Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Tư, 20 tháng 3 2024, 8:58 AM
Kết thúc lúc	Thứ Tư, 27 tháng 3 2024, 10:59 PM
Thời gian thực hiện	7 Các ngày 14 giờ

Câu hỏi **1**

Đúng

Đạt điểm 1,00

Implement method bubbleSort() in class SLinkedList to sort this list in ascending order. After each bubble, we will print out a list to check (using printList).

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

template <class T>
class SLinkedList {
public:
    class Node; // Forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    SLinkedList()
    {
      this->head = nullptr;
      this->tail = nullptr;
      this->count = 0;
    }
    ~SLinkedList(){};
    void add(T e)
    {
        Node *pNew = new Node(e);

    if (this->count == 0)
    {
        this->head = this->tail = pNew;
    }
    else
    {
        this->tail->next = pNew;
        this->tail = pNew;
    }

    this->count++;
}
int size()
{
    return this->count;
}
void printList()
{
    stringstream ss;
    ss << "[";
    Node *ptr = head;
    while (ptr != tail)
    {
        ss << ptr->data << ",";
```

```
            ptr = ptr->next;
        }

        if (count > 0)
            ss << ptr->data << "]";
        else
            ss << "]";
        cout << ss.str() << endl;
    }
public:
    class Node {
    private:
        T data;
        Node* next;
        friend class SLinkedList<T>;
    public:
        Node() {
            next = 0;
        }
        Node(T data) {
            this->data = data;
            this->next = nullptr;
        }
    };

    void bubbleSort();
};
```

For example:

Test	Result

`int arr[] = {9, 2, 8, 4, 1};`
`SLinkedList<int> list;`
`for(int i = 0; i <int(sizeof(arr))/4;i++)`
`  list.add(arr[i]);`
`list.bubbleSort();`
	`[2,8,4,1,9]`
	`[2,4,1,8,9]`
	`[2,1,4,8,9]`
	`[1,2,4,8,9]`

Answer: (penalty regime: 0 %)

Reset answer

```
template <class T>void SLinkedList<T>::bubbleSort(){
    if (count <= 1) return;
    bool swapped;    Node* current;
    Node* lptr = nullptr;
    int iteration = 1;
```

```
    do {
        swapped = false;
        current = head;
        while (current->next != lptr) {
            if (current->data > current->next->data) {
                T temp = current->data;
                current->data = current->next->data;
                current->next->data = temp;
                swapped = true;
            }
                current = current->next;
        }
                lptr = current;
                if (swapped) {
                    printList();
                    iteration++;
                }
    } while (swapped);
}
```

	Test	Expected	Got	

✓	int arr[] = {9, 2, 8, 4, 1};
	SLinkedList<int> list;
	for(int i = 0; i <int(sizeof(arr))/4;i++)
	    list.add(arr[i]);
	list.bubbleSort();
		[2,8,4,1,9]
		[2,4,1,8,9]
		[2,1,4,8,9]
		[1,2,4,8,9]
			[2,8,4,1,9]
			[2,4,1,8,9]
			[2,1,4,8,9]
			[1,2,4,8,9]
				✓

Passed all tests! ✓

Câu hỏi **2**

Đúng

Đạt điểm 1,00

Implement static method selectionSort in class **Sorting** to sort an array in ascending order. After each selection, we will print out a list to check (using printArray).

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

template <class T>
class Sorting
{
public:
    /* Function to print an array */
    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 selectionSort(T *start, T *end);
};
```

For example:

Test	Result
int arr[] = {9, 2, 8, 1, 0, -2}; Sorting<int>::selectionSort(&arr[0], &arr[6]);	-2, 2, 8, 1, 0, 9 -2, 0, 8, 1, 2, 9 -2, 0, 1, 8, 2, 9 -2, 0, 1, 2, 8, 9 -2, 0, 1, 2, 8, 9

Answer: (penalty regime: 0 %)

Reset answer

```
1  template <class T>
2  void Sorting<T>::selectionSort(T *start, T *end)
3  {
4      T* result = start;
5      while (start != end - 1){
6          T* min = start;
7          T* tmp = start + 1;
8          while (tmp != end){
9              if (*tmp < *min){
10                 min = tmp;
```

```
            }
            tmp++;
        }
        swap(*start, *min);
        Sorting<T>::printArray(result,end);
        start++;
    }
}
```

	Test	Expected	Got	

✓	`int arr[] = {9, 2, 8, 1, 0, -2};`
	`Sorting<int>::selectionSort(&arr[0], &arr[6]);`
		`-2, 2, 8, 1, 0, 9`
		`-2, 0, 8, 1, 2, 9`
		`-2, 0, 1, 8, 2, 9`
		`-2, 0, 1, 2, 8, 9`
		`-2, 0, 1, 2, 8, 9`
			`-2, 2, 8, 1, 0, 9`
			`-2, 0, 8, 1, 2, 9`
			`-2, 0, 1, 8, 2, 9`
			`-2, 0, 1, 2, 8, 9`
			`-2, 0, 1, 2, 8, 9`
				✓

Passed all tests! ✓

Câu hỏi **3**

Đúng

Đạt điểm 1,00

Implement static methods **sortSegment** and **ShellSort** 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 void printArray(T* start, T* end)
    {
        int size = end - start;
        for (int i = 0; i < size; i++)
            cout << start[i] << " ";
        cout << endl;
    }

public:
    // TODO: Write your code here
    static void sortSegment(T* start, T* end, int segment_idx, int cur_segment_total);
    static void ShellSort(T* start, T* end, int* num_segment_list, int num_phases);
};

#endif /* SORTING_H */
```

For example:

Test	Result

`int num_segment_list[] = {1, 3, 5};`
`int num_phases = 3;`
`int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };`
 	
`Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);`
	`5 segments: 5 4 3 2 1 10 9 8 7 6`
	`3 segments: 2 1 3 5 4 7 6 8 10 9`
	`1 segments: 1 2 3 4 5 6 7 8 9 10`

Answer: (penalty regime: 0 %)

Reset answer

```
1  // TODO: Write your code here
2
3  static void sortSegment(T* start, T* end, int segment_idx, int cur_segment_total) {
4      // TODO
5      T* left = start + segment_idx;
6      while (left <= end - cur_segment_total){
```

```
        T* tmp = start + segment_idx;
        while (tmp <= end - cur_segment_total - 1){
            if(* tmp > * (tmp + cur_segment_total)) swap(*tmp,* (tmp + cur_segment_total));
            tmp += cur_segment_total;
        }
        left += cur_segment_total;
    }
}

static void ShellSort(T* start, T* end, int* num_segment_list, int num_phases) {
    // TODO
    // Note: You must print out the array after sorting segments to check whether your algori
    for(int i = num_phases-1; i >= 0; i--){
        for(int j = 0; j < num_segment_list[i];j ++ ){
            sortSegment(start, end,j , num_segment_list[i]);
        }
        cout << num_segment_list[i] << " segments: ";
        Sorting<T>::printArray(start,end);
    }
}
```

	Test	Expected	Got	
✓	int num_segment_list[] = {1, 3, 5}; int num_phases = 3; int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 }; Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);	5 segments: 5 4 3 2 1 10 9 8 7 6 3 segments: 2 1 3 5 4 7 6 8 10 9 1 segments: 1 2 3 4 5 6 7 8 9 10	5 segments: 5 4 3 2 1 10 9 8 7 6 3 segments: 2 1 3 5 4 7 6 8 10 9 1 segments: 1 2 3 4 5 6 7 8 9 10	✓
✓	int num_segment_list[] = { 1, 2, 6 }; int num_phases = 3; int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 }; Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);	6 segments: 4 3 2 1 6 5 10 9 8 7 2 segments: 2 1 4 3 6 5 8 7 10 9 1 segments: 1 2 3 4 5 6 7 8 9 10	6 segments: 4 3 2 1 6 5 10 9 8 7 2 segments: 2 1 4 3 6 5 8 7 10 9 1 segments: 1 2 3 4 5 6 7 8 9 10	✓

	Test	Expected	Got	

✓	`int num_segment_list[] = { 1, 2, 5 };`
	`int num_phases = 3;`
	`int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);`
		5 segments: 5 4 3 2 1 10 9 8 7 6
		2 segments: 1 2 3 4 5 6 7 8 9 10
		1 segments: 1 2 3 4 5 6 7 8 9 10
			5 segments: 5 4 3 2 1 10 9 8 7 6
			2 segments: 1 2 3 4 5 6 7 8 9 10
			1 segments: 1 2 3 4 5 6 7 8 9 10
				✓

✓	`int num_segment_list[] = { 1, 2, 3 };`
	`int num_phases = 3;`
	`int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);`
		3 segments: 1 3 2 4 6 5 7 9 8 10
		2 segments: 1 3 2 4 6 5 7 9 8 10
		1 segments: 1 2 3 4 5 6 7 8 9 10
			3 segments: 1 3 2 4 6 5 7 9 8 10
			2 segments: 1 3 2 4 6 5 7 9 8 10
			1 segments: 1 2 3 4 5 6 7 8 9 10
				✓

✓	`int num_segment_list[] = { 1, 5, 8, 10 };`
	`int num_phases = 4;`
	`int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[15], &num_segment_list[0], num_phases);`
		10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12
		8 segments: 1 2 4 6 7 10 12 13 3 5 9 8 11 14 15
		5 segments: 1 2 4 3 5 9 8 11 6 7 10 12 13 14 15
		1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
			10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12
			8 segments: 1 2 4 6 7 10 12 13 3 5 9 8 11 14 15
			5 segments: 1 2 4 3 5 9 8 11 6 7 10 12 13 14 15
			1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
				✓

Passed all tests! ✓

Câu hỏi **4**

Đúng

Đạt điểm 1,00

Implement static methods **Partition** and **QuickSort** in class Sorting to sort an array in **DESCENDING 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 */
```

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: 17 2 1 0 7 6 1 0 1 0 0 0 3 0 1 0 0 0 0 0 Array after sorting: 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Answer: (penalty regime: 0 %)

Reset answer

```
static T* Partition(T* start, T* end) {
    T* arr = start;
    int low = 0, high = end - start - 1;
    int pivot = arr[low];
    int i = low, j = high + 1;
    do {
        do {
            i++;
        } while (arr[i] > pivot);
        do {
            j--;
        } while (arr[j] < pivot);
        if (i < j) {
            swap(arr[i], arr[j]);
        }
```

```
    } while (i <= j);
    if(i <= j) swap(arr[i], arr[j]);
    swap(arr[low], arr[j]);
    cout << j << " ";
    return start + j;
}

static void QuickSort(T* start, T* end) {
    if (start < end) {
        T* pivot = Partition(start, end);
        QuickSort(start, pivot);
        QuickSort(pivot + 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: 17 2 1 0 7 6 1 0 1 0 0 0 3 0 1 0 0 0 0 0 Array after sorting: 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	Index of pivots: 17 2 1 0 7 6 1 0 1 0 0 0 3 0 1 0 0 0 0 0 Array after sorting: 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	✓

Passed all tests! ✓

Câu hỏi **5**

Đúng

Đạt điểm 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

```
1 static void merge(T* left, T* middle, T* right){
```

```
static void merge(T* left, T* middle, T* right){
    /*TODO*/
    T* arr = left;
    int l = 0, m = middle - left, r = right - left;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for(int i = 0; i < n1; i++)
    L[i] = arr[l + i];
    for(int j = 0; j < n2; j++)
    R[j] = arr[m + 1 + j];
    int i = 0;
    int j = 0;
    int k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
    Sorting::printArray(left, right);
}
static void mergeSort(T* start, T* end){
    /*TODO*/
    if(start < end){
        T* mid = (end - start) / 2 + start;
        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! ✓

Câu hỏi **6**

Đúng

Đạt điểm 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

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 te

// Merge two sorted lists
ListNode* mergeLists(ListNode* a, ListNode* b) {
    if(a == NULL) return b;
    else if(b == NULL) return a;
    ListNode * result = new ListNode(0);
    ListNode* tmp = result;
    while(a || b){
        if(!b || (a && a->val < b->val)){
            tmp->next = a;
            a = a->next;
            tmp = tmp->next;
```

```
        } else {
            tmp->next = b;
            b = b->next;
            tmp = tmp->next;
        }
    }
    return result->next;
}

// Sort and unsorted list given its head pointer
ListNode* midnode(ListNode* a) {
    ListNode* tmp = a;
    while (tmp->next && tmp->next->next){
        a = a->next;
        tmp = tmp->next->next;
    }
    tmp = a->next;
    a->next = NULL;
    return tmp;
}
ListNode* mergeSortList(ListNode* head){
    if(head == NULL || head->next == NULL) return head;
    ListNode* mid = midnode(head);
    head = mergeSortList(head);
    mid = mergeSortList(mid);
    return mergeLists(head,mid);
}
```

	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	✓

	Test	Input	Expected	Got	

✓	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! ✓

Câu hỏi **7**

Sai

Đạt điểm 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) {
    // TODO
}

static void InsertionSort(T* start, T* end) {
    // TODO

}

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 m

}
```

	Test	Expected	
✗	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	✗

Some hidden test cases failed, too.

Câu hỏi **8**

Đúng

Đạt điểm 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 count = 0;

    for (int i = 0; i < people.size(); ++i) {
        while (roomIndex < rooms.size() && rooms[roomIndex] < people[i] - k) {
            roomIndex++;
        }
        if (roomIndex < rooms.size() && rooms[roomIndex] <= people[i] + k) {
            count++;
            roomIndex++;
        }
```

```
    }

    return count;
}
```

	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	✓

	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';`
		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! ✓

Câu hỏi 9

Đúng

Đạt điểm 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 minDiffPairs(int* arr, int n) {
    std::sort(arr, arr + n);
    int minDiff = arr[n - 1] - arr[0];
    std::stringstream result;
    bool isFirstPair = true;
    for (int i = 1; i < n; ++i) {
        if (arr[i] - arr[i - 1] < minDiff) {
            minDiff = arr[i] - arr[i - 1];
            result.str("");
            result.clear();
            result << "(" << arr[i - 1] << ", " << arr[i] << ")";
            isFirstPair = false;
        } else if (arr[i] - arr[i - 1] == minDiff) {
            if (!isFirstPair)
                result << ", ";
            result << "(" << arr[i - 1] << ", " << arr[i] << ")";
        }
    }
    return result.str();
}

```

	Test	Expected	Got	
✓	`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)`	`(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)`	`(-1, 10)`	✓

Passed all tests! ✓

Câu hỏi **10**

Đúng

Đạt điểm 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

```
#include <iostream>
#include <algorithm>

struct Element {
    int value;
    int frequency;
    int originalIndex;
};

bool compare(const Element &a, const Element &b) {
    if (a.frequency == b.frequency)
        return a.originalIndex < b.originalIndex;
    return a.frequency > b.frequency;
}

void sortByFrequency(int arr[], int n) {
    Element elements[n];

    for (int i = 0; i < n; ++i) {
        elements[i].value = arr[i];
        elements[i].originalIndex = i;
        elements[i].frequency = 0;
```

```
    }

    for (int i = 0; i < n; ++i) {
        bool found = false;
        for (int j = 0; j < i; ++j) {
            if (elements[j].value == elements[i].value) {
                found = true;
                ++elements[j].frequency;
                break;
            }
        }
        if (!found)
            elements[i].frequency = 1;
    }

    std::sort(elements, elements + n, compare);

    int index = 0;
    for (const auto &element : elements) {
        for (int i = 0; i < element.frequency; ++i)
            arr[index++] = element.value;
    }
}

```

	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! ✓

Câu hỏi **11**

Đúng

Đạt điểm 1,00

Given a list of points on the 2-D plane (**points[]** with **n** elements) and an integer **k**. Your task in this exercise is to implement the **closestKPoints** function to find K closest points to the given point (**des_point**) and print them by descending order of distances.

Prototype of `closestKPoints`:

```
void closestKPoints(Point points[], int n, Point& des_point, int k);
```

Note: The distance between two points on a plane is the Euclidean distance.

Template:

```
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <algorithm>
```

```
using namespace std;

class Point{
   public:
   int x, y;
   Point(int x = 0, int y = 0){
      this->x = x;
      this->y = y;
   }
   void display(){
      cout << "("<<x<<", "<<y<<")";
   }
};
```

For example:

Test	Result

`Point points[] = {{3, 3},{5, -1},{-2, 4}};`
`int n = sizeof(points)/sizeof(points[0]);`
`int k = 2;`
`Point des_point = {0,2};`
`closestKPoints(points, n, des_point, k);`
	`(-2, 4)`
	`(3, 3)`

`Point points[] = {{3, 3},{5, -1},{-2, 4}};`
`int n = sizeof(points)/sizeof(points[0]);`
`int k = 3;`
`Point des_point = {0,2};`
`closestKPoints(points, n, des_point, k);`
	`(-2, 4)`
	`(3, 3)`
	`(5, -1)`

Answer: (penalty regime: 0 %)

Reset answer

```
double distance(const Point& p1, const Point& p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) +
                (p1.y - p2.y) * (p1.y - p2.y));
}

bool compareDistance(const pair<double, Point>& a, const pair<double, Point>& b) {
    return a.first < b.first;
}

void closestKPoints(Point points[], int n, Point &des_point, int k) {
    pair<double, Point> distances[n];
    for (int i = 0; i < n; ++i) {
        double dist = distance(points[i], des_point);
        distances[i] = {dist, points[i]};
    }
    sort(distances, distances + n, compareDistance);
    for (int i = 0; i < min(k, n); ++i) {
        distances[i].second.display();
        cout << endl;
    }
}
```

	Test	Expected	Got	

✓	`Point points[] = {{3, 3},{5, -1},{-2, 4}};`
	`int n = sizeof(points)/sizeof(points[0]);`
	`int k = 2;`
	`Point des_point = {0,2};`
	`closestKPoints(points, n, des_point, k);`
		(-2, 4)
		(3, 3)
			(-2, 4)
			(3, 3)
				✓

✓	`Point points[] = {{3, 3},{5, -1},{-2, 4}};`
	`int n = sizeof(points)/sizeof(points[0]);`
	`int k = 3;`
	`Point des_point = {0,2};`
	`closestKPoints(points, n, des_point, k);`
		(-2, 4)
		(3, 3)
		(5, -1)
			(-2, 4)
			(3, 3)
			(5, -1)
				✓

Passed all tests! ✓