CS401 Lab 3: Runtime Complexity and Selection Sort

Overview

- This lab is to be completed individually.
- Focus: Understanding runtime complexity and implementing Selection Sort.
- Objectives:
  1. Analyze the runtime complexity of given methods.
  2. Implement and analyze the Selection Sort algorithm.

Part 1: Runtime Complexity Analysis

Task

Analyze and determine the Big-O complexity for *methodA*, *methodB*, and *methodC* in the provided *ComplexityExperiment* class.

Requirements

1. Do not run the code. Analyze it visually.
2. For each method (*methodA*, *methodB*, *methodC*):
   - Determine the Big-O complexity
   - Explain your reasoning
3. Document your analysis in a word or PDF file named "Complexity.pdf"

Provided Code

```
import java.util.Date;
import java.util.Timer;
public class ComplexityExperiment {
	public static void main (String args []) throws InterruptedException {
		run_method_A(250);
		run_method_A(500);
		run_method_A(1000);
		run_method_A(2000);
		System.out.println();
		run_method_B(250);
		run_method_B(500);
		run_method_B(1000);
		run_method_B(2000);
		System.out.println();
		run_method_C(250);
		run_method_C(500);
		run_method_C(1000);
		run_method_C(2000);
		System.out.println();
	}
	public static void run_method_A(int n) { // n must be bigger than 0 always
		int i = 0;
```

```
	double start, end;
	start = System.currentTimeMillis();
	methodA(n);
	end = System.currentTimeMillis() - start;
	System.out.println("methodA(n = " + n + ") time = " + end + "ms");
}
public static void run_method_B(int n) { // n must be bigger than 0 always
	int i = 0, loop = 1000;
	double start, end;
	start = System.currentTimeMillis();
	for (i = 0; i < loop; i++) {
		methodB(n);
	}
	end = System.currentTimeMillis() - start;
	System.out.println("methodB(n = " + n + ") time = " + end/loop + "ms");
	}
public static void run_method_C(int n ) { // n must be bigger than 0 always
	int i = 0;
	double start, end;
	start = System.currentTimeMillis();
	methodC(n);
	end = System.currentTimeMillis() - start;
	System.out.println("methodC(n = " + n + ") time = " + end + "ms");
}
public static void methodA(int n) {
	int i = 0;
	int j = 0;
	int k = 0;
	int total = 0;
	while (i < n) {
		while (j < n) {
			while (k < n) {
				total++;
				k++;
			}
			k = 0;
			j++;
		}
		j = 0;
		i++;
	}
}
public static void methodB(int n) {
	int i = 0;
	int j = 0;
	int total = 0;
```

```
            while (i < n) {
                while (j < n) {
                    total++;
                    j++;
                }
                i++;
            }
        }
        public static void methodC(int n) {
            int i = 0;
            int j = 0;
            int total = 0;
            j = n;
            while ((j = j / 2) > 0) {
                for (i = 0; i < 100 * n; i++) {
                    total++;
                }
            }
        }
}
```

Part 2: Selection Sort Implementation

Task
Implement the Selection Sort algorithm based on the description in Chapter 1 of your textbook.

Requirements
1. Create a *SelectionSort* class that:
   - Takes an array of integers as input
   - Implements the Selection Sort algorithm
   - Returns the sorted array
2. In the main method of *SelectionSort*:
   - Generate test data (consider using 10,000+ random values for accurate timing)
   - Call your sorting method
   - Measure and display the execution time
3. Analyze and document the Big-O complexity of your implementation
4. If execution time is "0 ms", use nanoseconds or larger datasets
5. What type of data? Any type of data: integer values, float values, or strings (up to your own convenience).

Code Documentation
- Include inline comments for all methods and complex code sections
- Provide a README file with:
  - Description of the program
  - Instructions on how to compile and run the program
  - Explanation of the Selection Sort implementation

- Command to run the JAR file

Submission Requirements

1. Complexity.pdf:
   - Part 1 analysis
   - Big-O complexity of your Selection Sort implementation
2. SelectionSort.java:
   - Source code of your Selection Sort implementation
   - Test code in the main method
3. Output.pdf:
   - Program outputs and/or screenshots
4. SelectionSort.jar:
   - Executable JAR file of your program
5. README.md:
   - Program description
   - Compilation and execution instructions
   - JAR file execution command

Important Notes

- Use only the textbook's algorithm description for Selection Sort
- Ensure thorough testing with various input sizes