BÁCH KHOA E-LEARNING

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}; 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)
{

    int i = 0;
    int min_idx = 0;
    // One by one move boundary of unsorted subarray
    for (i = 0;*(start+i) != *(end - 1); i++)
    {

        // Find the minimum element in unsorted array
        min_idx = i;
        for (int j = i+1; *(start+j) != *end; j++)
        {
            if (*(start+j) < *(start+ min_idx)) min_idx = j;
        }
        // Swap the found minimum element with the first element
        T temp = *(start + min_idx);
        *(start+min_idx) = *(start+i);
        *(start+i) = temp;
         printArray(start,end);
       // swap(&arr[min_idx], &arr[i]);
```

```
23      }
24  }
```

	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`
				✔

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 };`
 	
`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_t
    // TODO
}

static void ShellSort(T* start, T* end, int* num_segment_list, int num_phase
    // TODO
    // Note: You must print out the array after sorting segments to check wh

}
```

	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(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
// Hint: You should complete the function mergeLists first and validate it

// Merge two sorted lists
ListNode* mergeLists(ListNode* a, ListNode* b) {
    return nullptr;
}

// Sort and unsorted list given its head pointer
ListNode* mergeSortList(ListNode* head) {
    return nullptr;
}
```

	Test	Input	Expected	

✖	`int arr1[] = {1, 3, 5, 7, 9};`
	`int arr2[] = {2, 4, 6, 8};`
	
	`unordered_map<ListNode*, int> nodeAddr;`
	`ListNode* a = init(arr1, sizeof(arr1) / 4, nodeAddr);`
	`ListNode* b = init(arr2, sizeof(arr2) / 4, nodeAddr);`
	`ListNode* merged = mergeLists(a, b);`
	`try {`
	
	`printList(merged, nodeAddr);`
	`}`
	`catch(char const* err) {`
	`cout << err << '\n';`
	`}`
	`freeMem(merged);`
			`1 2 3 4 5 6 7 8 9`	✖

✖	`int size;`
	`cin >> size;`
	`int* array = new int[size];`
	`for(int i = 0; i < size; i++) cin >> array[i];`
	
	`unordered_map<ListNode*, int> nodeAddr;`
	`ListNode* head = init(array, size, nodeAddr);`
	`ListNode* sorted = mergeSortList(head);`
	`try {`
	
	`printList(sorted, nodeAddr);`
	`}`
	`catch(char const* err) {`
	`cout << err << '\n';`
	`}`
	`freeMem(sorted);`
	`delete[] array;`
		`9`
		`9 3 8 2 1 6 7 4 5`
			`1 2 3 4 5 6 7 8 9`	✖

	Test	Input	Expected	

✖	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;
		20
		360924 716925 459489 389399 824308 777676
		806187 986180 936828 433111 449243 141749
		250782 270849 955511 281354 960612 456616
		752793 292654
			141749 250782 270849 281354 292654 360924
			389399 433111 449243 456616 459489 716925
			752793 777676 806187 824308 936828 955511
			960612 986180
				✖

✖	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	✖

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

	Test	Input	Expected	
		100 100	100 100	

	Test	Input	Expected	
		100 100	100 100	

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 **TImSort** 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 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

Answer: (penalty regime: 0 %)

Reset answer

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

```
}

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

}

static void TimSort(T* start, T* end, int min_size) {
    // TODO
    // You must print out the array after using insertion sort and everytim

}
```

	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

```
bool isPermutation (string a, string b) {
    int arr[26];
    int brr[26];
    for (int i = 0; i < 26; ++i) arr[i] = 0;
    for (int i = 0; i < 26; ++i) brr[i] = 0;

    for (int i = 0; i < (int)a.size() ; ++i) ++arr[a[i]-97];
    for (int i = 0; i < (int)b.size() ; ++i) ++brr[b[i]-97];

    for (int i = 0; i < 26; ++i)
    {
        if (arr[i] != brr[i]) return false;
    }
    return true;
}
```

	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 remmember 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
vector<string> nums = { "Peter","Bob","Andrew","Peter","Bob","Peter","Bob","Peter" }; Neighbor s; cout << s.majorityNeighbor(nums);	Peter

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

Reset answer

```
class Neighbor {
public:
    string majorityNeighbor(vector<string>& neighbor) {
        int size = neighbor.size();
        vector<string> diff;
        for (int i = 0; i < size; ++i)
        {
            if (i == 0) {diff.push_back(neighbor[i]); continue;}
            for (int j = i-1; j >= 0; --j)
            {
                if (neighbor[j] == neighbor[i]) break;
                else
                {
                    if (j == 0) diff.push_back(neighbor[i]);
                }
            }
        }
        vector<int> app((int)diff.size());
        for (int i = 0; i < (int)app.size(); ++i)
        {
            app[i] = 0;
        }
        for (int j = 0; j < size; ++j)
        {
            for (int h = 0; h < (int)diff.size(); ++h)
            {
                if (diff[h] == neighbor[j]) ++app[h];
            }
        }
        int max_pos = 0;
        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

✔	`vector<string> nums = { "Zach", "Zach", "Mike", "Smith", "Andrew", "Peter", "Lan", "Ilan", "Kime" , "Neitt" };`
	`Neighbor s;`
	`cout << s.majorityNeighbor(nums);`
		Zach	Zach

✔	`vector<string> nums = { "Peter","Bob","Misheo" };`
	`Neighbor s;`
	`cout << s.majorityNeighbor(nums);`
		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 ►