Đã bắt đầu vào lúc	Thứ năm, 17 Tháng mười một 2022, 11:04 PM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ năm, 17 Tháng mười một 2022, 11:13 PM
Thời gian thực hiện	8 phút 39 giây
Điểm	**11,00** của 12,00 (**91,67**%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 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, 0, 5, 10, 15, ... %)

Reset answer

```
template <class T>
void SLinkedList<T>::bubbleSort()
{
    Node*tailtemp=tail;
    Node*prev=nullptr;
    Node*dummy=head;
    while(dummy!=tailtemp){
        while(dummy!=tailtemp){
            if(dummy->data>dummy->next->data){
                int temp=dummy->data;
                dummy->data=dummy->next->data;
                dummy->next->data=temp;
            }
            prev=dummy;
            dummy=dummy->next;
        }
        //headtemp=headtemp->next;
        tailtemp=prev;
        dummy=head;
        printList();
    }
}
```

	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! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

Chính xác

Điểm 1,00 của 1,00

QuickSort is one of the fastest sorting algorithms for sorting large data. When implemented well, it can be about two or three times faster than its main competitors, such as MergeSort and HeapSort. When the number of elements is below some threshold (min_size), switch to insertion sort - non-recursive sorting algorithm that performs fewer swaps, comparisons or other operations on such small arrays.

Implement static methods **hybridQuickSort** in class Sorting to sort an array in ascending order. If you do insertion sort, please print "Insertion sort: array" (using printArray) first. If you do quick sort, please print "Quick sort: array (using printArray)" first. Please read example carefully to know exactly what we print.

To match the test cases, please note:

- Using first element as pivot
- Recursively call the hybridQuickSort function to the left of the pivot first, then recursively to the right of the pivot
- Do insertion sort if sub array size smaller than min_size

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

template <class T>
class Sorting
{
private:
    static T *Partition(T *start, T *end);

public:
    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 insertionSort(T *start, T *end);

    static void hybridQuickSort(T *start, T *end, int min_size);
};
```

For example:

Test	Result

`int array[] = {1, 2, 6, 4, 7, 8, 5, 3};`
`int min_size = 4;`
`Sorting<int>::hybridQuickSort(&array[0], &array[8], min_size);`
	`Quick sort: 1, 2, 6, 4, 7, 8, 5, 3`
	`Quick sort: 2, 6, 4, 7, 8, 5, 3`
	`Quick sort: 6, 4, 7, 8, 5, 3`
	`Insertion sort: 5, 4, 3`
	`Insertion sort: 8, 7`

`int array[] = {2, 6, 4, 12, 23, 1, 0, -12};`
`int min_size = 4;`
`Sorting<int>::hybridQuickSort(&array[0], &array[8], min_size);`
	`Quick sort: 2, 6, 4, 12, 23, 1, 0, -12`
	`Insertion sort: 1, -12, 0`
	`Quick sort: 23, 12, 4, 6`
	`Insertion sort: 6, 12, 4`

Test	Result
int array[] = {30, 7, 20, 0, -13, 1, 19, 72}; int min_size = 3; Sorting<int>::hybridQuickSort(&array[0], &array[8], min_size);	Quick sort: 30, 7, 20, 0, -13, 1, 19, 72 Quick sort: 19, 7, 20, 0, -13, 1 Quick sort: -13, 7, 1, 0 Quick sort: 7, 1, 0 Insertion sort: 0, 1 Insertion sort: 20 Insertion sort: 72

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

Reset answer

```
template <class T>
T *Sorting<T>::Partition(T *start, T *end)
{
    T pivot=*start;
    T*starttemp=start+1;
    end--;
    while(end-starttemp>0){
        while(starttemp<end&&*starttemp<pivot){
            starttemp++;
        }
        while(*end>pivot){
            //cout<<"end= "<<*end<<endl;
            end--;
        }
        //cout<<end-starttemp<<endl;
        if(end-starttemp>0){
            //cout<<*end<<endl;
            swap(*end,*starttemp);
            end--;
            starttemp++;
        }
        //else starttemp++;
    }
    while(*end>pivot) end--;//get to the last element that is < pivot
    swap(*start,*end);
    //cout<<*start<<endl;
    //cout<<"index="<<start-end<<endl;
    return end;
}

template <class T>
void Sorting<T>::insertionSort(T *start, T *end)
{
    cout<<"Insertion sort: ";
    printArray(start,end);
    for(int i=1;i<end-start;i++){
        for(int j=i-1;j>=0;j--){
            if(start[j]>start[i]){
                swap(start[j],start[i]);
            }
            else break;
        }
    }
}

template <class T>
void Sorting<T>::hybridQuickSort(T *start, T *end, int min_size)
{
    if(start-end>=0) return ;
    if(end-start<min_size) {insertionSort(start,end);return; };
    cout<<"Quick sort: ";
    printArray(start,end);
    T *pivotpos=Partition(start,end);
    hybridQuickSort(start,pivotpos,min_size);
    hybridQuickSort(pivotpos+1,end,min_size);
}
```

	Test	Expected	Got	

✔	`int array[] = {1, 2, 6, 4, 7, 8, 5, 3};`
	`int min_size = 4;`
	`Sorting<int>::hybridQuickSort(&array[0], &array[8], min_size);`
		`Quick sort: 1, 2, 6, 4, 7, 8, 5, 3`
		`Quick sort: 2, 6, 4, 7, 8, 5, 3`
		`Quick sort: 6, 4, 7, 8, 5, 3`
		`Insertion sort: 5, 4, 3`
		`Insertion sort: 8, 7`
			`Quick sort: 1, 2, 6, 4, 7, 8, 5, 3`
			`Quick sort: 2, 6, 4, 7, 8, 5, 3`
			`Quick sort: 6, 4, 7, 8, 5, 3`
			`Insertion sort: 5, 4, 3`
			`Insertion sort: 8, 7`
				✔

✔	`int array[] = {2, 6, 4, 12, 23, 1, 0, -12};`
	`int min_size = 4;`
	`Sorting<int>::hybridQuickSort(&array[0], &array[8], min_size);`
		`Quick sort: 2, 6, 4, 12, 23, 1, 0, -12`
		`Insertion sort: 1, -12, 0`
		`Quick sort: 23, 12, 4, 6`
		`Insertion sort: 6, 12, 4`
			`Quick sort: 2, 6, 4, 12, 23, 1, 0, -12`
			`Insertion sort: 1, -12, 0`
			`Quick sort: 23, 12, 4, 6`
			`Insertion sort: 6, 12, 4`
				✔

✔	`int array[] = {30, 7, 20, 0, -13, 1, 19, 72};`
	`int min_size = 3;`
	`Sorting<int>::hybridQuickSort(&array[0], &array[8], min_size);`
		`Quick sort: 30, 7, 20, 0, -13, 1, 19, 72`
		`Quick sort: 19, 7, 20, 0, -13, 1`
		`Quick sort: -13, 7, 1, 0`
		`Quick sort: 7, 1, 0`
		`Insertion sort: 0, 1`
		`Insertion sort: 20`
		`Insertion sort: 72`
			`Quick sort: 30, 7, 20, 0, -13, 1, 19, 72`
			`Quick sort: 19, 7, 20, 0, -13, 1`
			`Quick sort: -13, 7, 1, 0`
			`Quick sort: 7, 1, 0`
			`Insertion sort: 0, 1`
			`Insertion sort: 20`
			`Insertion sort: 72`
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 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){
2      /*TODO*/
3      T* nleft=new T [(middle-left)];
4      T* nright=new T [(right-middle)];
5      int i;
6      for(i=0;i<middle-left+1;i++){
7          nleft[i]=left[i];
8      }
9      for(int i=0;i<right-middle;i++){
10         nright[i]=middle[i+1];
11     }
12     int countl,countr;
13     for(countl=0,countr=0;countl<middle-left+1&&countr<right-middle;){
```

```
            left[countl+countr]=nright[countr];
            countr++;
        }
        else{
            left[countl+countr]=nleft[countl];
            countl++;
        }
    }
    while(countl<middle-left+1){
        left[countl+countr]=nleft[countl];
        countl++;
    }
    while(countr<right-middle){
        left[countl+countr]=nright[countr];
        countr++;
    }
    Sorting::printArray(left, right);
}
static void mergeSort(T* start, T* end){
    /*TODO*/
    if(start>=end){
        return;
    }
    T* mid =start+(end-start)/2;
     Sorting::mergeSort(start,mid);
     Sorting::mergeSort((mid+1),end);
     Sorting::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! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

Chính xác

Điểm 1,00 của 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 example

// Merge two sorted lists
ListNode* mergeLists(ListNode* a, ListNode* b) {
    ListNode*result=nullptr;
    if(a==nullptr) return (b);
    if(b==nullptr) return (a);
    if(a->val<=b->val){
        result=a;
        result->next=mergeLists(a->next,b);
    }
    else{
        result=b;
        result->next=mergeLists(a,b->next);
    }
    return result;
}

// Sort and unsorted list given its head pointer
void split(ListNode*source,ListNode*&front,ListNode*&back){
    ListNode* slow=source;
    ListNode*fast =source->next;
    while(fast!=nullptr){
        fast=fast->next;
        if(fast!=nullptr){
            slow=slow->next;
            fast=fast->next;
        }
    }
    front=source;
    back=slow->next;
    //cout<<back->val<<endl;
    slow->next=nullptr;
}
ListNode* mergeSortList(ListNode* head) {
    if(head==nullptr||head->next==nullptr) return head;
    //cout<<head->val<<endl;
```

	Test	Input	Expected	Got	

	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! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 5

Chính xác

Điểm 1,00 của 1,00

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

Remind about odd even sort:

- This algorithm is divided into two phases - odd and even Phase. The algorithm runs until the array elements are sorted and in each iteration two phases mentioned above occurs.
- In the odd phase, we perform a bubble sort on odd indexed elements and in the even phase, we perform a bubble sort on even indexed elements.

```
#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 oddEvenSort(T *start, T *end);
};
```

For example:

Test	Result
`int arr[] = {3, 2, 3, 8, 5, 6, 4, 1};` `Sorting<int>::oddEvenSort(&arr[0], &arr[8]);`	2, 3, 3, 5, 8, 1, 6, 4 2, 3, 3, 1, 5, 4, 8, 6 2, 1, 3, 3, 4, 5, 6, 8 1, 2, 3, 3, 4, 5, 6, 8 1, 2, 3, 3, 4, 5, 6, 8

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

Reset answer

```
template <class T>
void Sorting<T>::oddEvenSort(T *start, T *end)
{
    bool issorted=0;
    while(issorted==0){
        issorted=1;
        //cout<<*(start+1)<<endl;
        for(int i=0;i<=end-start-2;i+=2){
            //cout<<*(start+i)<<endl;
            if(*(start+i)>*(start+i+1)){
                T temp=*(start+i);
                *(start+i)=*(start+i+1);
                *(start+i+1)=temp;
                issorted=0;
            }
```

```
16
17          }
18          for(int i=1;i<=end-start-2;i+=2){
19              if(*(start+i)>*(start+i+1)){
20                  T temp=*(start+i);
21                  *(start+i)=*(start+i+1);
22                  *(start+i+1)=temp;
23                  issorted=0;
24              }
25
26          }
27          printArray(start,end);
28      }
29  }
```

	Test	Expected	Got	
✔	int arr[] = {3, 2, 3, 8, 5, 6, 4, 1}; Sorting<int>::oddEvenSort(&arr[0], &arr[8]);	2, 3, 3, 5, 8, 1, 6, 4 2, 3, 3, 1, 5, 4, 8, 6 2, 1, 3, 3, 4, 5, 6, 8 1, 2, 3, 3, 4, 5, 6, 8 1, 2, 3, 3, 4, 5, 6, 8	2, 3, 3, 5, 8, 1, 6, 4 2, 3, 3, 1, 5, 4, 8, 6 2, 1, 3, 3, 4, 5, 6, 8 1, 2, 3, 3, 4, 5, 6, 8 1, 2, 3, 3, 4, 5, 6, 8	✔
✔	int arr[] = {9, 30, 1, -7, 7, -9, 5, 6,-1,-2}; Sorting<int>::oddEvenSort(&arr[0], &arr[10]);	9, -7, 30, -9, 1, 5, 7, -2, 6, -1 -7, -9, 9, 1, 30, -2, 5, -1, 7, 6 -9, -7, 1, -2, 9, -1, 30, 5, 6, 7 -9, -7, -2, -1, 1, 5, 9, 6, 30, 7 -9, -7, -2, -1, 1, 5, 6, 7, 9, 30 -9, -7, -2, -1, 1, 5, 6, 7, 9, 30	9, -7, 30, -9, 1, 5, 7, -2, 6, -1 -7, -9, 9, 1, 30, -2, 5, -1, 7, 6 -9, -7, 1, -2, 9, -1, 30, 5, 6, 7 -9, -7, -2, -1, 1, 5, 9, 6, 30, 7 -9, -7, -2, -1, 1, 5, 6, 7, 9, 30 -9, -7, -2, -1, 1, 5, 6, 7, 9, 30	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 6

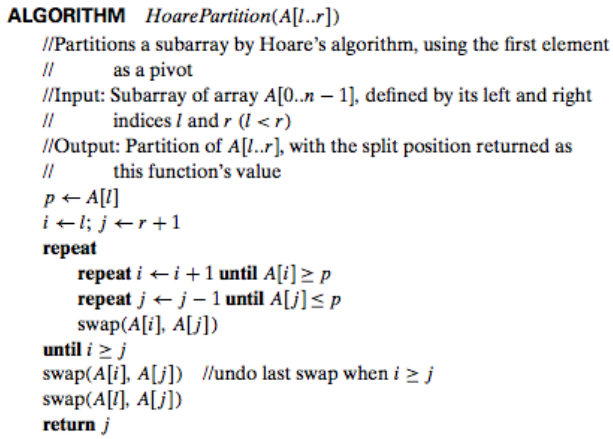
Chính xác

Điểm 1,00 của 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, 0, 0, 5, 10, 15, ... %)

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     int size = end - start;
4     int left = 0, right = size - 1;
5     int lower = left + 1, upper = right;
6     T pivot = start[left];
7     while (lower <= upper) {
8         while (start[lower] < pivot) lower++;
```

```
        if (lower < upper) {
            swap(start[lower], start[upper]);
            lower++;
            upper--;
        }
        else lower++;
    }
    swap(start[upper], start[left]);
    return &start[upper];
}

static void QuickSort(T* start, T* end) {
    // TODO
    // In this question, you must print out the index of pivot in subarray after everytime calling method Pa
    if (end - start <= 0) return;
    T* pivot_pos = Partition(start, end);
    cout << pivot_pos - start << " ";
    QuickSort(start, pivot_pos);
    QuickSort(pivot_pos + 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! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 7

Chính xác

Điểm 1,00 của 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;
    }

    static void sortSegment(T* start, T* end, int segment_idx, int cur_segment_total) ;

public:
    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, 0, 5, 10, 15, ... %)

Reset answer

```
static void sortSegment(T* start, T* end, int segment_idx, int cur_segment_total) {
    // TODO
    int size = end - start;
    for (int curr = segment_idx + cur_segment_total; curr < size; curr += cur_segment_total) {
        int tmp = start[curr];
        int i;
        for (i = curr - cur_segment_total; i >= 0 && start[i] > tmp;
            i -= cur_segment_total) {
                start[i + cur_segment_total] = start[i];
            }
        start[i + cur_segment_total] = tmp;
    }
}

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 algorithm is true.
    for (int phase = num_phases - 1; phase >= 0; phase--) {
        int step = num_segment_list[phase];
```

```
20          for (int segment = 0; segment < step; segment++) {
21              sortSegment(start, end, segment, step);
22          }
23          cout << step << " segments: ";
24          printArray(start, end);
25      }
26  }
```

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 8

Chính xác

Điểm 1,00 của 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, 0, 5, 10, 15, ... %)

Reset answer

```
template <class T>
void Sorting<T>::selectionSort(T *start, T *end)
{
    for(int i=0;i<end-start-1;i++){
        int min=i;
        for(int j=i+1;j<end-start;j++){
            if(start[j]<start[min]){
                min=j;
            }
        }
        swap(start[i],start[min]);
        printArray(start, end);
    }
}
```

	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! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 9

Chính xác

Điểm 1,00 của 1,00

Two strings are called permutation of each other when they have exactly the same number of character, and the number of appearance of each character in each string must be the same.

For example:

String a = "abba" and String b = "baba" are said to be permutation of each other. While String a = "abbc" and String b = "baba" are not.

Your task in this exercise is to implement the **isPermutation** function. Note that, you can write one or more functions in order to achieve this exercise.

```
#ifndef SORTINGAPPLICATION_H
#define SORTINGAPPLICATION_H
#include <iostream>
#include <string>
```

```
#include <algorithm>
using namespace std;
bool isPermutation (string a, string b) {}
#endif /* SORTINGAPPLICATION_H */
```

For example:

Test	Result

`string a = "abba";`
`string b="baba";`
`cout << isPermutation(a, b);`
	1

`string a = "abbac";`
`string b="baba";`
`cout << isPermutation(a, b);`
	0

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

Reset answer

```
/* Your helping functions go here */
bool isPermutation (string a, string b) {
    //TODO
    sort(a.begin(),a.end());
    sort(b.begin(),b.end());
    return !a.compare(b);
}

```

	Test	Expected	Got	

✔	`string a = "abba";`
	`string b="baba";`
	`cout << isPermutation(a, b);`
		1	1	✔

✔	`string a = "abbac";`
	`string b="baba";`
	`cout << isPermutation(a, b);`
		0	0	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 10

Chính xác

Điểm 1,00 của 1,00

You are given a list of integers `nums` with `n` elements (`3 ≤ n ≤ 100000`), the value of each element is between `-1000` to `1000`.

Request: Implement function:

```
int maximumProduct(vector<int>& nums);
```

Where `nums` is the given list of integers. This function returns the maximum product of `3` elements which can be found from the given list `nums`.

Example:

The list of integers is `{5, 4, 1, 3, -2, -2}`. Therefore, the maximum product is `60` by choosing 3 elements of the given list are `5`, `4`, `3`.

Note:

In this exercise, the libraries `iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered_map, set, unordered_set, functional, algorithm` has been included and `namespace std` are used. You can write helper functions and classes. Importing other libraries is allowed, but not encouraged, and may result in unexpected errors.

For example:

Test	Result
`vector<int>nums{5, 4, 1, 3, -2, -2};` `cout << maximumProduct(nums);`	60
`vector<int>nums{4,4,1,2,-5,3,-2,-5,0,-2};` `cout << maximumProduct(nums);`	100

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

Reset answer

```
int maximumProduct(vector<int>& nums) {
    sort(nums.begin(),nums.end());
    int pro=1;
    for(unsigned int i=nums.size()-1;i>nums.size()-4;i--){
        pro*=nums[i];
    }
    int temp=nums[0]*nums[1]*nums[nums.size()-1];
    if(pro<temp){
        pro=temp;
    }
    //int temp=
    return pro;
}
```

	Test	Expected	Got	
✔	`vector<int>nums{5, 4, 1, 3, -2, -2};` `cout << maximumProduct(nums);`	60	60	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 11

Không trả lời

Điểm 0,00 của 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

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) {

}
```

Câu hỏi 12

Chính xác

Điểm 1,00 của 1,00

Trong ngày hội CSE Job Fair 2020, Ban tổ chức đã treo cờ dọc theo đường dẫn vào tòa H6, có N lá cờ được đánh số từ 1 đến N, lá cờ thứ i có màu A_i. Tuy nhiên, sau khi treo cờ lên, người tổ chức sự kiện - bạn Quang Anh thấy dãy cờ có nhiều màu sắc khác nhau là không hợp lý. Phía Đoàn hội tiến hành rà soát và cho biết còn dư M lá cờ, đánh số từ 1 đến M, lá cờ thứ j có màu B_j. Ban tổ chức quyết định sẽ thay thế một số lá cờ trong số N lá cờ ban đầu để được dãy cờ mới có ít màu nhất có thể. Lá cờ bị thay xuống sẽ bị rách nên không thể dùng lại cho các lần thay tiếp theo, cũng như do thời gian có hạn, bất kì lá cờ nào thay lên cũng không được phép gỡ xuống.

Yêu cầu: Hãy tìm cách thay một số (hoặc giữ nguyên) lá cờ đã treo bằng một số lá cờ trong số cờ còn dư sao cho tổng số màu xuất hiện trên dãy cờ chính thức (N lá) là ít nhất.

Dữ liệu:

- Dòng đầu ghi hai số N và M lần lượt là số lá cờ chính thức và số lá cờ còn dư
- Dòng thứ hai ghi N số nguyên A_i cho biết màu của các lá cờ đã treo ($0 <= A_i <= 255$, $1<= i <= N$)
- Dòng thứ ba ghi M số nguyên B_j cho biết màu của các lá cờ còn dư ($0 <= B_j <= 255$, $1<= j <= M$)

Kết quả: in ra một số duy nhất, không có bất kì khoảng trắng nào, là số màu mới của dãy cờ chính thức sau khi được thay thế.

Giải thích testcase 1:

Dãy cờ mới sẽ là 1 2 5 **5 2 5 5** 5 5 -> Có 3 màu trong dãy cờ mới.

Các thư viện ĐÃ có trong bài, các bạn KHÔNG được phép thêm bất kì thư viện nào khác:

#include <iostream>

#include <fstream>

#include <string>

#include <cmath>

#include <vector>

#include <algorithm>

#include <stack>

#include <queue>

#include <map>

For example:

Test

`int n=9;`
`int m=4;`
`vector<int> A({1,2,5,4,8,9,3,5,5});`
`vector<int> B({2,5,5,5});`
`cout << flag(n, m,A,B);`

`int n=45;`
`int m=1;`
`vector<int>`
`A({32,134,131,44,194,254,63,209,140,181,29,108,94,153,165,117,159,2,33,31,133,229,255,47,144,74,120,15,88,0,111,30,137,143,156,75,4`
`vector<int> B({151});`
`cout << flag(n, m,A,B);`

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

Reset answer

```
1  //Helping functions goes here
2  struct ele {
3      int count, index, val;
4  };
```

```
        //element[i].index = i;

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

        // Fill values in structure
        // elements
        //element[i].val = A[i];
        element.push_back(temp);
    }

    /* Sort the structure elements according to value,
       we used stable sort so relative order is maintained.
     */
    stable_sort(element.begin(), element.end(), mycomp);
    //for(int i=element.size()-1;i>=0;i--){
```

	Test

✔	`int n=9;`
	`int m=4;`
	`vector<int> A({1,2,5,4,8,9,3,5,5});`
	`vector<int> B({2,5,5,5});`
	`cout << flag(n, m,A,B);`

✔	`int n=45;`
	`int m=1;`
	`vector<int>`
	`A({32,134,131,44,194,254,63,209,140,181,29,108,94,153,165,117,159,2,33,31,133,229,255,47,144,74,120,15,88,0,111,30,137,143,1`
	`vector<int> B({151});`
	`cout << flag(n, m,A,B);`

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.