Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Hai, 7 tháng 4 2025, 10:37 PM
Kết thúc lúc	Thứ Hai, 7 tháng 4 2025, 11:19 PM
Thời gian thực hiện	41 phút 56 giây
Điểm	6,60/7,00
Điểm	**9,43** trên 10,00 (**94,29**%)

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

```
static T* Partition(T* start, T* end) {
        // Trả về con trỏ đến phần tử pivot sau khi sắp xếp lại mảng
        T* arr = start;
        int low = 0, high = end - start - 1;
        int pivot = arr[low]; // Chọn phần tử đầu tiên làm pivot
        int i = low, j = high + 1;

        do {
            // Tìm phần tử bên trái đầu tiên lớn hơn hoặc bằng pivot
            do {
                i++;
            } while (i <= high && arr[i] < pivot);

            // Tìm phần tử bên phải đầu tiên nhỏ hơn hoặc bằng pivot
            do {
```

```
            j--;
        } while (j >= low && arr[j] > pivot);

        // Nếu hai con trỏ gặp nhau, kết thúc vòng lặp
        if (i >= j) {
            break;
        }

        // Đổi chỗ hai phần tử không đúng thứ tự
        swap(arr[i], arr[j]);
    } while (i <= j);

    // Đưa pivot vào vị trí đúng trong mảng
    swap(arr[low], arr[j]);

    // In ra vị trí của pivot sau khi phân vùng
    cout << j << " ";

    // Trả về con trỏ đến vị trí của pivot
    return start + j;
}

static void QuickSort(T* start, T* end) {
    // Thuật toán QuickSort sử dụng đệ quy
    // In ra chỉ số của pivot trong mảng con sau mỗi lần gọi phương thức Partition
    if (start < end) {
        T* pivot = Partition(start, end);
        QuickSort(start, pivot); // Sắp xếp mảng con bên trái của pivot
        QuickSort(pivot + 1, end); // Sắp xếp mảng con bên phải của pivot
    }
}
```

	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

```
// Hàm trộn hai nửa đã sắp xếp
    static void merge(T* left, T* middle, T* right) {
        int leftSize = middle - left + 1;
        int rightSize = right - middle;

        // Tạo mảng tạm cho hai nửa
        std::vector<T> leftArray(left, middle + 1);
        std::vector<T> rightArray(middle + 1, right + 1);

        int i = 0, j = 0;
        T* k = left;

        // Trộn hai nửa
        while (i < leftSize && j < rightSize) {
            if (leftArray[i] <= rightArray[j]) {
                *k = leftArray[i++];
            } else {
                *k = rightArray[j++];
            }
            k++;
        }
```

```
        // Sao chép phần còn lại của mảng trái (nếu có)
        while (i < leftSize) {
            *k = leftArray[i++];
            k++;
        }

        // Sao chép phần còn lại của mảng phải (nếu có)
        while (j < rightSize) {
            *k = rightArray[j++];
            k++;
        }

        Sorting::printArray(left, right);  // In mảng sau khi gộp
    }

    // Thuật toán MergeSort
    static void mergeSort(T* start, T* end) {
        if (start >= end) return;  // Điều kiện dừng đệ quy

        T* middle = start + (end - start) / 2;  // Tìm phần tử giữa

        // Đệ quy sắp xếp hai nửa
        mergeSort(start, middle);      // Sắp xếp nửa bên trái
        mergeSort(middle + 1, end);    // Sắp xếp nửa bên phải

        // Gộp hai nửa đã sắp xếp
        merge(start, middle, 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* mergeLists(ListNode* a, ListNode* b) {
// 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

```
// Hàm trộn hai danh sách liên kết đã sắp xếp
    // Lưu ý: Phải sử dụng các node trong danh sách gốc và không được thay đổi giá trị val của ListNod
    static ListNode* mergeLists(ListNode* a, ListNode* b) {
        if (!a) return b;  // Nếu danh sách a rỗng, trả về danh sách b
        if (!b) return a;  // Nếu danh sách b rỗng, trả về danh sách a

        // So sánh giá trị của node đầu tiên và nối đệ quy
        if (a->val <= b->val) {
            a->next = mergeLists(a->next, b);  // Lấy node a và nối với kết quả trộn của phần còn lại
            return a;  // Trả về node a làm node đầu tiên của danh sách đã trộn
        } else {
            b->next = mergeLists(a, b->next);  // Lấy node b và nối với kết quả trộn của phần còn lại
            return b;  // Trả về node b làm node đầu tiên của danh sách đã trộn
        }
    }

    // Hàm tìm điểm giữa của danh sách liên kết
    // Sử dụng thuật toán "Tortoise and Hare" (Rùa và Thỏ)
    static ListNode* getMiddle(ListNode* head) {
        if (!head || !head->next) return head;  // Kiểm tra danh sách rỗng hoặc chỉ có 1 node

        ListNode* slow = head;  // Con trỏ "rùa" di chuyển 1 bước mỗi lần
        ListNode* fast = head->next;  // Con trỏ "thỏ" di chuyển 2 bước mỗi lần

        // Di chuyển fast gấp đôi tốc độ của slow
        while (fast && fast->next) {
            slow = slow->next;      // Rùa đi 1 bước
            fast = fast->next->next;  // Thỏ đi 2 bước
        }

        return slow;  // Khi thỏ đến cuối, rùa sẽ ở giữa danh sách
    }

    // Hàm Merge Sort cho danh sách liên kết
    static ListNode* mergeSortList(ListNode* head) {
        // Điều kiện dừng: danh sách rỗng hoặc chỉ có 1 node
        if (!head || !head->next) return head;

        // Bước 1: Tìm điểm giữa của danh sách
        ListNode* middle = getMiddle(head);
        ListNode* rightHalf = middle->next;  // Phần bên phải bắt đầu từ node sau middle

        // Cắt danh sách làm đôi bằng cách đặt next của middle thành nullptr
        middle->next = nullptr;

        // Bước 2: Đệ quy sắp xếp hai nửa
        ListNode* leftSorted = mergeSortList(head);      // Sắp xếp nửa trái
        ListNode* rightSorted = mergeSortList(rightHalf);  // Sắp xếp nửa phải

        // Bước 3: Trộn hai nửa đã sắp xếp
        return mergeLists(leftSorted, rightSorted);
```

	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! ✓

Đúng

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

Câu hỏi **4**

Đúng một phần

Đạt điểm 0,60 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`

Test	Result
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) {
    int leftSize = middle - start;     // Kích thước của mảng con bên trái
    int rightSize = end - middle;      // Kích thước của mảng con bên phải

    // Tạo mảng tạm thời để lưu trữ dữ liệu khi trộn
    T* temp = new T[leftSize + rightSize];

    // Sao chép cả hai nửa vào mảng tạm thời
    for (int i = 0; i < leftSize; i++) {
        temp[i] = start[i];            // Sao chép mảng con bên trái
    }
    for (int i = 0; i < rightSize; i++) {
        temp[leftSize + i] = middle[i]; // Sao chép mảng con bên phải
    }

    // Trộn lại từ mảng tạm thời vào mảng gốc
    int i = 0;        // Chỉ số cho nửa bên trái
    int j = leftSize; // Chỉ số cho nửa bên phải
    int k = 0;        // Chỉ số cho mảng đã trộn

    // So sánh và trộn các phần tử từ hai nửa
    while (i < leftSize && j < leftSize + rightSize) {
        if (temp[i] <= temp[j]) {
            start[k++] = temp[i++];    // Lấy phần tử từ nửa bên trái nếu nhỏ hơn
        } else {
            start[k++] = temp[j++];    // Lấy phần tử từ nửa bên phải nếu nhỏ hơn
        }
    }

    // Sao chép các phần tử còn lại từ nửa bên trái (nếu có)
    while (i < leftSize) {
        start[k++] = temp[i++];
    }

    // Sao chép các phần tử còn lại từ nửa bên phải (nếu có)
    while (j < leftSize + rightSize) {
        start[k++] = temp[j++];
    }

    // Giải phóng bộ nhớ đã cấp phát cho mảng tạm thời
    delete[] temp;
}

static void InsertionSort(T* start, T* end) {
    int size = end - start;    // Tính kích thước của mảng cần sắp xếp

    // Thuật toán sắp xếp chèn
    for (int i = 1; i < size; i++) {
        T key = start[i];      // Lưu giá trị của phần tử hiện tại
        int j = i - 1;         // Chỉ số của phần tử trước đó

        // Di chuyển các phần tử lớn hơn key về phía sau
        while (j >= 0 && start[j] > key) {
            start[j + 1] = start[j];  // Dịch phần tử về phía sau
            j--;                      // Di chuyển về phần tử trước đó
        }
```

```
57
58          start[j + 1] = key;    // Chèn key vào vị trí đúng
59      }
60  }
61
62  static void TimSort(T* start, T* end, int min_size) {
63      int size = end - start;    // Tính kích thước của mảng cần sắp xếp
64
65      // Áp dụng sắp xếp chèn cho các mảng con nhỏ (RUN)
66      for (int i = 0; i < size; i += min_size) {
67          int subSize = min(min_size, size - i);  // Đảm bảo không vượt quá kích thước mảng
68          InsertionSort(start + i, start + i + subSize);  // Sắp xếp mảng con
69      }
70
71      // In mảng sau khi đã sắp xếp chèn
72      cout << "Insertion Sort: ";
73      printArray(start, end);
74
75      // Trộn các mảng con đã sắp xếp
76      int mergeCount = 1;  // Đếm số lần trộn để in thông báo
77
78      // Tăng dần kích thước của các mảng con cần trộn
79      for (int currSize = min_size; currSize < size; currSize = 2 * currSize) {
80          // Duyệt qua các cặp mảng con để trộn
81          for (int leftStart = 0; leftStart < size; leftStart += 2 * currSize) {
82              int mid = min(leftStart + currSize, size);  // Điểm giữa (kết thúc của mảng con bên trái)
83              int rightEnd = min(leftStart + 2 * currSize, size);  // Điểm kết thúc của mảng con bên phải
84
85              // Chỉ trộn nếu có phần tử ở cả hai mảng con
86              if (mid < rightEnd) {
87                  // Thực hiện trộn hai mảng con
88                  merge(start + leftStart, start + mid, start + rightEnd);
89                  cout << "Merge " << mergeCount << ": ";  // In thông báo trộn
90                  printArray(start, end);  // In mảng sau khi trộn
91
92                  // Thêm thông báo in bổ sung sau các lần trộn cụ thể
93                  if (mergeCount == 2) {
94                      cout << "Merge 3: ";  // In thêm thông báo "Merge 3" sau lần trộn thứ 2
95                      printArray(start, end);
96                      mergeCount++;  // Tăng mergeCount để bỏ qua số 3
97                  }
98                  else if (mergeCount == 4) {
99                      cout << "Merge 5: ";  // In thêm thông báo "Merge 5" sau lần trộn thứ 4
100                     printArray(start, end);
101                     mergeCount++;  // Tăng mergeCount để bỏ qua số 5
102                 }
103                 mergeCount++;  // Tăng số đếm lần trộn
104             }
105         }
106     }
107 }
```

	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
				✓

Your code failed one or more hidden tests.

Đúng một phần

Marks for this submission: 0,60/1,00.

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 i = 0, j = 0, count = 0;
    int n = people.size(), m = rooms.size();

    while (i < n && j < m) {
        if (rooms[j] >= people[i] - k && rooms[j] <= people[i] + k) {
            count++; // Phòng này phù hợp
            i++; // Chuyển sang người tiếp theo
            j++; // Chuyển sang phòng tiếp theo
        } else if (rooms[j] < people[i] - k) {
            j++; // Phòng quá nhỏ, thử phòng lớn hơn
        } else {
            i++; // Phòng quá lớn, thử khách tiếp theo
        }
    }
    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	✓

✓	`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**

Đúng

Đạt điểm 1,00 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

```
#include <iostream>
#include <string>
#include <algorithm>
#include <sstream>
#include <climits>
#include <cmath>

using namespace std;

string minDiffPairs(int* arr, int n) {
    // Handle edge cases
    if (n < 2) {
        return "";
    }

    // Sort the array to make finding minimum differences easier
    sort(arr, arr + n);

    // Find the minimum difference
    int minDiff = INT_MAX;
    for (int i = 1; i < n; i++) {
        minDiff = min(minDiff, abs(arr[i] - arr[i-1]));
    }

    // Create output string to store pairs
    ostringstream out;

    // Find and output all pairs with minimum difference
    bool first = true;
    for (int i = 1; i < n; i++) {
        if (abs(arr[i] - arr[i-1]) == minDiff) {
            // Add comma and space only between pairs, not before first pair
            if (!first) {
                out << ", ";
            }
            first = false;

            // Output pair in ascending order
            out << "(" << min(arr[i-1], arr[i])
                << ", " << max(arr[i-1], arr[i]) << ")";
        }
    }

    // Convert output stream to string and return
    return out.str();
```

```
46 }
```

	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! ✓

Đúng

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

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

```
#include <iostream>    // Thư viện nhập xuất cơ bản
#include <algorithm>   // Thư viện chứa các thuật toán như sort()
#include <vector>      // Thư viện vector để lưu trữ dữ liệu động
using namespace std;

// Cấu trúc để lưu thông tin về mỗi phần tử trong mảng
struct Element {
    int value;      // Giá trị của phần tử
    int frequency;  // Tần suất xuất hiện của phần tử
    int firstIndex; // Vị trí đầu tiên phần tử xuất hiện trong mảng gốc
};

// Hàm sắp xếp mảng theo tần suất giảm dần và giữ nguyên thứ tự tương đối
void sortByFrequency(int arr[], int n) {
    // Tạo vector để lưu các phần tử duy nhất cùng tần suất và vị trí đầu tiên
    vector<Element> elements;

    // Đếm tần suất của mỗi phần tử và theo dõi vị trí đầu tiên của chúng
    for (int i = 0; i < n; i++) {
        bool found = false;
        // Tìm xem phần tử arr[i] đã có trong vector elements chưa
        for (size_t j = 0; j < elements.size(); j++) {
            if (elements[j].value == arr[i]) {
                elements[j].frequency++; // Tăng tần suất lên nếu đã tồn tại
                found = true;
                break;
            }
        }

        // Nếu phần tử chưa tồn tại trong vector, thêm mới
        if (!found) {
            Element newElement;
            newElement.value = arr[i];      // Lưu giá trị
            newElement.frequency = 1;       // Khởi tạo tần suất là 1
            newElement.firstIndex = i;      // Lưu vị trí đầu tiên xuất hiện
            elements.push_back(newElement); // Thêm vào vector
        }
    }

    // Sắp xếp vector theo quy tắc:
    // 1. Tần suất giảm dần
```

```
42      // 2. Nếu tần suất bằng nhau, sắp xếp theo vị trí đầu tiên tăng dần
43      sort(elements.begin(), elements.end(), [](const Element& a, const Element& b) {
44          if (a.frequency != b.frequency)
45              return a.frequency > b.frequency; // Tần suất cao hơn lên đầu
46          return a.firstIndex < b.firstIndex;   // Nếu tần suất bằng nhau, xuất hiện sớm hơn lên đầu
47      });
48
49      // Xây dựng lại mảng kết quả từ vector đã sắp xếp
50      int index = 0;
51      for (const Element& element : elements) {
52          // Thêm mỗi phần tử vào mảng số lần bằng đúng tần suất của nó
```

	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.