CSE 143X: Accelerated Computer Programming I/II

Efficiency

What does it mean to have an “efficient program”?

```
1 System.out.println("hello");
vs.
1 System.out.print("h");
2 System.out.print("e");
3 System.out.print("l");
4 System.out.print("l");
5 System.out.println("o");
```

```
OUTPUT
>> left average run time is 1000 ns.
>> right average run time is 5000 ns.
```

We're measuring in NANOSECONDS!

Both of these run very very quickly. The first is definitely better style, but it's not "more efficient."

Comparing Programs: Timing

```
public boolean hasDuplicate1(int[] array) {
    for (int i=0; i < array.length; i++) {
        for (int j=0; j < array.length; j++) {
            if (i != j && array[i] == array[j]) {
                return true;
            }
        }
    }
    return false;
}
```

```
public boolean hasDuplicate2(int[] array) {
    for (int i=0; i < array.length - 1; i++) {
        if (array[i] == array[i+1]) {
            return true;
        }
    }
    return false;
}
```

```
OUTPUT
>> hasDuplicate1 average run time is 5254712 ns.
>> hasDuplicate2 average run time is 2384 ns.
```

Comparing Programs: # Of Steps

```
public int stepsHasDuplicate1(int[] array) {
    int steps = 0;
    for (int i=0; i < array.length; i++) {
        for (int j=0; j < array.length; j++) {
            steps++; // The if statement is a step
            if (i != j && array[i] == array[j]) {
                return steps;
            }
        }
    }
    return steps;
}
```

```
OUTPUT
>> hasDuplicate1 average number of steps is 9758172 steps.
>> hasDuplicate2 average number of steps is 170 steps.
```

Comparing Programs: Plotting

This still isn’t good enough! We’re only trying a single array!

Instead, let's try running on arrays of size 1, 2, 3, ..., 1000000, and plot:
Comparing Programs: Analytically

Runtime Efficiency
We've made the following observations:
- All "simple" statements (println("hello"), 3 + 7, etc.) take one step to run.
- We should look at the "number of steps" a program takes to run.
- We should compare the growth of the runtime (not just one input).

```java
1 statement1;
2 statement2;
3 statement3;
4 5 for (int i = 0; i < N; i++) {
5 6 7 } statement4;
6 8 9 10 for (int i = 0; i < N; i++) {
11 12 13 statement5;
14 15 statement6;
16 17 statement7;
18 19 statement8;
20 }
```

More Examples

```java
1 statement1;
2 statement2;
3 statement3;
4 5 for (int i = 0; i < N; i++) {
5 6 7 } statement4;
6 7 for (int j=0; j < N/2; j++) {
8 9 } statement5;
10 11 12 13 for (int i = 0; i < N; i++) {
14 15 16 statement6;
17 18 19 statement7;
20 21 statement8;
22 23 statement9; } 0.5N^2 + 5N + 3
```

So, the entire thing is $O(N^2)$, because the quadratic term overtakes all the others.

ArrayList Efficiency

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(val)</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>add(idx, val)</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>get(idx)</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>set(idx, val)</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>remove(idx)</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>size()</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Investigating Our Answer With Pictures

What are the time complexities of these functions?

```java
1 public static void numbers1(int max) {
2 3 4 for (int i = 1; i < max; i++) {
5 6 list.add(i); //O(1)
7 list.add(i); //O(1)
8 }
9 }
10 }
```

```java
1 public static void numbers2(int max) {
2 3 4 for (int i = 1; i < max; i++) {
5 6 list.add(i); //O(1)
7 list.add(i); //O(1)
8 }
9 }
```

Big-Oh

We measure algorithmic complexity by looking at the growth rate of the steps vs. the size of the input.

The algorithm on the previous slide ran in $5N + 3$ steps. As $N$ gets very large, the "5" and the "3" become irrelevant.

We say that algorithm is $O(N)$ ("Big-Oh-of-N") which means the number of steps it takes is linear in the input.

Some Common Complexities

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>Constant</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>Linear</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>Quadratic</td>
</tr>
<tr>
<td>$O(2^n)$</td>
<td>Exponential</td>
</tr>
</tbody>
</table>

The number of steps doesn’t depend on $n$.

If you double $n$, the number of steps doubles.

If you double $n$, the number of steps quadruples.

The number of steps gets infeasible at $n < 100$.
Find the Runtime

1 public boolean is10(int number) {
2     return number == 10;
3 }
4
5 public boolean two10s(int num1, int num2, int num3) {
6     return (is10(num1) && is10(num2) && !is10(num3)) ||
7          (is10(num1) && !is10(num2) && is10(num3)) ||
8          (!is10(num1) && is10(num2) && is10(num3));
9 }
10
11 public void loops(int N) {
12     for (int i = 0; i < N; i++) {
13         for (int j = 0; j < N; j++) {
14             System.out.println(i + " " + j);
15         }
16     }
17     for (int i = 0; i < N; i++) {
18         System.out.println(N - i);
19     }
20 }
21
22 O(n^2)
23 O(n)
24 O(1)
25
It's the WORST CASE!

1 public static int has5(int[] array) {
2     for (int i = 0; i < array.length; i++) {
3         if (array[i] == 5) {
4             return true;
5         }
6     }
7     return false;
8 }
9
10 O(n)
11 O(1)
12
Sometimes, these will finish in fewer than array.length steps, but in the worse case, we have to go through the whole array. This makes both of them $O(n)$. 