Midterm 2018-2019 Sem 1

Brendan Cheong Ern Jie

10/02/2022

1 Question 01

(1) Time Complexity
[O(1)]

(2) Time Complexity
[O(N)]

(3) Time Complexity
[O(N)]

(4) Time Complexity
[$O(N^2)$]

(5) Time Complexity
[O($N \log N$)]

(6) Time Complexity
[O(N)]

(7) Time Complexity
[O(1)]

(8) Time Complexity
[O(N)]

(9) Time Complexity
[O(N)]

(10) Time Complexity
[O(1)]

2 Question 02

True or False? Explain your answers.

(1) When writing a C++ class, it is possible to not implement/define any constructors and still be able to compile the program.

[True. It is possible to not implement any constructors as the compiler will automatically create a constructor for you for that class**]

(2) There is no sorting algorithm that is both in-place and stable. (You do not have to define in-place or stable in your responses).

[False, there is a sorting algorithm that does both of these. It is the selection sort and bubble sort]

(3) In Mergesort, the time complexity is $O \log_2 N$here is a total of $log_2 n$ layers when we split the array into 2 at each divide step. If we instead divide the array into n parts of 1 element each, we can improve Mergesort to run in $O(N \log_2 N)$ which simplifies to $O(N)$

[False. Despite dividing the array into n parts, we still need to merge the subarrays together in $O(N)$ time. This combined with the splitting of the array in $\log_2 N$ time will always lead to a time complexity of $O(N \log(N))$]

(4) Insertion Sort will never run faster than Mergesort when they are used to sort the same array.

[False. Insertion Sort can run in $O(N)$ speed if the array is already sorted. However, Mergesort will still run at a slower $O(N \log(N))$ with a sorted array]

(5) It is possible to print/output a Singly Linked List of n integers in reversed order with $O(N)$ time complexity.

[True. You can just iterate through the singly linked list in reverse by removing the tail of the linked list all the way to the head, or removing and storing the head of the linked list into a stack until the linked list is empty and then emptying the stack in $O(N)$ time by popping the stack until its empty]

3 Question 03

Create Test Cases for each scenario below. Each valid test case is worth 5 marks.

(1) Answer: HARD

[For the most efficient quicksort, we must split pivot the array into 2 parts that is equal, or rather when **the pivot is the median of the array** Ex: [4, 3, 2, 1, 5, 6, 7]]

(2) Answer: HARD

[Selection sort worse case is $O(N^2)$ especially when the array is unsorted non-increasing array, thus a test case would be : [999 ... 1], Radix sort worst case is $O(N \times \frac{k}{d})$], so a test case would be an array with each element having many digits, basically making d as large as possible: [100 ... 999]

(3) Answer: HARD

[For the stack, the half of the array reversed must be the same for the second half of the array, so its [4,3,2,1,5,1,2,3,4]]

4 Question 04

(1) What happens when K = N

[When K=N, we must include all the students into the group. We first sort the input array in $O(N \log(N))$ using STL sort after which we will then take the first and last elements of the array in O(1) time and just take last - first, as the last is the max element and the first is the minimum element. Thus, the fastest algorithm is $O(\log(N))$]

(2) What happens when K = 2

[When we face K = 2, we sort the students for each pair in $O(\log(N))$ and then calculate the minimum pair in $O(N)$ time, resulting in an overall time complexity of $O(N \log(N))$]

(3) Design an efficient algorithm to solve the general case of this problem for all values of K between 2 and N. Do include the time complexity analysis for your code.

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

int main() {

    int N,K;
    cin >> N >> K;
    int scores[N];

    for (int i = 0; i < N; ++i) {
        cin >> scores[i];
    }

    sort(scores, scores + N); // O(N log N)

    int min = scores[N - 1] - scores[0];

    for (int i = 0; i + K - 1 < N; ++i) { // O(N)
        if (scores[i + K - 1] - scores[i] < min) {
            min = scores[i + K - 1] - scores[i];
        }
    }

    cout << min;

    // total: O(N log N) + O(N) = O(N log N)

    return 0;
}
```

5 Question 05

(1) Quick Question

[It resembles a stack data structure]

(2) Bus Rides

```
Bracket Matching Algorithm
Each person is a unique pair of 'brackets'.
At the boarding station: Open Bracket
At the alighting station: Closed Bracket

#include <bits/stdc++.h> // satck, pair(in utility)
using namespace std;
typedef pair<int, int> pii;
typedef pair<int, pii> passenger;
```

```
int main() {
    stack<pii> start;
    int N, P;
    cin >> N >> P;

    vector<passenger> passengers;
    for (int i = 0; i < P; ++i) {
        int s, d;
        cin >> s >> d;
        if (s < d) {
            cout << "No";
            return 0;
        }
        passengers.push_back({i, {s, d}});
    }

    // sort(passengers.begin(), passengers.end(), [](passenger &p1, passenger &p2) { return p1.
    bool valid = true;
    for (auto p : passengers) {
        if (start.empty()) {
            start.push(p.second.first);
        } else {
            if (start.top() > p.second.second) {
                valid = false;
                break;
            } else {
                start.pop();
            }
        }
    }

    if (valid) cout << "Yes";
    else cout << "No";

    return 0;
} // the overall time complexity of my C++ code above is O(n) consider to sort as we need to si
```