CS401 Lab 3: Runtime Complexity and Selection Sort

Complexity

Seonghwan Lim

A20586593

Part 1

methodA

```
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); // Runs 1 time
        end = System.currentTimeMillis() - start;
        System.out.println("methodA(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) { // Loop 1: Runs n times
                while (j < n) { // Loop 2: For each iteration of the Loop 1, runs n times
                        while (k < n) { // Loop 3: For each iteration of the while Loop 2, runs n times
                                total++;
                                k++;
                        }
                        k = 0;
                        j++;
                }
                j = 0;
                i++;
        }
```

- methodA(n) is a bounded time operation $O(1)$.
- Loop 1 runs n times.
- Loop 2 runs $n * n = n^2$ times.
- Loop 3 runs $n * n * n = n^3$ times.
- methodA Big-O complexity is $O(N^3)$.

methodB

```
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++) { // Runs 1000 times
                methodB(n);
        }
        end = System.currentTimeMillis() - start;
        System.out.println("methodB(n = " + n + ") time = " + end/loop + "ms");
        }

public static void methodB(int n) {
        int i = 0;
        int j = 0;
        int total = 0;
        while (i < n) { // Loop 1: Runs n times
        while (j < n) { // Loop 2: For each iteration of the Loop 1, Runs n times
            total++;
            j++;
        }
        i++;
    }
```

- methodB(n) is a bounded time operation $O(1)$.
- Loop 1 runs n times.
- Loop 2 runs $n * n = n^2$ times.
- methodB Big-O complexity is $O(N^2)$.

methodC

```
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); // Runs 1 times
    end = System.currentTimeMillis() - start;
    System.out.println("methodC(n = " + n + ") time = " + end + "ms");
}

public static void methodC(int n) {
    int i = 0;
    int j = 0;
    int total = 0;
    j = n;
    while ((j = j / 2) > 0) { // Loop 1: Runs log2(n) times
        for (i = 0; i < 100 * n; i++) { // Loop 2: For each iteration of the Loop 1, runs 100 * n times
            total++;
        }
    }
}
```

- methodC(n) is a bounded time operation $O(1)$.
- Loop 1 runs $\log_2(n)$ times.
- Loop 2 runs $n * \log_2(n) = n \log_2(n)$ times.
- methodC Big-O complexity is $O(N \log_2(N))$.

Comparison of Methods

Method	methodA	methodB	methodC
Big-O	$O(N^3)$	$O(N^2)$	$O(N \log_2(N))$.
N = 250	15,625,000	62,500	1,991
N = 500	125,000,000	250,000	4,482
N = 1000	1,000,000,000	1,000,000	9,965
N = 2000	8,000,000,000	4,000,000	21,931

Big-O complexity of my Selection Sort implementation

```
// Method sorts the given array using the selection sort algorithm.
    public static int[] selectionSort(int[] arr) {
        int n = arr.length;

        // Loop through the array to find the minimum element in each iteration.
        for (int i = 0; i < n; i++) {
            int min = i;  // Assume the current index holds the smallest value.

            // Inner loop to find the smallest value in the remaining unsorted portion.
            for (int j = i + 1; j < n; j++) {
                if(arr[j] < arr[min]) {  // If a smaller value is found, update the minimum index.
                    min = j;
                }
            }
            // Swap the smallest found value with the value at the current index.
            int temp = arr[min];
            arr[min] = arr[i];
            arr[i] = temp;
        }
        return arr;  // Return the sorted array.
    }
```

- The outer loop runs n times.
- The inner loop runs from i+1 to n.

$$\sum_{i=0}^{n-1}(n-i-1)=\frac{n(n-1)}{2}$$

- Big-O complexity: $O(N^2)$