Answer to the question no : 01

```
#include <bits/stdc++.h>
using namespace std;

void PrintArray(int array [],int size){

    for(int i=0; i<size; i++){
        cout<< array[i]<<" ";
    }
    cout<<endl;
}

int main() {

    int size;
    cout<<"Please Enter Your Array Size : ";
    cin>>size;

    int array[size];

    cout<<"Please Enter Your Array Value : ";
    for (int i=0; i<size; i++){
        cin>>array[i];
    }

    cout << "Before sort : ";
    PrintArray(array,size);
    cout<<endl;

    for(int i=1; i<size; i++)
    {
        int flag= 0;
        cout<<"Iteration No : "<< i <<endl<<endl;
        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;
    }


    cout << "After sort !";
    PrintArray(array,size);
    cout<<endl;

    return 0;
}
```

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 in
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

```
#include<iostream>
#define MAX 255
using namespace std;

void countSort(int array[], int size) {
    int output[MAX];
    int count[MAX];
    int max = array[0];

    // Step-01 Finding Max
    for (int i = 1; i < size; i++) { if (array[i] > max)
            max = array[i];
    }
```

```
    // step-02 Initialize of array to 0
    for (int i = 0; i <= max; ++i) {
        count[i] = 0;
    }

    // step-03 Frequency Calculation
    for (int i = 0; i < size; i++) {
        count[array[i]]++;
    }

    // step -04 Store the cummulative sum
    for (int i = 1; i <= max; i++)
          {
                  count[i] += count[i - 1];
          }

          // step-05 Final array --> Backword Travarsal of Basic array
          for (int i = size - 1; i >= 0; i--)
    {
        output[count[array[i]] - 1] = array[i];
        count[array[i]]--;
    }
    for (int i = 0; i < size; i++) {
        array[i] = output[i];
    }
}

// print an array
void display(int array[], int size) {
    for (int i = 0; i < size; i++)
        cout << array[i] << " ";
    cout << endl;
}


int main() {
    int array[] = {7,2,9,2,7,4,5,1};
    int n = sizeof(array) / sizeof(array[0]);
```

```
    countSort(array, n);

    display(array, n);

    return 0;
}
```

Answer to the question no : 04

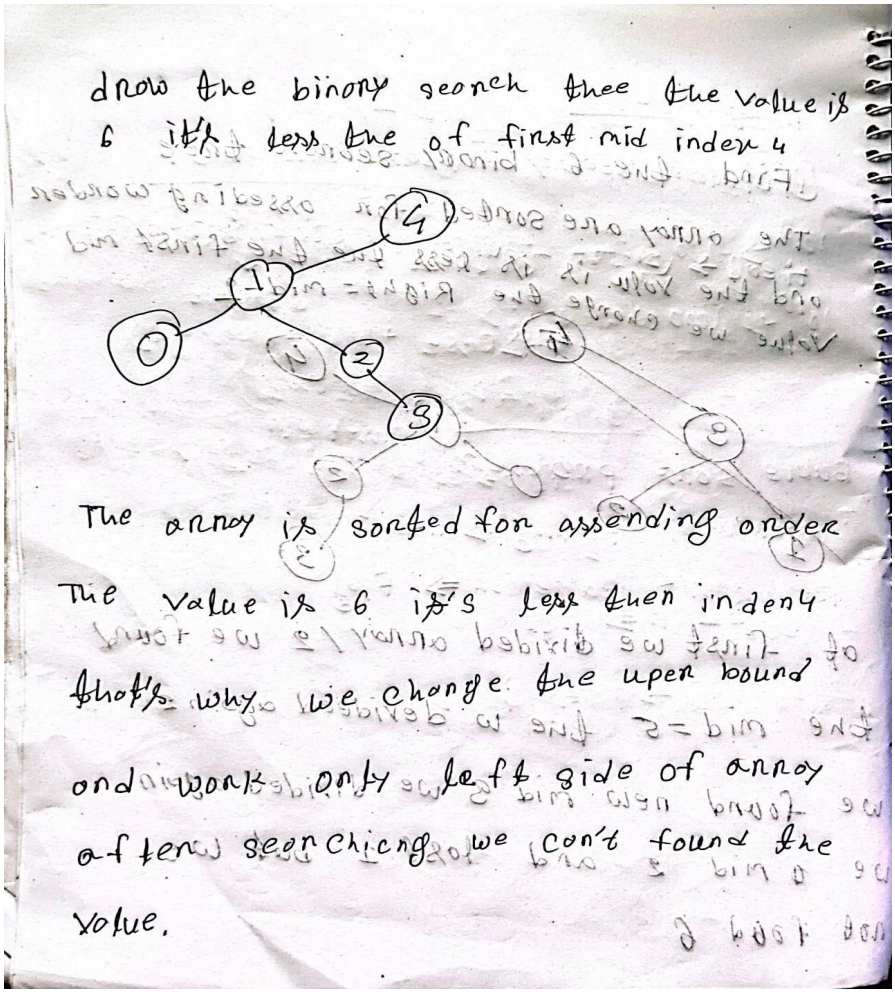
```
#include <bits/stdc++.h>
using namespace std;

void PrintArray(int array [],int size){

    for(int i=0; i<size; i++){
        cout<< array[i]<<" ";
    }
    cout<<endl;
}

int main() {

    int array[] = {5,1,3,8,2};
    int size = sizeof(array) / sizeof(array[0]);
    cout << "Before sort : ";
    PrintArray(array,size);
    for(int i=1; i<size; i++)
    {
        int key= array[i];
        cout<<endl<<"i : "<< i <<" key : "<< key <<endl;

        int j = i-1;

        while (array[j]>key && j>=0)

        {
                array[j+1]=array[j];
                 PrintArray(array,size);
                 j--;
        }
```

```
            array[j+1]=key;
            PrintArray(array,size);

    }
    cout <<endl<< "After sort ";
    PrintArray(array,size);
    cout<<endl;

    return 0;
}
```

Answer to the question no : 05

```
#include <bits/stdc++.h>
using namespace std;
int binarySearch(int arr[], int p, int r, int num) {
  if (p <= r) {
    int mid = (p + r)/2;
    if (arr[mid] == num)
      return mid ;
    if (arr[mid] > num)
      return binarySearch(arr, p, mid-1, num);
    if (arr[mid] < num)
      return binarySearch(arr, mid+1, r, num);
  }
  return -1;
}
int main(void) {
  int arr[] = {1, 2,9,11,17,24,26,31,32,45};
  int n = sizeof(arr)/ sizeof(arr[0]);
  int num;
  cout << "Enter the number to search: \n";
  cin >> num;
  int index = binarySearch (arr, 0, n-1, num);
  if(index == -1){
    cout<< num <<" is not present in the array";
  }else{
    cout<< num <<" is present at index "<< index <<" in the array";
  }
```

```
    return 0;
}
```



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.

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

```
#include <iostream>
using namespace std;

class node {
public:
    int data;
    node* next;

    node(int value)
    {
        data = value;

        next = NULL;
    }
};

void insertathead(node*& head, int val)
{
    node* n = new node(val);
    n->next = head;
    head = n;
}
```

```
void insertafter(node* head, int key, int val)
{
    node* n = new node(val);
    if (key == head->data) {
        n->next = head->next;
        head->next = n;
        return;
    }
}
void insertattail(node*& head, int val)
{
    node* n = new node(val);
    if (head == NULL) {
        head = n;
        return;
    }

    node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = n;
}


void print(node*& head)
{
    node* temp = head;

    while (temp != NULL) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

int main()
{
    node* head = NULL;

    insertathead(head, 1);
```

```
    insertathead(head, 2);
    cout << "After insertion at head: ";
    print(head);
    cout << endl;

    insertattail(head, 4);
    insertattail(head, 5);
    cout << "After insertion at tail: ";
    print(head);
    cout << endl;


      return 0;
}
```

Answer to the question no : 09

What is the value of Head?
Answer : 5
What is the value of ? marked address location?
Answer : 3
What will be the value of Head->Next->Next->Value?
Answer : 14
What will be the value of **Sum** following pseudocode snippets?
Answer: -2