



BÁCH KHOA E-LEARNING

[Dashboard](#) / [Courses](#) / [Học kỳ I năm học 2021-2022 \(Semester 1 - Academic year 2021-2022\)](#)/ [Đại Học Chính Qui \(Bachelor program \(Full-time study\)\)](#)/ [Khoa Khoa học và Kỹ thuật Máy tính \(Faculty of Computer Science and Engineering.\)](#) / [Khoa Học Máy Tính](#)/ [Cấu trúc dữ liệu và giải thuật \(thực hành\) \(CO2004\) Trần Khánh Tùng \(DH_HK211\)](#) / Lab 2: Doubly Linked List + Stack + Queue + Sorting/ [Lab 2: Sorting](#)

| | |
|--------------|----------------------------------|
| Started on | Sunday, 3 October 2021, 3:13 PM |
| State | Finished |
| Completed on | Monday, 4 October 2021, 12:21 AM |
| Time taken | 9 hours 8 mins |
| Marks | 2.70/6.00 |
| Grade | 4.50 out of 10.00 (45%) |

Question 1

Partially correct

Mark 0.70 out of 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}; | -2, 2, 8, 1, 0, 9 |
| Sorting<int>::selectionSort(&arr[0], &arr[6]); | -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

```
1 template <class T>
2 void Sorting<T>::selectionSort(T *start, T *end)
3 {
4
5     int i = 0;
6     int min_idx = 0;
7     // One by one move boundary of unsorted subarray
8     for (i = 0; *(start+i) != *(end - 1); i++)
9     {
10
11         // Find the minimum element in unsorted array
12         min_idx = i;
13         for (int j = i+1; *(start+j) != *end; j++)
14         {
15             if (*(start+j) < *(start+ min_idx)) min_idx = j;
16         }
17         // Swap the found minimum element with the first element
18         T temp = *(start + min_idx);
19         *(start+min_idx) = *(start+i);
20         *(start+i) = temp;
21         printArray(start,end);
22         // swap(&arr[min_idx], &arr[i]);
```

```
23 | }  
24 | }
```

| | Test | Expected | Got | |
|---|--|--|--|---|
| ✓ | <pre>int arr[] = {9, 2, 8, 1, 0, -2}; Sorting<int>::selectionSort(&arr[0], &arr[6]);</pre> | <pre>-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</pre> | <pre>-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</pre> | ✓ |

Your code failed one or more hidden tests.

Partially correct

Marks for this submission: 0.70/1.00.

Question **2**

Incorrect

Mark 0.00 out of 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 */
```

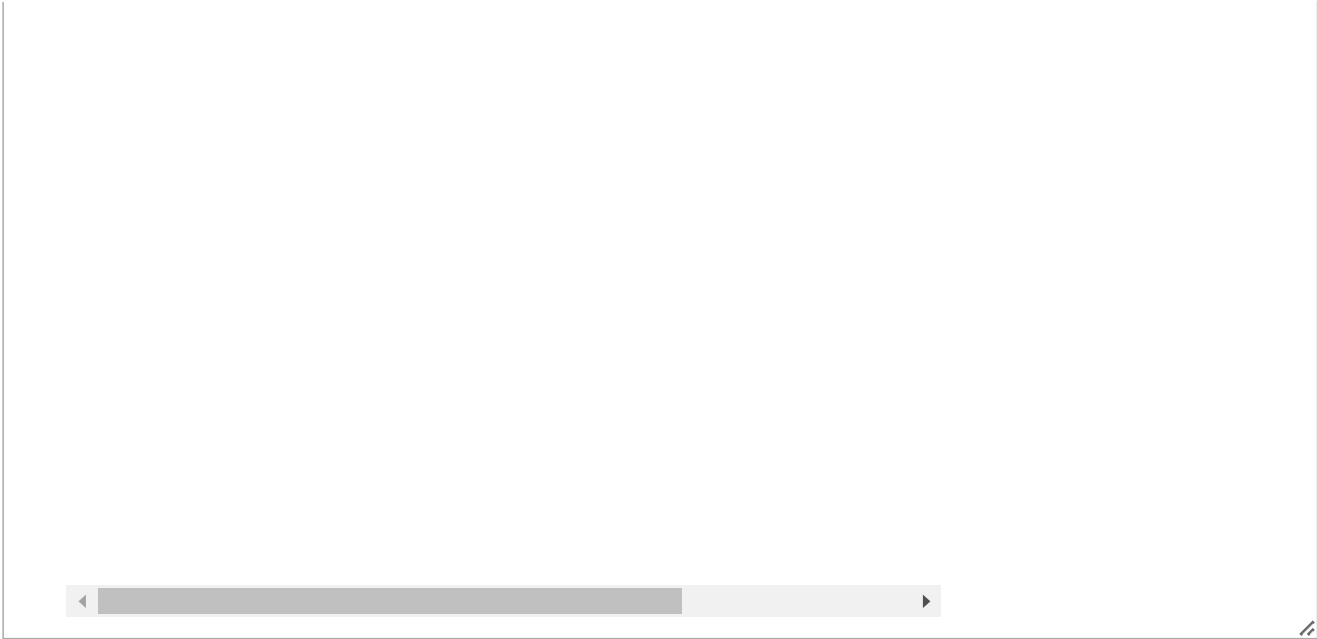
For example:

| 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 }; | 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 |
| Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases); | |

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

Reset answer

```
1 | static void sortSegment(T* start, T* end, int segment_idx, int cur_segment_t
2 |     // TODO
3 | }
4 |
5 | static void ShellSort(T* start, T* end, int* num_segment_list, int num_phase
6 |     // TODO
7 |     // Note: You must print out the array after sorting segments to check wh
8 |
9 | }
```



| Test | Expected |
|------|----------|
| | |

Your code failed one or more hidden tests.

Incorrect

Marks for this submission: 0.00/1.00.

Question **3**

Incorrect

Mark 0.00 out of 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 $O(n \log n)$ 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:

$0 \leq \text{list.length} \leq 10^4$

$0 \leq \text{node.val} \leq 10^6$

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 |
|--|-------|-------------------|
| <pre>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);</pre> | | 1 2 3 4 5 6 7 8 9 |

| Test | Input | Result |
|---|----------------------------------|--------------------------------|
| <pre> 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; </pre> | <pre> 9 9 3 8 2 1 6 7 4 5 </pre> | <pre> 1 2 3 4 5 6 7 8 9 </pre> |

Answer: (penalty regime: 0 %)

Reset answer

```

1 // You must use the nodes in the original list and must not modify ListNode
2 // Hint: You should complete the function mergeLists first and validate it
3
4 // Merge two sorted lists
5 ▼ ListNode* mergeLists(ListNode* a, ListNode* b) {
6     return nullptr;
7 }
8
9 // Sort and unsorted list given its head pointer
10 ▼ ListNode* mergeSortList(ListNode* head) {
11     return nullptr;
12 }

```

| | Test | Input | Expected | |
|---|---|------------------------|-------------------|---|
| ✖ | <pre> 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); </pre> | | 1 2 3 4 5 6 7 8 9 | ✖ |
| ✖ | <pre> 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; </pre> | 9 9 3 8 2 1 6 7 4 5 | 1 2 3 4 5 6 7 8 9 | ✖ |

[illegible]

10/22

11/22

[illegible]

Some hidden test cases failed, too.

Incorrect

Marks for this submission: 0.00/1.00.

Question 4

Incorrect

Mark 0.00 out of 1.00

Implement static methods **maerge**, **InsertionSort** and **TlmSort** 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 - 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 TlmSort(T* start, T* end, int min_size) ;
};
#endif /* SORTING_H */
```

For example:

| Test | Result |
|---|---|
| <pre>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>::TlmSort(&array[0], &array[20], min_size);</pre> | <p>Insertion Sort: 17 18 19 20 12 13 14 15 1 2 6 9 4 7 11 16 3 5 8 10</p> <p>Merge 1: 12 13 14 15 17 18 19 20 1 2 6 9 4 7 11 16 3 5 8 10</p> <p>Merge 2: 12 13 14 15 17 18 19 20 1 2 4 6 7 9 11 16 3 5 8 10</p> <p>Merge 3: 12 13 14 15 17 18 19 20 1 2 4 6 7 9 11 16 3 5 8 10</p> <p>Merge 4: 1 2 4 6 7 9 11 12 13 14 15 16 17 18 19 20 3 5 8 10</p> <p>Merge 5: 1 2 4 6 7 9 11 12 13 14 15 16 17 18 19 20 3 5 8 10</p> <p>Merge 6: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20</p> |

Answer: (penalty regime: 0 %)

Reset answer

```
1 static void merge(T* start, T* middle, T* end) {
2     // TODO
3 }
```

```
4 |}  
5 |  
6 | static void InsertionSort(T* start, T* end) {  
7 |     // TODO  
8 |  
9 | }  
10 |  
11 | static void TimSort(T* start, T* end, int min_size) {  
12 |     // TODO  
13 |     // You must print out the array after using insertion sort and everytim  
14 |  
15 | }
```

| | Test | Expected | |
|--|------|----------|--|
| | | | |

Your code failed one or more hidden tests.

Incorrect

Marks for this submission: 0.00/1.00.

Question 5

Correct

Mark 1.00 out of 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>
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, 5, 10, 15, ... %)

Reset answer

```
1 bool isPermutation (string a, string b) {
2     int arr[26];
3     int brr[26];
4     for (int i = 0; i < 26; ++i) arr[i] = 0;
5     for (int i = 0; i < 26; ++i) brr[i] = 0;
6
7     for (int i = 0; i < (int)a.size() ; ++i) ++arr[a[i]-97];
8     for (int i = 0; i < (int)b.size() ; ++i) ++brr[b[i]-97];
9
10    for (int i = 0; i < 26; ++i)
11    {
12        if (arr[i] != brr[i]) return false;
13    }
14    return true;
15 }
```


| | 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! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 6

Correct

Mark 1.00 out of 1.00

Ann has gone to New York City for 1 week. Ann is friendly and helpful person so she decided to visit all her neighbor

in this city. But she can not remember that she has visited some neighbors before or not. Therefore, she may visit one neighbor many

times. Find the neighbor she has visited the most? If there are many results, return result which followed by alphabet.

You have to implement `majorityNeighbor(vector<string>& neighbor);` to return neighbor that Ann has visit the most.

Example: ["Peter", "Bob", "Andrew", "Peter", "Bob", "Peter", "Bob", "Peter"] -> Peter

Note that: we included `iostream`, `vector`, `list`, `string` so that you don't need to include them again.

Constraints:

`1 <= neighbor <= 1000`

`neighbor[i]` is a string that contains letter only.

For example:

| Test | Result |
|---|--------|
| <pre>vector<string> nums = { "Peter", "Bob", "Andrew", "Peter", "Bob", "Peter", "Bob", "Peter" }; Neighbor s; cout << s.majorityNeighbor(nums);</pre> | Peter |

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

Reset answer

```

1 class Neighbor {
2 public:
3     string majorityNeighbor(vector<string>& neighbor) {
4         int size = neighbor.size();
5         vector<string> diff;
6         for (int i = 0; i < size; ++i)
7         {
8             if (i == 0) {diff.push_back(neighbor[i]); continue;}
9             for (int j = i-1; j >= 0; --j)
10            {
11                if (neighbor[j] == neighbor[i]) break;
12                else
13                {
14                    if (j == 0) diff.push_back(neighbor[i]);
15                }
16            }
17        }
18        vector<int> app((int)diff.size());
19        for (int i = 0; i < (int)app.size(); ++i)
20        {
21            app[i] = 0;
22        }
23        for (int j = 0; j < size; ++j)
24        {
25            for (int h = 0; h < (int)diff.size(); ++h)
26            {
27                if (diff[h] == neighbor[j]) ++app[h];
28            }
29        }
30        int max_pos = 0;
31        int max = app[0];

```

```

32     for (int j = 1; j < (int)app.size(); ++j)
33     {
34         if (max < app[j])
35         {
36             max = app[j];
37             max_pos = j;
38         }
39     }
40     string res = diff[max_pos];
41     string curr = "";
42
43     for (int j = max_pos+1; j < (int)app.size(); ++j)
44     {
45         if (max == app[j])
46         {
47             curr = diff[j];
48             if (res > curr ) res = curr;
49         }
50     }
51     return res;
52 }
53 };
54

```

| | Test | Expected | Got |
|---|---|----------|--------|
| ✓ | vector<string> nums = { "Peter","Bob","Andrew","Peter","Bob","Peter","Bob","Peter" }; Neighbor s; cout << s.majorityNeighbor(nums); | Peter | Peter |
| ✓ | vector<string> nums = { "Peter","Bob","Andrew","Peter","Bob","Peter","Bob","Peter","Andrew","Peter","Bob","Peter","Bob","Peter" }; Neighbor s; cout << s.majorityNeighbor(nums); | Peter | Peter |
| ✓ | vector<string> nums = { "Smith","Bob","Andrew","Peter","Cat","Peter","Bob","Peter","Dog","Fish", "Andrew","Micle","Andrew","Peter","Cat","Dog","Fish","Andrew","Micle","Andrew" }; Neighbor s; cout << s.majorityNeighbor(nums); | Andrew | Andrew |
| ✓ | vector<string> nums = { "Micle", "Mike", "Mike", "Sith", "Miol", "Bob", "Mike", "Sith", "Miol", "Sith", "Miol", "Bob", "Mike" }; Neighbor s; cout << s.majorityNeighbor(nums); | Mike | Mike |
| ✓ | vector<string> nums = { "Peter","Bob","Bob" }; Neighbor s; cout << s.majorityNeighbor(nums); | Bob | Bob |
| ✓ | vector<string> nums = { "Bob", "Bob", "Mike", "Smith", "Andrew", "Peter", "Lan", "Ilan", "Kime", "Neitt" }; Neighbor s; cout << s.majorityNeighbor(nums); | Bob | Bob |

| | Test | Expected | Got |
|---|---|----------|------|
| ✓ | <pre>vector<string> nums = { "Zach", "Zach", "Mike", "Smith", "Andrew", "Peter", "Lan", "Ilan", "Kime" , "Neitt" }; Neighbor s; cout << s.majorityNeighbor(nums);</pre> | Zach | Zach |
| ✓ | <pre>vector<string> nums = { "Peter","Bob","Misheo" }; Neighbor s; cout << s.majorityNeighbor(nums);</pre> | Bob | Bob |

Passed all tests! ✓



Correct

Marks for this submission: 1.00/1.00.

◀ Lab 2: Doubly LL, Stack and Queue

Jump to...

Lab 3: Preparation ▶

Copyright 2007-2021 Ho Chi Minh City University of Technology. All Rights Reserved.
Address: Hochiminh City University Of Technology - 268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City.
Email: elearning@hcmut.edu.vn
Powered by Moodle