CSCI 3320: Data Structures
HW 5: Finding a Number Pair
Due: 12/15 (Fri) 11:59 pm (10 points)

1. Project Details

In this project, you will implement three algorithms using diverse data structures and algorithms we have discussed during the semester. Your program will generate random numbers and check if there are two numbers as a pair of which sum equals a given number. For instance, if the random numbers are 19, 21, 2, 5, and 49 and the given number is 60, your program finds and prints the pair (21, 49). If there is no pair found, e.g., the given number is 65, your program prints "No pair found".

To find the pair, you will implement 3 different algorithms for this program as follows:

1. O (N^2) algorithm to find the pair
   - Hint: **Nested loops**

2. O (N logN) algorithm to find the pair.
   - Hint: **Sorting algorithms**

3. O (N) algorithm to find the pair.
   - Hint: **Hash table**

Your program will take three parameters.

0. `N`: The first parameter indicates how many numbers are randomly generated. The range of random numbers is 0 ~ 999,999.
1. `K`: The second parameter is the sum of two numbers your program is looking for.
2. `Algorithm`: The last parameter indicates which algorithm will be used for finding a pair.
   - 0: O (N^2) algorithm
   - 1: O (N logN) algorithm
   - 2: O (N) algorithm

Your program **must** take these parameters from **your command line arguments** as follows:

- Python3 findPair.py 10000 1000 0 or Java findPair 10000 1000 0
  - Your program generates 10000 random numbers and searches a pair (two numbers) whose sum is 1000 using the O (N^2) algorithm.
- Python3 findPair.py 50000 2000 2 or Java findPair 50000 2000 2
  - Your program generates 50000 random numbers and searches a pair (two numbers) whose sum is 2000 using the O (N) algorithm.

2. Project Details

You will make a report for performance comparison among your algorithms using a graph and detailed explanations of your ideas and analyze the complexity of each algorithm.

To compare your algorithms' performance, **you will measure elapsed time for each algorithm execution** using the Python function `time.time()` or java function `System.nanoTime()` as done in the previous HW. You need to measure execution time for **your algorithms only**, i.e., execution time for generating input data must not be included in elapsed time. You must run at least 10 times and plot the averaged **worst execution time** for each algorithm with various input sizes, 10000, 20000, 40000, 80000, and 160000. You may set the `K` value to 0 (or less than 0) to measure the worst-case execution time for each algorithm. In your graph, you must show the input sizes, execution time, and each algorithm. You need to use tools, e.g., Microsoft Excel, to draw plots.

You can utilize any built-in data structures to complete this project. **You can reuse code you implemented from the previous HWs for generating numbers and measuring execution time.** A skeleton code file (**findPair.java and findPair.java**) is provided. Please add comments in your code for the instructor to understand your code. You can discuss this project with your classmates to get some hints. However, **it is not allowed to 1) share the solution with your classmates, 2) post this project to websites e.g., chegg.com to get solutions, 3) just copy and paste the code you found on the Internet, or other references (classmates).** In short, **you must complete the project by yourself**. The exams may ask about your project implementation details.

3. Deliverables

Please submit a zipped file (your code and report) on Canvas by the due date.

For full credit, your code must be well documented with comments. Your programs must contain enough comments. Programs without comments or with insufficient and/or vague comments will cost you 30%. Inline comments should be utilized as necessary (but not overused) to make algorithms clear to the reader. **Put your information in the code.**

4. Grading

The following factors will be considered while grading your submission:

Code Correctness: 60%

Report (performance measurement, idea explanation, and algorithm complexity): 30%

Coding and style (indentations, readability of code, comments in code) and error handling: 10%