Midterm CS2040C Sem 1 2021-2022

Brendan Cheong Ern Jie

08/02/2022

1 Question 01

(1) What will be printed if we run that code? (any answer with the first two most significant digits correct will be accepted)

[$N \times \log_2(2N) \times 2 \rightarrow 2048 \times 12 \times 2 = 49152$]

(2) What is the time complexity of the code above in terms of N?

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

(3) If N in line 5 is changed to 1000000 (1 Million), do you think your code can run in less than 1s if your computer can do 100 Million basic operations in 1s? Write Y/N in the box:

[N]

(4) If the comment "//" in line 8 is removed so that line 8 is part of the code, will the time complexity that you have answered in part 2 changes? Write Y/N in the box:

[N]

(5) If line 10 is changed to for (int k = 0; k < N; ++k), the time complexity of the code above in terms of N changes into O(5)

[$N^2 \log(N)$]

2 Question 02

(1) `#include <1>`

[algorithm]

(2) `for (auto 2name : names)`

[&]

(3) `std::sort(names.3(), names.end(), [](const std::string &a, const std::string &b) {}`

[begin]

(4) `if (a.length() 4 b.length())`

[! =]

(5) `return a.length() 5 b.length();`

[<]

3 Question 03

(1) Choose the Data Structure to be used

```
std::vector
```

(2) How do you implement add(v)

[using the vector method

```
push_back(v)
```

with the input being the element. This method implementation has a O(1) time complexity]

(3) How do you implement count()

[using the vector method size() with the output being the size of the vector or the number of items in the ADT. The time complexity will be O(1)]

(4) How do you implement returnAnyWithEqualProbability()

[let the index be:

```
int index = rand() % size();
```

where this will randomly choose an index within the size of the ADT. Then once the index is found, use the STL vector in built

```
vector.remove(index);
```

Finding the random index is O(1) while removing the element is O(N)]

4 Question 04

What time complexity?
[O(N)]

5 Question 05

What is the time complexity?
[O(N)]

6 Question 06

What is the time complexity?
[O($N \log(N)$)]

7 Question 07

What is the time complexity?
[O(N^2), because we have no tail, we must traverse until we find the end]

8 Question 08

explain how newSort() works and why it can sort properly (HARD)
[Now the Quicksort doesn't sort the whole array entirely, leaving some portions of it unsorted. Thus, with the addition of insertionSort, it will sort the rest of the array in O(K) times, meaning the time complexity is $O((N-K)\log(N-K)+K)$]

9 Question 09

1. What is the time complexity?

   [O(N^2), since K is large, we reach once high-low+1 <= K very early and have to resort to insertion Sort]

2. What is the time complexity?

   [O($N\log(N)$), since K is very small, we are always high-low+1 > K and we keep recursing, resulting in mostly quickSort all the way]

10 Question 10

What is the expected time complexity of newSort() in terms of K (that can range from 1 to N) and N? (HARD)

from O($N(\log(N)-\log(K)$), we only quick sort a portion of the array. Whats that portion you may ask? why thats $N-K$. Thus, we quicksort O($N(\log(N)-\log(K))$ = O($N\log(\frac{N}{K})$)). When partioning, we have to multiply by $O(N)$. Next, we insertionSort the rest of whats left, which is $length <= K$ For each element it's correct location must be in the same subarray. So each element is at most K swaps away from correct position. Thus, O(NK) for insertion sort.
Therefore answer is: O($N\log(\frac{N}{K})+NK$)

11 Question 11

(1) Fill In the Blanks

[1]

(2) Fill In the Blanks

[3]

(3) Fill In the Blanks

[9]

(4) Fill In the Blanks

[7]

(5) Fill In the Blanks

[N]

12 Question 12

```
#include <iostream>
#include <algorithm> // blanks
#include <vector>
#include <utility>
#include <unordered_map>

using namespace std;
typedef pair<int, int> pii;
typedef pair<int, pii> pip;
typedef unordered_map<int, pii> mip; // {value, {firstOccur, count}}

int main() {
    int N, K; cin >> N >> K; // 1 <= N <= 1M, 1 <= K <= N

    mip map;
    for (int i  = 0; i < N; ++i) { // O(N) here
        int v; cin >> v;
        if (map.find(v) != map.end()) {
            ++map[v].second;
        } else {
            map[v] = {i, 1};
        }
    }

    vector<pip> A;
    for (auto &e : map) {
        A.push_back({e.first, e.second});
    }

    // sort
    sort(A.begin(), A.end(), [](pip &a, pip &b){ // {value, {firstOccur, count}}
        if (a.second.second != b.second.second) {
            return a.second.second > b.second.second;
        } else {
            return a.second.first < b.second.first;
        }
    });

    for (auto a : A) {
        for (auto i = 0; i < a.second.second; ++i)
            cout << a.first << " ";
    }
    cout << endl;
    return 0;
}
```

[the overall time complexity of my C++ code above is $O(N + N2\log(N2))$, where N2is the number of unique numbers in N inputs]