Midterm 2017-2018 Sem 1

Brendan Cheong Ern Jie

15/02/2022

1 Question 01

(1) O(N)

(2) O(1)

(3) O(N)

(4) O(N)

(5) O(N)

(6) O(N)

(7) O(1)

(8) O(1)

(9) O($N \log(N)$)

(10) O(N)

2 Question 02

(1) Ans: O($N \log(N)$)

[the first loop is in O(N), the second loop cuts the iteration by i elements. This means that the big O notation is O($\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \ldots + \frac{n}{n-1} + \frac{n}{n-1}$) which converges to the Harmonic series which is O($N \log(N)$)]

(2) Ans: No, counting sort or radix sort can sort in O(N)

[Quicksort best case is O($N \log(N)$) while non-comparison sorts like counting and radix sort can sort in the time complexity of O(N). Optimised bubbleSort and insertion sort can also sort in O(N) given an already sorted array.]

(3) Ans: Yes.

[A single-linked list can push(key) is O(1) time rather than O(N) time of STL vector, since vectors would have to update and increment the rest of the elements in the vector starting from the back element to the 1 index vector, if an element was pushed to the front of the vector, which would take O(N) time. Moreover, popping by removing the top of the stack in vector also takes O(N) time for the same reason. However, for SLL, it only need to update the head vector or top of the stack in O(1) time.]

(4) Ans: Yes

[A SLL is better at dequeuing in O(1) time compared to the O(N) time of STL vector. Dequeing takes up O(N) time as to remove from the head of a vector, I must update the rest of the elements index in O(N) time by iterating through all of them, starting from the last element to the i=1 element. Enqueing is the same O(1) operation for both vector and SLL]

(5) Yes.

[The smallest element is the bottommost rightmost element. A binary heap must be a complete binary tree and must fulfil the max heap property. This is because when creating a binary tree, the root element is the biggest, while the leaf elements are the smallest, with the smallest being the rightmost leaf.]

3 Question 03 Anagram

(1) HARD!!!

The best algorithm is to use Counting sort, as it takes O(N) time to validate an anagram.
1)To do this, first sort A using Counting sort in O(N).
2)Then sort B in O(N).
3)Then iterate through both A and B from i...N, N being length of either A or B. if A[i] != B[i] then its not an anagram and the Answer is "No". Else, the answer is "Yes" if A[i] = B[i] for i...N.

```
#include <bits/stdc++.h>
using namespace std;

void countSort(char arr[]) {
    // The output character array
    // that will have sorted arr
    char output[strlen(arr)];

    // Create a count array to store count of individual
    // characters and initialize count array as 0
    int count[RANGE + 1], i;
    memset(count, 0, sizeof(count));

    // Store count of each character
    for (i = 0; arr[i]; ++i)
        ++count[arr[i]];

    // Change count[i] so that count[i] now contains actual
    // position of this character in output array
    for (i = 1; i <= RANGE; ++i)
        count[i] += count[i - 1];

    // Build the output character array
    for (i = 0; arr[i]; ++i) {
        output[count[arr[i]] - 1] = arr[i];
        --count[arr[i]];
    }
```

```
    /*
    For Stable algorithm
    for (i = sizeof(arr)-1; i>=0; --i)
    {
        output[count[arr[i]]-1] = arr[i];
        --count[arr[i]];
    }

    For Logic : See implementation
    */

    // Copy the output array to arr, so that arr now
    // contains sorted characters
    for (i = 0; arr[i]; ++i)
        arr[i] = output[i];
}
int main() {
    char[] A, B;
    cin >> A >> B;
    countSort(A);
    countSort(B);

    (strcmp(A, B) == 0)
        ? cout << "Yes"
        : cout << "No";

    return 0;
}
```

4 Question 04 SLL Partition

After choosing the head element as the pivot,
1) push the head vertex into the SLL, store the value of the head as a variable
2) If the following number is less than the pivot, push the numbers to the front of the SLL, Example:, push 1 and then push 2 using InsertAtHead.
3) If more than pivot, push to the back of the SLL using InsertAfterTail.
4) Finally, print the SLL.
Algorithm should be O(N)

```
#include <bits/stdc++.h>
using namespace std;
class SLL {
    struct Vertex {
        int item;
        Vertex *next;
    };
private:
    Vertex *head, *tail;
public:
SLL() {
    head = tail = NULL;
```

```
    }
    void InsertAtHead(int value) {
            Vertex *vtx = new Vertex();
            vtx->item = value;
            vtx->next = head;
            if (head == NULL) tail = vtx; // first insertion
            head = vtx;
        }
    void InsertAfterTail(int value) {
        if (head == NULL)
            InsertAtHead(value); // special case
        else {
            Vertex *vtx = new Vertex();
            vtx->item = value;
            tail->next = vtx;
            tail = vtx;
        }
    }

    void PrintList() {
        for (Vertex *vtx = head; vtx != NULL; vtx = vtx->next)
            cout << vtx->item << " ";
        cout << endl;
    }

    SLL Partition() {
            SLL result; // do the required operations here
            Vertex pivot = *head;
            for (Vertex *vtx = head; vtx != NULL; vtx = vtx->next) {
                if (vtx->item <= pivot->item) {
                    result.InsertAtHead(vtx->item)
                } else {
                    result.InsertAtTail(vtx->item);
                }
            }
            return result;
        }
    };
int main() {
    SLL L, Lmod;
    L.InsertAfterTail(5); L.InsertAfterTail(8); L.InsertAfterTail(6);
    L.InsertAfterTail(1); L.InsertAfterTail(7); L.InsertAfterTail(2);
    Lmod = L.Partition();
    L.PrintList(); // should remain 5->8->6->1->7->2
    Lmod.PrintList(); // [smaller than 5] -> 5 -> [bigger than 5]
    return 0;
}
```

5 Question 05 Stack with FindMax()

(1) HARD!!!

Basically, to do this, you must implment 2 stacks S1, S2.
1) for each element from 2, 7... 5, push into stack s1. Do this from 1...N elements given
2) for stack s2, if the stack is empty, push in an element. For subsequent elements, before pushing to stack s2, check if s2.top() element is bigger than incoming element E. If E ¿ s2.top(), push E into s2. Else, push s2.top() into s2, like this: s2.push(s2.top()).
3) For find max, simply do s2.top().
4) When I pop() from s1, pop() from s2 as well. This means that whenever we pop from s1, we always know the next maximum element given N elements by looking at the top of s2 stack.

```
#include <bits/stdc++.h>
using namespace std;
class StackRememberMax {
private:
    stack<int> S1;
    stack<int> S2;
public:
    StackRememberMax() {
    }
    void push(int value) { // put value at the top of Stack
        S1.push(value);
        if (S2.empty())
            S2.push(value);
        else
            S2.push(max(value, S2.peek()));
    }
    int top() { // return the current top integer of Stack
        return S1.top();
    }

    int pop() { // pop the topmost integer from the stack
        S1.pop();
        S2.pop();
        return S1.top();
    }

    int findMax() { // report the max integer currently in this stack
        return S2.top();
    }
};
int main() {
    StackRememberMax S;
    cout << S.findMax() << endl; // should be -1
    S.push(2); S.push(7); S.push(1);
    S.push(6); S.push(8); S.push(5);
    cout << S.top() << endl; // should be 5
    cout << S.findMax() << endl; // should be 8
    S.pop(); // pop 5 out
    cout << S.top() << endl; // should be 8 now
    cout << S.findMax() << endl; // should still be 8
    S.pop(); // pop 8 out
    cout << S.top() << endl; // should be 6 now
```

```
        cout << S.findMax() << endl; // should now be 7
    }
```

6 Question 06 Application

To do this question, the time complexity will be O(N) time complexity.
1) first, take O(N) time to cin ¡¡ element and add each element into a std::list¡int¿ data structure using push_back(element) 2) Then, given a stream of characters in O(N) time, if char == 'R', reverse the list using reverse(list.begin(), list.end()); 3) if char == 'D', then pop_front() the list in O(1) time. 4) finally, when there are no more characters, print the list using a for loop like

```
for (auto& i : list) {
    cout << i << ' ' << endl;
}
```

in O(N) time. This will overall lead to a time complexity of O(N) time since its O(N + N + RN + D + N), where R is the number of 'R' characters in the stream of commands.