Thời gian còn lại 0:08:13

Câu hỏi 1

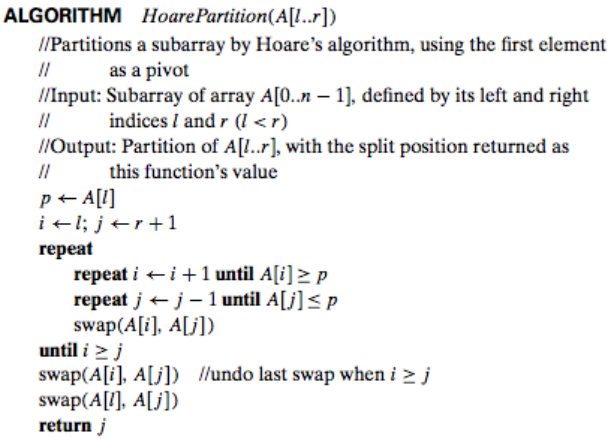
Chính xác

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 %)

Reset answer

```
1  static T* Partition(T* start, T* end)
2  {
3      const T pivot = *start;
4      T* j = end;
5      T* i = start;
6      do {
7          do i++; while(*i < pivot);
8          do j--; while(*j > pivot);
```

```
        const T temp = *i;
        *i = *j;
        *j = temp;
    }while(i < j);
    T temp = *i;
    *i = *j;
    *j = temp;
    temp = *start;
    *start = *j;
    *j = temp;
    return j;
}
static void QuickSort(T* start, T* end)
{
    if (start == end) return;
    T* pivotPos = Partition(start,end);
    cout << (pivotPos - start) << " ";
    QuickSort(start,pivotPos);
    QuickSort(pivotPos + 1,end);
}
```

Precheck | Kiểm tra

	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 2

Đúng một phần

Điểm 0,70 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 exampl

// Merge two sorted lists
ListNode* middle(ListNode* head) {
    ListNode* slow = head;
    ListNode* fast = head->next;
    while(!(slow->next) && (!fast && !(fast->next))) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}
ListNode* mergeLists(ListNode* a, ListNode* b) {
    ListNode* merged;
    ListNode* temp;
    if (a->val <= b->val) {
        temp = a;
        a=a->next;
    } else {
        temp = b;
        b=b->next;
    }
    merged=temp;
    while(a&&b) {
        if (a->val <= b->val) {
            temp->next =a;
            temp=a;
            a=a->next;
        } else {
            temp->next =b;
            temp=b;
            b=b->next;
        }
    }
    while(a) {
        ListNode* tempnode =a;
        a=a->next;
```

Precheck Kiểm tra

	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`	✔

	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;	100000 100	100 100	***Time limit exceeded***	✖

	Test	Input	Expected	Got	
		100 ...snip... 0 100	100 10 ...snip... 0 100		

	Test	Input	Expected	Got	
		100 100	100 100		

	Test	Input	Expected	Got	
		100 100	100 100		

Testing was aborted due to error.

Show differences

Đúng một phần

Điểm cho bài nộp này: 0,70/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 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
    int size1 = middle - start + 1;
    int size2 = end - middle;
    int i1 = 0, i2 = 0, k = 0;
    int* l = new int[size1];
    int* r = new int[size2];
    for(int i = 0; i < size1; i++){
        l[i] = start[i];
    }
    for(int j = 0; j <size2; j++){
        r[j] = middle[j+1];
    }
    while(i1<size1 && i2 < size2){
        if(l[i1] <= r[i2]){
            start[k++] = l[i1++];
        }
        else{
            start[k++] = r[i2++];
        }
    }
    while(i1<size1){
        start[k++] = l[i1++];
    }
    while(i1<size2){
        start[k++] = r[i2++];
    }
}

static void InsertionSort(T* start, T* end) {
    // TODO
    int n = end - start;
    for(int i = 1; i<n;i++){
```

```
        for(int j = i; j>0 && start[j] < start[j-1]; j--){
            swap(start[j],start[j-1]);
        }
    }
}

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

Precheck Kiểm tra

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

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

Chính xác

Điểm 1,00 của 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){
    sort(arr, arr+n);
    int minDiff = INT32_MAX;
    for (int i=1; i<n; i++) {
        int dif = arr[i] - arr[i-1];
        minDiff = min(minDiff, dif);
    }
    bool found = false;
    for (int i = 1; i < n; ++i)
    {
        int diff = arr[i] - arr[i - 1];
        if (diff == minDiff)
        {
            if (found) cout << ", ";
            cout << "(" << arr[i - 1] << ", " << arr[i] << ")";
            found = true;
        }
    }
    cout << endl;
    return "";
}
```

Precheck | Kiểm tra

	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);`			✔

	Test	Expected	Got	
✔	`int arr[] = {10, -1, -150, 200};` `cout << minDiffPairs(arr, 4);`	`(-1, 10)`	`(-1, 10)`	✔

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

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 <vector>

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

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

void sortByFrequency(int* arr, int n) {
    std::vector<Element> elements;
    for (int i = 0; i < n; ++i) {
        bool found = false;
        for (auto& element : elements) {
            if (element.value == arr[i]) {
                element.frequency++;
```

Precheck Kiểm tra

	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! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 6

Chính xác

Điểm 1,00 của 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

```
void closestKPoints(Point points[], int n, Point &des_point, int k){
    //TODO
    int i, j;
    int max = 0;
    Point tmp;
    for (i = 1; i < n; i++) {
        max = pow(points[i].x - des_point.x, 2) + pow(points[i].y - des_point.y, 2);
        tmp = points[i];
        j = i - 1;
        while (j >= 0 && pow(points[j].x - des_point.x, 2) + pow(points[j].y - des_point.y, 2) > max) {
```

```
            points[j + 1] = points[j];
            j = j - 1;
        }
        points[j + 1] = tmp;
    }
    if(k > n) k = n;
    for (int i = 0; i < k; i++) {
        points[i].display();
            cout << endl;
    }
}
```

Precheck | Kiểm tra

	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! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.