
70	80	75	85

lb=6, ub=8, pivot = 70.

6	8	6≤8 (T)	70≤70 (T)	—	—
7	8	7≤8 (T)	80≤70 (F)	75≥70 (T)	—
7	7	7≤7 (T)			
7	6	7≤6 (F)	80≤70 (F)	80≥70 (T)	

a[low] = a[6]

a[6] = 70

return 6;

6	7	8
70	80	75

lb = 7, ub = 8, pivot = 80

7	8	7≤8 (T)	80≤80 (T)	—	—
8	8	8≤8 (T)	75≤80 (T)	—	—
9	8	9≤8 (F)			

a[low] = a[8]

a[8] = 80

return 8;

6	7
75	80

∴ The list-2 elements are sorted.

∴ The sorted list is

1	2	3	4	5	6	7	8	9
45	50	55	60	65	70	75	80	85

⇒ Time complexity of Quick sort :-

(i) Worst case :-

$$\text{Let } T(n) = \begin{cases} a_T, & n=1 \\ T(n-1)+n, & n>1 \end{cases}$$

Now,

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

Put $n=n-1$ we get

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

Sub $T(n-1)$ in eq (1)

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

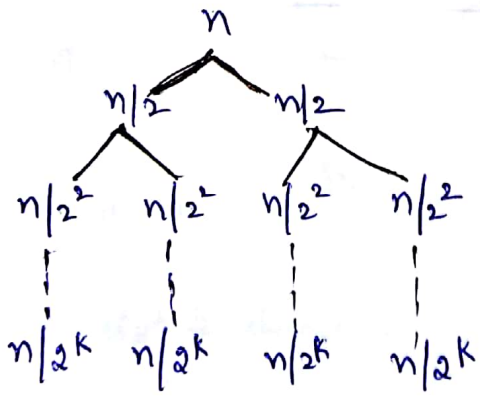
$$= T(n-3)+(n-2)+(n-1)+n$$

$$\vdots$$

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

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

(ii) Best case and average case :-



∴ The recurrance relation is

$$T(n) = \begin{cases} 2T(n/2)+n, & n>1 \\ a, & n=0 \end{cases}$$

Let $T(n) = 2T(n/2) + n$ — (1)

Apply masters method.

$T(n) = aT(n/b) + F(n)$

$a = 2$
$b = 2$, $d = 1$

$a = b^d$

$2 = 2^1$ (condition is true)

$= \Theta(n^d \log n)$

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

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

(or)

Let $T(n) = 2T(n/2) + n$ — (1)

put $n = n/2$ in eq (1)

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

Sub $T(n/2)$ in eq (1)

$T(n) = 2\left[2 \times T(n/2^2) + (n/2)\right] + n$

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

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

⋮

$\boxed{T(n) = 2^k + (n/2^k) + kn}$

put $n = 2^k$ i.e, $k = \log_2^n$, $T(1) = 1$

i.e $T(n) = nT(2^k/2^k) + \log_2^n \cdot n$

$= nT(1) + \log_2^n \cdot n$

$= n(1) + \log_2^n \cdot n$

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

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

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

⇒ Strassen's matrix multiplication :-

As in general we take two matrices A and B whose size is $m \times n$. The product of two matrice is for i to n

```
{
  for j to n
  {
    c[i,j] := 0
    for k to n
    c[i,j] = c[i,j] + A[i,k] *
                      B[k,j]
  }}
```

By using divide and conquer strategy. we can divide the size of matrix and then perform the multiplication operation and their product as shown in below.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

Here, for multiplying two matrices whose size 2x2 requires

8 multiplication and 4 addition operations. Therefore, the general time complexity of these matrix multiplication is

$$\boxed{T(n) = 8T(n/2) + 4n^2}$$

To reduce matrix multiplication and time complexity we go for "Strassen formulas".

i.e $P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = (A_{21} + A_{22})B_{11}$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12})B_{22}$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

∴ The recurrance relation is $\boxed{T(n) = 7T(n/2) + cn^2}$

⟹ Find matrix multiplication by using Strassen's formula.

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

Sol :- Given that

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Where,

$A_{11}=1 \quad B_{11}=0 \quad C_{11}=?$

$A_{12}=1 \quad B_{12}=0 \quad C_{12}=?$

$A_{21}=1 \quad B_{21}=1 \quad C_{21}=?$

$A_{22}=1 \quad B_{22}=1 \quad C_{22}=?$

$P=(A_{11}+A_{22})(B_{11}+B_{22})$

$=(1+1)(0+1)$

$=2(1)=2$

$\boxed{P=2}$

$Q=(A_{21}+A_{22})B_{11}$

$=(1+1)0$

$\boxed{Q=0}$

$R=A_{11}(B_{12}-B_{22})$

$=1(0-1)$

$=1(-1)$

$\boxed{R=-1}$

$S=A_{22}(B_{21}-B_{11})$

$=1(1-0)$

$\boxed{S=1}$

$T=(A_{11}+A_{12})B_{22}$

$=(1+1)1$

$\boxed{T=2}$

$U=(A_{21}-A_{11})(B_{11}+B_{12})$

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

$\boxed{U=0}$

$V=(A_{12}-A_{22})(B_{21}+B_{22})$

$=(1-1)(1+1)$

$\boxed{V=0}$

$C_{11}=P+S-T+V=2+1-2+0=1$

$C_{12}=R+T=-1+2=1$

$C_{21} = Q + S = 0 + 1 = 1$

$C_{22} = P + R - Q + U = 2 - 1 - 0 + 0$

$= 1$

$$\therefore \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Imp: Find product of the following matrices of size 4×4 by using Strassen's matrix multiplication method.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{bmatrix}, \quad B = \begin{bmatrix} 8 & 9 & 1 & 2 \\ 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \end{bmatrix}$$

Sol:- The given matrices of ordes 4×4

Hence we will divide the problem into 2×2 submatrices i.e

$$AB = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} & \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix} \\ \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \\ \begin{bmatrix} 7 & 8 \\ 2 & 3 \end{bmatrix} & \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} \end{bmatrix}$$

Where, $A_{11} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}$ $B_{11} = \begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix}$

$A_{12} = \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix}$ $B_{12} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}$

$A_{21} = \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}$ $B_{21} = \begin{bmatrix} 7 & 8 \\ 2 & 3 \end{bmatrix}$

$A_{22} = \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}$ $B_{22} = \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}$

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$= \left[\begin{bmatrix}1 & 2\\ 5 & 6\end{bmatrix} + \begin{bmatrix}3 & 4\\ 7 & 8\end{bmatrix}\right]\left[\begin{bmatrix}8 & 9\\ 3 & 4\end{bmatrix} + \begin{bmatrix}1 & 2\\ 5 & 6\end{bmatrix}\right]$$

$$= \begin{bmatrix}4 & 6\\ 12 & 14\end{bmatrix}\begin{bmatrix}9 & 11\\ 8 & 9\end{bmatrix}$$

$$Q = (A_{21} + A_{22})B_{11}$$

$$= \left[\begin{bmatrix}9 & 1\\ 4 & 5\end{bmatrix} + \begin{bmatrix}2 & 3\\ 6 & 7\end{bmatrix}\right]\begin{bmatrix}8 & 9\\ 3 & 4\end{bmatrix}$$

$$= \begin{bmatrix}11 & 4\\ 10 & 12\end{bmatrix}\begin{bmatrix}8 & 9\\ 3 & 4\end{bmatrix}$$

$$= \begin{bmatrix}100 & 115\\ 116 & 138\end{bmatrix}$$

$$R = A_{11}(B_{12} - B_{22})$$

$$= \begin{bmatrix}1 & 2\\ 5 & 6\end{bmatrix}\left[\begin{bmatrix}1 & 2\\ 5 & 6\end{bmatrix} - \begin{bmatrix}9 & 1\\ 4 & 5\end{bmatrix}\right]$$

$$= \begin{bmatrix}1 & 2\\ 5 & 6\end{bmatrix}\begin{bmatrix}-8 & 1\\ 1 & 1\end{bmatrix}$$

$$= \begin{bmatrix}-6 & 3\\ -34 & 11\end{bmatrix}$$

$$S = A_{22}(B_{21} - B_{11})$$

$$= \begin{bmatrix}2 & 3\\ 6 & 7\end{bmatrix}\left[\begin{bmatrix}7 & 8\\ 2 & 3\end{bmatrix} - \begin{bmatrix}8 & 9\\ 3 & 4\end{bmatrix}\right]$$

$$= \begin{bmatrix}2 & 3\\ 6 & 7\end{bmatrix}\begin{bmatrix}-1 & -1\\ -1 & -1\end{bmatrix} = \begin{bmatrix}-5 & -5\\ -13 & -13\end{bmatrix}$$

$$T = (A_{11} + A_{12})B_{22}$$

$$= \left[\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix} \right] \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 6 \\ 12 & 14 \end{bmatrix} \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 60 & 34 \\ 164 & 82 \end{bmatrix}$$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$$= \left[\begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} - \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \right] \left[\begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 8 & -1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 9 & 11 \\ 8 & 10 \end{bmatrix} = \begin{bmatrix} 64 & 78 \\ -17 & -21 \end{bmatrix}$$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

$$= \left[\begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix} - \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \right] \left[\begin{bmatrix} 7 & 8 \\ 2 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 16 & 9 \\ 6 & 8 \end{bmatrix} = \begin{bmatrix} 22 & 17 \\ 22 & 17 \end{bmatrix}$$

$C_{11} = P + S - T + V$

$$= \begin{bmatrix} 86 & 75 \\ 278 & 227 \end{bmatrix} + \begin{bmatrix} -5 & -5 \\ -13 & -13 \end{bmatrix} - \begin{bmatrix} 60 & 34 \\ 164 & 82 \end{bmatrix} + \begin{bmatrix} 22 & 17 \\ 22 & 17 \end{bmatrix}$$

$$= \begin{bmatrix} 45 & 53 \\ 123 & 149 \end{bmatrix}$$

$C_{12} = R + T$

$$= \begin{bmatrix} -6 & 3 \\ -34 & 11 \end{bmatrix} + \begin{bmatrix} 60 & 34 \\ 164 & 82 \end{bmatrix} = \begin{bmatrix} 54 & 37 \\ 130 & 93 \end{bmatrix}$$

$C_{21} = Q + S$

$$= \begin{bmatrix} 100 & 115 \\ 116 & 138 \end{bmatrix} + \begin{bmatrix} -5 & -5 \\ -13 & -13 \end{bmatrix}$$

$$= \begin{bmatrix} 95 & 110 \\ 103 & 125 \end{bmatrix}$$

$C_{22} = P + R - Q + O$

$$= \begin{bmatrix} 86 & 75 \\ 278 & 227 \end{bmatrix} + \begin{bmatrix} -6 & 3 \\ -34 & 11 \end{bmatrix} - \begin{bmatrix} 100 & 115 \\ 116 & 138 \end{bmatrix} + \begin{bmatrix} 64 & 78 \\ -17 & -21 \end{bmatrix}$$

$$= \begin{bmatrix} 44 & 41 \\ 111 & 79 \end{bmatrix}$$

⇒ Time complexity of strassen matrix :-

$T(n) = 7T(n/2) + cn^2$ ——— (1)

put $n = n/2$ in eq (1) we get

$T(n/2) = 7T(n/2^2) + c(n/2)^2$

Sub $T(n/2)$ in eq (1) we get

$T(n) = 7\left[7T(n/2^2) + c(n/2)^2\right] + cn^2$

$= 7^2 T(n/2^2) + (7/4)cn^2 + cn^2$

$= 7^3 T(n/2^3) + (7/4)^2 cn^2 + \left(\frac{7}{4}\right)cn^2 + cn^2$

$= 7^4 T(n/2^4) + \left[\left(\frac{7}{4}\right)^3 cn^2 + \left(\frac{7}{4}\right)^2 cn^2 + \left(\frac{7}{4}\right)cn^2 + cn^2\right]$

$\therefore \boxed{T(n) = 7^k T(n/2^k) + (7/4)^k \cdot cn^2}$

put $n = 2^k$ i.e $k = \log_2^n$

$T(n) = 7^{\log_2^n} T(2^k/2^k) + (7/4)^{\log_2^n} \cdot cn^2$

$$= 7^{\log_2^n} \cdot T(1) + \frac{7^{\log_2^n}}{4^{\log_2^n}} \cdot cn^2$$

since $\boxed{a^{\log_c^b} = b^{\log_c^a}}$

$$T(n) = n^{\log_2^7} + \frac{n^{\log_2^7}}{n^2} \times cn^2$$

$$T(n) = n^{\log_2^7} [1+c]$$

$$T(n) = O\left(n^{\log_2^7}\right)$$

$$T(n) = O(n^{2.80})$$

$$\therefore \boxed{T(n) = O\left(n^{\log_2^7}\right) = O(n^{2.80})}$$

[Recursion ni Solve chey]

Mr Y. Subba Rayudu
B.Tech, M.Tech, MISTE
(ph.D)