# **INDEX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SL. NO. | ASSIGNTMENTS | DATE OF EXECUTION | DATE OF SUBMISSION | REMARKS | SIGNATURE |
| 1. | 1.1: Write an algorithm to find minimum and maximum from an 1-D array. Implement the algorithm using C programming  1.2: Algorithm to search an element in 1-D array and write a program in C | 12.08.2022  12.08.2022 |  |  |  |
| 2. | Write an algorithm to insert in 1st , last and p-th position in an 1-D array and write a C program | 19.08.2022 |  |  |  |
| 3. | Algorithm to merge two sorted array into another sorted array and write a C program | 26.08.2022 |  |  |  |
| 4. | Algorithm to delete 1st, last and p-th position in 1-D array | 26.08.2022 |  |  |  |
| 5. | Write an algorithm to implement a singly link-list and write a C program | 02.09.2022 |  |  |  |
| 6. | Write an algorithm for all operations of insertion in singly linked list | 09.09.2022 |  |  |  |
| 7 | Write an algorithm and C program for all operations of deletion in singly linked list | 23.09.2022 |  |  |  |
| 8 | Write an algorithm and C program for reverse a singly link list | 23.09.2022 |  |  |  |
| 9 | Write an algorithm and C program for create and traverse circular link list | 14.10.2022 |  |  |  |
| 10 | Write an algorithm and C program for insertion operations in a circular link list | 14.10.2022 |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SL. NO. | ASSIGNMENT | DATE OF EXECUTION | DATE OF SUBMISSION | REMARKS | SIGNATURE |
| 11 | Write an algorithm and C program for deletion operations in a circular link list | 14.10.2022 |  |  |  |
| 12 | Write an algorithm and C program for all the insertion and traverse operations in a doubly link list | 21.10.2022 |  |  |  |
| 13 | Write an algorithm and C program for all the deletion operations of a doubly link list | 18.11.2022 |  |  |  |
| 14 | Write an algorithm and C program for all basic operations of stack | 18.11.2022 |  |  |  |
| 15 | Write an algorithm and C program for all basic operations for a linear queue | 18.11.2022 |  |  |  |
| 16 | Write an algorithm and C program for binary search in a sorted array | 18.11.2022 |  |  |  |
| 17. | Write an algorithm and C program for Bubble Sort | 18.11.2022 |  |  |  |
| 18. | Write an algorithm and C program for Insertion Sort | 18.11.2022 |  |  |  |
| 19 | Write an algorithm and C program for Quick Sort | 02.12.2022 |  |  |  |
| 20 | Write an algorithm and C program for Merge Sort | 02.12.2022 |  |  |  |
| 21 | Write an algorithm and C program for Selection Sort | 02.12.2022 |  |  |  |

* 1. Write an algorithm to find minimum and maximum from an 1-D array. Implement the algorithm using C programming.

**Algorithm:**

**Step 1:** Begin Min(arr)

**Step 2:** set min:=arr[0], i:=1

**Step 3:** if arr[i] < min, set min:=arr[i]

**Step 4:** set i := i +1

**Step 5:** if i < n, go to step 3

**Step 6:**  return min

**Step 7:** end Min(arr)

**Step 1:** Begin Max(arr)

**Step 2:** set max:=arr[0], i:=1

**Step 3:** if arr[i] > max, set max:=arr[i]

**Step 4:** set i := i +1

**Step 5:** if i < n, go to step 3

**Step 6:**  return max

**Step 7:** end Max(arr)

**Program:**

int Min(int arr[])

{

int min = arr[0];

for (int i = 0; i < 6; i++)

{

if (min>arr[i])

{

min=arr[i];

}

}

return min;

}

int Max(int arr)

{

int max = arr[0];

for (int i = 0; i < 6; i++)

{

if (max<arr[i])

{

max=arr[i];

}

}

return max;

}

**Output:**

Input: 4 2 15 67 99 1

Output: Min = 1, Max = 99

* 1. Algorithm to search an element in 1-D array and write a program in C

**Algorithm:**

**Step 1:** Begin search(arr,item)

**Step 2:** set i:=0

**Step 3:** if arr[i] = item,

print “FOUND”, go to step 7

**Step 4:** else, set i := i+1,

**Step 5:** if i < n, go to step 7

**Step 6:** print “NOT FOUND”

**Step 7:** end

**Program:**

int search(int arr[], int item, int size)

{

for(int i=0; i < size; i++)

{

if (arr[i] == item)

{

printf(“FOUND”);

break;

}

}

if (i==size)

printf(“NOT FOUND”);

return 0;

}

**Output:**

**Case 1:**

Input: 2 5 66 34 4 8 7

Item: 66

Output: FOUND

**Case 2:**

Input: 2 5 66 34 4 8 7

Item: 55

Output: NOT FOUND

1. Write an algorithm to insert in 1st , last and p-th position in an 1-D array and write a C program. Let, arr[]={2 5 4 8 7}

**Algorithm:**

**Step 1:** Begin insert\_first(arr,item)

**Step 2:** set i:=size-1

**Step 3:** set arr[i+1] := arr[i]

**Step 4:** set i := i-1

**Step 5:** if i >= 0, go to step 3

**Step 6:**  set arr[0] := item

**Step 7:**  return arr

**Step 8:**  end insert\_first(arr, item)

**Step 1:** Begin insert\_last(arr,item)

**Step 2:**  set arr[size] := item

**Step 3:**  return arr

**Step 4:**  end insert\_last(arr, item)

**Step 1:** Begin insert\_pth\_position(arr,item, p)

**Step 2:** set i:=size-1

**Step 3:** set arr[i+1] := arr[i]

**Step 4:** set i := i-1

**Step 5:** if i >= p-1, go to step 3

**Step 6:**  set arr[p-1] := item

**Step 7:**  return arr

**Step 8:**  end insert\_pth\_position(arr, item)

**Program:**

void insert\_first(int\* arr[], int size, int item)

{

for(int i=size-1; i >=0; i - -)

{

arr[i+1] = arr[i];

}

arr[0] = item;

}

void insert\_last(int\* arr[], int size, int item)

{

arr[size] = item;

}

void insert\_pth\_position(int\* arr[], int size, int item, int p)

{

for(int i=size-1; i >=p-1; i - -)

{

arr[i+1] = arr[i];

}

arr[p-1] = item;

}

**Output:**

Press 1 to insert first, 2 to insert at last, 3 to insert at p-th position.

**Case 1:**

Input: 1

Item: 66

Output: 66 2 5 4 8 7

**Case 2:**

Input: 2

Item: 55

Output: 66 2 5 4 8 7 55

**Case 3:**

Input: 2

Position: 3

Item: 34

Output: 66 2 34 5 4 8 7 55

3. Algorithm to merge two sorted array into another sorted array and write a C program.

**Algorithm:**

**Step 1:** Begin merge\_sort(arr1,size1,arr2, size2)

**Step 2:** set, i := 0, j:=0, k := 0

**Step 3:** if arr1[i] <arr2[j], then, set arr3[k]:= arr1[i], set i := i+1, set k := k+1.

**Step 4:** if arr1[i] >arr2[j], then set arr3[k] := arr2[j], set j:=j+1, k:=k+1

**Step 5:** if i<size1 and j<size2, go to step 3

**Step 6:** if i<size1, set arr3[k]:=arr1[i], k:=k+1, i:=i+1, and repeat step 6.

**Step 7:** if j<size2, set arr3[k]:=arr2[j], k:=k+1, j:=j+1, and repeat step 7.

**Step 8:** print arr3

**Step 9:** end

**Program:**

int\* merge(int \*a1, int size1, int \*a2, int size2){

while (i<size1 && j<size2)

{

if (a1[i]<a2[j])

{

marray[k]=a1[i];

i++;

k++;

}

else

{

marray[k]=a2[j];

j++;

k++;

}

}

while (i<size1)

{

marray[k]=a1[i];

k++;

i++;

}

while (j<size2)

{

marray[k]=a2[j];

k++;

j++;

}

return &marray

}

**Output:**

**Input:**

First Array: 2 4 6 8

Second Array: 1 3 5 7

Merged Array: 1 2 3 4 5 6 7 8

4. Algorithm to delete 1st, last and p-th position in 1-D array

**Algorithm:**

**Step 1:** Begin delete\_first(arr,size)

**Step 2:** set i:=1`

**Step 3:** set arr[i-1] := arr[i]

**Step 4:** set i := i+1

**Step 5:** if i < size, go to step 3

**Step 6:**  set size := size-1

**Step 7:**  traverse (arr, size)

**Step 8:**  end delete\_first(arr, size)

**Step 1:** Begin delete\_last(arr,size)

**Step 2:**  set size:= size-1

**Step 3:**  traverse (arr,size)

**Step 4:**  end delete\_last(arr, size)

**Step 1:** Begin delete\_pth\_position(arr,item, p)

**Step 2:** set i:=pos

**Step 3:** set arr[i-1] := arr[i]

**Step 4:** set i := i+1

**Step 5:** if i < size, go to step 3

**Step 6:**  set size := size-1

**Step 7:** traverse (arr, size)

**Step 8:**  end delete\_pth\_position(arr,size)

**Program:**

void firstDeletion(int arr[], int size)

{

for (int i = 1; i < size-1; i++)

{

arr[i-1] = arr[i];

}

size--;

traverse(arr, size);

}

void firstDeletion(int arr[], int size, int index )

{

size--;

traverse(arr, size);

}

void firstDeletion(int arr[], int size, int pos)

{

for (int i = pos; i < size-1; i++)

{

arr[i-1] = arr[i];

}

size--;

traverse(arr, size);

}

**Output:**

Array: 1 2 3 4 5 6

**Case 1:** Output: 2 3 4 5 6

**Case 2:** Output: 1 2 3 4 5

**Case 3:** Input: 3

Output: 1 2 4 5 6

5. Write an algorithm to implement a singly link-list and write a C program

**Algorithm:**

Creation(struct list \*head)

Step 1: Begin

Step 2: set new: = allocate memory

Step 3: If new!= NULL then

Step 31: Input item.

Step 3.2; Set new-> data := item.

Step 3. 3: Set new-> link:=NULL

Step3 4: if head = NULL then set head := new

Step 3.5: else go to step 3.6

Step 3.6: Set temp := head

Step 3.7: if temp-> link != NULL, then set temp:= temp->link

Step 3.8: Set temp-> link := new

Step 4: else print " No Memory" go to step 5

Step 5:return head pointer

Step 6: End

traversal (struct list \*head)

Step 1: Begin

Step 2: If head = NULL then print "NO list exist"

Step 3: Set temp:= head

Step 4: if temp != NULL then

Step 4.1: print temp -> data

Step 4.2: Set temp:= temp-> link go to step 3

Step 5: End

**Program:**

struct Node \*creation(struct Node \*head, int data)

{

struct Node \*NewNode = (struct Node \*)malloc(sizeof(struct Node));

NewNode->data = data;

struct Node \*p = head;

while (p->next!=NULL)

{

p = p->next;

}

p->next = NewNode;

NewNode->next = NULL;

return head;

}

void linklisttreversal(struct Node \*ptr)

{

while (ptr != NULL)

{

printf("%d ", ptr->data);

ptr = ptr->next;

}

}

**Output:**

Inputs: 2 4 6 8 9

Traverse: 2-> 4-> 6-> 8-> 9-> NULL

6. Write an algorithm for all operations of insertion in singly linked list

**Algorithm:**

Insert\_first(struct list \*head)

Step 1: Begin

Step 2: Set new: = allocate memory

Step 3: Input item

Step 4: Set new->data := item

Step 5: Set new->link:=head

Step 6: Set head := new

Step 7: End

Insert\_last (struct list head)

Step 1: Begin

step 2: set new: =allocatememory

Step 3: if new!= NULL then

Step 3.1: input item

Step 3.2 Set new->data := item.

Step 3.3: Set new->link:= NULL

Step 3.4: Set temp := head.

Step 3.5: if temp->link !=NULL then set temp:= temp->link repeat

Step 3.6: Set temp->link := new go to Step 5

Step 4: Else print "No memory" go to step 5

Step 5: End

Insert\_pth (struct list \*head, int pos)

Step 1: Begin

Step 2: Set new: = allocate memory

Step 3: if new!= NULL then

Step 3.1: input item

Step 3.2: Set new data: item

Step 3.3: Set new-> link: = NULL

Step 3.4: Set count :=1

Step 3.5: if count < pos then set prev:= temp Set temp := temp->link, set count = count+1 repeat

Step 3.6: Set prev -> link := new

Step 3.7: Set new-> link:= temp go to stop 5

Step 4: else print "No memory"

Step 5: End

**Program:**

struct Node \*insertAtFirst(struct Node \*head, int data)

{

struct Node \*NewNode = (struct Node \*)malloc(sizeof(struct Node));

NewNode->next = head;

NewNode->data = data;

return NewNode;

}

struct Node \*insertAtEnd(struct Node \*head, int data)

{

struct Node \*NewNode = (struct Node \*)malloc(sizeof(struct Node));

NewNode->data = data;

struct Node \*p = head;

while (p->next!=NULL)

{

p = p->next;

}

p->next = NewNode;

NewNode->next = NULL;

return head;

}

struct Node \*insertAtIndex(struct Node \*head, int data, int index)

{

struct Node \*NewNode = (struct Node \*)malloc(sizeof(struct Node));

struct Node \*p = head;

int count = 1;

while (count < pos)

{

p = p->next;

count++;

}

NewNode->data = data;

NewNode->next = p->next;

p->next = NewNode;

return head;

}

**Output:**

Link list: 7 3 5 9 6 12

**Case 1 (**Insert\_First**):**

Item: 45

Output: 45->7-> 3-> 5-> 9-> 6-> 12-> NULL

**Case 2 (**Insert\_Last**):**

Item: 45

Output: 7-> 3-> 5-> 9-> 6-> 12-> 45-> NULL

**Case 3 (**Insert\_pth**):**

Item: 45

Pos: 4

Output: 7-> 3-> 5-> 45-> 9-> 6-> 12-> NULL

7. Write an algorithm and C program for all operations of deletion in singly linked list

**Algorithm:**

**STEP 1:** IF HEAD = NULL

**STEP 2:** SET TEMP = HEAD

**STEP 3:** SET I = 0

**STEP 4:** REPEAT STEP 5 TO 8 UNTIL I<loc< li=""> </loc<>

**STEP 5:** TEMP1 = TEMP

**STEP 6:** TEMP = TEMP → NEXT

**STEP 7:** IF TEMP = NULL

WRITE "DESIRED NODE NOT PRESENT"  
    GOTO STEP 12  
    END OF IF

**STEP 8:** I = I+1

**STEP 9:** TEMP1 → NEXT = TEMP → NEXT

**STEP 10:** FREE TEMP

**STEP 11:** EXIT

**Program:**

#include<stdio.h>

#include<stdlib.h>

void create(int);

void delete\_specified();

struct node

{

    int data;

    struct node \*next;

};

struct node \*head;

void main ()

{

    int choice,item;

    do

    {

        printf("\n1.Append List\n2.Delete node\n3.Exit\n4.Enter your choice?");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:

            printf("\nEnter the item\n");

            scanf("%d",&item);

            create(item);

            break;

            case 2:

            delete\_specified();

            break;

            case 3:

            exit(0);

            break;

            default:

            printf("\nPlease enter valid choice\n");

        }

    }while(choice != 3);

}

void create(int item)

    {

        struct node \*ptr = (struct node \*)malloc(sizeof(struct node \*));

        if(ptr == NULL)

        {

            printf("\nOVERFLOW\n");

        }

        else

        {

            ptr->data = item;

            ptr->next = head;

            head = ptr;

            printf("\nNode inserted\n");

        }

    }

void delete\_specified()

    {

        struct node \*ptr, \*ptr1;

        int loc,i;

        scanf("%d",&loc);

        ptr=head;

        for(i=0;i<loc;i++)

        {

            ptr1 = ptr;

            ptr = ptr->next;

            if(ptr == NULL)

            {

                printf("\nThere are less than %d elements in the list..\n",loc);

                return;

            }

        }

        ptr1 ->next = ptr ->next;

        free(ptr);

        printf("\nDeleted %d node ",loc);

 }

**Output:**

1.Append List

2.Delete node

3.Exit

4.Enter your choice?1

Enter the item

12

**9.** Write an algorithm and C program for create and traverse circular link list

**Algorithm:**

* **STEP 1:** SET PTR = HEAD
* **STEP 2:** IF PTR = NULL
* **STEP 4:** REPEAT STEP 5 AND 6 UNTIL PTR → NEXT != HEAD
* **STEP 5:** PRINT PTR → DATA
* **STEP 6:** PTR = PTR → NEXT
* **STEP 7:** PRINT PTR→ DATA
* **STEP 8:** EXIT

**Program:**

void traverse()

{

    struct node \*ptr;

    ptr=head;

    if(head == NULL)  {

printf("\nnothing to print");

    }

    else

    {

        printf("\n printing values ... \n");

        while(ptr -> next != head) {

            printf("%d\n", ptr -> data);

            ptr = ptr -> next;

        }

        printf("%d\n", ptr -> data);   } }

10. Write an algorithm and C program for insertion operations in a circular link list

* **Step 1:** IF PTR = NULL
* **Step 2:** SET NEW\_NODE = PTR
* **Step 3:** SET PTR = PTR -> NEXT
* **Step 4:** SET NEW\_NODE -> DATA = VAL
* **Step 5:** SET TEMP = HEAD
* **Step 6:** Repeat Step 8 while TEMP -> NEXT != HEAD
* **Step 7:** SET TEMP = TEMP -> NEXT
* **Step 8:** SET NEW\_NODE -> NEXT = HEAD
* **Step 9:** SET TEMP → NEXT = NEW\_NODE
* **Step 10:** SET HEAD = NEW\_NODE
* **Step 11:** EXIT

**Program:**

void beg\_insert(int item)

{

    struct node \*ptr = (struct node \*)malloc(sizeof(struct node));

    struct node \*temp;

    if(ptr == NULL)

    {

        printf("\nOVERFLOW");

    }

    else

    {

        ptr -> data = item;

        if(head == NULL)

        {

            head = ptr;

            ptr -> next = head;

        }

        else

        {

            temp = head;

            while(temp->next != head)

                temp = temp->next;

            ptr->next = head;

            temp -> next = ptr;

            head = ptr;

        }

    printf("\nNode Inserted\n");

    }

11. Write an algorithm and C program for deletion operations in a circular link list

**Algorithm:**

* **Step 1:** IF HEAD = NULL
* **Step 2:** SET PTR = HEAD
* **Step 3:** Repeat Steps 4 and 5 while PTR -> NEXT != HEAD
* **Step 4:** SET PREPTR = PTR
* **Step 5:** SET PTR = PTR -> NEXT
* **Step 6:** SET PREPTR -> NEXT = HEAD
* **Step 7:** FREE PTR
* **Step 8:** EXIT

**Program:**

void last\_delete()  {

struct node \*ptr, \*preptr;

    if(head==NULL)

    {

        printf("\nUNDERFLOW\n");

    }

    else if (head ->next == head)

    {

        head = NULL;

        free(head);

        printf("\nNode Deleted\n");

    }

    else

    {

        ptr = head;

        while(ptr ->next != head)

        {

            preptr=ptr;

            ptr = ptr->next;

        }

        preptr->next = ptr -> next;

        free(ptr);

        printf("\nNode Deleted\n");

    }

}

13. Write an algorithm and C program for all the deletion operations of a doubly link list

**Algorithm:**

* **STEP 1:** IF HEAD = NULL
* **STEP 2:** SET PTR = HEAD
* **STEP 3:** SET HEAD = HEAD → NEXT
* **STEP 4:** SET HEAD → PREV = NULL
* **STEP 5:** FREE PTR
* **STEP 6:** EXIT

**Program:**

void beginning\_delete()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW\n");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nNode Deleted\n");

}

else

{

ptr = head;

head = head -> next;

head -> prev = NULL;

free(ptr);

printf("\nNode Deleted\n");

}  }

Output:

1.Append List

2.Delete node from beginning

3.Exit

4.Enter your choice?1

Enter the item

12

Node Inserted

14. Write an algorithm and C program for all basic operations of stack

Algorithm (PUSH):

* **STEP 1** START
* **STEP 2** Store the element to push into array
* **STEP 3** Check if top== (MAXSIZE-1) then stack is full else goto step 4
* **STEP 4** Increment top as top = top+1
* **STEP 5** Add element to the position stk[top]=num
* **STEP 6** STOP

**Program:**

void push ()

{

int num;

if (s.top == (MAXSIZE - 1))

{

printf ("Stack is Full\n");

return;

}

else

{

printf ("Enter the element to be pushed\n");

scanf ("%d", &num);

s.top = s.top + 1;

s.stk[s.top] = num;

}

return;

}

Algorithm (POP):

**STEP 1** begin procedure pop: stack

**STEP 2** if stack is empty

return null

endif

**STEP 3** data ← stack[top]

**STEP 4** top ← top - 1

**STEP 5** return data

**STEP 6** end procedure

Program:

void Pop()

{

if(Top==-1)

{

printf("\nUnderflow!!");

}

else

{

printf("\nPopped element: %d",inp\_array[Top]);

Top=Top-1;

}

}

Output:

Operations performed by Stack

1.Push the element

2.Pop the element

3.Show

4.End

Enter the choice:1

Enter element to be inserted to the stack:10

Operations performed by Stack

1.Push the element

2.Pop the element

3.Show

4.End

Enter the choice:3

Elements present in the stack:

10

Operations performed by Stack

1.Push the element

2.Pop the element

3.Show

4.End

Enter the choice:2

Popped element: 10

Operations performed by Stack

1.Push the element

2.Pop the element

3.Show

4.End

Enter the choice:3

Underflow!!

15. Write an algorithm and C program for all basic operations for a linear queue

Algorithm (INSERT):

* **STEP 1** START
* **STEP 2** Store the element to insert
* **STEP 3** Check if REAR= MAX-1 then write Overflow else goto step 5
* **STEP 4** Check if REAR= -1 then  
               set FRONT=REAR=0  
               else  
               set REAR=REAR+1
* STEP 5 set QUEUE [REAR]=NUM
* STEP 6 STOP

Program:

insert()

{

int add\_item;

if (rear == MAX - 1)

printf("Queue Overflow \n");

else

{

if (front == - 1)

/\*If queue is initially empty \*/

front = 0;

printf("Inset the element in queue : ");

scanf("%d", &add\_item);

rear = rear + 1;

queue\_array[rear] = add\_item;

}

}

Output:

If you want to enter the element in queue press 1 else 0

1

Inset the element in queue : 10

If you want to enter the element in queue press 1 else 0

1

Inset the element in queue : 20

If you want to enter the element in queue press 1 else 0

1

Inset the element in queue : 30

If you want to enter the element in queue press 1 else 0

0

Queue is :

10 20 30

Algorithm (Delete):

* **STEP 1** START
* **STEP 2** Store the element to insert
* **STEP 3** Check if FRONT=-1 or FRONT > REAR writes Underflow else goto step 4
* **STEP 4** Set VAL=QUEUE [FRONT]  
               Set FRONT= FRONT + 1
* STEP 5 STOP

Program:

void delete()

{

if (front == - 1 || front > rear)

{

printf("Queue Underflow \n");

return ;

}

else

{

printf("Element deleted from queue is : %d\n", queue\_array[front]);

front = front + 1;

}

}

16. Write an algorithm and C program for binary search in a sorted array

Algorithm:

* Step 1: set beg = lower\_bound, end = upper\_bound, pos = - 1
* Step 2: repeat steps 3 and 4 while beg <=end
* Step 3: set mid = (beg + end)/2
* Step 4: if a[mid] = val
* set pos = mid
* print pos
* go to step 6
* else if a[mid] > val
* set end = mid - 1
* else
* set beg = mid + 1
* [end of if]
* [end of loop]
* Step 5: if pos = -1
* print "value is not present in the array"
* [end of if]
* Step 6: exit

Program:

int binarySearch(int a[], int beg, int end, int val)

{

int mid;

if(end >= beg)

    {        mid = (beg + end)/2;

if(a[mid] == val)

{

return mid+1;

}

else if(a[mid] < val)

{

return binarySearch(a, mid+1, end, val);

}

else

{

return binarySearch(a, beg, mid-1, val);

}

}

return -1;

}

Output:

The Elements of the array are – 11 14 25 30 40 41 52 57 70

Element to be searched is – 40

Element is present at 5 position of array

17. Write an algorithm and C program for Bubble Sort -

**Algorithm:**

Step 1: begin BubbleSort(arr)

Step 2: for all array elements

Step 3: if arr[i] > arr[i+1]

Step 4: swap(arr[i], arr[i+1])

end if

end for

Step 5: return arr

Step 6: end BubbleSort

**Program:**

void bubbleSort(int \*A, int n)

{

int temp;

for (int i = 0; i < n - 1; i++) // For number of pass

{

for (int j = 0; j < n - 1 - i; j++) // For comparison in each pass

{

if (A[j] > A[j + 1])

{

temp = A[j];

A[j] = A[j + 1];

A[j + 1] = temp;

}

}

}

}

**Output:**

Input: 5 3 13 4 8 9

Output: 3 4 5 8 9 13

18. Write an algorithm and C program for Insertion Sort

**Algorithm:**

Insertion\_sort (A, size)

Step 1: Begin

step 2: Repeat step 3 to 5 for i=1 to size

step 3: set temp := A[i], j:=i-1

Step 4: Repeat while temp < A[j] and j>=0

Set A[j+1]= A[j]

set j:=j-1

[end inner loop]

Step 5: set A[j+1]:= temp

[ end of outer loop]

step 6: End

**Program:**

void insertionSort(int arr[], int n)

{

    int i, key, j;

    for (i = 1; i < n; i++) {

        key = arr[i];

        j = i - 1;

        /\* Move elements of arr[0..i-1], that are

          greater than key, to one position ahead

          of their current position \*/

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

**Output:**

Input: 5 3 13 4 8 9

Output: 3 4 5 8 9 13