# Midterm 2017-2018 Sem 2

Brendan Cheong Ern Jie

10/02/2022

## 1 Question 01 Worst Case Time Complexity

(1) No 1

[O(1), only need to shift 1 element]

(2) No 2

[O(N), still need to shift all the elements]

(3) No 3

[O(N)]

(4) No 4

[$O(N \log(N))$]

(5) No 5 (NOTE!!!)

[$O(\log(N))$, binary search is log N]

(6) No 6

[O(N), you have to index every element thus you can't go any faster than O(N)]

(7) No 7

[O(1), double linked list can go backwards so only 3 elements]

(8) No 8

[O(1), just pop and pop from the stack]

(9) No 9

[O(N), you can only remove from the head in queue]

(10) No 10

[O(1), for deque, you can remove both head and tail, thus its O(1)]

## 2 Question 02 Analysis

(1) No 1

[False, C libraries scanf and printf are compatible with C++, C++ is just a superset of the C programming language]

(2) No 2

[False, *stable_sort* uses merge sort, Randomized quick sort is not the fastest sort as it has the worst case scenario of O($N^2$). Furthermore, Randomized quick sort is not stable]

(3) No 3

[True, each vertex in a DLL has a prev vertex and a next vertex, double that of a SLL, thus double the memory per vertex]

(4) No 4

[True, getting any element is O(1) just like the stack's ability to get the first element or rather the head. Note: if this was remove first element, vector is O(N) while stack is O(1), as vectors have to shift all the elements to the right]

(5) No 5

[True, it can compile and has more or less the same methods and operations as a vector]

# 3 Question 03 Alternative Implications

(1) No 2

[the time complexity of *push_back* for the List ADT is O(1) for this vector implementation, which is the same as the normal STL list]

(2) No 3

[Removing an element in the list takes O(N) time as we need shift the rest of elements to the right starting from the last element.]

(3) No 4

[Inserting an element in the middle and the front is O(N), as we would need to shift the index of all the elements to the right starting from the element at the back.]

(4) No 5

[Removing an element from the back is O(1)]

(5) No 6

[Inserting an element from the back is O(1)]

(6) No 7

[Sorting the array is takes an extra O(N) time because finding the specific elements of the array based on the index will require iterating from the start of the array at index 0 to the end. This makes the worst-time complexity with a comparison sort like bubble sort at $O(N^3)$ time]

# 4 Question 04 Singly/e Linked List without Tail Pointer

(1) No 2

[Removing the tail element will still be O(N) as we need to now the prev vertex by iterating through the list]

(2) No 3

[Removing the middle element is still O(N)]

(3) No 4

[Inserting at the front would be O(1) as we still know the head vertex]

(4) No 5

[Removing at the front would be O(1) as we still know the head vertex and only need to attach a new vertex to the head of the linked list]

(5) No 6

[Inserting a new tail element will take O(N) time instead of O(1), as we would have to iterate N - 1 times starting from the head vertex to the N-1 vertex to attach a new vertex, as we do not know the tail vertex]

# 5 Question 05 Special Sorting Criteria

```
#include <bits/stdc++.h> // you have to complete this question using C/C++ code
using namespace std;

int main() {
    string name;
    vector<string> listOfNames;
    while (cin >> name, !cin.eof()) { // if can read name in one line (not EOF yet)
        string reversed = reverse(name.begin(), name.end()); // O(M), M being length of string
        listOfNames.push_back(reversed); // O(N)
    }

    stable_sort(listOfNames.begin(), listOfNames.end()); // O (N log N)

    for (auto& name : listOfNames) { // O(N)
        cout << name << '\n';
    }

    return 0;
} // the overall time complexity of my C++ code above is O(MN + N log N)
```

# 6 Question 06 Continuous Median (version IV)

```
#include <bits/stdc++.h> // you have to complete this question using C++ code
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cout.ite(NULL);
    cin.tie(NULL); // N is gigantic, we need fast I/O

    int TC, N, Ai;
    cin >> TC; // always 1 test case for this version IV
    cin >> N;
    long long ans = 0;
    vector<int> arr;
```

```
    for (int i = 0; i < N; i++) {
        cin >> Ai; // this time, 1 <= Ai <= 3
        arr.push_back(Ai) // O(1)

        long long sizeOfArr = arr.size(); // O(1)

        if (sizeOfArr % 2!= 0) {
            ans += arr[sizeOfArr / 2];
        } else {
            int first = arr[(sizeOfArr / 2) - 1];
            int second = arr[sizeOfArr / 2];
            ans += (first + second) / 2;
        }
    }

    cout << ans << endl;
    return 0;
} // the overall time complexity of my C++ code above is O(N)
```