Day 5

Name: -Shivam Yadav

UID: -22BCS15259

Date: -26/12/24

Section : - 620-A

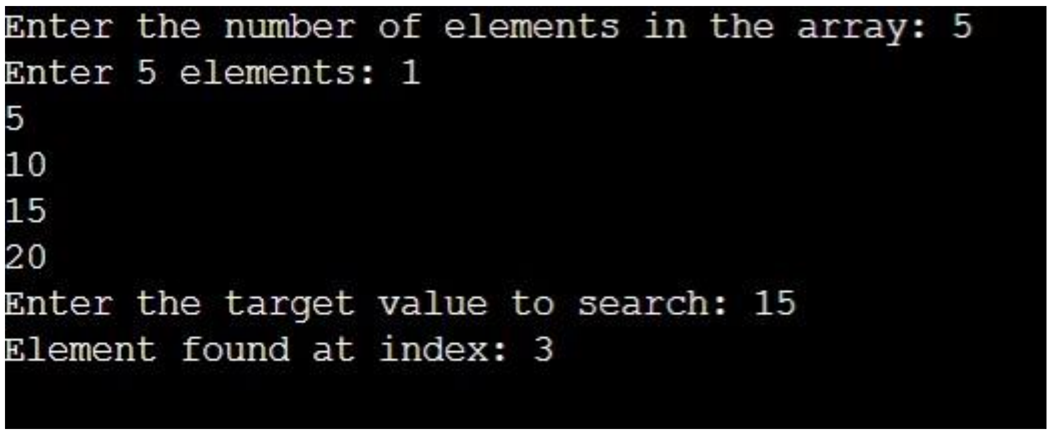
Question 1: -Implement the linear search algorithm to find the target value in the array #include <iostream> using namespace std; void inputArray(int arr[], int& size)

```
{
   cout << "Enter the number of elements in the array: ";
cin >> size;
   cout << "Enter " << size << " elements: ";
for (int i = 0; i < size; i++)
   {
      cin >> arr[i];
   }
}
int linearSearch(int arr[], int size, int target)
{
   for (int i = 0; i < size; i++)
   {
      if (arr[i] == target)
```

```
        {
return i;
        }
    }
    return -1;
}
int main()
{
    int size, target;     int
arr[100];
inputArray(arr, size);
    cout << "Enter the target value to search: ";
cin >> target;
    int result = linearSearch(arr, size, target);

    if (result != -1)
    {
        cout << "Element found at index: " << result << endl;
    } else
    {
        cout << "Element not found." << endl;
    }

    return 0;
```

}

OUTPUT: -



```
Enter the number of elements in the array: 5
Enter 5 elements: 1
5
10
15
20
Enter the target value to search: 15
Element found at index: 3
```

Question 2: -

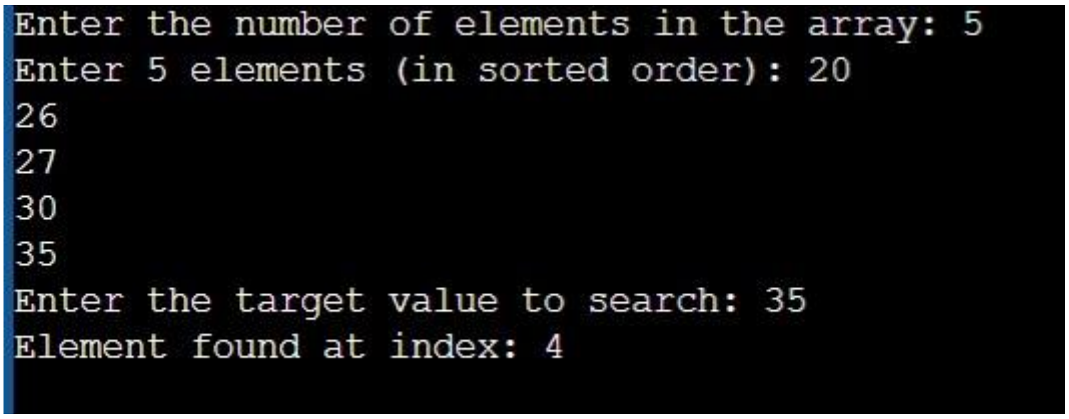
write a function to implement binary search on sorted array the function should return the index of the target value

```
#include <iostream>

#include <algorithm>   using

namespace std;


int binary(int arr[], int size, int target) {

int left = 0;     int right = size - 1;


   while (left <= right) {

      int mid = left + (right - left) / 2;

if (arr[mid] == target) {

return mid;

      }
```

```
        if (arr[mid] < target) {
left = mid + 1;
        }        else {
right = mid - 1;
        }
    }

    return -1;
}
int main() {     int
size, target;
    cout << "Enter the number of elements in the array: ";
cin >> size;     int arr[size];
    cout << "Enter " << size << " elements (in sorted order): ";
for (int i = 0; i < size; i++) {         cin >> arr[i];
    }
    cout << "Enter the target value to search:
    "; cin >> target; sort(arr, arr + size);
    int result = binary(arr, size, target);
    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found" << endl;
    }
```

```
   return 0;
}
```

OUTPUT: -



```
Enter the number of elements in the array: 5
Enter 5 elements (in sorted order): 20
26
27
30
35
Enter the target value to search: 35
Element found at index: 4
```

Question3: -

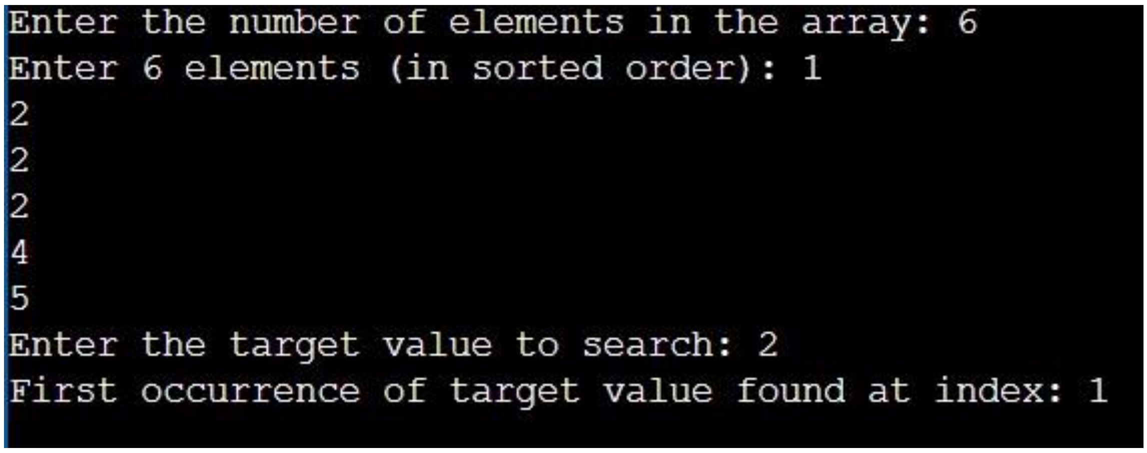
write a program of binary search to find the first occurrence of target value in sorted element #include <iostream> using namespace std;

```
int binarySearchFirstOccurrence(int arr[], int size, int target) {
int left = 0;    int right = size - 1;    int result = -1;


   while (left <= right) {
      int mid = left + (right - left) / 2;
if (arr[mid] == target) {
result = mid;            right = mid -
1;
      }
```

```
        else if (arr[mid] > target) {
right = mid - 1;
        }
                 else {
left = mid + 1;
        }
    }
    return result;
}
int main() {     int
size, target;
    cout << "Enter the number of elements in the array:
    "; cin >> size; int arr[size];
    cout << "Enter " << size << " elements (in sorted order): ";
                                                          for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    cout << "Enter the target value to search: ";
cin >> target;
    int result = binarySearchFirstOccurrence(arr, size, target);
if (result != -1) {
        cout << "First occurrence of target value found at index: " <<
result << endl;
```

```
    } else {
        cout << "Element not found" << endl;
    }
    return 0;
}
```

OUTPUT: -



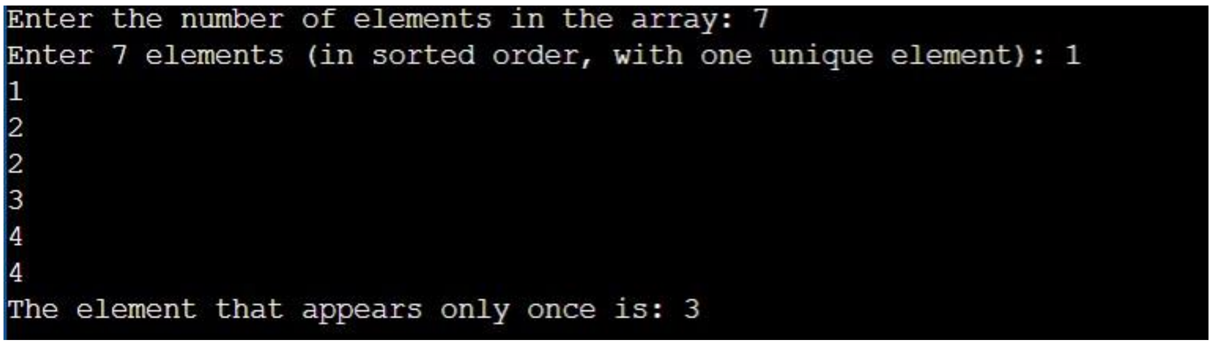
```
Enter the number of elements in the array: 6
Enter 6 elements (in sorted order): 1
2
2
2
4
5
Enter the target value to search: 2
First occurrence of target value found at index: 1
```

Question 4: - find the element that appear only once in a sorted array.

```
#include <iostream>
using namespace std;
int findSingleElement(int arr[], int size) {
int left = 0;     int right = size - 1;     while
(left < right) {
        int mid = left + (right - left) / 2;
if (mid % 2 == 1) {           mid--;
        }
        if (arr[mid] == arr[mid + 1]) {
left = mid + 2;
```

```
        } else {
right = mid;
        }
    }
    return arr[left];
}

int main() {
int size;
    cout << "Enter the number of elements in the array:
    "; cin >> size; int arr[size];
    cout << "Enter " << size << " elements (in sorted order, with one
unique element): ";
    for (int i = 0; i < size; i++) {
cin >> arr[i];
    }
    int result = findSingleElement(arr, size);
    cout << "The element that appears only once is: " << result <<
endl;

    return 0;
}
```

OUTPUT: -



```
Enter the number of elements in the array: 7
Enter 7 elements (in sorted order, with one unique element): 1
1
2
2
3
4
4
The element that appears only once is: 3
```

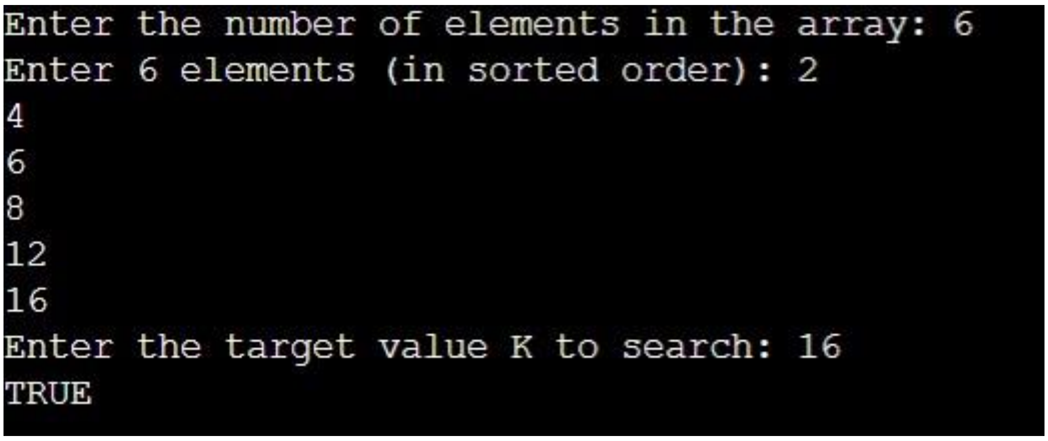
Question 5: -

given an array sorted in ascending order and an integer K return true if k is present in the array other wise false

```
#include <iostream> using

namespace std;


bool binarySearch(int arr[], int size, int K) {

int left = 0;

   int right = size - 1;

while (left <= right) {

      int mid = left + (right - left) / 2;


      if (arr[mid] == K) {

return true;

      }

      else if (arr[mid] < K) {

left = mid + 1;

      }

else {
```

```
            right = mid - 1;
        }
    }

    return false;
}
int main() {
int size, K;
    cout << "Enter the number of elements in the array: ";
cin >> size;     int arr[size];
    cout << "Enter " << size << " elements (in sorted order): ";
for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    cout << "Enter the target value K to search: ";
cin >> K;
    if (binarySearch(arr, size, K)) {
cout << "TRUE" << endl;
    } else {
        cout << "FALSE" << endl;
    }
    return 0;
}
```

OUTPUT: -



```
Enter the number of elements in the array: 6
Enter 6 elements (in sorted order): 2
4
6
8
12
16
Enter the target value K to search: 16
TRUE
```

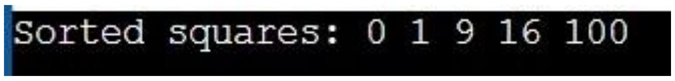
Question 6: -

given an integer array number sorted in non decreasing order return an array of square of each number sorted in non decreasing order.

```
#include <iostream>

#include <vector>

#include <algorithm> using

namespace std;


vector<int> sortedSquares(const vector<int>& nums) {

int n = nums.size();     vector<int> result(n);     int left =

0, right = n - 1;     int pos = n - 1;      while (left <= right)

{

      int leftSquare = nums[left] * nums[left];

int rightSquare = nums[right] * nums[right];
```

```
        if (leftSquare > rightSquare) {
result[pos] = leftSquare;
left++;        } else {
            result[pos] = rightSquare;
right--;
        }
pos--;
    }
    return result;
}
int main() {
    vector<int> nums = {-4, -1, 0, 3, 10};
vector<int> result = sortedSquares(nums);
cout << "Sorted squares: ";    for (int num :
result) {        cout << num << " ";
    }
    cout << endl;


    return 0;
}
```

OUTPUT: -



```
Sorted squares: 0 1 9 16 100
```

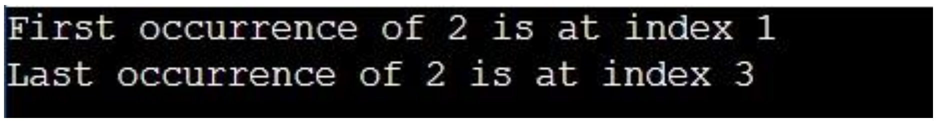
Question 7:-

Left most and Right most index.

```
#include <iostream>

#include <vector> using

namespace std;


int findFirstOccurrence(const vector<int>& arr, int X) {

int low = 0, high = arr.size() - 1;     int result = -1;


   while (low <= high) {

      int mid = low + (high - low) / 2;


      if (arr[mid] == X) {

result = mid;             high =

mid - 1;          } else if

(arr[mid] < X) {            low

= mid + 1;

      } else {

high = mid - 1;

      }

   }

   return result;

}

int findLastOccurrence(const vector<int>& arr, int X) {

int low = 0, high = arr.size() - 1;     int result = -1;
```

```
   while (low <= high) {        int mid
= low + (high - low) / 2;

      if (arr[mid] == X) {
result = mid;            low
= mid + 1;
      } else if (arr[mid] < X) {
         low = mid + 1;
      } else {
high = mid - 1;
      }
   }
   return result;
}

int main() {
   vector<int> arr = {1, 2, 2, 2, 3, 4, 5, 5, 5};
int X = 2;

   int first = findFirstOccurrence(arr, X);
int last = findLastOccurrence(arr, X);

   if (first != -1 && last != -1) {
        cout << "First occurrence of " << X << " is at index " << first <<
endl;
        cout << "Last occurrence of " << X << " is at index " << last <<
endl;
    } else {
        cout << X << " is not present in the array." << endl;
    }

    return 0;
}
```

OUTPUT: -



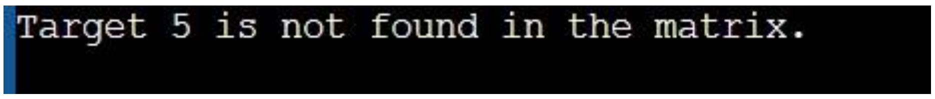
```
First occurrence of 2 is at index 1
Last occurrence of 2 is at index 3
```

Question 8:-

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

bool searchMatrix(vector<vector<int>>& matrix, int target) {
int m = matrix.size();        int n = matrix[0].size();

    int low = 0, high = m * n - 1;

    while (low <= high) {        int mid
= low + (high - low) / 2;
```

```
        int row = mid / n;
int col = mid % n;

        if (matrix[row][col] == target) {
return true;
        } else if (matrix[row][col] < target) {
low = mid + 1;
        } else {
high = mid - 1;
        }
    }

    return false;
}

int main() {
    vector<vector<int>> matrix = {
        {1, 4, 7, 11},
        {2, 5, 8, 12},
        {3, 6, 9, 16},
        {10, 13, 14, 17}
    };
    int target = 5;

    if (searchMatrix(matrix, target)) {
        cout << "Target " << target << " is found in the matrix." << endl;
    } else {
        cout << "Target " << target << " is not found in the matrix." <<
endl;
    }

    return 0;
}
```

OUTPUT: -



```
Target 5 is not found in the matrix.
```

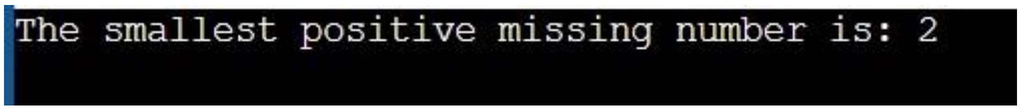
Question 9: -

Smallest Positive Missing Number.

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

int firstMissingPositive(vector<int>& arr) {
int n = arr.size();     for (int i = 0; i < n; i++) {
        while (arr[i] > 0 && arr[i] <= n && arr[arr[i] - 1] != arr[i]) {
swap(arr[i], arr[arr[i] - 1]);
```

```
        }
    }

    for (int i = 0; i < n; i++) {
if (arr[i] != i + 1) {
return i + 1;
        }
    }
    return n + 1;
}

int main() {
    vector<int> arr = {3, 4, -1, 1};

    cout << "The smallest positive missing number is: " <<
firstMissingPositive(arr) << endl;

    return 0;
}
```

OUTPUT: -



```
The smallest positive missing number is: 2
```

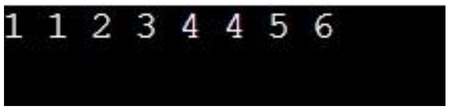
Question 10: - Merge

k Sorted Lists.

```
#include <iostream>

#include <vector>

#include <queue> using

namespace std; struct

ListNode {

   int val;

   ListNode *next;

   ListNode(int x) : val(x), next(NULL) {}

};

struct Compare {

   bool operator()(ListNode* a, ListNode* b) {

return a->val > b->val;

   }

};

class Solution { public:

   ListNode*      mergeKLists(vector<ListNode*>&      lists)      {
priority_queue<ListNode*,    vector<ListNode*>,    Compare>
minHeap;

      for (auto list : lists) {           if

(list) minHeap.push(list);

      }

      ListNode* dummy = new ListNode(0);

ListNode* current = dummy;        while

(!minHeap.empty()) {           ListNode*
node = minHeap.top();
minHeap.pop();           current->next =
node;           current = current->next;

          if (node->next) minHeap.push(node->next);
       }
       return dummy->next;
   }
};
void printList(ListNode* head) {
while (head) {        cout <<
head->val << " ";        head =
head->next;
   }
   cout << endl;
}
ListNode* createList(const vector<int>& nums) {
   ListNode* dummy = new ListNode(0);
ListNode* current = dummy;     for (int
num : nums) {        current->next = new
ListNode(num);        current = current-
>next;
   }
   return dummy->next;
```

```
}
int main() {
   vector<ListNode*>        lists        =        {
createList({1, 4, 5}),           createList({1, 3,
4}),        createList({2, 6})
   };
   Solution solution;
   ListNode* mergedList = solution.mergeKLists(lists);
printList(mergedList);     return 0;
}
```

OUTPUT: -



```
1 1 2 3 4 4 5 6
```