MODULE-1

Introduction :- BASIC CONCEPTS

1) Structure & union
2) Introduction to data structure and its classification
3) The need for data structure
4) Algorithm specification
5) performance analysis & Measurements
6) polynomials
7) sparse matrices

NOTES:- Data Structure with Applications
Shalini. K. B
Assistant professor
ISE, DSCE

1) Structure and union;->

→ Arrays are collection of data of the same type. Data of different types can be grouped together using structure.

→ Arrays can be used to represent a group of data items that belong to the same type such as int & float.

→ Structures can be used to represent a collection of data items of different types using single name.

"Definition";-> Structure is a collection of logically related elements of similar/dissimilar data types having the single name.

"Declaration of structure";->

It is declared using a keyword struct followed by the name of the structure. The variables of the structure are declared within the structure

"Syntax";- struct struct_name

{

datatype var_name;

datatype var_name;

⋮ ⋮

};

Struct is the keyword. struct_name is the name of the structure and varname is the field of the structure

example:-
```
struct student
{
 int id;
 char name[10];
 int gradepoints;
};
```

There are three different ways to declare the structure

1) variable structure
2) Tagged structure
3) Typedef structure

1) variable structure:- It consists of keyword struct followed by a field list and an variables

Syntax:-
```
struct tag-name
{
field list
} variablename;
```

Example:-
```
struct student
{
 char idname[10];
 int id;
 int marks;
} s1;
```

Note;- It creates a structure that can be used for only one variable definition

b) Tagged structure; →

By giving the structure a tag-name we can use it to define variables (more than one) parameter and return type

Syntax;- Struct tagname
{
datatype memb1;
datatype memb2;
:
:
};
Struct tagname varname;

Example;-

```
Struct student
{
char name[10];
int id;
int marks;
} s1, s2;
```

(or)

```
Struct student
{
char name[10];
int id;
int marks;
};
Struct student s1, s2;
```

c) Type def structure; →

We can create one own structure data types using the typedef statement as follows

Syntax:- typedef struct
{
datatype membl;
datatype membr;
} NEWTYPE;
NEWTYPE variables;
↓
User defined type

Example:- typedef struct
{
char name[10];
int id;
int marks;
} STUDENT
STUDENT S1, S2;

"Accessing structures"

The members of the structure can be assigned values in member of ways. The members themselves are not variables. They should be linked to the structure variables in order to make the meaningful members using dot operator

example:- S1.id;
S1.name;
S1.marks;

values to the members can be assigned using strcpy (S1.name, "Harshal");

S1.id = 1
S1.marks = 95

scanf can also be used
scanf ("%s", S1.name);
scanf ("%s", S1.id);
scanf ("%s", S1.marks);

Write a program to student details [At least 5 record] and the print the student details; [NOTE;- Refer class notes] :-

Nested structure;->

```
#include<stdio.h>
 Struct student
   {
    int rno;
    float marks;
     Struct date;
     {
      int d, m, y;
     }dob, doadm, doexam;

  };
 void main()
   {
     Struct student- s;
     printf("\n Enter roll no & marks\n");
     scanf("%d %f", &s.rno, &s.marks);
     printf("Enter the date of birth\n");
      scanf("%d %d %d", &s.dob.d,
            &s.dob.m, &s.dob.y);
     printf("Enter date of admission\n");
     scanf("%d %d %d", &s.doadm.d,
            &s.doadm.m, &s.doadm.y);
     printf("Enter date of examination\n");
```

```
scanf("%d %d %d", &s.doexam.d, &s.doexam.m,
                  &s.doexam.y);
printf("The student details are \n");
printf("\n Rollno = %d", s.rno);
printf("\n marks = %f", s.marks);
printf("\n Date of birth = %d - %d - %d",
        s.dob.d, s.dob.m, s.dob.y);
printf("\n Date of admission = %d - %d - %d",
        s.doadm.d, s.doadm.m, s.doadm.y);
printf("\n Date of exam = %d - %d - %d",
        s.doexam.d, s.doexam.m, s.doexam.y);
getch();
}
```

Syntax:-

```
struct structure_name
{
data membl;
data membz;
struct structurename
{
data membl;
data membz;
}variable list
};
```

Self-Referential Structures:

Self-referential structure is one in which one or more of its components is a pointer to itself.

```
typedef struct
{
    int data;
    struct list *link;
} list;
```

Each instance of the structure list have two components, data and link. Data is a single character which link is a pointer to a list structure. The value of link is either the address in memory of an instance (variable) of list.

program:- Define the structure for the employee write the following fields

Emp Id (integer), emp-name (string), Emp-sal (float), empdept-(string) Empage (integer) write the following functions to process employee data

1) function to read an Employee record
2) function to print an employee record

```
Struct employe
{
 int emp_id;
 chas empname[20];
 float empsal;
 chas empdept[10];
  int empage;
};
void main()
{
Struct employe e[10];
printf("Enter no of employee records");
 scanf("%d",&n);
 readempdata();
  printempdata();
}
readdata()
{
 printf("Enter the details of employee");
  for(i=1;i<=n;i++)

{
 printf("Enter details of emplo");
  scanf("%d%s%f%s%d", &e[i].emp_id,
  e[i].empname, e[i].empbasic, e[i].empdepts
      e[i].empage);
  }
}
```

```
printempdata()
{
for(i=1;i<=n;i++)
  {
  printf("The employe are");
  printf("%d %s %f %s %d", e[i].empid, e[i].
    empname, e[i].empsal, e[i].empdate,
      e[i].empage);
  }
}
```

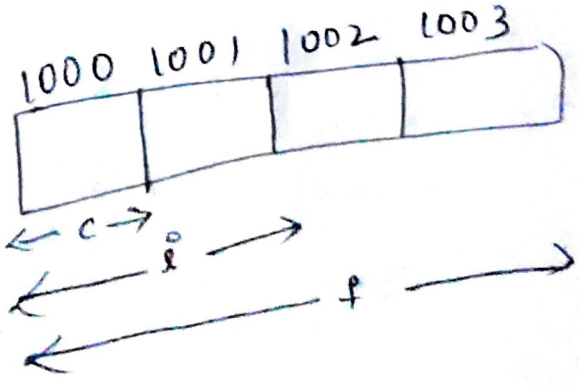
"UNION"- union delaration is similar to an structure but the fields of a union must share their memory space. This means that only one field of the union is active at any given time

Syntax:- union tagname
{
datatype member1;
datatype member2;
.
.
};

(OR)

typedef union
{
datatype member1;
datatype member2;
.
.
};

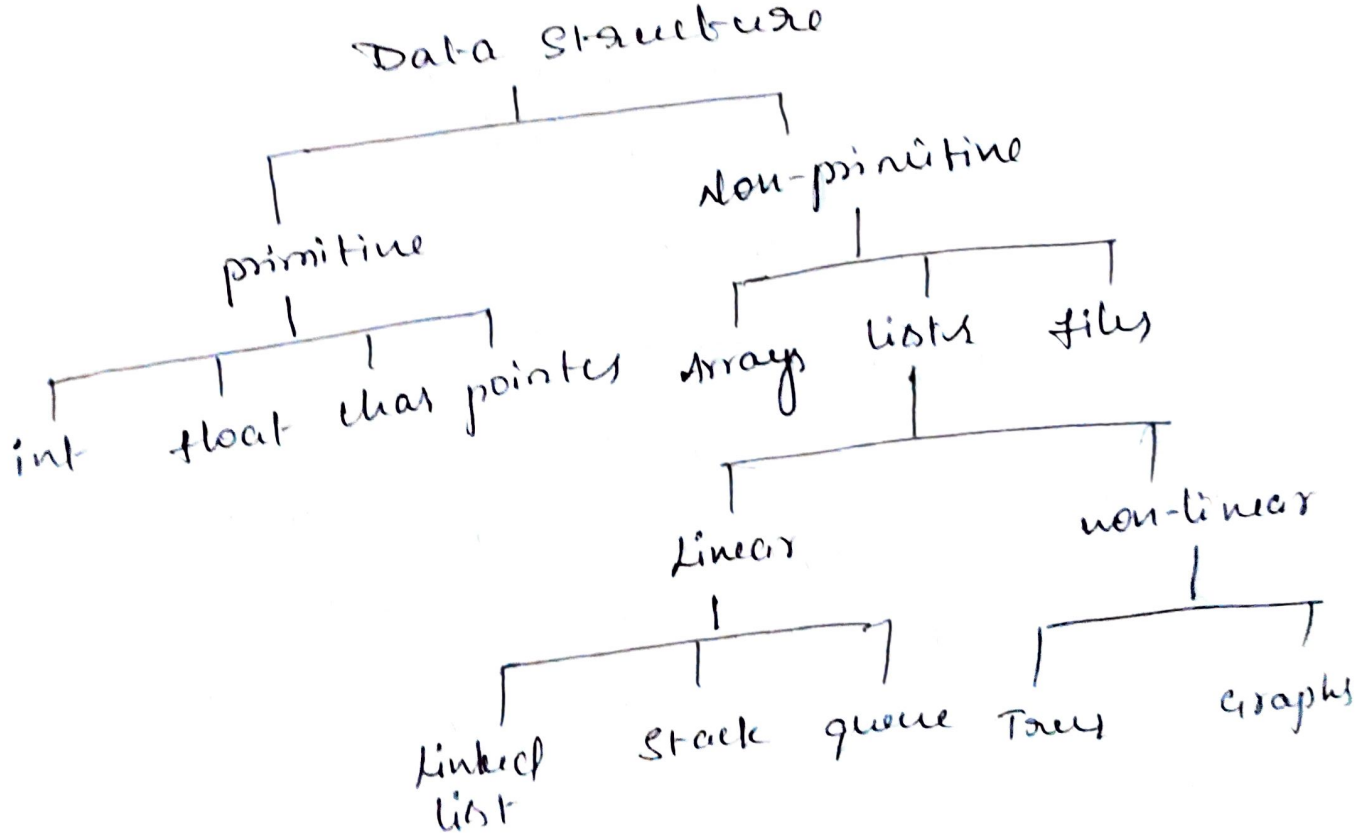
Example:- typedef union
{
char c;
int i;
float f;
} DATA;



2] "Introduction to Data structure and its classifications"

"Definition" :- Data structure deals with the study of how the data is organized in the memory, how efficiently the data can be retrieved and manipulated and possible ways in which different items are logically related

classification of data structure



primitive data structure's

These are the data structure that can be manipulated (operated) directly by machine instructions.

Eg:- int, float, char and double

Non-primitive data structures

These are the data structures that cannot be manipulated directly by machine instructions

Non-primitive data structures are classified into two types

* linear data structure
* Non-linear data structure

The data structures that shows the relationship of logically adjacency between the elements are called linear data structure

Ex:- Stacks, queue, singly linked list, doubly linked list.

Non-linear data structure

Ex:- Trees, graphs, files

The need for data structures

1) The data structure organize data
   * which gives more efficient programs
2) The choice of data structure and algorithm can make the difference between a program running in a few seconds (or) many days
3) A solution is said to be efficient if it solves the problem within the resource constraints.

   i.e space and time

3] Algorithm specification :-

Definition :- An algorithm is a finite set of instructions that is followed to accomplish particular task. The algorithm must satisfy the following criteria

Input :- There are zero (or) more quantities that are externally supplied

output :- Atleast one quantity is produced

Definiteness :- Each instruction is clear and unambiguous

Finiteness :- Algorithm should terminate after a finite no of steps

Effectiveness :- It should be definite & feasible

Algorithm representation :-

1) using natural language
2) graphical represent-ation like flowchart
3) combination of english & C

Example :-

Selection Sort :- From those integers are currently unsolved. find the smallest and place it next in the sorted list

```
for (i=1; i<=n; i++)
{
```

examine List[i] to List[n]
and suppose that the smallest integer
is at list[min];
Interchange list[i] and list[min];

}

Ex;-

	1	2	3	4	5
	(20)	40	10	30	5
		i=2	3 min	4	5
	5	(40)	10	30	20
			i 3	4	5 min
	5	10	(40)	30	20
				i=min 3	4
	5	10	20	(30)	40
	5	10	20	30	40

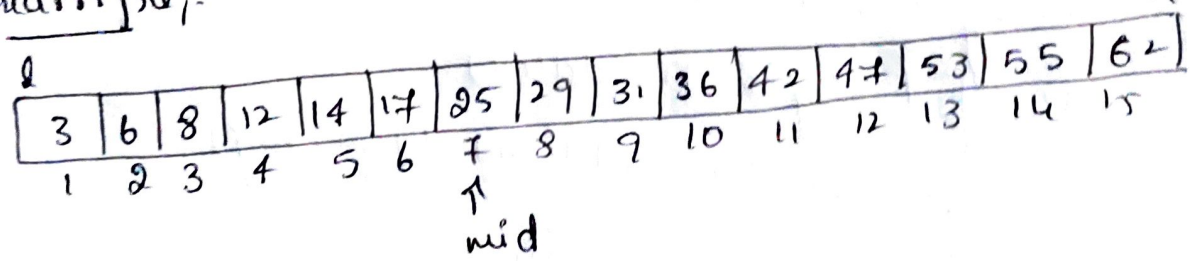
Swap function;-

```
main()
 {
  int a, b;
  a=10, b=20;
   swap(&a,&b)
    }
swap(int *x, int *y)
      {
        int temp;
        temp *x;
        *x = *y;
         *y= temp;
       }
```

Binary Search [iterative method];-

```
while (there are more integers to check)
{
    middle = (left + right)/2;
    if (searchnum == list[middle])
        return middle
    if (searchnum == list[middle])
        right = middle - 1
    else
        left = middle + 1
}
```

Example;-



key = 42

$mid = \left[\frac{l+h}{2}\right]$

In the 11th position we found the key element.

$$\frac{l \quad h}{1 \quad 15} = \frac{1+15}{2} = 8$$

$$9 \quad 15 = \frac{9+15}{2} = 12$$

$$9 \quad 11 = \frac{9+11}{2} = 10$$

Recursive algorithm:-

* Function that call themselves (direct recursion)
or they may call other function that invokes the calling function again (indirect recursion)

Recursive implementation of binary search

```
Binarysearch (int a[], int low, int high, int key)
{
  If (low < high)
  return -1;
  middle = (low + high) /2;
  If (key < a[middle])
  Binarysearch (a, low, middle-1, key)
  else if (key > a[middle])
  Binarysearch (a, middle+1, high, key)
  elseif (key == a[middle])
  return(middle);
}
```

Definition:- Recursion is a method of solving the problem where the solution to a problem depends on solution to smaller instance of the same problem.

That call itself during execution

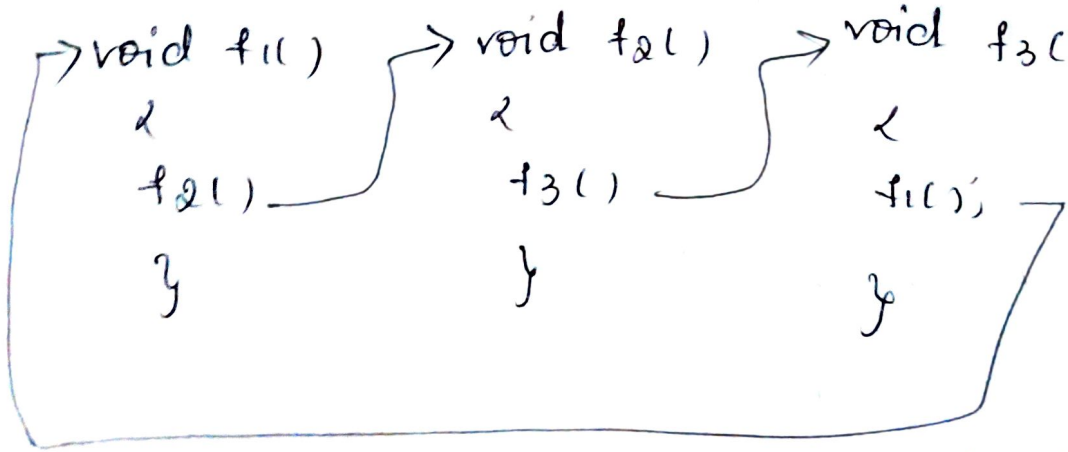
The various types of recursion

1) Direct recursion
2) Indirect recursion

Direct recursion:- A recursive function that invokes itself is said to have direct-recursion

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

Indirect recursion:- A function which contains a call to another function which intern calls another function and eventually calls the first function is said to be called indirect recursion



Note:- 1) Recursive function must terminate condition

2) Recursive function should never generate infinite sequence of calls itself

factorial of number:-

```
int fact(n)
{
if n==0 return 1
 else
return n*fact(n-1);
}
void main()
{
  int n;
  printf("Enter n");
  scanf("%d",&n);
    r=fact(n);
    printf("%d",r);
}
```

o/p:-

1! = 1*0! = 1*1 = 1

2! = 2*1! = 2*1 = 2

3! = 3*2! = 3*2 = 6

4! = 4*3! = 4*6 = 24

5! = 5*4! = 5*24 = 120

comparsion of two integers:-

```
int compare (int x, int y)
{
  If (x==y) return 0;
  else if (x<y) return -1
   else
   return 1;
}
```

4] "performance analysis" :-

1] Does the program meet the original specifications of the task.

2] Does it work correctly?

3] Does the program contain documentation that shows how to use it and how it works

4] Does the program efficiently use functions to create logical units

5] Is the program's code readable

6] Does the program efficiently use primary & secondary storage

7] Is the programs running time acceptable for the task

The first field we focus on obtaining estimates of time & space that machine independent. this field is called performance analysis

The second field which we call performance measurement. obtains - machine dependent- running time. these times are used to identify inefficient code segment-

1) space complexity
2) Time complexity

"Space complexity" - The space complexity of a program is the amount of memory that it needs to run to completion of program.

"Time complexity" is the time complexity of a program is the amount of computer time that it needs to run to completion program.

Two types of space requirements

→ fixed space requirements

→ variable space requirements

→ fixed space requirements is Refers to space requirements that do not depend on the no and size of the program input/output.

* Instruction space
* space for simple variable
* fixed-size structure
* constants

→ variable space requirements is this component consists of the space needed by structured variable whose size depends on the particular instance 'I' of the problem being solved.

→ Time complexity is Time taken by program P the sum of its compile & runtime compile time is similar to the fixed space component.

Asymptotic Notations:→

Asymptotic Notations which provides upper bound and/or lower bound on the value of f for suitably large values of n.

Definition:- Asymptotic notations which provides upper bound / lower bound of a given function.

"The asymptotic notation is the study of how the value of a function varies for large value of n where n is the size of the input using the asymptotic behaviour of a function", we can find the time efficiency of an algorithm.

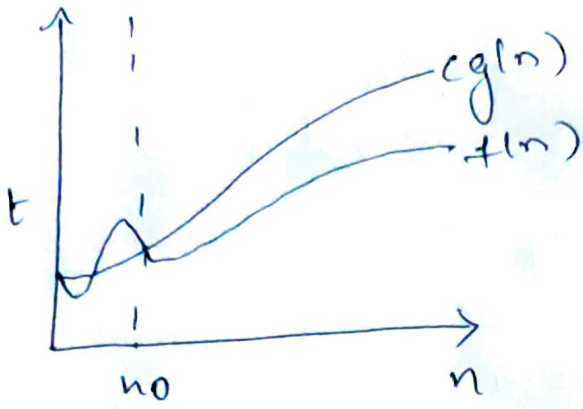
- Asymptotic notations
  - O (Big oh)
  - Ω (Big omega)
  - Θ (Big theta)

Big oh (O):- Big (O) is a measure of longest amount of time taken by the algorithm to complete the execution.

$$f(n) \leq c \cdot g(n) \quad \text{for all } n \geq n_0$$

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

such that there is a positive constant c and positive integer n_0



The upper bound on f(n) indicates that function f(n) will not consume more than c.g(n) so we can say that f(n) is faster than g(n).

↓

Means g(n) is slower than f(n)

↓

Means g(n) is takes more time to execute

Examples:-

① $f(n) = 100n + 5$

$g(n) = f(n)$

$g(n) = 100^{n} + 1$ [replace n]

$g(n) = 101n$

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

$100n + 5 \leq c \cdot 101n$

$n = 5$

$100(5) + 5 \leq 101 \cdot 5$

$505 \leq 505$

② $f(n) = 3n + 2$ $g(n) = n$

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

$3n + 2 \leq c \cdot n$ $c = 3$

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

$3(4) + 2 \leq 4(4)$

$12 + 2 \leq 16$

$14 < 16$

f(n) is less time

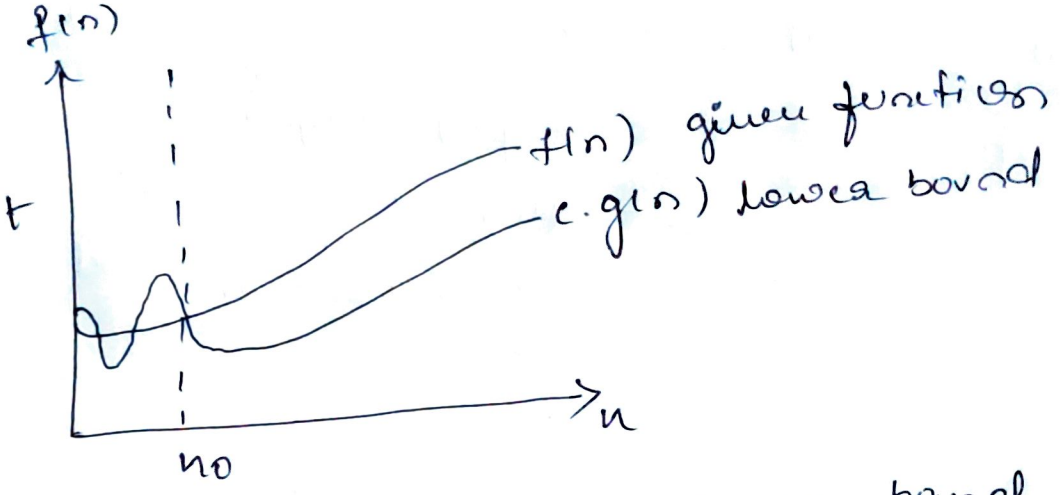
g(n) is more time

Ω (Big - omega) :→

Big(Ω) is a measure of the least amount of time taken by the algorithm to complete execution.

$f(n) \geq c.g(n)$ for all $n \geq n_0$

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



This notation gives the lower bound on a function $f(n)$ within a constant factor.

The lower bound $f(n)$ indicates that $f(n)$ function will consume atleast the specified time $c.g(n)$

Let $f(n) = 100n + 5$ express $f(n)$ using big omega.

$g(n) = f(n)$

$= 100n + 5$ [replace with 0)

$g(n) = 100n$

$f(n) \geq c.g(n)$ $n \geq n_0$

$100n + 5 \geq 100n$

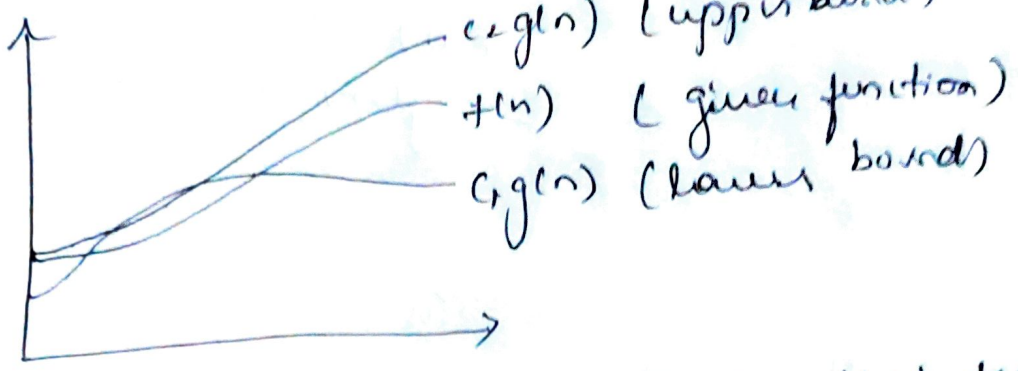
Θ (Big-Theta) :→

Definition :→ Big Θ is a measure of the least- as well as longest amount of time taken by the algorithm to complete

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

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for } n \geq n_0$$

such that there is exists some positive constants c_1 and c_2 and non-negative integer n_0



upper bound on $f(n)$ indicates that function $f(n)$ will not consume more than specified time $c_2 g(n)$ the lower bound on $f(n)$ indicates that the function $f(n)$ in the best case will consume atleast specified time $c_1 g(n)$.

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

$$100 * n \leq 100n + 5 \leq 105 * n \quad n \geq n_0, \; n \geq 1$$

[NOTE :→ Along with the above example refer class notes also for more information].

"Performance Measurement"

#include <time.h>. The functions we need to time events are part of c's standard library and are accessed through the statement-

	Method 1	Method 2
Start timing	Start = clock();	Start = time(NULL);
Stop timing	Stop = clock();	Stop = time(NULL)
Type returned	clock_t()	time_t()
Result in seconds	duration = ((double)(Stop - Start)/ CLOCKS_PER_SEC);	duration = (double) difftime (Stop, Start);

```
#include<stdio.h>
#include<time.h>
#include<conio.h>
#include<stdlib.h>
#define MAX_SIZE 30000
#define NTIMES 5000
void sort(int a[], int n)
{
  int i, j, s;
  for(i=0; i<n-1; i++)
  {
    for(j=0; j<n-i-1; j++)
    {
      if(a[j] > a[j+1])
      {
        s=a[j];
        a[j]=a[j+1];
```

```
        a[j+1]= s
    }
}
void main()
{
  int a[MAX_SIZE], i,n,t;
  clock_t start, end;
  double runtime;
  clrscr();
  printf("Enter the Size of array \n");
  scanf("%d",&n);
  printf("ntimes");
  for(t=0; t<=NTIMES; t++)
    {
      srand();
      for(i=0; i<n;i++)
        a[i]= rand();
        start = clock();
        SORT(a,n);
        end = clock();
        runtime = runtime + ((end-start)/
                             CLK_TCK);
    }
  runtime = runtime/NTIMES;
  printf("The sorted array is");
  for(i=0;i<n;i++)
  printf("%d\n",a[i]);
  printf("Time taken for sorting is %lf seconds",
                        runtime);
  getch();
}
```

Sparse matrix:

Sparse matrix is a matrix that has very few non zero elements spread out convensibly a matrix which has more no of zero element than very few non-zero element

Example:-

	0	1	2	3
0	10	0	0	40
1	11	0	22	0
2	20	0	50	0
3	0	15	0	25

Syntax:-

```
#define MAX_TERMS 101
typedef struct
{
 int row;
 int col;
 int val;
} Term;
Term a[MAX_TERM]
```

To represent sparse matrix in triplet format

	row	col	val
a[0]	5	4	8
a[1]	0	0	10
a[2]	0	3	40
a[3]	1	0	11
a[4]	1	2	22
a[5]	3	0	20
a[6]	3	3	50
a[7]	4	1	15
a[8]	4	3	25
a[A1]			

[NOTE: The code for the above refer class Notes]

[To read and print the sparse matrix and also search element from the sparse matrix refer lab program].

Disadvantage: The disadvantage of sparse matrix is a sparse matrix contains many zeros. If we are manipulating only non-zero value then we are wasting the Memory space by storing unnecessary zero values

Transpose of a Matrix: Transpose is obtained by changing row element to a column and column to row

Example:- The same above given matrix

	row	col	val
a[0]	5	4	8
a[1]	0	0	10
a[2]	0	3	40
a[3]	1	0	11
a[4]	1	2	22
a[5]	3	0	20
a[6]	3	3	50
a[7]	4	1	15
a[8]	4	3	25

Transpose of a matrix

	row	col	val
b[0]	4	5	8
b[1]	0	0	10
b[2]	0	1	11
b[3]	0	3	20
b[4]	1	4	15
b[5]	2	1	22
b[6]	3	0	40
b[7]	3	3	50
b[8]	3	4	25

```
void transpose (term a[], term b[])
{
 int i, j, k;
 b[0].row = a[0].col;
 b[0].col = a[0].row;
 b[0].val = a[0].val;
  k = 1;
for (i=0; i<a[0].col; i++)
{
for (j=1; j<=a[0].val; j++)
 {
  If (a[j].col == i)
  {
   b[k].row = a[j].col;
   b[k].col = a[j].row;
   b[k].val = a[j].val;
    k++;
   }
  }
 }
```

[NOTE:- For tracing part refer class notes]

ADT of sparse matrix;

Definition;→ ADT sparse matrix is the one that follows the various operation that can be performed on sparse matrix. It consists of various objects (variable)

object-i→ A set of triples <row, column, value> whose row and column are integer and form a unique combination, and value comes from the set item.

functions;-

for all a, b ∈ sparse matrix, x ∈ item, i, j, maxcol, maxRow ∈ index

sparsematrix create (maxRow, maxcol) ::=
→ return a sparsematrix that can hold up to maxItems = maxRow x maxcol and whose maximum row size is maxRow and whose maximum col size is maxcol

Sparsematrix Transpose (a) ::=
→ return the matrix produced by interchanging the row and column value of every triple

Sparsematrix Add (a, b) ::=
→ If the dimensions of a & b are the same return the matrix produced by adding corresponding items namely those with identical row and column value else return error

Sparse matrix (a,b) ::=

If no of columns in a equals no of rows in b return matrix d produced by multiplying a by b according to the formula d[i][j] = Σ(a[i][k] * b[k][j]) where d(i,j) is the (i,j)th element else set error.

Polynomials:- polynomials is the application of ADT If the polynomials is $-10+3x+5x^2$ then we can write it as

$-10x^0 + 3x^1 + 5x^2$

poly is the name of the array

	0	1	2
poly	-10	3	5

int poly[3];

A polynomial of a single variable A(x) can be written as $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ where $a_n \neq 0$ and degree of A(x) is n

	0	1	2	...	n-1	n
poly	a_0	a_1	a_2	...	a_{n-1}	a_n

for a polynomial of degree n, n+1 terms are required

polynomial addition;→

Simple addition

$X_1 = 3x^1 + 5x^2 + 7x^3$

Degree of X_1 is $M = 3$

$X_2 = 10x^0 + 3x^1 + 5x^2$

Degree of X_2 is $N = 2$

* Identify the value of highest degree polynomials
* write polynomial X_3 with degree (max (degree of X_1 and degree of X_2))

$X_1 = 3x^1 + 5x^2 + 7x^3$

$X_1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

i =	0	1	2	3
	0	3	5	7

$X_2 = 10x^0 + 3x^1 + 5x^2$

$X_2 = 10x^0 + 3x^1 + 5x^2 + 0x^3$

j =	0	1	2	3
	10	3	5	0

$X_3 = 10x^0 + 6x^1 + 5x^2 + 7x^3$

	0	1	2	3
	10	6	5	7

Step 1:- $a_0 = 0 + 10 = 10$

```
i = j = k = 0
while (i <= m)
{
C[k] = A[i] + B[j]
i = i++;
j = j++;
k = k++;
```

Representation

```
#define MAX_DEGREE 101
typedef struct {
    int degree;
    float coef[MAX_DEGREE];
} polynomial
```

Now if a is of type polynomial and $n < MAX\_DEGREE$ the polynomial

$$A(x) = \sum_{i=0}^{n} a_i x^i$$

a.degree = n

a.coeff[i] = a_{n-i} , $0 \le i \le n$

ADT - polynomial:-

ADT polynomial is object = $p(x) = a_1 x^{e_1} + \dots + a_n x^{e_n}$ a set of ordered pairs of (e_i, a_i) where a_i is co-efficients and e_i is exponent e_i are integers $>= 0$

functions:- for all poly, poly1, poly2 ∈ polynomial coef ∈ coefficients, expon ∈ exponents

1) polynomial zero() ::= return the polynomial $p(x) = 0$

2) Boolean Iszero (poly) ::= if (poly) return FALSE
else return TRUE

3) coefficient coef (poly, expon) ::= if (expon ∈ poly)
return its co-efficient else
return zero

4) Exponent Leadexp (poly) ::= return the largest
exponent in poly

5) polynomial Attach (poly, coef, Expon) ::=
if (expo ∈ poly) return error else return
the polynomial poly with the term where
with term (coef, expon) inserted

6) polynomial Remove (poly, expon) ::=
if (expon ∈ poly) return the polynomial
poly with the term where exponent is
expon deleted else return error

7) polynomial Singmult (poly, coef, expon) ::=
return the polynomial poly
coef x. expon

8) polynomial Add (poly1, poly2) ::= return the polynomial
poly1 + poly2

9) polynomial mult (poly1, poly2) ::= return the
polynomial poly1, poly2

polynomial addition:→

$x_1 = 7x^4 + 5x^2 + 3x^1$ $\quad x_2 = 5x^3 + 3x^1 + 8x^0$

x_1 =

	i=0		
co-efficient	7	5	3
exponent	4	2	1

x_2 =

	j=0		
co-efficient	5	3	8
exponent	3	1	0

x_3 =

co-efficient	7	5	5	6	-8
exponent	4	3	2	1	0

Exponent $4 > 3$

case1:- If the exponent of term pointed by j in x_2 is less than the exponent of the current term pointed by i of x_1 then copy the current term of x_1 pointed by i in the location pointed by k in polynomial x_3 Advance the pointers i and k to the next term

```
if (x1[i].expo > x2[jj.expo)
{
    x3[k].co-eff = x1[i].coeff;
    x3(k).expo = x1[i].expo;
    i=i++;
}  k=k++;
```

case :- 2 $a < 3$

If the exponent of the term pointed by j in x_2 is greater than the exponent of the current term pointed by i of x_1 then copy of the current term of x_2 pointed by j in the location pointed by k in polynomial x_3. Advance the pointers j and k the next term.

```
if (x1[i].expo < x2[j].expo)
{
  x3[k].coeff = x2[j].coeff;
  x3[k].coeff = x2[j].coeff;
  j = j+1;
  k = k+1;
}
```

case 3:- If the exponents of the two terms of polynomials x_1 and x_2 are equal then the coefficients are added and the new term is stored in the resultant polynomial x_3 and advance i, j and k to track to the next term.

```
... if (x1[i].expo == x2[j].expo)
    {
      x3[k].coeff = x1[i].coeff + x2[j].coeff
      x3[k].expo = x1[i].expo;
       i = i++;
       j = j++;
       k = k++;
    }
```

case 4:- If there is no more elements in x1 and there are few elements remaining in x2 then copy the rest of the elements in x2 to x3 and advance j & k to reach to the next terms

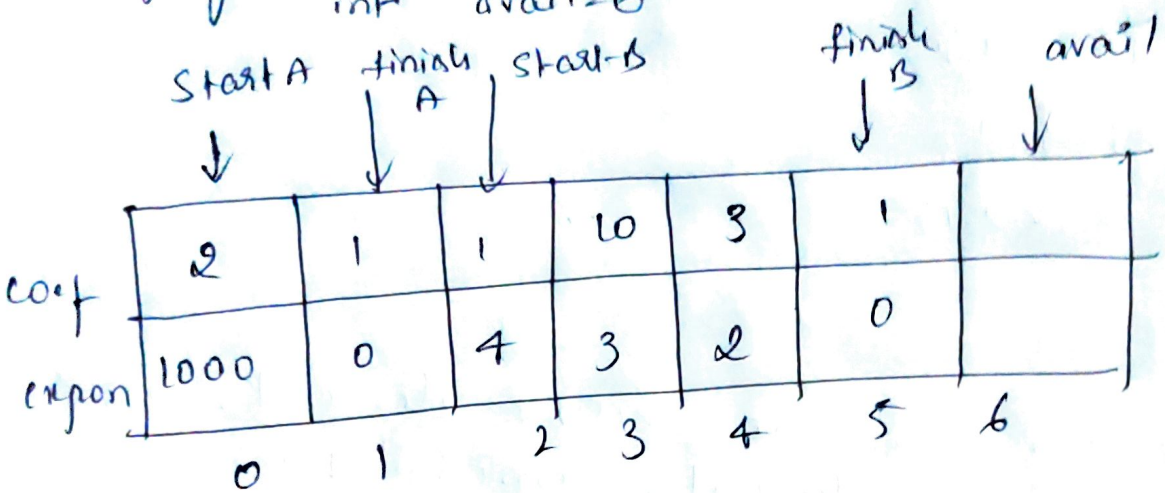
```
while (j<n) do
    {
      x3[k].coeff = x2[j].coeff;
       x3[k].expo = x2[j].expo;
         j = j++;
         k = k++;
    }
```

Array representation of two polynomials is

```
MAX_TERMS 100 /* Size of term array */
  typedef struct
    {
      float coef;
```

```
int expon;
} polynomial;

polynomial terms [MAX-TERMS];
int avail=0
```



$A(x) = 2x^{1000} + 1$

$B(x) = x^4 + 10x^3 + 3x^2 + 1$