Bắt đầu vào lúc	Thứ Tư, 20 tháng 3 2024, 2:07 PM
Trạng thái	Đã xong
Kết thúc lúc	Thứ Ba, 2 tháng 4 2024, 7:45 PM
Thời gian thực hiện	13 Các ngày 5 giờ
Điểm	6,00/7,00
Điểm	**8,57** trên 10,00 (**85,71**%)

Câu hỏi **1**

Đúng

Đạt điểm 1,00 trên 1,00

Implement static methods **Partition** and **QuickSort** in class Sorting to sort an array in ascending order.

```
#ifndef SORTING_H
#define SORTING_H
#include <sstream>
#include <iostream>
#include <type_traits>
using namespace std;
template <class T>
class Sorting {
private:
    static T* Partition(T* start, T* end) ;
public:
    static void QuickSort(T* start, T* end) ;
};
#endif /* SORTING_H */
```

You can read the pseudocode of the algorithm used to in method Partition in the below image.

ALGORITHM *HoarePartition*$(A[l..r])$
//Partitions a subarray by Hoare's algorithm, using the first element
// as a pivot
//Input: Subarray of array $A[0..n-1]$, defined by its left and right
// indices l and r $(l < r)$
//Output: Partition of $A[l..r]$, with the split position returned as
// this function's value
$p \leftarrow A[l]$
$i \leftarrow l$; $j \leftarrow r+1$
repeat
  repeat $i \leftarrow i+1$ **until** $A[i] \geq p$
  repeat $j \leftarrow j-1$ **until** $A[j] \leq p$
  swap$(A[i], A[j])$
until $i \geq j$
swap$(A[i], A[j])$ //undo last swap when $i \geq j$
swap$(A[l], A[j])$
return j

For example:

Test	Result

`int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11, 16, 17, 18, 20, 19 };`
`cout << "Index of pivots: ";`
`Sorting<int>::QuickSort(&array[0], &array[20]);`
`cout << "\n";`
`cout << "Array after sorting: ";`
`for (int i : array) cout << i << " ";`
	`Index of pivots: 2 0 0 6 1 0 2 1 0 0 2 1 0 0 0 0 0 0 1 0`
	`Array after sorting: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20`

Answer: (penalty regime: 0 %)

Reset answer

```
1  static T* Partition(T* start, T* end) {
2      // TODO: return the pointer which points to the pivot after rearrange the array.
3      T pivot = start[0];
4      int i = 0;
5      int j = end - start;
6      do{
7          do{
8              i++;
```

```
        }while(start[i] <= pivot);
        do{
            j--;
        }while(start[j] > pivot);
        swap(start[i], start[j]);
    }while(i < j);
    swap(start[i], start[j]);
    swap(start[0], start[j]);
    return start + j;
}

static void QuickSort(T* start, T* end) {
    // TODO
    // In this question, you must print out the index of pivot in subarray after everytime calling metho
    if(start < end){
        int *i = Partition(start, end);
        cout<<i - start << " ";
        QuickSort(start,i);
        QuickSort(i + 1, end);
    }
}
```

	Test	Expected	Got	
✓	int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11, 16, 17, 18, 20, 19 }; cout << "Index of pivots: "; Sorting<int>::QuickSort(&array[0], &array[20]); cout << "\n"; cout << "Array after sorting: "; for (int i : array) cout << i << " ";	Index of pivots: 2 0 0 6 1 0 2 1 0 0 2 1 0 0 0 0 0 0 1 0 Array after sorting: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	Index of pivots: 2 0 0 6 1 0 2 1 0 0 2 1 0 0 0 0 0 0 1 0 Array after sorting: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **2**

Đúng

Đạt điểm 1,00 trên 1,00

Implement static methods **Merge** and **MergeSort** in class Sorting to sort an array in ascending order. The Merge method has already been defined a call to method printArray so you do not have to call this method again to print your array.

```
#ifndef SORTING_H
#define SORTING_H
#include <iostream>
using namespace std;
template <class T>
class Sorting {
public:
    /* Function to print an array */
    static void printArray(T *start, T *end)
    {
        long size = end - start + 1;
        for (int i = 0; i < size - 1; i++)
            cout << start[i] << ", ";
        cout << start[size - 1];
        cout << endl;
    }

    static void merge(T* left, T* middle, T* right){
        /*TODO*/
        Sorting::printArray(left, right);
    }
    static void mergeSort(T* start, T* end) {
       /*TODO*/
    }
};
#endif /* SORTING_H */
```

For example:

Test	Result
int arr[] = {0,2,4,3,1,4}; Sorting<int>::mergeSort(&arr[0], &arr[5]);	0, 2 0, 2, 4 1, 3 1, 3, 4 0, 1, 2, 3, 4, 4
int arr[] = {1}; Sorting<int>::mergeSort(&arr[0], &arr[0]);	

Answer: (penalty regime: 0, 0, 0, 5, 10, 15, ... %)

Reset answer

```
static void merge(T* left, T* middle, T* right){
        /*TODO*/
        int left_size = middle - left + 1;
        int right_size = right - middle;

        int leftArr[left_size], rightArr[right_size];
        for (int i = 0; i < left_size; ++i)
            leftArr[i] = left[i];
        for (int j = 0; j < right_size; ++j)
            rightArr[j] = middle[j + 1];

        int leftIdx = 0;
        int rightIdx = 0;
```

```
        int mergedArrIdx = 0;

        while (leftIdx < left_size && rightIdx < right_size) {
            if (leftArr[leftIdx] <= rightArr[rightIdx]) {
                left[mergedArrIdx] = leftArr[leftIdx];
                ++leftIdx;
            }
            else {
                left[mergedArrIdx] = rightArr[rightIdx];
                ++rightIdx;
            }
            ++mergedArrIdx;
        }

        while (leftIdx < left_size)
        {
            left[mergedArrIdx] = leftArr[leftIdx];
            ++leftIdx;
            ++mergedArrIdx;
        }
        while (rightIdx < right_size) {
            left[mergedArrIdx] = rightArr[rightIdx];
            ++rightIdx;
            ++mergedArrIdx;
        }

        Sorting::printArray(left, right);
    }
    static void mergeSort(T* start, T* end) {
        /*TODO*/
        if (start < end) {
            T* mid = start + (end - start) / 2;
            mergeSort(start, mid);
            mergeSort(mid + 1, end);
            merge(start, mid, end);
        }
    }

```

	Test	Expected	Got	

✓	`int arr[] = {0,2,4,3,1,4};`
	`Sorting<int>::mergeSort(&arr[0], &arr[5]);`
		`0, 2`
		`0, 2, 4`
		`1, 3`
		`1, 3, 4`
		`0, 1, 2, 3, 4, 4`
			`0, 2`
			`0, 2, 4`
			`1, 3`
			`1, 3, 4`
			`0, 1, 2, 3, 4, 4`
				✓

✓	`int arr[] = {1};` `Sorting<int>::mergeSort(&arr[0], &arr[0]);`			✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **3**

Đúng

Đạt điểm 1,00 trên 1,00

The best way to sort a singly linked list given the head pointer is probably using merge sort.

Both Merge sort and Insertion sort can be used for linked lists. The slow random-access performance of a linked list makes other algorithms (such as quick sort) perform poorly, and others (such as heap sort) completely impossible. Since worst case time complexity of Merge Sort is O(nLogn) and Insertion sort is O(n^2), merge sort is preferred.

Additionally, Merge Sort for linked list only requires a small constant amount of auxiliary storage.

To gain a deeper understanding about Merge sort on linked lists, let's implement **mergeLists** and **mergeSortList** function below

Constraints:

0 <= list.length <= 10^4
0 <= node.val <= 10^6

Use the nodes in the original list and don't modify ListNode's val attribute.

```
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int _val = 0, ListNode* _next = nullptr) : val(_val), next(_next) { }
};

// Merge two sorted lists
ListNode* mergeSortList(ListNode* head);

// Sort an unsorted list given its head pointer
ListNode* mergeSortList(ListNode* head);
```

For example:

Test	Input	Result

`int arr1[] = {1, 3, 5, 7, 9};`
`int arr2[] = {2, 4, 6, 8};`
`unordered_map<ListNode*, int> nodeAddr;`
`ListNode* a = init(arr1, sizeof(arr1) / 4, nodeAddr);`
`ListNode* b = init(arr2, sizeof(arr2) / 4, nodeAddr);`
`ListNode* merged = mergeLists(a, b);`
`try {`
`printList(merged, nodeAddr);`
`}`
`catch(char const* err) {`
`cout << err << '\n';`
`}`
`freeMem(merged);`
		1 2 3 4 5 6 7 8 9

Test	Input	Result

int size;
cin >> size;
int* array = new int[size];
for(int i = 0; i < size; i++) cin >> array[i];
unordered_map<ListNode*, int> nodeAddr;
ListNode* head = init(array, size, nodeAddr);
ListNode* sorted = mergeSortList(head);
try {
printList(sorted, nodeAddr);
}
catch(char const* err) {
cout << err << '\n';
}
freeMem(sorted);
delete[] array;
	9
	9 3 8 2 1 6 7 4 5
		1 2 3 4 5 6 7 8 9

Answer: (penalty regime: 0 %)

Reset answer

```
// You must use the nodes in the original list and must not modify ListNode's val attribute.
// Hint: You should complete the function mergeLists first and validate it using our first testcase exam

// Merge two sorted lists
ListNode* mergeLists(ListNode* a, ListNode* b) {
    ListNode* LastSorted=new ListNode;
    ListNode* source=LastSorted;
    ListNode* first=a;
    ListNode* second=b;
    while(first!=nullptr && second!=nullptr){
        if(first->val<=second->val){
            LastSorted->next=first;
            LastSorted=LastSorted->next;
            first=first->next;
        }
        else{
            LastSorted->next=second;
            LastSorted=LastSorted->next;
            second=second->next;
        }
    }
    if(first==nullptr){
        LastSorted->next=second;
        second=nullptr;
    }else{
        LastSorted->next=first;
        first=nullptr;
    }
    return source->next;
}

ListNode* findMiddle(ListNode* head) {
    if (head == nullptr) return nullptr;

    ListNode* slow = head;
    ListNode* fast = head->next;

    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }

    return slow;
}
// Sort and unsorted list given its head pointer
ListNode* mergeSortList(ListNode* head) {
    if (head == nullptr || head->next == nullptr) return head;
```

```
48
49      ListNode* mid = findMiddle(head);
50      ListNode* nextToMid = mid->next;
51      mid->next = nullptr;
52
```

	Test	Input	Expected	Got	

✓	`int arr1[] = {1, 3, 5, 7, 9};`
	`int arr2[] = {2, 4, 6, 8};`
	`unordered_map<ListNode*, int> nodeAddr;`
	`ListNode* a = init(arr1, sizeof(arr1) / 4, nodeAddr);`
	`ListNode* b = init(arr2, sizeof(arr2) / 4, nodeAddr);`
	`ListNode* merged = mergeLists(a, b);`
	`try {`
	`printList(merged, nodeAddr);`
	`}`
	`catch(char const* err) {`
	`cout << err << '\n';`
	`}`
	`freeMem(merged);`
			1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	✓

✓	`int size;`
	`cin >> size;`
	`int* array = new int[size];`
	`for(int i = 0; i < size; i++) cin >> array[i];`
	`unordered_map<ListNode*, int> nodeAddr;`
	`ListNode* head = init(array, size, nodeAddr);`
	`ListNode* sorted = mergeSortList(head);`
	`try {`
	`printList(sorted, nodeAddr);`
	`}`
	`catch(char const* err) {`
	`cout << err << '\n';`
	`}`
	`freeMem(sorted);`
	`delete[] array;`
		9
		9 3 8 2 1 6 7 4 5
			1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **4**

Đúng

Đạt điểm 1,00 trên 1,00

Implement static methods **merge, InsertionSort** and **TimSort** in class **Sorting** to sort an array in ascending order.

merge is responsible for merging two sorted subarrays. It takes three pointers: start, middle, and end, representing the left, middle, and right portions of an array.

InsertionSort is an implementation of the insertion sort algorithm. It takes two pointers, start and end, and sorts the elements in the range between them in ascending order using the insertion sort technique.

TimSort is an implementation of the TimSort algorithm, a hybrid sorting algorithm that combines insertion sort and merge sort. It takes two pointers, start and end, and an integer min_size, which determines the minimum size of subarrays to be sorted using insertion sort. The function first applies insertion sort to small subarrays, prints the intermediate result, and then performs merge operations to combine sorted subarrays until the entire array is sorted.

```
#ifndef SORTING_H
#define SORTING_H
#include <sstream>
#include <iostream>
#include <type_traits>
using namespace std;
template <class T>
class Sorting {
private:
    static void printArray(T* start, T* end)
    {
        int  size = end - start;
        for (int i = 0; i < size - 1; i++)
            cout << start[i] << " ";
        cout << start[size - 1];
        cout << endl;
    }

    static void merge(T* start, T* middle, T* end) ;
public:
    static void InsertionSort(T* start, T* end) ;
    static void TimSort(T* start, T* end, int min_size) ;
};
#endif /* SORTING_H */
```

For example:

Test	Result
int array[] = { 19, 20, 18, 17 ,12, 13, 14, 15, 1, 2, 9, 6, 4, 7, 11, 16, 10, 8, 5, 3 }; int min_size = 4; Sorting<int>::TimSort(&array[0], &array[20], min_size);	Insertion Sort: 17 18 19 20 12 13 14 15 1 2 6 9 4 7 11 16 3 5 8 10 Merge 1: 12 13 14 15 17 18 19 20 1 2 6 9 4 7 11 16 3 5 8 10 Merge 2: 12 13 14 15 17 18 19 20 1 2 4 6 7 9 11 16 3 5 8 10 Merge 3: 12 13 14 15 17 18 19 20 1 2 4 6 7 9 11 16 3 5 8 10 Merge 4: 1 2 4 6 7 9 11 12 13 14 15 16 17 18 19 20 3 5 8 10 Merge 5: 1 2 4 6 7 9 11 12 13 14 15 16 17 18 19 20 3 5 8 10 Merge 6: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
int array[] = { 3, 20, 18, 17 ,12, 13, 14, 15, 1, 2, 9, 6, 4, 7, 11, 16, 10, 8, 5, 19 }; int min_size = 4; Sorting<int>::TimSort(&array[0], &array[20], min_size);	Insertion Sort: 3 17 18 20 12 13 14 15 1 2 6 9 4 7 11 16 5 8 10 19 Merge 1: 3 12 13 14 15 17 18 20 1 2 6 9 4 7 11 16 5 8 10 19 Merge 2: 3 12 13 14 15 17 18 20 1 2 4 6 7 9 11 16 5 8 10 19 Merge 3: 3 12 13 14 15 17 18 20 1 2 4 6 7 9 11 16 5 8 10 19 Merge 4: 1 2 3 4 6 7 9 11 12 13 14 15 16 17 18 20 5 8 10 19 Merge 5: 1 2 3 4 6 7 9 11 12 13 14 15 16 17 18 20 5 8 10 19 Merge 6: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Answer: (penalty regime: 0 %)

Reset answer

```
static void merge(T* start, T* middle, T* end) {
    T* left = start;
        T* right = end;
        T i, j, k;
        T n1 = middle - left + 1;
        T n2 = right - middle;
        /* Tạo các mảng tạm */
        T *L = new T[n1], *R = new T[n2];
        /* Copy dữ liệu sang các mảng tạm */
        for (i = 0; i < n1; i++)
            L[i] = left[i];
        for (j = 0; j < n2; j++)
            R[j] = middle[1 + j];

        /* Gộp hai mảng tạm vừa rồi vào mảng arr*/
        i = 0; // Khởi tạo chỉ số bắt đầu của mảng con đầu tiên
        j = 0; // Khởi tạo chỉ số bắt đầu của mảng con thứ hai
        k = 0; // IKhởi tạo chỉ số bắt đầu của mảng lưu kết quả
        while (i < n1 && j < n2)
        {
            if (L[i] <= R[j])
            {
                left[k] = L[i];
                i++;
            }
            else
            {
                left[k] = R[j];
                j++;
            }
            k++;
        }
```

```
        /* Copy các phần tử còn lại của mảng L vào arr nếu có */
        while (i < n1)
        {
            left[k] = L[i];
            i++;
            k++;
        }

        /* Copy các phần tử còn lại của mảng R vào arr nếu có */
        while (j < n2)
        {
            left[k] = R[j];
            j++;
            k++;
        }
}

 static void InsertionSort(T* start, T* end) {
        // TODO
        for (int i = 0; i<end - start; i++) {
            char min = start[i];
            int id = i;
            for (int j = i; j<end - start; j++){
                if (start[j]<min){
                    id = j;
                    min = start[j];
                }
            }
            int temp = start[i];
            start[i] = start[id];
            start[id] = temp;
        }
    }

    static void TimSort(T* start, T* end, int min_size) {
        // TODO
        // You must print out the array after using insertion sort and everytime calling method merge.
        cout << "Insertion Sort: ";
        for (int i = 0; i < end - start; i += min_size){
            if (start+i+min_size<end) InsertionSort(start + i, start+i+min_size);
            else InsertionSort(start + i, end);
        }
        printArray(start, end);
        int i = 1;
        for (int size = min_size; size < end - start; size = 2 * size){
            for (int left = 0; left < end - start; left += 2 * size,i++){
                cout << "Merge " << i << ": ";
                int mid = (left + size - 1) < (end - start - 1) ? (left + size - 1) : (end - start - 1);
                int right = (left + 2 * size - 1) < (end - start - 1) ? (left + 2 * size - 1) : (end - s
                merge(&start[left], &start[mid], &start[right]);
                printArray(start, end);
            }
        }
    }
```

	Test	Expected	Got	
✓	int array[] = { 19, 20, 18, 17 ,12, 13, 14, 15, 1, 2, 9, 6, 4, 7, 11, 16, 10, 8, 5, 3 }; int min_size = 4; Sorting<int>::TimSort(&array[0], &array[20], min_size);	Insertion Sort: 17 18 19 20 12 13 14 15 1 2 6 9 4 7 11 16 3 5 8 10 Merge 1: 12 13 14 15 17 18 19 20 1 2 6 9 4 7 11 16 3 5 8 10 Merge 2: 12 13 14 15 17 18 19 20 1 2 4 6 7 9 11 16 3 5 8 10 Merge 3: 12 13 14 15 17 18 19 20 1 2 4 6 7 9 11 16 3 5 8 10 Merge 4: 1 2 4 6 7 9 11 12 13 14 15 16 17 18 19 20 3 5 8 10 Merge 5: 1 2 4 6 7 9 11 12 13 14 15 16 17 18 19 20 3 5 8 10 Merge 6: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	Insertion Sort: 17 18 19 20 12 13 14 15 1 2 6 9 4 7 11 16 3 5 8 10 Merge 1: 12 13 14 15 17 18 19 20 1 2 6 9 4 7 11 16 3 5 8 10 Merge 2: 12 13 14 15 17 18 19 20 1 2 4 6 7 9 11 16 3 5 8 10 Merge 3: 12 13 14 15 17 18 19 20 1 2 4 6 7 9 11 16 3 5 8 10 Merge 4: 1 2 4 6 7 9 11 12 13 14 15 16 17 18 19 20 3 5 8 10 Merge 5: 1 2 4 6 7 9 11 12 13 14 15 16 17 18 19 20 3 5 8 10 Merge 6: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	✓
✓	int array[] = { 3, 20, 18, 17 ,12, 13, 14, 15, 1, 2, 9, 6, 4, 7, 11, 16, 10, 8, 5, 19 }; int min_size = 4; Sorting<int>::TimSort(&array[0], &array[20], min_size);	Insertion Sort: 3 17 18 20 12 13 14 15 1 2 6 9 4 7 11 16 5 8 10 19 Merge 1: 3 12 13 14 15 17 18 20 1 2 6 9 4 7 11 16 5 8 10 19 Merge 2: 3 12 13 14 15 17 18 20 1 2 4 6 7 9 11 16 5 8 10 19 Merge 3: 3 12 13 14 15 17 18 20 1 2 4 6 7 9 11 16 5 8 10 19 Merge 4: 1 2 3 4 6 7 9 11 12 13 14 15 16 17 18 20 5 8 10 19 Merge 5: 1 2 3 4 6 7 9 11 12 13 14 15 16 17 18 20 5 8 10 19 Merge 6: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	Insertion Sort: 3 17 18 20 12 13 14 15 1 2 6 9 4 7 11 16 5 8 10 19 Merge 1: 3 12 13 14 15 17 18 20 1 2 6 9 4 7 11 16 5 8 10 19 Merge 2: 3 12 13 14 15 17 18 20 1 2 4 6 7 9 11 16 5 8 10 19 Merge 3: 3 12 13 14 15 17 18 20 1 2 4 6 7 9 11 16 5 8 10 19 Merge 4: 1 2 3 4 6 7 9 11 12 13 14 15 16 17 18 20 5 8 10 19 Merge 5: 1 2 3 4 6 7 9 11 12 13 14 15 16 17 18 20 5 8 10 19 Merge 6: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **5**

Đúng

Đạt điểm 1,00 trên 1,00

A hotel has m rooms left, there are n people who want to stay in this hotel. You have to distribute the rooms so that as many people as possible will get a room to stay.
However, each person has a desired room size, he/she will accept the room if its size is close enough to the desired room size.
More specifically, if the maximum difference is k, and the desired room size is x, then he or she will accept a room if its size is between x - k and x + k
Determine the maximum number of people who will get a room to stay.

input:
vector<int> rooms: rooms[i] is the size of the ith room
vector<int> people: people[i] the desired room size of the ith person
int k: maximum allowed difference. If the desired room size is x, he or she will accept a room if its size is between x - k and x + k

output:
the maximum number of people who will get a room to stay.

Note: The iostream, vector and algorithm library are already included for you.

Constraints:
1 <= rooms.length, people.length <= 2 * 10^5
0 <= k <= 10^9
1 <= rooms[i], people[i] <= 10^9

Example 1:

Input:
rooms = {57, 45, 80, 65}
people = {30, 60, 75}
k = 5
Output:
2

Explanation:
2 is the maximum amount of people that can stay in this hotel.
There are 3 people and 4 rooms, the first person cannot stay in any room, the second and third person can stay in the first and third room, respectively

Example 2:

Input:
rooms = {59, 5, 65, 15, 42, 81, 58, 96, 50, 1}
people = {18, 59, 71, 65, 97, 83, 80, 68, 92, 67}
k = 1000
Output:
10

For example:

Test	Input	Result

int peopleCount, roomCount, k;
cin >> peopleCount >> roomCount >> k;
 	
vector<int> people(peopleCount);
vector<int> rooms(roomCount);
 	
for(int i = 0; i < peopleCount; i++)
cin >> people[i];
for(int i = 0; i < roomCount; i++)
cin >> rooms[i];
cout << maxNumberOfPeople(rooms, people, k) << '\n';
	3 4 5
	30 60 75
	57 45 80 65
		2

int peopleCount, roomCount, k;
cin >> peopleCount >> roomCount >> k;
 	
vector<int> people(peopleCount);
vector<int> rooms(roomCount);
 	
for(int i = 0; i < peopleCount; i++)
cin >> people[i];
for(int i = 0; i < roomCount; i++)
cin >> rooms[i];
cout << maxNumberOfPeople(rooms, people, k) << '\n';
	10 10 1000
	18 59 71 65 97 83 80 68 92 67
	59 5 65 15 42 81 58 96 50 1
		10

Answer: (penalty regime: 0 %)

Reset answer

```
int maxNumberOfPeople(vector<int>& rooms, vector<int>& people, int k) {
    sort(rooms.begin(), rooms.end());
    sort(people.begin(), people.end());

    int roomIndex = 0;
    int peopleIndex = 0;
    int maxPeople = 0;

    while (roomIndex < rooms.size() && peopleIndex < people.size()) {
        int roomSize = rooms[roomIndex];
        int desiredRoomSize = people[peopleIndex];

        if (abs(roomSize - desiredRoomSize) <= k) {
            maxPeople++;
            roomIndex++;
            peopleIndex++;
        } else if (roomSize < desiredRoomSize) {
            roomIndex++;
        } else {
            peopleIndex++;
        }
    }

    return maxPeople;
}
```

	Test	Input	Expected	Got	

✓	`int peopleCount, roomCount, k;`
	`cin >> peopleCount >> roomCount >> k;`
	
	`vector<int> people(peopleCount);`
	`vector<int> rooms(roomCount);`
	
	`for(int i = 0; i < peopleCount; i++)`
	`  cin >> people[i];`
	`for(int i = 0; i < roomCount; i++)`
	`  cin >> rooms[i];`
	`cout << maxNumberOfPeople(rooms, people, k) << '\n';`
		3 4 5
		30 60 75
		57 45 80 65
			2	2	✓

✓	`int peopleCount, roomCount, k;`
	`cin >> peopleCount >> roomCount >> k;`
	
	`vector<int> people(peopleCount);`
	`vector<int> rooms(roomCount);`
	
	`for(int i = 0; i < peopleCount; i++)`
	`  cin >> people[i];`
	`for(int i = 0; i < roomCount; i++)`
	`  cin >> rooms[i];`
	`cout << maxNumberOfPeople(rooms, people, k) << '\n';`
		10 10 1000
		18 59 71 65 97 83 80 68 92 67
		59 5 65 15 42 81 58 96 50 1
			10	10	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **6**

Sai

Đạt điểm 0,00 trên 1,00

Given a list of distinct unsorted integers nums.

Your task is to implement a function with following prototype:

```
int minDiffPairs(int* arr, int n);
```

This function identify and return all pairs of elements with the smallest absolute difference among them. If there are multiple pairs that meet this criterion, the function should find and return all of them.

Note: Following libraries are included: iostream, string, algorithm, sstream

For example:

Test	Result
`int arr[] = {10, 5, 7, 9, 15, 6, 11, 8, 12, 2};` `cout << minDiffPairs(arr, 10);`	`(5, 6), (6, 7), (7, 8), (8, 9), (9, 10), (10, 11), (11, 12)`
`int arr[] = {10};` `cout << minDiffPairs(arr, 1);`	
`int arr[] = {10, -1, -150, 200};` `cout << minDiffPairs(arr, 4);`	`(-1, 10)`

Answer: (penalty regime: 0 %)

Reset answer

```
string minDiffPair(int* arr, int n){
    // STUDENT ANSWER
}
```

Syntax Error(s)

```
__tester__.cpp: In function 'std::string minDiffPair(int*, int)':
__tester__.cpp:14:1: error: no return statement in function returning non-void [-Werror=return-type]
   14 | }
      | ^
__tester__.cpp: In function 'int main()':
__tester__.cpp:20:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   20 | cout << minDiffPairs(arr, 10);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:25:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   25 | cout << minDiffPairs(arr, 1);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:30:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   30 | cout << minDiffPairs(arr, 4);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:35:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   35 | cout << minDiffPairs(arr, 16);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:40:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   40 | cout << minDiffPairs(arr, 20);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:45:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   45 | cout << minDiffPairs(arr, 20);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:50:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   50 | cout << minDiffPairs(arr, 49);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:55:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   55 | cout << minDiffPairs(arr, 40);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:60:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   60 | cout << minDiffPairs(arr, 30);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
__tester__.cpp:65:9: error: 'minDiffPairs' was not declared in this scope; did you mean 'minDiffPair'?
   65 | cout << minDiffPairs(arr, 90);;
      |         ^~~~~~~~~~~~
      |         minDiffPair
cc1plus: all warnings being treated as errors
```

Sai

Marks for this submission: 0,00/1,00.

Câu hỏi **7**

Đúng

Đạt điểm 1,00 trên 1,00

Print the elements of an array in the decreasing frequency order while preserving the relative order of the elements.

Students are not allowed to use map/unordered map.

iostream, algorithm libraries are included.

For example:

Test	Result

`int arr[] = {-4,1,2,2,-4,9,1,-1};`
`int n = sizeof(arr) / sizeof(arr[0]);`
 	
`sortByFrequency(arr, n);`
 	
`for (int i = 0; i < n; i++)`
`  cout << arr[i] << " ";`
	`-4 -4 1 1 2 2 9 -1`

`int arr[] = {-5,3,8,1,-9,-9};`
`int n = sizeof(arr) / sizeof(arr[0]);`
 	
`sortByFrequency(arr, n);`
 	
`for (int i = 0; i < n; i++)`
`  cout << arr[i] << " ";`
	`-9 -9 -5 3 8 1`

Answer: (penalty regime: 0 %)

Reset answer

```
struct ele {
    int count, index, val;
};

// Used for sorting by value
bool mycomp(struct ele a, struct ele b)
{
    return (a.val < b.val);
}

// Used for sorting by frequency. And if frequency is same,
// then by appearance
bool mycomp2(struct ele a, struct ele b)
{
    if (a.count != b.count)
        return (a.count < b.count);
    else
        return a.index > b.index;
}

void sortByFrequency(int arr[], int n)
{
    struct ele element[n];
    for (int i = 0; i < n; i++) {

        // Fill Indexes
        element[i].index = i;

        // Initialize counts as 0
        element[i].count = 0;

        // Fill values in structure
        // elements
        element[i].val = arr[i];
```

```
35      }
36
37      /* Sort the structure elements according to value,
38         we used stable sort so relative order is maintained.
39       */
40      stable_sort(element, element + n, mycomp);
41
42      /* initialize count of first element as 1 */
43      element[0].count = 1;
44
45      /* Count occurrences of remaining elements */
46      for (int i = 1; i < n; i++) {
47
48          if (element[i].val == element[i - 1].val) {
49              element[i].count += element[i - 1].count + 1;
50
51              /* Set count of previous element as -1, we are
52                 doing this because we'll again sort on the
```

	Test	Expected	Got	
✓	\tint arr[] = {-4,1,2,2,-4,9,1,-1}; \tint n = sizeof(arr) / sizeof(arr[0]); \tsortByFrequency(arr, n); \tfor (int i = 0; i < n; i++) \t\tcout << arr[i] << " ";	-4 -4 1 1 2 2 9 -1	-4 -4 1 1 2 2 9 -1	✓
✓	\tint arr[] = {-5,3,8,1,-9,-9}; \tint n = sizeof(arr) / sizeof(arr[0]); \tsortByFrequency(arr, n); \tfor (int i = 0; i < n; i++) \t\tcout << arr[i] << " ";	-9 -9 -5 3 8 1	-9 -9 -5 3 8 1	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.