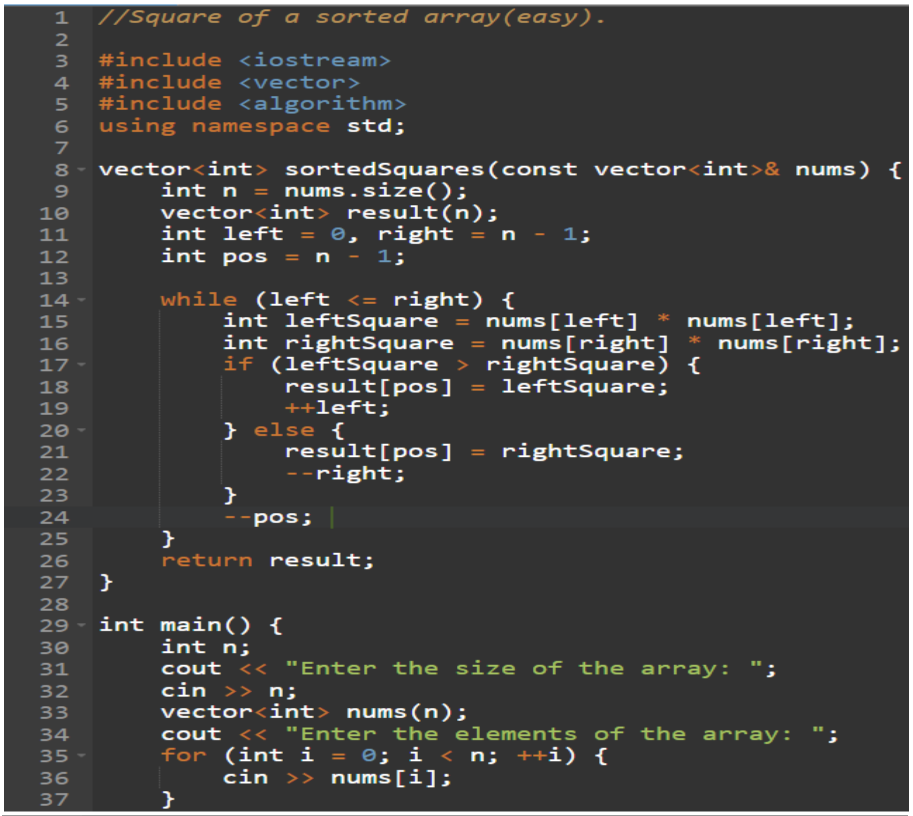
WINTER DOMAIN CAMP

NAME: Vikram Kumar **UID:22BCS12220**

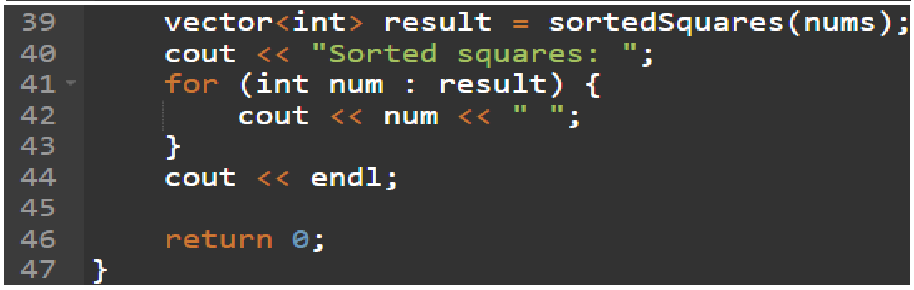
SECTION: KPIT-901-B **DATE: 26/12/2024**

Q1. Square of a sorted array(Easy)

Sol.

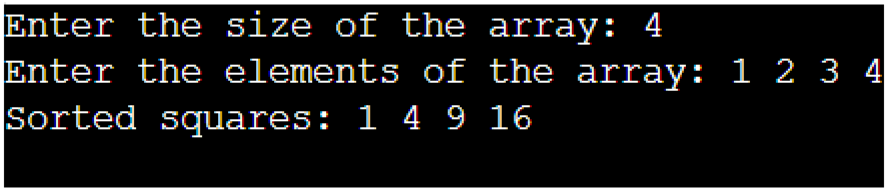


```cpp
//Square of a sorted array(easy).

#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() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }
```



```cpp
    vector<int> result = sortedSquares(nums);
    cout << "Sorted squares: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

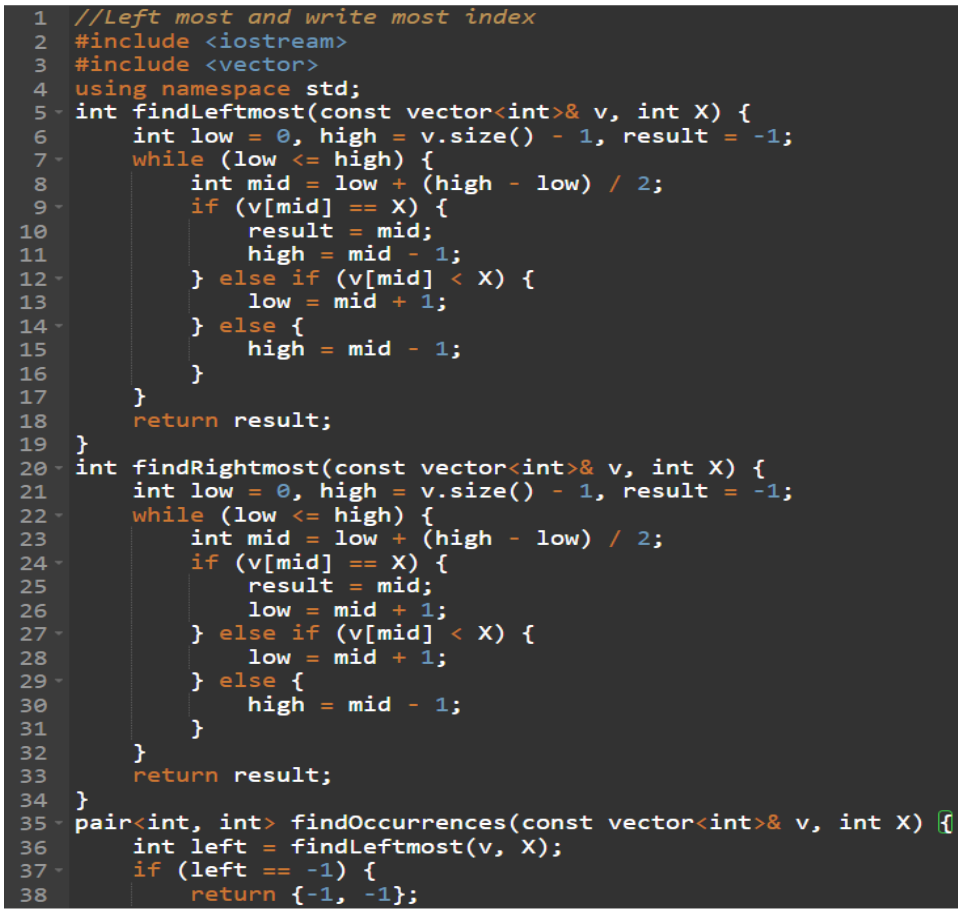
OUTPUT:



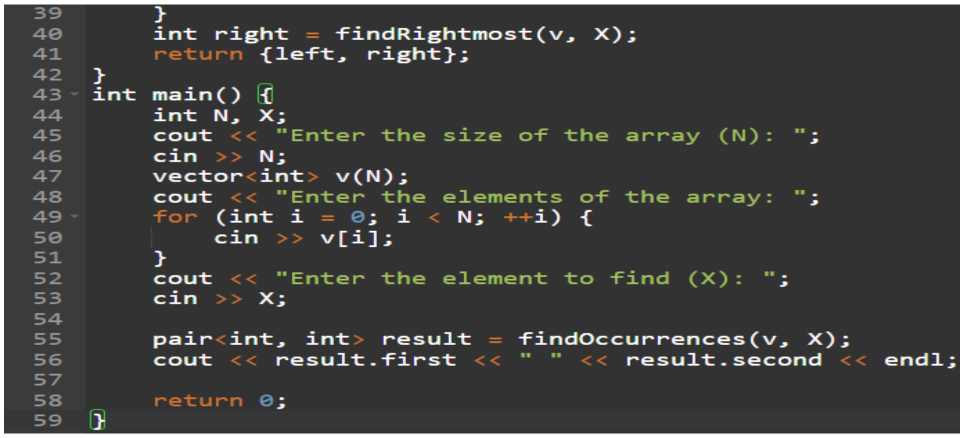
```
Enter the size of the array: 4
Enter the elements of the array: 1 2 3 4
Sorted squares: 1 4 9 16
```

Q2. Left most and write most index(medium)

Sol.

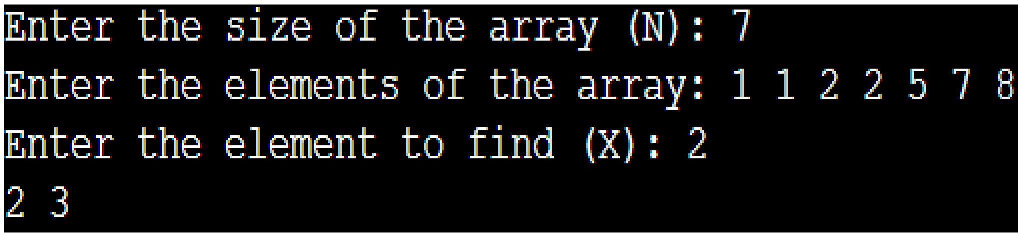


```cpp
//Left most and write most index
#include <iostream>
#include <vector>
using namespace std;
int findLeftmost(const vector<int>& v, int X) {
    int low = 0, high = v.size() - 1, result = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (v[mid] == X) {
            result = mid;
            high = mid - 1;
        } else if (v[mid] < X) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return result;
}
int findRightmost(const vector<int>& v, int X) {
    int low = 0, high = v.size() - 1, result = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (v[mid] == X) {
            result = mid;
            low = mid + 1;
        } else if (v[mid] < X) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return result;
}
pair<int, int> findOccurrences(const vector<int>& v, int X) {
    int left = findLeftmost(v, X);
    if (left == -1) {
        return {-1, -1};
```



```cpp
    }
    int right = findRightmost(v, X);
    return {left, right};
}
int main() {
    int N, X;
    cout << "Enter the size of the array (N): ";
    cin >> N;
    vector<int> v(N);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < N; ++i) {
        cin >> v[i];
    }
    cout << "Enter the element to find (X): ";
    cin >> X;

    pair<int, int> result = findOccurrences(v, X);
    cout << result.first << " " << result.second << endl;

    return 0;
}
```

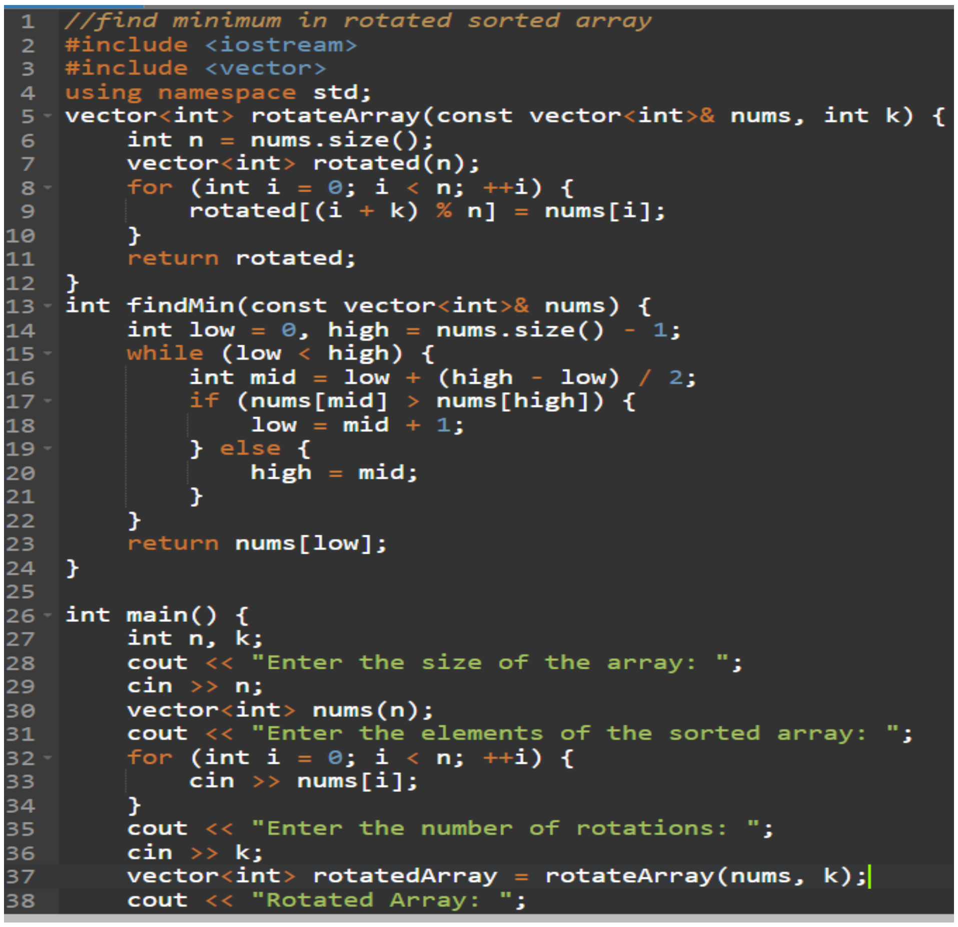
OUTPUT:



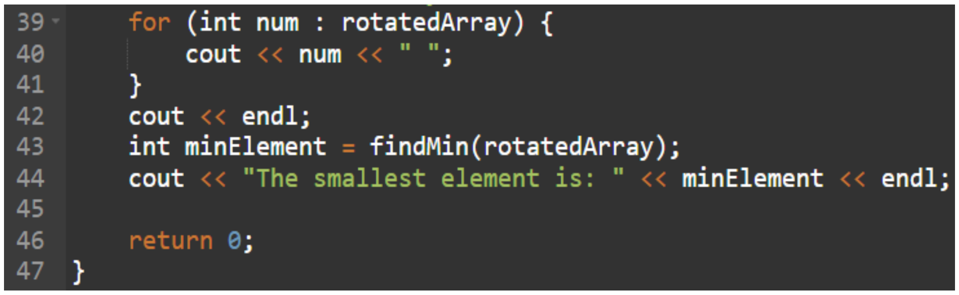
```
Enter the size of the array (N): 7
Enter the elements of the array: 1 1 2 2 5 7 8
Enter the element to find (X): 2
2 3
```

Q3. Find minimum in rotated sorted array(medium).

Sol.

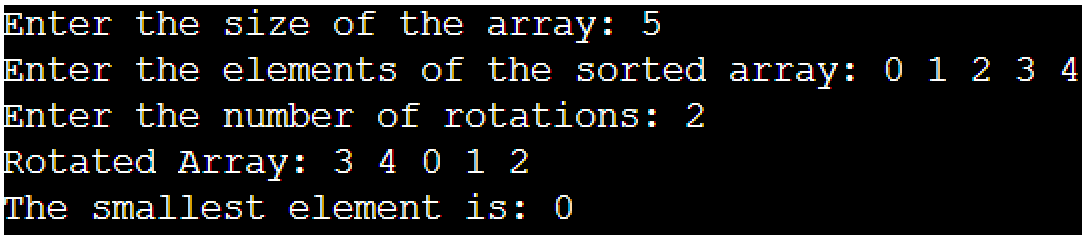


```cpp
//find minimum in rotated sorted array
#include <iostream>
#include <vector>
using namespace std;
vector<int> rotateArray(const vector<int>& nums, int k) {
    int n = nums.size();
    vector<int> rotated(n);
    for (int i = 0; i < n; ++i) {
        rotated[(i + k) % n] = nums[i];
    }
    return rotated;
}
int findMin(const vector<int>& nums) {
    int low = 0, high = nums.size() - 1;
    while (low < high) {
        int mid = low + (high - low) / 2;
        if (nums[mid] > nums[high]) {
            low = mid + 1;
        } else {
            high = mid;
        }
    }
    return nums[low];
}

int main() {
    int n, k;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the sorted array: ";
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }
    cout << "Enter the number of rotations: ";
    cin >> k;
    vector<int> rotatedArray = rotateArray(nums, k);
    cout << "Rotated Array: ";
```



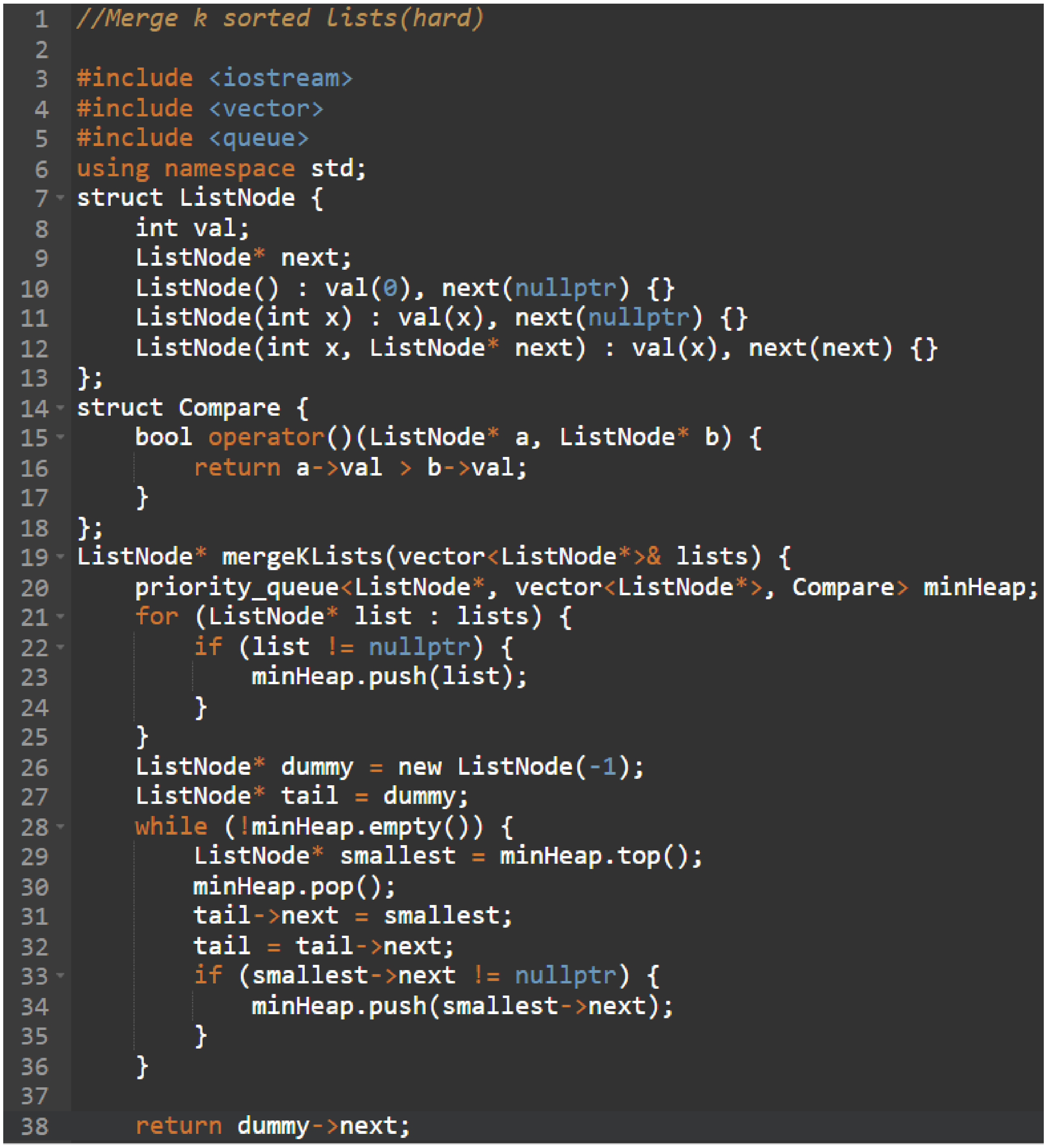
```cpp
    for (int num : rotatedArray) {
        cout << num << " ";
    }
    cout << endl;
    int minElement = findMin(rotatedArray);
    cout << "The smallest element is: " << minElement << endl;

    return 0;
}
```

OUTPUT:

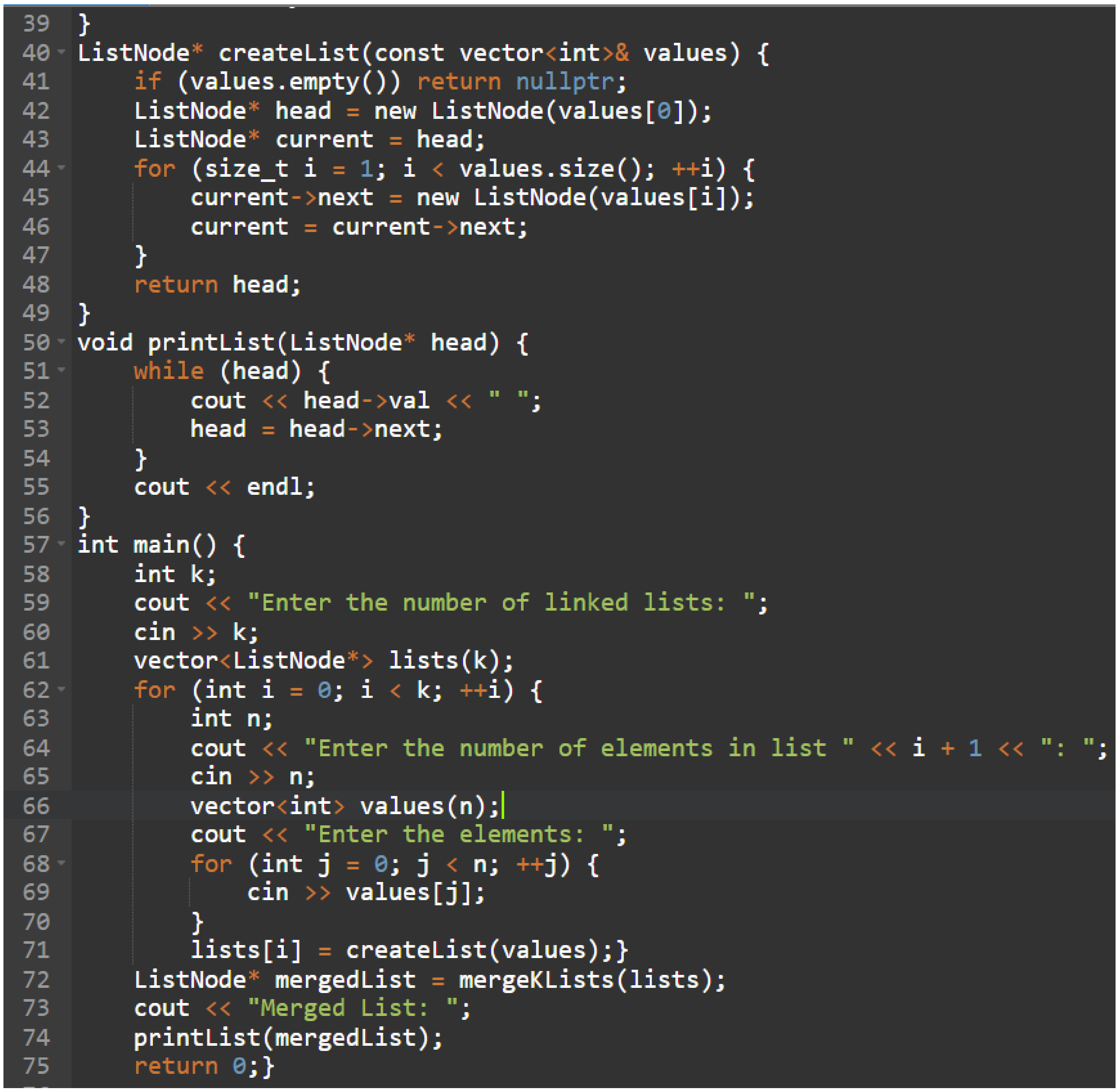


```
Enter the size of the array: 5
Enter the elements of the sorted array: 0 1 2 3 4
Enter the number of rotations: 2
Rotated Array: 3 4 0 1 2
The smallest element is: 0
```

Q4.Merge k sorted lists(hard)
Sol.

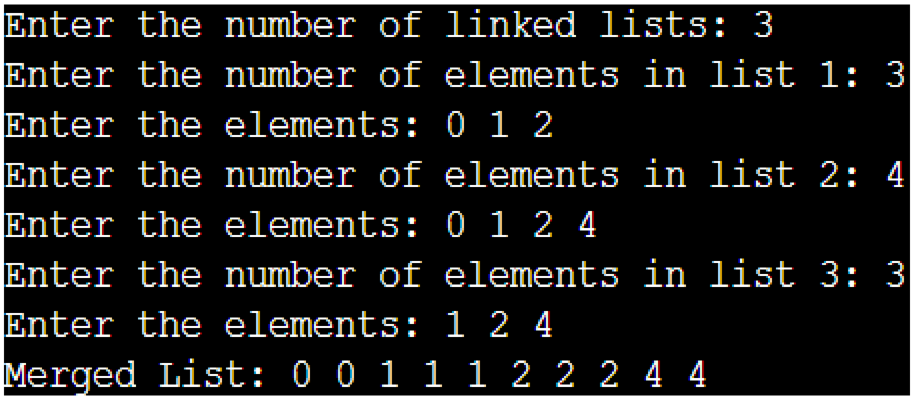


```cpp
//Merge k sorted lists(hard)

#include <iostream>
#include <vector>
#include <queue>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};
struct Compare {
    bool operator()(ListNode* a, ListNode* b) {
        return a->val > b->val;
    }
};
ListNode* mergeKLists(vector<ListNode*>& lists) {
    priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;
    for (ListNode* list : lists) {
        if (list != nullptr) {
            minHeap.push(list);
        }
    }
    ListNode* dummy = new ListNode(-1);
    ListNode* tail = dummy;
    while (!minHeap.empty()) {
        ListNode* smallest = minHeap.top();
        minHeap.pop();
        tail->next = smallest;
        tail = tail->next;
        if (smallest->next != nullptr) {
            minHeap.push(smallest->next);
        }
    }

    return dummy->next;
```



```cpp
}
ListNode* createList(const vector<int>& values) {
    if (values.empty()) return nullptr;
    ListNode* head = new ListNode(values[0]);
    ListNode* current = head;
    for (size_t i = 1; i < values.size(); ++i) {
        current->next = new ListNode(values[i]);
        current = current->next;
    }
    return head;
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}
int main() {
    int k;
    cout << "Enter the number of linked lists: ";
    cin >> k;
    vector<ListNode*> lists(k);
    for (int i = 0; i < k; ++i) {
        int n;
        cout << "Enter the number of elements in list " << i + 1 << ": ";
        cin >> n;
        vector<int> values(n);
        cout << "Enter the elements: ";
        for (int j = 0; j < n; ++j) {
            cin >> values[j];
        }
        lists[i] = createList(values);}
    ListNode* mergedList = mergeKLists(lists);
    cout << "Merged List: ";
    printList(mergedList);
    return 0;}
```

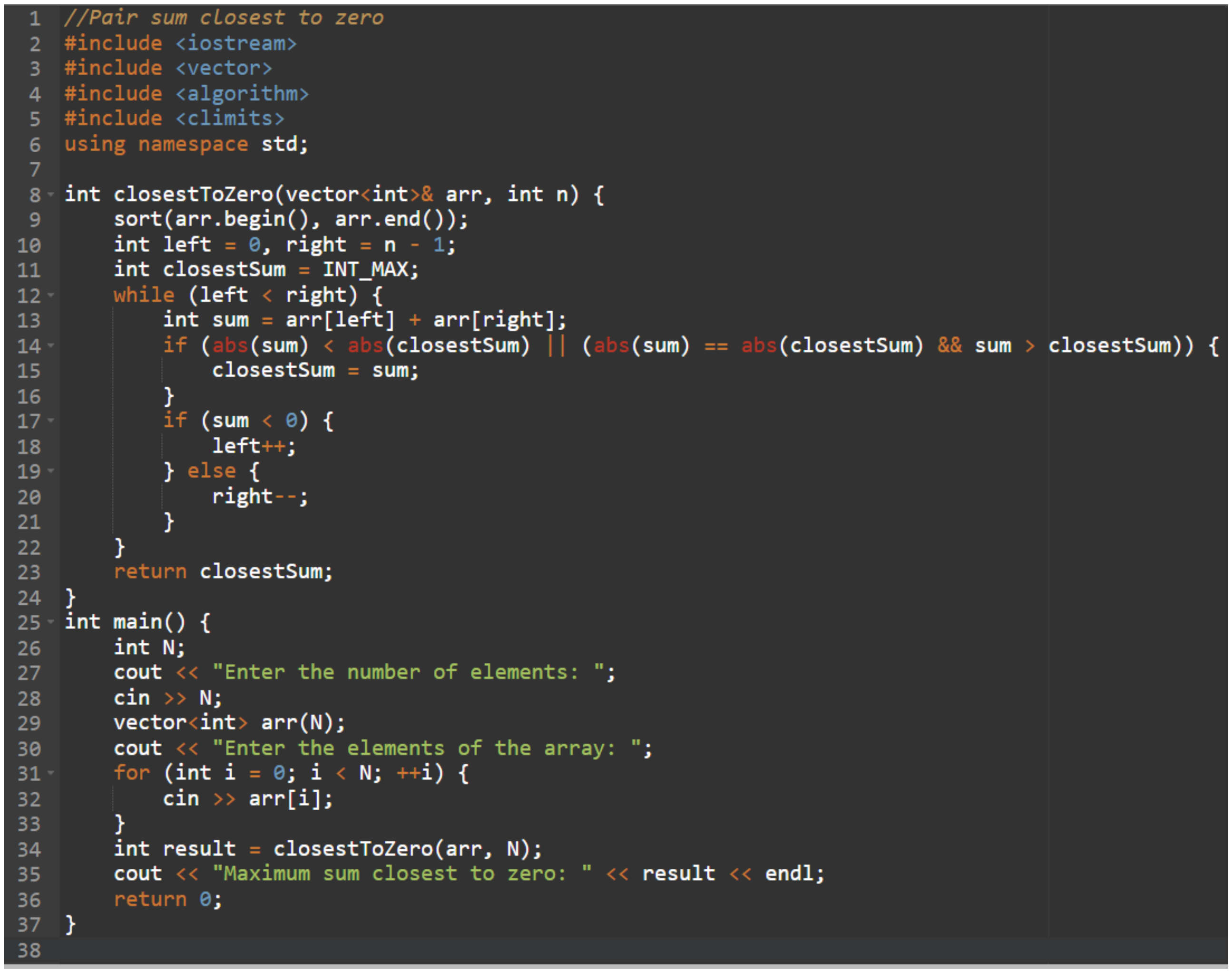
Output:



```
Enter the number of linked lists: 3
Enter the number of elements in list 1: 3
Enter the elements: 0 1 2
Enter the number of elements in list 2: 4
Enter the elements: 0 1 2 4
Enter the number of elements in list 3: 3
Enter the elements: 1 2 4
Merged List: 0 0 1 1 1 2 2 2 4 4
```

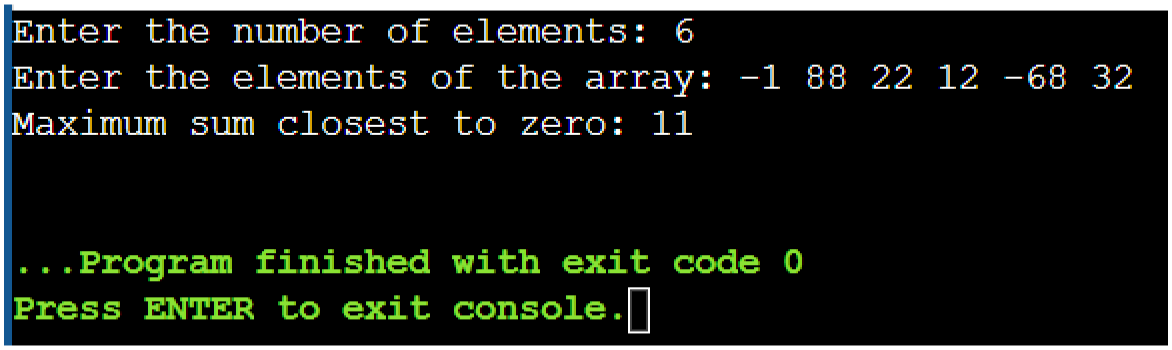
Q5.Pair sum closest to zero.(hard)

Sol.



```cpp
//Pair sum closest to zero
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

int closestToZero(vector<int>& arr, int n) {
    sort(arr.begin(), arr.end());
    int left = 0, right = n - 1;
    int closestSum = INT_MAX;
    while (left < right) {
        int sum = arr[left] + arr[right];
        if (abs(sum) < abs(closestSum) || (abs(sum) == abs(closestSum) && sum > closestSum)) {
            closestSum = sum;
        }
        if (sum < 0) {
            left++;
        } else {
            right--;
        }
    }
    return closestSum;
}
int main() {
    int N;
    cout << "Enter the number of elements: ";
    cin >> N;
    vector<int> arr(N);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < N; ++i) {
        cin >> arr[i];
    }
    int result = closestToZero(arr, N);
    cout << "Maximum sum closest to zero: " << result << endl;
    return 0;
}
```

OUTPUT:



```
Enter the number of elements: 6
Enter the elements of the array: -1 88 22 12 -68 32
Maximum sum closest to zero: 11


...Program finished with exit code 0
Press ENTER to exit console.
```