09-03-22

DEPARTMENT OF COMPUTER SCIENCE

Insertion of element at given Index

Aim:- Write a program to read 'N' numbers of elements into an array and also perform the following Operations on an Array.

a). Insert an element at given Index of array

b). Delete an existing element

Procedure:-

a). Insertion at given Index (Step's are:)-

1. Get the element value which needs to be inserted.
2. Get the position value.
3. Check whether the position value is valid or not.
4. If it is valid, shift all the elements from the last index to position index by 1 position to the right.
   Insert the new element in arr[position]
5. Otherwise,
   Invalid position.

b). Delete an Existing element:-

1. Find the given element in the given array and note the index.
2. If the element found, shift all the elements from index+1 by 1 position to the left. Reduce the array size by 1.
3. Otherwise, print "Element not Found"

1. Program:-

1a) Program: Inserting an element in the array

```
#include<stdio.h>
#include<conio.h>
#define max 50
void main()
{
    int arr[max],size;
    int element, pos, i;

    printf("Enter Array size:");
    scanf("%d",&size);
    printf("enter %d elements into Array:",size);
    for(i=0;i<size;i++)
    scanf("%d",&arr[i]);

    printf("Enter position and element:\n");
    scanf("%d%d",&pos,&element);

    if(pos <= size && pos > 0)
    {

        for(i = size-1; i >= pos-1; i--)
            arr[i+1] = arr[i];

        arr[pos-1] = element;
        size++;
        printf("Array after new element:\n");
        for(i = 0; i < size; i++)
            printf("%d ", arr[i]);
    }
    else
        printf("Invalid Position\n");

    getch();
}
```

Output:-

```
Enter Array size: 5

Enter 5 elements into Array: 87 55 34 23 12

Enter position and element:

4

99

Array after new element:

87 55 34 99 23 12
```

J. A

DEPARTMENT OF COMPUTER SCIENCE

Deletion of an Existing element

Aim:

Write a program to read 'N' numbers of elements into an array and perform following Operation on An array:

Procedure :-

b). Delete an Existing element :-

1. Find the given element in the given Array and note the Index.
2. If the element found, shift all the elements from index+1 by 1 position to the left. Reduce the array size by 1.
3. Otherwise, print "Element Not found".

1b) Program: Remove an element in the array

```
#include<stdio.h>
#include<conio.h>
#define size 5
voidmain()
{
int arr[size] = {1, 20, 5, 78, 30};
int key, i, index = -1;
    printf("Enter element to delete\n");
    scanf("%d",&key);
for(i = 0; i < size; i++)
    {
if(arr[i] == key)
        {
            index = i;
break;
        }
    }

if(index != -1)
    {
for(i = index; i < size - 1; i++)
            arr[i] = arr[i+1];

        printf("New Array : ");
for(i = 0; i < size - 1; i++)
            printf("%d ",arr[i]);
    }
else
        printf("Element Not Found\n");

getch();
}
```

Output:-

```
Enter element to delete
78
New Array: 1 20 5 30
```

I.A

DEPARTMENT OF COMPUTER SCIENCE

STACK OPERATIONS

Aim:- Write a program to Implement the stack Operations push and pop Using an Array.

Procedure:-

Algorithm: Push (Insert) Operation in stack

Step 1: IF Top = Max-1
Print "OverFlow: Stack is Full" and Exit
End IF
Step 2: Top = Top + 1
Step 3: Stack[Top] = Element
Step 4: END

Algorithm: Pop (Delete) Operation in Stack

Step 1: If TOP = -1
Print "Underflow: Stack is empty" and Exit
End if
Step 2: Set Del_Element = stack [Top]
Step 3: Top = Top - 1
Step 4: Del_Element
Step 5: End

2. Program:-

```
#include <stdio.h>
#include <conio.h>
#define MAX 3
int st[MAX], top=-1;
void push(int st[], int val);
int pop(int st[]);
void display(int st[]);
void main()
{
int val, option;
do
{
printf("\n *****MAIN MENU*****");
printf("\n 1. PUSH");
printf("\n 2. POP");
printf("\n 3. DISPLAY");
printf("\n 4. EXIT");
printf("\n Enter your option: ");
scanf("%d", &option);
switch(option)
{
case 1:
printf("\n Enter the number to be pushed on stack: ");
scanf("%d", &val);
push(st, val);
break;
case 2:
val = pop(st);
if(val != -1)
printf("\n The value deleted from stack is: %d", val);
break;
case 3:
display(st);
break;
}
}while(option != 4);
```

```
getch();
}

void push(int st[], int val)
{
if(top == MAX-1)
{
printf("\n STACK OVERFLOW");
}
else
{
top++;
st[top] = val;
}
}
int pop(int st[])
{
int val;
if(top == -1)
{
printf("\n STACK UNDERFLOW");
return -1;
}
else
{
val = st[top];
top--;
return val;
}
}
void display(int st[])
{
int i;
if(top == -1)
printf("\n STACK IS EMPTY");
else
{
for(i=top;i>=0;i--)
printf("\n %d",st[i]);
printf("\n");
}
}
```

Output:-

```
*****MAIN MENU*****
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 5
 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 9
 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT

 Enter your option: 1
```

DEPARTMENT OF COMPUTER SCIENCE

Infix to Postfix expression

Aim:- Write a program using stacks to Convert infix expression to postfix expression.

Procedure :-

Step 1 : Add ")" to the end of the infix expression.

Step 2 : Push "(" on to the stack

Step 3 : Repeat Until each character in the infix notation is Scanned

IF a "(" is encountered, push it on the stack

IF an Operend (whether a digit or a character) is encountered, add it postfix expression.

IF a ")" is encountered, then

a. Repeadtly pop from stack and add it to the postfix expression until a "(" is encountered.

b. Discard the "(". That is remove the "(" from stack and do not add it to the postfix expression.

IF an Operator is encountered, then

a. Repeadtly pop from stack and add each Operator to the postfix expression which has the Same preccedence or a higher precedence (than) then

b. Push the Operator to the stack

[End of IF]

Step 4 : Repeadtly pop from the stack and add it to the postfix expression Until the stack is Empty.

Step 5 : Exit

3. Program:-

```
#include<stdio.h>
#include<conio.h>
#define SIZE 100
char stack[SIZE];
int top = -1;
void push(char item)
{
            if(top >= SIZE-1)
            {
             printf("\nStack Overflow.");
            }
            else
            {
             top = top+1;
             stack[top] = item;
            }
}
char pop()
{
            char item ;
            if(top <0)
            {
             printf("stack under flow: invalid infix
expression");
             exit(1);
            }
            else
            {
             item = stack[top];
             top = top-1;
             return(item);
            }
            return 'a';
}
```

```
int is_operator(char symbol)
{
            if(symbol == '^' || symbol == '*' || symbol ==
'/' || symbol == '+' || symbol =='-')
            {
             return 1;
            }
            else
            {
            return 0;
            }
}
int precedence(char symbol)
{
            if(symbol == '^')
            {
             return(3);
            }
            else if(symbol == '*' || symbol == '/')
            {
             return(2);
            }
            else if(symbol == '+' || symbol == '-')
            {
             return(1);
            }
            else
            {
             return(0);
            }
}
void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
            int i, j;
            char item;
            char x;
            push('(');
            strcat(infix_exp,")");
            i=0;
            j=0;
```

```
            i++;
            item = infix_exp[i];
            }
            if(top>0)
            {
             printf("\nInvalid infix Expression.\n");
             exit(1);
            }
            if(top>0)
            {
             printf("\nInvalid infix Expression.\n");
             exit(1);
            }
            postfix_exp[j] = '\0';
}
void main()
{
            char infix[SIZE], postfix[SIZE];
            clrscr();
            printf("ASSUMPTION: The infix expression
contains single letter variables and single digit constants
only.\n");
            printf("\nEnter Infix expression : ");
            gets(infix);
            InfixToPostfix(infix,postfix);
            printf("Postfix Expression: ");
            puts(postfix);
      getch();
}
```

Output:-

```
ASSUMPTION: The infix expression contains single letter
variables and single digit constants only.

Enter Infix expression: A+B-C*D/E

Postfix Expression: AB+CD*E/-
```

I. A

```
item=infix_exp[i];
        while(item != '\0')
        {
         if(item == '(')
         {
              push(item);
         }
         else if( isdigit(item) || isalpha(item))
         {
              postfix_exp[j] = item;
              j++;
         }
         else if(is_operator(item) == 1)
         {
              x=pop();
              while(is_operator(x) == 1 &&
precedence(x)>= precedence(item))
              {
                   postfix_exp[j] = x;
                   j++;
                   x = pop();
              }
              push(x);
              push(item);
         }
         else if(item == ')')
         {
              x = pop();
              while(x != '(')
              {
                   postfix_exp[j] = x;
                   j++;
                   x = pop();
              }
         }
         else
         {
              printf("\nInvalid infix Expression.\n");
              exit(1);
         }
```

4. Program:-

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node
{
int data;
struct node *next;
};
struct queue
{
struct node *front;
struct node *rear;
};
struct queue *q;
void create_queue(struct queue *);
struct queue *insert(struct queue *,int);
struct queue *delete_element(struct queue *);
struct queue *display(struct queue *);
int peek(struct queue *);
int main()
{
int val, option;
create_queue(q);
clrscr();
do
{
printf("\n *****MAIN MENU*****");
printf("\n 1. INSERT");
printf("\n 2. DELETE");
printf("\n 3. PEEK");
printf("\n 4. DISPLAY");
printf("\n 5. EXIT");
printf("\n Enter your option : ");
scanf("%d", &option);
switch(option)
{
```

```
case 1:
printf("\n Enter the number to insert in the queue:");
scanf("%d", &val);
q = insert(q,val);
break;
case 2:
q = delete_element(q);
break;
case 3:
val = peek(q);
if(val! = -1)
printf("\n The value at front of queue is : %d", val);
break;
case 4:
q = display(q);
break;
}
}while(option != 5);
getch();
return 0;
}
void create_queue(struct queue *q)
{
q -> rear = NULL;
q -> front = NULL;
}
struct queue *insert(struct queue *q,int val)
{
struct node *ptr;
ptr = (struct node*)malloc(sizeof(struct node));
ptr -> data = val;
if(q -> front == NULL)
{
q -> front = ptr;
q -> rear = ptr;
```

24/03/22

DEPARTMENT OF COMPUTER SCIENCE

Queue Operations Using LinkedList

Aim :- Write Program to Implement the Queue Operations Using Linked List ?

Procedure :-

Algorithm for Insertion :

Step 1 : Allocate the Space for the new node PTR

Step 2 : Set PTR → DATA = VAL

Step 3 : IF FRONT = NULL
SET FRONT = REAR = PTR
SET FRONT ⇒ NEXT = REAR → NEXT = NULL
ELSE
SET REAR → NEXT = PTR
SET REAR = PTR
SET REAR → NEXT = NULL
[END OF IF]

Step 4 : END

Algorithm for Deletion

Step 1: IF FRONT = NULL
Write "Underflow"
Go to step 5
[End of if]

Step 2: Set ptr = FRONT

step 3: Set FRONT = FRONT → NEXT

step 4: FREE PTR

Step 5: END

```
printf("\n The value being deleted is : %d", ptr -> data);
free(ptr);
}
return q;
}


int peek(struct queue *q)
{
if(q->front==NULL)
{
printf("\n QUEUE IS EMPTY");
return -1;
}
else
return q->front->data;
}
```

```
q -> front -> next = q -> rear -> next = NULL;
}
else

{
q -> rear -> next = ptr;


q -> rear = ptr;
q -> rear -> next = NULL;
} return q;
}
struct queue *display(struct queue *q)
{
struct node *ptr;
ptr = q -> front;
if(ptr == NULL)
printf("\n QUEUE IS EMPTY");
else
{
printf("\n");
while(ptr!=q -> rear)
{
printf("%d\t", ptr -> data);
ptr = ptr -> next;
}
printf("%d\t", ptr -> data);
}
return q;
}
struct queue *delete_element(struct queue *q)
{
struct node *ptr;
ptr = q -> front;
if(q -> front == NULL)
printf("\n UNDERFLOW");
else
{
q -> front = q -> front -> next;
```

DEPARTMENT OF COMPUTER SCIENCE

Polynomial addition Using Single linked list

Aim:- Write a program for Polynomial addition Using Single Linked list.

Procedure:-

Polynomial is a mathematical expression that Consists of variables and co-efficients.

For example $x^2 - 4x + 7$

In the Polynomial Linked list, the co-efficient and exponents of the polynomial are defined as the data node of the list.

For adding two polynomials that are stored as a linked list.

In a linked list node A linked list that is used to store Polynomial looks like- Polynomial: $4x^7 + 12x^2 + 45$

Step 1: loop around all values of linked list and follow step 2 & 3.

Step 2: if the value of a node's exponent is greater, copy this node to result node and head towards the next node.

Step 3: if the values of both node's exponent is Same add the co-efficients and then Copy the added value with node to the result.

Step 4: Print the resultant node.

5. Program:-

```
#include<stdio.h>
#include<malloc.h>
#include<conio.h>
struct link{
       int coeff;
       int pow;
       struct link *next;
       };
struct link *poly1=NULL,*poly2=NULL,*poly=NULL;
void create(struct link *node)
{
 int ch;
 do
 {
  printf("\n enter coeff:");
  scanf("%d",&node->coeff);
  printf("\n enter power:");
  scanf("%d",&node->pow);
  node->next=(struct link*)malloc(sizeof(struct link));
  node=node->next;
  node->next=NULL;
  printf("\n continue(1/0):");
  scanf("%d",&ch);
 }
 while(ch==1);
}
void show(struct link *node)
{
 while(node->next!=NULL)
 {
  printf("%dx^%d",node->coeff,node->pow);
```

Output:-

```
*****MAIN MENU*****
1. INSERT
2. DELETE
3. PEEK
4. DISPLAY
5. EXIT
Enter your option: 3
QUEUE IS EMPTY
Enter your option: 5
```

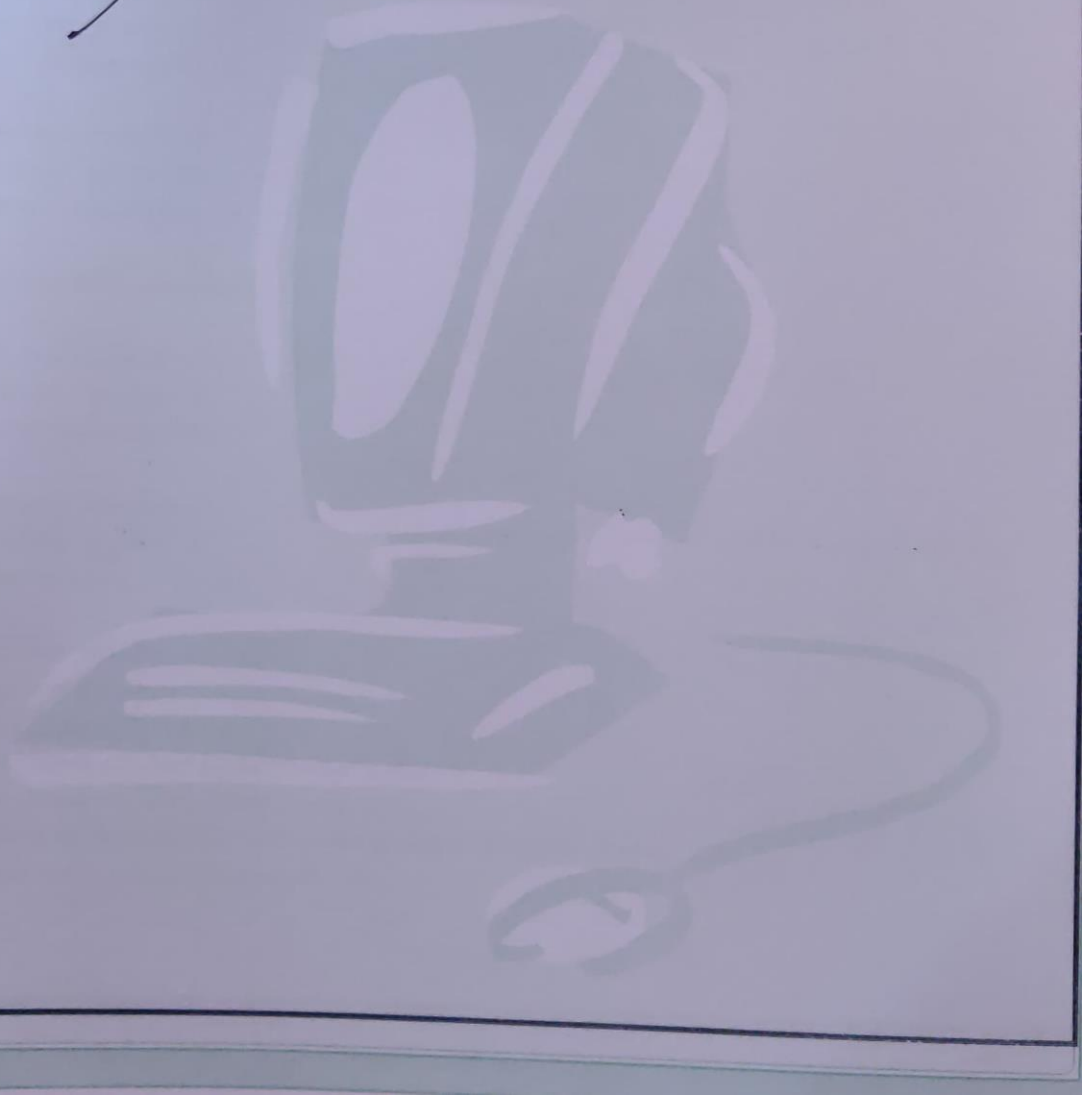
```
 if(poly1->next)
       {
        poly->pow=poly1->pow;
        poly->coeff=poly1->coeff;
        poly1=poly1->next;

  }
      if(poly2->next)
      {
       poly->pow=poly2->pow;
       poly->coeff=poly2->coeff;
       poly2=poly2->next;
       }
       poly->next=(struct link *)malloc(sizeof(struct
link));
       poly=poly->next;
       poly->next=NULL;
       }
}
void main()
{
      char ch;
      clrscr();
      do{
      poly1=(struct link *)malloc(sizeof(struct link));
      poly2=(struct link *)malloc(sizeof(struct link));
      poly=(struct link *)malloc(sizeof(struct link));
      printf("\nenter 1st number:");
      create(poly1);
      printf("\nenter 2nd number:");
      create(poly2);
      printf("\n1st Number:");
      show(poly1);
      printf("\n2nd Number:");
      show(poly2);
      polyadd(poly1,poly2,poly);
```

```
printf("\nAdded polynomial:");
      show(poly);



printf("\n add two more numbers:");
      scanf("%c",&ch);
      }
      while(ch=='y' || ch=='Y');
      getch();
}
```



```
  node=node->next;
   if(node->next!=NULL)


   printf("+");
  }
 }
 void polyadd(struct link *poly1,struct link *poly2,struct
 link *poly)
 {
      while(poly1->next &&  poly2->next)
      {
       if(poly1->pow>poly2->pow)
       {
        poly->pow=poly1->pow;
        poly->coeff=poly1->coeff;
        poly1=poly1->next;
        }
       else if(poly1->pow<poly2->pow)
       {
        poly->pow=poly2->pow;
        poly->coeff=poly2->coeff;
        poly2=poly2->next;
        }
       else
       {
        poly->pow=poly1->pow;
        poly->coeff=poly1->coeff+poly2->coeff;
        poly1=poly1->next;
        poly2=poly2->next;
        }
      poly->next=(struct link *)malloc(sizeof(struct
link));
      poly=poly->next;
      poly->next=NULL;
     }
     while(poly1->next || poly2->next)
     {
```

DEPARTMENT OF COMPUTER SCIENCE

Searching : a). Linear search

Aim:- Write a program to search an item in a given list using the following Algorithm's: a) Linear b) Binary

Procedure:-

a). **Linear Search:** A linear search, also known as a Sequential search, is a method of finding an element within a list. It checks each element of the list sequentially until a match is found or the whole list has been searched.

1. Begin with the left most element of arr[ ] and one by one compare x with each other.
2. If x matches with the (middle) element then return the Index.
3. If x does not match with any of the elements then return -1.

b). **Binary Search:** Search a sorted array by repeatedly dividing the search interval in the half. Being with an internal covering the whole array. If the value of search key is less than the item in the middle of the interval, narrow the internal to the lower half. Repeteadly check until the value is found or is Empty:

1. Compare x with the middle element
2. If x matches with the middle element, then x Can only lie in the right half.
3. (Sub) Else if x is greater than the mid element, then sub array after the mid element recur for the right half.
4. Else (x is smaller) recur for the left half.

6. Program:-

6A. Linear search:-

```
#include<stdio.h>

void main()
{
    int a[20],i,x,n;
    printf("How many elements?");
    scanf("%d",&n);

    printf("Enter array elements:\n");
    for(i=0;i<n;++i)
        scanf("%d",&a[i]);

    printf("\nEnter element to search:");
    scanf("%d",&x);

    for(i=0;i<n;++i)
        if(a[i]==x)
            break;

    if(i<n)
        printf("Element found at index %d",i);
    else
        printf("Element not found");

    getch();
}
```

Output:-

```
enter 1st number:

 enter coeff:5
 enter power:4
 continue(1/0):1

 enter coeff:6
 enter power:3
 continue(1/0):1

 enter coeff:3
 enter power:2
 continue(1/0):1

 enter coeff:8
 enter power:0
 continue(1/0):0

enter 2nd number:

 enter coeff:4
 enter power:3
 continue(1/0):1

 enter coeff:6
 enter power:2
 continue(1/0):1

 enter coeff:9
 enter power:1
 continue(1/0):0
1st Number:5x^4+6x^3+3x^2+8x^0
2nd Number:4x^3+6x^2+9x^1
Added polynomial:5x^4+10x^3+9x^2+9x^1+8x^0
 add two more numbers: no
```

Searching : b). Binary search

Aim: Write a program to search an item in a given list using the following Algorithm : b). Binary search .

Procedure :

b). Binary Search :

Search a sorted array by repeatedly dividing the search interval in the half. Being with an internal covering the whole array. If the value of search key is less than the item in the mid of the interval, narrow the internal to the lower half. Repeatedly check until the value is found or it is Empty.

1. Compare X with middle element.
2. If it matches with the middle element, then X can only lie in the right half.
3. Else if, X is greather than the mid element, then sub array after the mid element recur for the Right half.
4. Else (x is smaller) recur for left half.

6B.Binary search:-

```
#include <stdio.h>
#include<conio.h>
int binarySearch(int arr[], int l, int r, int x)
{
     if (r >= l) {
          int mid = l + (r - l) / 2;

               if (arr[mid] == x)
               return mid;

               if (arr[mid] > x)
               return binarySearch(arr, l, mid - 1, x);

          return binarySearch(arr, mid + 1, r, x);
     }


     return -1;
}

void main()
{
     int arr[] = { 2, 3, 4, 10, 40 };
     int n = sizeof(arr) / sizeof(arr[0]);
     int x,result;
     clrscr();
     printf("Enter a element u want to search:");
     scanf("%d",&x);
     result = binarySearch(arr, 0, n - 1, x);
     (result == -1) ? printf("Element is not present in
array") : printf("Element is present at index %d", result);
     getch();
}
```

Output:-

```
How many elements? 6

Enter array elements:

5

8

9

3

2

1

Enter element to search: 3

Element found at index 3
```

I. A

13/04/22

DEPARTMENT OF COMPUTER SCIENCE

Bubble sort

Aim :- Write a program for Implementation of Bubble sort Algorithm

Procedure :-

The basic methodology of the working of bubble sort is as follows:

a). In Pass 1, A[0] and A[1] are Compared, then A[1] is compared with A[2], A[2] is Compared with A[3], and so on. Finally, A[N-2] is Compared with A[N-1]. Pass 1 involves n-1 Comparisons and places the biggest element at the highest index of the Array.

b). In Pass 2, A[0] and A[1] are Compared, then A[1] is Compared with A[2], A[2] is Compared with A[3], and so on. Finally, A[N-3] is Compared with A[N-2]. Pass 2 involves n-2 Comparisions and places the second biggest element at the Second highest index of array.

c). In Pass 3, A[0] and A[1] are Compared, then A[1] is compared with A[2], A[2] is Compared with A[3], and so on. Finally A[N-4] is Compared with A[N-3]. Pass 3 Involves n-3 Comparisons and places the third biggest element at the third highest Index of the array.

d). In Pass n-1, A[0] and A[1] are Compared so that A[0] < A[1]. After this step, all the elements of the array are arranged in Ascending Order.

7. Program:-

```
#include <stdio.h>
void bubble_sort(int a[], int n) {
    int i = 0, j = 0, tmp;
    for (i = 0; i < n; i++)
 {
        for (j = 0; j < n - i - 1; j++)
 {
            if (a[j] > a[j + 1])
 {
                tmp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = tmp;
            }
        }
    }
}
void main() {
  int a[100], n, i, d, swap;
  printf("Enter number of elements in the array:\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  bubble_sort(a, n);
  printf("Printing the sorted array:\n");
  for (i = 0; i < n; i++)
     printf("%d\n", a[i]);
  getch();
}
```

Output:-

```
Enter a element u want to search: 10

Element is present at index 3
```

I.A

25/04/22

DEPARTMENT OF COMPUTER SCIENCE

Insertion Sort

Aim :- Write a program for Implementation of Insertion sort Algorithm.

Procedure :-

Insertion sort: Insertion sort is a very simple Sorting algorithm in which the sorted array (or list) is built one element at a time. We all are familiar with this techniqve of sorting, as we usually use it for Ordering a deck of cards while playing bridge.

* The array of values to be sorted is divided into two sets. One that stores sorted values and another that contains Unsorted values.
* The Sorting algorithm will proceed until there are elements in the Unsorted set.
* The first element of the Unsorted partition has array index 1 (if LB = 0).

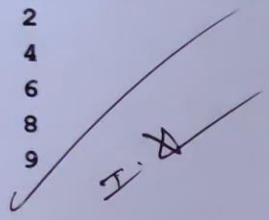
Algorithm :-

Step 1: If it is first element, it is already sorted. return 1;
Step 2: Pick next element
Step 3: Compare with all elements in the sorted sub-list
Step 4: Shift all the elements in the sorted sub-list that is greather than the value to be sorted.
Step 5: Insert the value
Step 6: Repeat until the list is sorted.

Output:-

```
Enter number of elements in the array:
5
Enter 5 integers
8
6
4
2
9

Printing the sorted array:
2
4
6
8
9
```



Output:-

```
Original elements before sorting: 12 11 13 5 6

After insertion Sort: 5 6 11 12 13
```

I.A

05/05/22

DEPARTMENT OF COMPUTER SCIENCE

Dept First Search graph (DFS)

Aim:- Write a program to Implement Dept First Search graph traversals Algorithm.

Procedure:-

The dept-first search algorithm progresses by Expanding the starting node of G and then going deeper and deeper until the goal node is found, or until a node that has no children is encountered. When a dead-end is reached, the algorithm backtracks, returning to the most recent node that has not been completely explored. In other words, dept-first search begins at a starting node A which becomes the current node. Then, it examines each node N along a path P which begins At A. That is, we process a neighbour of A, then a neighbour of neighbor of A, and so on. then we backtrack to the current node.

Step 1: SET STATUS = 1 for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2.

Step 3: Repeat Step 4 and 5 until STACK is Empty.

Step 4: Pop the top node N. Process it and set its STATUS = 3

Step 5: Push on the stack all the neighbours of N that are in the ready state and set their STATUS = 2 [END OF LOOP].

Step 6: Exit

8. Program:-

```
#include <math.h>
#include <stdio.h>

void insertionSort(int arr[], int n)
{
     int i, key, j;
     for (i = 1; i < n; i++) {
          key = arr[i];
          j = i - 1;


          while (j >= 0 && arr[j] > key) {
               arr[j + 1] = arr[j];
               j = j - 1;
          }
          arr[j + 1] = key;
     }
}


void printArray(int arr[], int n)
{
     int i;
     for (i = 0; i < n; i++)
          printf("%d ", arr[i]);
     printf("\n");
}


int main()
{
     int arr[] = { 12, 11, 13, 5, 6 };
     int n = sizeof(arr) / sizeof(arr[0]);
    printf("original elements Before sorting:");
    printArray(arr,n);
     insertionSort(arr, n);
     printf("After insertion Sort:");
     printArray(arr, n);

     return 0;
}
```

Output:-

```
Enter the adjacency matrix:
0 1 0 1 0
1 0 1 1 0
0 1 0 0 1
1 1 0 0 1
0 0 1 1 0
DFS Traversal:A->B->C->E->D->
```

I-A

11/05/22

DEPARTMENT OF COMPUTER SCIENCE

Breadth First Search Graph (BFS)

Aim:- Write a program to Implement Breadth First Search Graph traversals Algorithm

Procedure:-

Breadth-First Search (BFS) is a graph Search Algorithm that begins at the root node and explores all the neighbours nodes. Then for each of those nearest nodes, the algorithm explores their Unexplored neighbor nodes, and so on, until it finds the goal. This means that we need to track the neighbours of the node and guartee that every node in the graph is processed and no node is processed more that once. This is Accomplished by using a Queue that will hold the nodes that are waiting for further processing and a variable STATUS to represent the Current State of the node.

Step 1: Set Status = 1 for each node in G.
Step 2: Enqueue the Starting node A and set its Status = 2
Step 3: Repeat Steps 4 and 5 until Queue is Empty.
Step 4: Dequeue a node N. Process it and set its Status = 3
Step 5: Enqueue all the neighbours of N that are in the ready state and set their Status = 2.
[END OF LOOP].
Step 6: Exit

9. Program:-

```
#include <stdio.h>
#define MAX 5
void depth_first_search(int adj[ ][MAX],int visited[ ],int
start)
{
int stack[MAX];
int top = -1, i;
printf("%c->",start + 65);
visited[start] = 1;
stack[++top] = start;
while(top ! = -1)
{
start = stack[top];
for(i = 0; i < MAX; i++)
{
if(adj[start][i] && visited[i] == 0)
{
stack[++top] = i;
printf("%c->", i + 65);
visited[i] = 1;
break;
}
}
if(i == MAX)
top--;
}
}
int main()
{
int adj[MAX][MAX];
int visited[MAX] = {0}, i, j;
printf("\n Enter the adjacency matrix: ");
for(i = 0; i < MAX; i++)
for(j = 0; j < MAX; j++)
scanf("%d", &adj[i][j]);
printf("DFS Traversal: ");
depth_first_search(adj,visited,0);
printf("\n");
return 0;
}
```

Output:-

```
Enter the adjacency matrix:
1 0 1 0 1
1 1 1 1 0
0 0 1 0 1
0 0 0 0 1
1 1 1 1 1
Breadth First traversal:
A    C    E    B    D
```

I-A

10. Program:-

```
#include <stdio.h>
#define MAX 5
void breadth_first_search(int adj[][MAX],int visited[],int
start)
{
int queue[MAX],rear = -1,front =-1, i;
queue[++rear] = start;
visited[start] = 1;
while(rear != front)
{
start = queue[++front];
printf("%c \t",start + 65);
for(i = 0; i < MAX; i++)
{
if(adj[start][i] == 1 && visited[i] == 0)
{
queue[++rear] = i;
visited[i] = 1;
}
}
}
}
void main()
{
int visited[MAX] = {0};
int adj[MAX][MAX], i, j;
clrscr();
printf("\n Enter the adjacency matrix: ");
for(i = 0; i < MAX; i++)
for(j = 0; j < MAX; j++)
scanf("%d", &adj[i][j]);
printf("Breadth First traversal:\n");
breadth_first_search(adj,visited,0);
getch();
}
```