FALL 2023 CS431102 Algorithms
Bonus programming Assignment 1 Selection Algorithm
Student: Victor D. Lopez
ID: 110062426
Professor: JANG PING SHEU

1. How to execute **(Note input file name must be input.txt)**

In the directory where 110062426_bonus1.cpp is located, please, run the following command to compile:

$ g++ -Wall -std=c++11 -o bonus 110062426_bonus1.cpp

In addition, you must have a file name **input.txt** with a single line consisting of the input parameters N,K,G in that order.

Then, simple execute *./bonus* (or whatever you name the executable) in the current directory. This will output the kth smallest element in **output.txt**. Note the smallest element is counted from 1 and not from 0 in this result, thus our result corresponds to the element with index K-1 in the sorted array.

2. Average execution time (SELECT ALGORITHM ONLY)

We considered an input size of N = 10,000,000 and varied the values of K among 50 experiments.

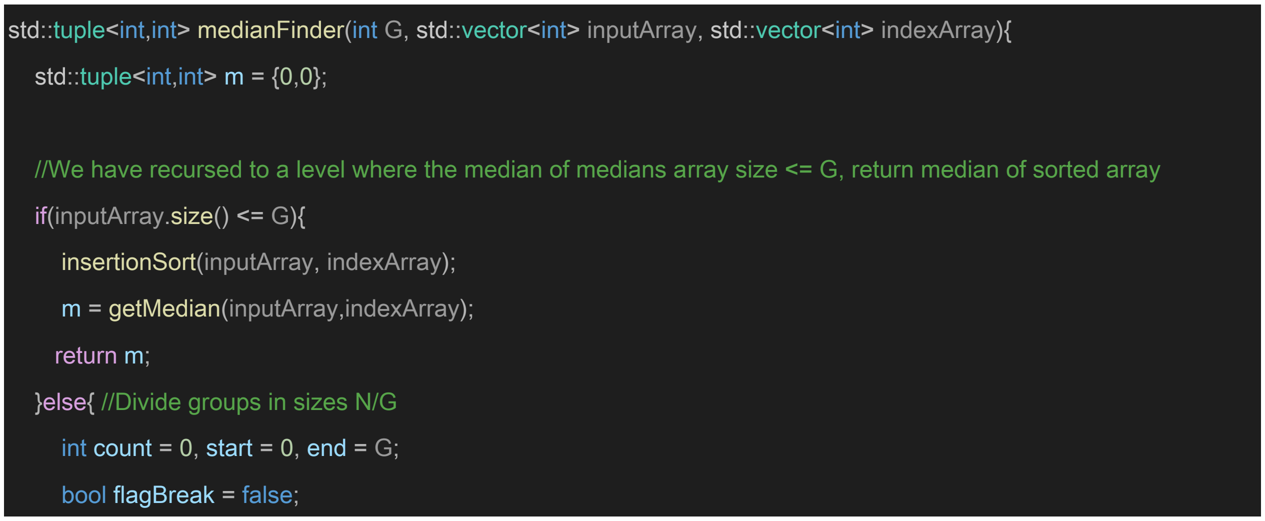
K	Average execution time in seconds (50 tries)
3	4.62
5	3.92
7	3.72
9	3.75

As expected, the algorithm slows down and is no longer linear on the input Θ(N) when we consider groups of size G = 3;

3. Differences with slides/book and some important pseudocode:

The algorithm implemented is the same one implemented in the book (4th edition), however, to find the median of medians, we don't use the SELECT algorithm, instead we created the *medianFinder()* algorithm which is also recursive but clearer than the one shown in the book.

The pseudocode is shown below:



```
std::tuple<int,int> medianFinder(int G, std::vector<int> inputArray, std::vector<int> indexArray){
  std::tuple<int,int> m = {0,0};

  //We have recursed to a level where the median of medians array size <= G, return median of sorted array
  if(inputArray.size() <= G){
    insertionSort(inputArray, indexArray);
    m = getMedian(inputArray,indexArray);
   return m;
  }else{ //Divide groups in sizes N/G
    int count = 0, start = 0, end = G;
    bool flagBreak = false;
```

```cpp
        std::vector<int> temp, tempIdx, medians, medianIdx;

        while(true){
            for(int i = start; i < end; i++){
                temp.emplace_back(inputArray[i]);
                tempIdx.emplace_back(indexArray[i]);
            }

            //Sort each array group
            insertionSort(temp, tempIdx);

            //Get the median and its index and add it to the arrays
            m = getMedian(temp, tempIdx);
            medianIdx.emplace_back(std::get<0>(m));
            medians.emplace_back(std::get<1>(m));

            //Update indexes for next group
            count+=1;
            start = G*count;
            end = start + G;
            temp.clear();
            tempIdx.clear();

            if(end >= inputArray.size()){
                end = inputArray.size();
                if(!flagBreak){
                    flagBreak = true;
                }else{
                    break;
                }
            }
        }
        //Recurse to find median of medians
        m = medianFinder(G,medians, medianIdx);
    }
    return m;
}
```

This is the main ingredient of the SELECT algorithm. We designed it this way to easily debug and verify our algorithm behaved correctly in the intermediate steps of the SELECT algorithm, for instance, when partitioning. Keeping track of the index and value of the median was possible thanks to the *tuple* data structure.

Other than this, the programming was straightforward for insertionSort, partition and SELECT.

4. References

[1] Chapter 9 Slides for FALL 2023

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Introduction to Algorithms, 4th Edition.