Answer to the question no : 01

STEP :

1st Iteration :

1st Step : **7 2** 11 2 13 4 —-> 2 7 11 2 13 4
2nd Step : 2 **7 11** 2 13 4 —-> 2 7 11 2 13 4
3rd Step : 2 7 **11 2** 13 4 —-> 2 7 2 11 13 4
4th Step : 2 7 2 **11 13** 4 —-> 2 7 2 11 13 4
5th Step : 2 7 2 11 **13 4** —-> 2 7 2 11 4 13

2nd Iteration :

1st Step : **2 7** 2 11 4 13 —-> 2 7 2 11 4 13
2nd Step : 2 **7 2** 11 4 13 —-> 2 2 7 11 4 13
3rd Step : 2 2 **7 11** 4 13 —-> 2 2 7 11 4 13
4th Step : 2 2 7 **11 4** 13 —-> 2 2 7 4 11 13

3rd Iteration :

1st Step : **2 2** 7 4 11 13 —-> 2 2 7 4 11 13
2nd Step : 2 **2 7** 4 11 13 —-> 2 2 7 4 11 13
3rd Step : 2 2 **7 4** 11 13 —-> 2 2 4 7 11 13

4th Iteration :

1st Step : **2 2** 4 7 11 13 —-> 2 2 4 7 11 13
2nd Step : 2 **2 4** 7 11 13 —-> 2 2 4 7 11 13

After sort : 2 2 4 7 11 13

Answer to the question no : 02

Disadvantages of the Bubble Sort

● The main disadvantage of the bubble sort method is the time it requires. With a running time of O(n^2), it is highly inefficient for large data sets.also

Also disadvantage of bubble sort

1.Redundenty testing Even after being sort in a array, only iteration is taking place thereby increasing the time complexity

Solution : So we'll run an array size - i -1 to now that will reduce its redundancy testing and reduce the time complexity a bit and optimize the algorithm

.

2. Redundant iteration

Solution : To reduce the redundent iteration we will use a variable named flug value is 0 so that if there is only one variable can swap in the loop then the loop will run and fag value is 1 otherwise the loop will break this will reduce the number of iterations and make the itaretion much smaller and the time complexity will be reduced to a large extent.

```
for(int i=1; i<size; i++)
{
int flag= 0;
for(int j=0; j<size-i; j++)
{
if (array[j]>array[j+1])
{
swap(array[j],array[j+1]);
flag = 1;
}
cout<<"Step "<<j+1<<": ";
PrintArray(array,size);
}
cout<<endl;
if(flag == 0) break;
}
```

Optimizing the Algorithm

● One way to make the bubble sort more efficient is to take into account the fact that after the ith pass, the last i numbers will be in their correct places .

● This reduces the running time by half. ○ (n)(n/2)=(n^2)/2

● However, many argue that while this optimizes the running time for a worst-case scenario, this renders the term "bubble sort" invalid for this type of algorithm

Answer to the question no : 03

STEP :

Before sort : 7 2 9 2 7 4 5 1

Maximum : 9

Frequency : 0 1 2 0 1 1 0 2 0 1

Cumulative Sum : 0 1 3 3 4 5 5 7 7 8

After sort : 1 2 2 4 5 7 7 9

The reason of backward traversing :

After Traversing the Original array , we prefer from last Since we want to add Elements on their proper position so when we subtract the index , the Element will be added to lateral position.
But if we start traversing from beginning , then there will be no meaning for taking the cummulative sum since we are not adding according to the Elements placed. We are adding hap -hazardly which can be done even if we not take their cummulative sum.

Answer to the question no : 04

1st Iteration : -------------- Key : 1

Step 1 : 5 **1** 3 8 2 2 -------> 5 5 3 8 2 2
Step 2 : **5** 5 3 8 2 2 -------> 1 5 3 8 2 2

2nd Iteration : -------------- Key : 3

Step 1 : 1 5 **3** 8 2 2 -------> 1 5 5 8 2 2
Step 2 : 1 **5** 5 8 2 2 -------> 1 3 5 8 2 2

3rd Iteration : -------------- Key : 8

Step 1 : 1 3 5 **8** 2 2 -------> 1 3 5 8 2 2

4th Iteration : -------------- Key : 2

Step 1 : 1 3 5 8 **2** 2 -------> 1 3 5 8 8 2
Step 2 : 1 3 **5** 8 8 2 -------> 1 3 5 5 8 2
Step 3 : 1 **3** 5 5 8 2 -------> 1 3 3 5 8 2
Step 4 : 1 **3** 3 5 8 2 -------> 1 2 3 5 8 2

5th Iteration : -------------- Key : 2

Step 1 : 1 2 3 5 8 **2** -------> 1 2 3 5 8 8
Step 2 : 1 2 3 5 **8** 8 -------> 1 2 3 5 5 8
Step 3 : 1 2 3 **5** 5 8 -------> 1 2 3 3 5 8
Step 4 : 1 2 **3** 3 5 8 -------> 1 2 2 3 5 8

After sort : 1 2 2 3 5 8

Answer to the question no : 05

STEP 1:

Here upper bound 9 and lower bound 0 , so the mid is…..
Mid : 4

17 is greater than 6 , 6 is on left side. Because the array is sorted on ascending order.

STEP 2:

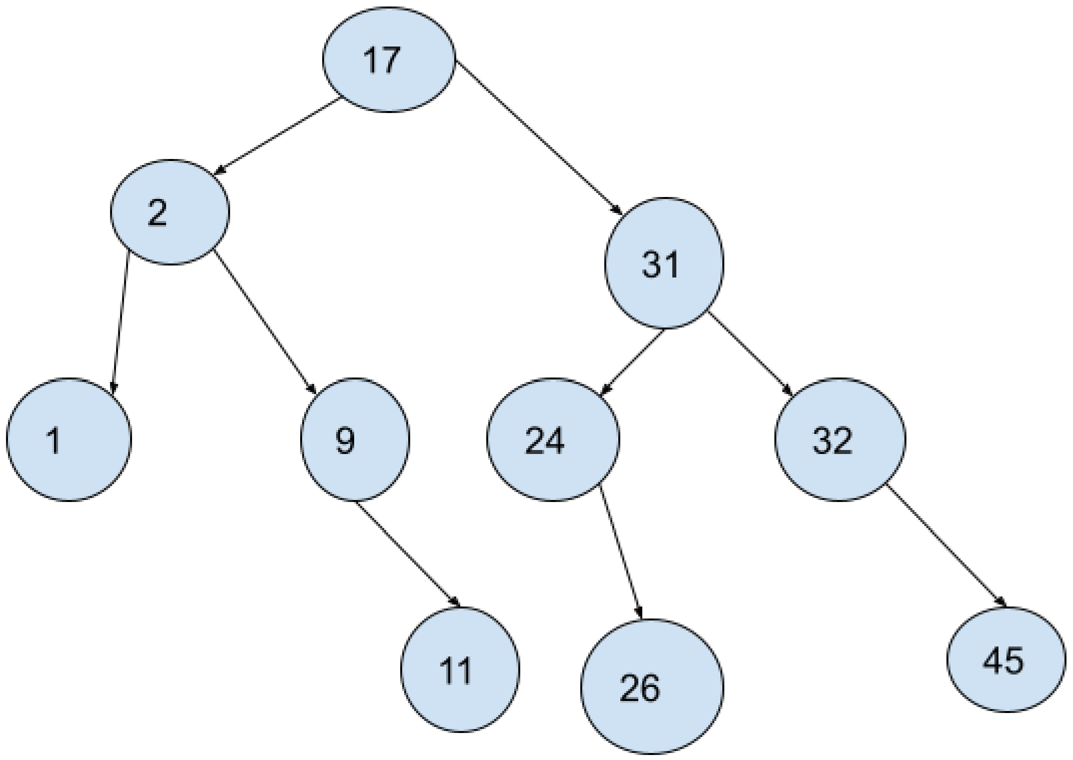
Here upper bound 4 and lower bound 0 , so the mid is…..
Mid : 2

9 is greater than 6 , 6 is on left side.

STEP 3:

Here upper bound 2 and lower bound 0 , so the mid is…..
Mid : 1
2 is less than 6 , so **6 is not found.**

Binary Search Tree :



Answer to the question no : 06

Arr[50][45]
Given that arr[0][0] = 760.

Now we Find out the location of arr[6][13]

= 760+4[45(6-0)+(13-0)
= 760+4[270+13]
= 760+4283
= 760+1132
= 1892

Answer to the question no : 07

Three Limitations of the array :

1. In an array, elements are stored in contiguous memory location or consecutive manner in the memory.

2. In array, Insertion and Deletion operation takes more time, as the memory locations are consecutive and fixed.

3. Memory is allocated as soon as the array is declared, at compile time. It's also known as Static Memory Allocation.

These can be solved by the use of Linked List :

1. In a linked list, new elements can be stored anywhere in the memory.

2. Address of the memory location allocated to the new element is stored in the previous node of the linked list, hence forming a link between the two nodes/elements. In case of linked list, a new element is stored at the first free and available memory location, with only a single overhead step of storing the address of memory location in the previous node of linked list. Insertion and Deletion operations are fast in the linked list.

3. Memory is allocated at runtime, as and when a new node is added. It's also known as Dynamic Memory Allocation.

Answer to the question no : 08

Insert an element at the head and Tail of the Linked List :

```
#include<bits/stdc++.h>

using namespace std;

class Node
{
public:
    int value;
    Node* next;

    Node(int val)
    {
        value= val;
        next = NULL;
    }
};

void insert_At_Head(Node* &head, int val)
{
    Node *newNode = new Node(val);

    newNode->next = head;
    head = newNode;
}


void insert_At_Tail(Node* &head, int val)
{
    Node *newNode = new Node(val);

    if(head==NULL)
    {
        head = newNode;
        return;
    }
    Node *temp = head;

    while(temp->next!=NULL)
    {
        temp = temp->next;
    }

    temp->next = newNode;
}


void display(Node* n)
{
    while(n!=NULL)
    {
        cout<< n->value;
        if(n->next!=NULL) cout<<" -> ";
        n = n->next;
    }

    cout<<endl<<endl;

}

int main()
{
    Node *head = NULL;

    int n;

    cout<<"Choice 1: Insertion at Head" << endl << "Choice 2: Insertion at Tail" << endl
<<"Choice 3: Exit" << endl;
    int choice = 2;


    while(choice == 1 || choice == 2)
    {
        cout<<"Enter The Value : ";
        cin>>n;
        cout<<endl;
        if(choice == 1) insert_At_Head(head,n);
        else if(choice == 2) insert_At_Tail(head,n);
        cout<<"Next Choice :";
        cin>>choice;
        cout<<endl;

    }

    display(head);
    return 0;
}
```

Print a linear linked list in a reverse way :

```
#include<bits/stdc++.h>

using namespace std;

class Node
{
public:
   int value;
   Node* next;

   Node(int val)
   {
      value= val;
      next = NULL;
   }
};

void insert_At_Head(Node* &head, int val)
{
   Node *newNode = new Node(val);

   newNode->next = head;
   head = newNode;
}


void insert_At_Tail(Node* &head, int val)
{
   Node *newNode = new Node(val);
```

```
    if(head==NULL)
    {
        head = newNode;
        return;
    }

    Node *temp = head;

    while(temp->next!=NULL)
    {
        temp = temp->next;
    }

    temp->next = newNode;
}

void printReverse(Node* head)
{

    if (head == NULL)
    return;

    printReverse(head->next);

    cout << head->value << " ";
}

int main()
{
    Node *head = NULL;

    int n;

    cout<<"Choice 1: Insertion at Head" << endl << "Choice 2: Insertion at Tail" << endl
<<"Choice 3: Exit" << endl;
    int choice = 2;


    while(choice == 1 || choice == 2)
    {
        cout<<"Enter The Value : ";
        cin>>n;
        cout<<endl;
        if(choice == 1) insert_At_Head(head,n);
        else if(choice == 2) insert_At_Tail(head,n);
        cout<<"Next Choice :";
        cin>>choice;
        cout<<endl;

    }

    printReverse(head);
    return 0;
}
```

Answer to the question no : 09

A. What is the value of Head?

Ans : 500

B. What is the value of ? marked address location?

Ans : 1020

C. What will be the value of Head->Next->Next->Value?

Ans : 1

D. What will be the value of **Sum** following pseudocode snippets?

Ans : -1