# Analysis of Algorithms

## BLG 335E

# Project 1

Res. Assist. Meral Kuyucu
korkmazmer@itu.edu.tr

Instructors:
Hazım Kemal Ekenel
İlkay Öksüz

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 21.11.2023

# 1. Implementation

## 1.1. Naive Implementation of Quicksort

To warm up, you are expected to implement the basic QuickSort sorting algorithm without any optimization. The goal of this part is to ensure that you have a solid foundation of the core principles of the QuickSort algorithm. Your code should:

1. Choose a pivot element from the array (for this part, use the last element as your pivot).
2. Partition the array into two subarrays around the pivot. Elements on the left hand side subarray should be smaller in value than the elements on the right hand side subarray. Your algorithm is only expected to sort in **ascending** order.
3. Recursively apply QuickSort on the subarrays until you reach the base case.

## 1.2. Implementation of Quicksort Using Different Pivoting Strategies

Now it is time to expand your code. Modify your implementation of QuickSort from the previous part to work with the following three pivoting strategies:

- **Last Element:** This is the pivoting strategy implemented in the previous part of the assignment. The last element of the array should be taken as the pivot value.
- **Random Element:** Choose one element of the array randomly and pivot around it. You can refer to this link if you need help generating a random number.
- **Median of 3:** Choose three elements at random and select the median of these elements as the pivot.

## 1.3. Hybrid Implementation of Quicksort and Insertion Sort

Let's take your QuickSort implementation to the next level. In recitation, you saw that insertion sort may perform better than divide and conquer approaches (such as MergeSort or QuickSort) for nearly sorted arrays of smaller size.

In this part, you are asked to develop a hybrid method. This method should run based on a threshold **"k"**. For input arrays which are larger than **"k"**, QuickSort should run as usual with the selected pivoting strategy. When the input size is smaller than or equal to the threshold **"k"**, the input array should be sorted with **InsertionSort**.

# 2. DataSet

For this assignment you are given a simplified version of the DataSet given in this link. The original DataSet contains 11 columns, however, the simplified DataSet contains the following columns:

- **City [String]**
- **Population [Integer]**

Be informed that multiple permutations of the same simplified DataSet titled **“population1.csv”**, **“population2.csv”**, **“population3.csv”**, and **“population4.csv”** are provided to test your code and report your results.

The report template provided has some guidelines for you to test your code out. However, you are highly encouraged to test your code on smaller portions of these DataSets and incrementally increase the size of your data and discuss your results.

# 3. Deliverables

## 3.1. Code [70%]

Submit a code file titled **“QuickSort.cpp”**. This file should run on DataSets in the format of the DataSet files provided in the assignment description. You are free to delegate the tasks of the assignment into functions as you choose, however, bear in mind that you will be graded for **both** efficiency and readability. Your code should accept the following command line arguments:

1. The name of the DataSet file.
2. The pivoting strategy implemented
    - ’l’: Last Element
    - ’r’: Random Element
    - ’m’: Median of Three
3. The value of the threshold **“k”**. If the value of **“k”** is 1, naive QuickSort should run. Otherwise, the hybrid approach with InsertionSort should run. Assume that **“k”** will always take value greater than or equal to 1.
4. The name of the output file. The sorted output should be written to a csv file with the same format as the input. Please refer to the example output for **“population1.csv”** given in **“out.csv”**.
5. Whether or not a verbose output is provided. The last command line argument can be **“v”** for verbose. It may or may not be provided by the user. In the case which it isn’t provided, do not print verbose output. In the case that this argument is provided, a **“log.txt”** file should be generated printing out either the partition details for each call to partition in QuickSort. You do not need to print anything for InsertionSort.

The shell command to compile this file is:

```
g++ QuickSort.cpp -o QuickSort
```

An example shell command to run the compiled executable is:

```
./QuickSort [DataSetFileName].csv r 5 out.csv v
```

This particular command line prompt should run QuickSort using a random pivoting strategy. When the sizes of the subarrays are equal to 5, it should run to termination with insertion sort. The output should be written to a file titled **“out.csv”**. For example, given dummy data **[2 1 6 9 7 4 3 8 5]**, verbose output should be written to the **“log.txt”** file in the following format:

```
Pivot: 5 Array: [2, 1, 6, 4, 5, 9, 3, 7, 8]
...
```

Your code should print to the command line the amount of time it took to sort the input array in the following format:

```
Time taken by QuickSort with pivot strategy 'r'
and threshold 5: [Insert Elapsed Time] ns.
```

If you are not sure how to measure time elapsed, please refer to this link.

## 3.2. Report [30%]

Upload the provided report template to Overleaf. Fill in your report and submit only the pdf file.

## 3.3. Guidelines and Grading

Your homework will be graded under the following criteria:

- Code: Part 1: 15%, Part 2: 15%, Part 3: 25%, Implementation: 15%
- Report: 30 %

Please note that a **50% overall penalty** will be imposed on code that fails to compile and run within the provided **docker** environment. Additionally, the grade for the report will be capped at the number of points achieved in the code implementation part.

You will be graded on the following:

- Your efficient C++ **Implementation** of the assignment requirements. [70 pts]
  - Correctness
  - Clarity
  - Good Coding Practice
- **Report** detailing your implementation. [30 pts]
  - Content
  - Organization
  - Relevance
  - Quality

**IMPORTANT NOTE:**

- Please be informed that your code and report will be checked for plagiarism using plagiarism detection tools and disciplinary action will be taken upon detection of plagiarism. Please write your code and report on your own.

- Copying code fragments from any source including books, websites, or classmates is considered plagiarism.
- You are free to conduct research on the topics of the assignment to gain a better understanding of concepts at an algorithmic level. Make sure to cite any such sources that you refer to.
- Refrain from posting your code/report on any public platform (i.e. Github) before the deadline of the assignment.
- You are free to use STL for data structures ONLY (i.e. you can use vector, but not built-in sorting algorithms).