Middle East Technical University | Department of Computer Engineering

CENG 315

Algorithms

Fall 2024-2025

Take-Home Exam 1

Due date: 27 October 2024, Sunday, 23.59

1 Problem Definition

Momo is a little menace who loves to steal people's rummy (okey or 101) tiles when they are not looking at the game table. Over the years, he accumulated an immense amount of tiles. He wants to display them proudly and asked for your help to sort them in increasing order. Fortunately, he took note of the number of each tile he stole in a digital file. He sent the file to you so that you can work on the sorting task digitally.

But Momo decided to create a game out of these tiles just to annoy you. He gives you a set of tiles and a number `k`. He wants you to tell him the number of the `kth` smallest tile.

You want to use QuickSort algorithm to sort the tiles and select the `kth` smallest tile. However, since the tile numbers are repetitive, you want to update the partition approach of QuickSort to include a third partition with the elements equal to the pivot. After some thinking, you decided to use the following algorithm for partitioning, where `A` is the array of tiles (where indices start from 0), `size` is the length of A, and `swap` is the count of swaps so far:

```
Partition(A,size,swap)
    i <- 1
    j <- 1
    k <- size-1
    pivot <- A[0]
    while(j <= k)
        if A[j] = pivot
            j <- j+1
        else if A[j] < pivot
            swap(A[i],A[j])
            i <- i+1
            j <- j+1
        else if A[j] > pivot
            swap(A[j],A[k])
            k <- k-1
        end
    end
    swap(A[i-1],A[0])
    return <i-1,j>   //left and right pivotal points for recursive calls
```

1.1 select_k_with_quick_sort3

As the first task of this exam, you are asked to complete `select_k_with_quick_sort3` function definition to sort the array `arr` of size `size` in ascending order. Your function should also count the number of `swap`s executed during this sorting process. You should return a `pair`, where the first element is the `kth` smallest tile for `k=index)`, and the second element is the number of recursive calls.

```
std::pair<unsigned short, unsigned int>
select_k_with_quick_sort3(unsigned short * arr,
                          unsigned int size,
                          unsigned int index,
                          long &swap);
```

1.2 quick_select3

Momo is not satisfied with your implementation and asked you to do it faster. To achieve a shorter selection time for the `kth` element, you decided to sort only the parts needed to select it, not the whole list. For instance, if you know that the elements on one side of the partition belong to that side, and their inner ordering does not affect the `kth` element, you should **not** make a recursive call to sort that side. Keep in mind that on average, QuickSelect algorithm aims to achieve $O(n)$ complexity.

For the second task of this exam, you are asked to complete `quick_select3` function definition. Unlike `select_k_with_quick_sort3`, this time, you are not expected to sort the whole list, but stop when you find the `kth` smallest tile for `k=index`. The other arguments and inputs are identical to the previous task.

```
std::pair<unsigned short, unsigned int>
quick_select3(unsigned short *arr,
              unsigned int size,
              unsigned int index,
              long &swap);
```

2 Example Run

Example 1

```
Size: 6
Using: select_k_with_quick_sort3
Array elements: {3, 3, 12, 10, 3, 3}
Element at index 5 when sorted: 12
Swap: 5
Number of recursive calls: 5
```

Example 2

```
Size: 10
Using: quick_select3
Array elements: {12, 12, 10, 10, 1, 12, 1, 5, 10, 8}
Element at index 7 when sorted: 12
Swap: 8
Number of recursive calls: 1
```

3 Constraints and Limits

In this exam, the complexity of your implementations will be checked with your reporting of swap count and recursive call count. Hence the system limitations are not strict, and are as follows:

- a maximum execution time of 2 minutes
- a 4 MB maximum memory limit
- a stack size of 128 MB for function calls (ie. recursive solutions)

There are some important points to keep in mind:

- You should keep track of the count of `swap` operations complying with the pseudocode. If your implementation differs in a way, please make sure you keep track of the swaps correctly. This means that even if you think a swap operation is unnecessary (for instance, the arguments of swap refer to the same element), you should still count it as a swap operation.
- You are expected to keep track of the count of recursive function calls. You may need helper functions to keep track of this count.
- Count of swaps and recursive calls will show whether or not your implementation obeys the expected complexity requirements, so please pay extra attention to them.
- Even though rummy tiles are numbered from 1 to 13, you are expected to handle an unsigned short array. Make sure your implementation does not depend on this assumption.
- Maximum `size` for the array `arr` is 2^{16}.

4 Specifications

- You will implement your solutions in the `the1.cpp` file.
- Do not change the first line of `the1.cpp`, which is `#include "the1.h"`.
- Do not change the arguments and the return value of the given functions in the file `the1.cpp`, but you are free to add other functions to `the1.cpp`.
- Do not include any other library or write include anywhere in your `the1.cpp` file (not even in comments).
- You are given a `test.cpp` file to test your work on ODTUCLASS or your locale. You can and you are encouraged to modify this file to add different test cases.
- You can test your `the1.cpp` on the virtual lab environment. If you click run, your function will be compiled and executed with `test.cpp`. If you click evaluate, you will get feedback for your current work and your work will be temporarily graded with a limited number of inputs.
- The grade you see in VPL is not your final grade, your code will be reevaluated with more inputs after the exam.
- If you want to test your work and see your outputs on your locale you can use the following commands:

```
> g++ test.cpp the1.cpp -Wall -std=c++11 -o test
> ./test
```

The `test.cpp` file has a random input generation function so that you can try your code with random input arrays. When you **Run** your code in the VPL, this random test function will run. You can set the `random_fill` variable as `false` in `test.cpp` file to disable random tests.

In the random input generation function, `quick_select` variable controls whether to use `quick_select3` or `select_k_with_quick_sort3`. The variables `minimum_value` and `interval` control the numbers to be randomly sampled from. For example, with a `minimum_value` of 1 and `interval` of 13, a random list of rummy tiles can be generated.

5 Regulations

- **Implementation and Submission:** The template files are available in the Virtual Programming Lab (VPL) activity called "THE0" on ODTUCLASS. At this point, you have two options:
  - You can download the template files, complete the implementation, and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
  - You can directly use the editor of the VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submitting a file.

  Please make sure that your code runs on ODTUCLASS. There is no limitation in running your code online. The last save/submission will determine your final grade.

- **Programming Language:** You must code your program in C++. Your submission will be tested on the VPL environment in ODTUCLASS, hence you are expected to make sure your code runs successfully there.

- **Cheating: This assignment is designed to be worked on individually.** Additionally, the use of any LLMs (chatgpt, copilot, the other one that you are thinking about...) and copying code directly from the internet for implementations is strictly forbidden. Your work will be evaluated for cheating, and disciplinary action may be taken if necessary.

- **Evaluation:** Your program will be evaluated automatically using "black-box" testing, so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to worry about invalid cases. **Important Note:** The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your final grade after the deadline.