Câu hỏi 1

Chính xác

Chấm điểm 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      // TODO: return the pointer which points to the pivot after rearrange the array.
4      T pivot = *start; // Chọn phần tử đầu tiên làm pivot
5      T *i = start + 1; // Con trỏ i bắt đầu từ phần tử thứ hai
6      T *j = end - 1;   // Con trỏ j bắt đầu từ phần tử cuối cùng
7      while (true)
8      {
```

```
        while (i <= pivot && i != end)
10          ++i; // Tăng i cho đến khi tìm thấy phần tử lớn hơn pivot
11      while (*j > pivot && j != start)
12          --j; // Giảm j cho đến khi tìm thấy phần tử nhỏ hơn hoặc bằng pivot
13      if (i >= j)
14          break;          // Nếu i và j gặp nhau hoặc vượt qua nhau, dừng vòng lặp
15      std::swap(*i, *j); // Hoán đổi giá trị của *i và *j
16  }
17  std::swap(*start, *j); // Hoán đổi giá trị của pivot và *j
18  return j;              // Trả về vị trí của pivot sau khi đã sắp xếp xong mảng
19  }
20
21  static void QuickSort(T *start, T *end)
22  {
23      // TODO
24      // In this question, you must print out the index of pivot in subarray after everytime calling method P
25      if (start < end)
26      {                                       // Nếu mảng có ít nhất hai phần tử
27          T *pivot = Partition(start, end); // Gọi phương thức Partition để chia mảng thành hai nửa
28          cout << pivot - start << " ";     // In ra vị trí của pivot sau khi đã chia mảng
29          QuickSort(start, pivot);          // Gọi đệ quy QuickSort cho nửa mảng bên trái pivot
30          QuickSort(pivot + 1, end);        // Gọi đệ quy QuickSort cho nửa mảng bên phải pivot
31      }
32  }
```

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! ✔

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn

Câu hỏi 2

Chính xác

Chấm điểm 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  {
3      int i, j, k;
4      int n1 = middle - left + 1;
5      int n2 = right - middle;
6
7      /* create temp arrays */
8      T L[n1], R[n2];
9
10     /* Copy data to temp arrays L[] and R[] */
11     for (i = 0; i < n1; i++)
12         L[i] = left[i];
13     for (j = 0; j < n2; j++)
```

```
14          R[j] = middle[j + 1];
15
16      /* Merge the temp arrays back into arr[l..r]*/
17      i = 0; // Initial index of first subarray
18      j = 0; // Initial index of second subarray
19      k = 0; // Initial index of merged subarray
20      while (i < n1 && j < n2)
21      {
22          if (L[i] <= R[j])
23          {
24              left[k] = L[i];
25              i++;
26          }
27          else
28          {
29              left[k] = R[j];
30              j++;
31          }
32          k++;
33      }
34
35      /* Copy the remaining elements of L[], if there are any */
36      while (i < n1)
37      {
38          left[k] = L[i];
39          i++;
40          k++;
41      }
42
43      /* Copy the remaining elements of R[], if there are any */
44      while (j < n2)
45      {
46          left[k] = R[j];
47          j++;
48          k++;
49      }
50
51      Sorting::printArray(left, right);
52  }
53  static void mergeSort(T *start, T *end)
54  {
55      if (start < end)
56      {
57          // Same as (l+r)/2, but avoids overflow for large l and h
58          T *middle = start + (end - start) / 2;
59
60          // Sort first and second halves
61          mergeSort(start, middle);
62          mergeSort(middle + 1, end);
```

Precheck Kiểm tra

	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 3

Chính xác

Chấm điểm 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

// Merge two sorted lists
ListNode *mergeLists(ListNode *a, ListNode *b)
{
    // Base cases
    if (!a)
        return b;
    if (!b)
        return a;

    // Start with the linked list whose head data is the least
    if (a->val < b->val)
    {
        a->next = mergeLists(a->next, b);
        return a;
    }
    else
    {
        b->next = mergeLists(a, b->next);
        return b;
    }
}

// Sort and unsorted list given its head pointer
ListNode *mergeSortList(ListNode *head)
{
    // Base case: if head is null or there is only one element in the list
    if (!head || !head->next)
        return head;

    // Get the middle of the list
    ListNode *slow = head, *fast = head, *prev = nullptr;
    while (fast && fast->next)
    {
        prev = slow;
        slow = slow->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	✔

Passed all tests! ✔

Câu hỏi 4

Chính xác

Chấm điểm 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 leftSize = middle - start;
    int rightSize = end - middle;
    T *left = new T[leftSize];
    T *right = new T[rightSize];

    for (int i = 0; i < leftSize; i++)
        left[i] = start[i];
    for (int i = 0; i < rightSize; i++)
        right[i] = middle[i];

    int i = 0, j = 0, k = 0;
    while (i < leftSize && j < rightSize)
    {
        if (left[i] <= right[j])
            start[k++] = left[i++];
        else
            start[k++] = right[j++];
    }

    while (i < leftSize)
        start[k++] = left[i++];
    while (j < rightSize)
        start[k++] = right[j++];

    delete[] left;
    delete[] right;
}

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

```
    for (T *i = start + 1; i != end; ++i)
    {
        T key = *i;
        T *j = i - 1;

        while (j >= start && *j > key)
        {
            *(j + 1) = *j;
            --j;
        }
        *(j + 1) = key;
    }
}

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.
    for (T *i = start; i < end; i += min_size)
    {
        InsertionSort(i, min((T *)(i + min_size), end));
    }
    cout << "Insertion Sort: ";
    printArray(start, end);

    int count = 0;
    for (int size = min_size; size < end - start; size *= 2)
    {
        for (T *leftStart = start; leftStart < end; leftStart += 2 * size)
        {
            T *midPoint = min((T *)(leftStart + size), end);
            T *rightEnd = min((T *)(leftStart + 2 * size), end);
            merge(leftStart, midPoint, rightEnd);
            cout << "Merge " << ++count << ": ";
            printArray(start, end);
        }
    }
}

```

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! ✔

Câu hỏi 5

Chính xác

Chấm điểm 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

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)
{
    // Sắp xếp các phòng và người theo thứ tự tăng dần
    sort(rooms.begin(), rooms.end());
    sort(people.begin(), people.end());

    int i = 0, j = 0;
    int count = 0;
    // Duyệt qua danh sách phòng và người
    while (i < rooms.size() && j < people.size())
    {
        // Nếu kích thước phòng thỏa mãn yêu cầu của người thì tăng biến đếm
        if (abs(rooms[i] - people[j]) <= k)
        {
            count++;
            i++;
            j++;
        }
        else if (rooms[i] < people[j])
        {
            // Nếu kích thước phòng nhỏ hơn yêu cầu thì chuyển sang phòng tiếp theo
            i++;
        }
        else
        {
            // Nếu kích thước phòng lớn hơn yêu cầu thì chuyển sang người tiếp theo
            j++;
        }
    }
    // Trả về số lượng người tối đa có thể ở
    return count;
}
```

Precheck Kiểm tra

	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! ✔