CS 61B Spring 2021

Asymptotics

Topical Review Session 4: February 28, 2021

Here is a review of some formulas that you will find useful when doing asymptotic analysis.

- $\sum_{i=1}^{N} i = 1 + 2 + 3 + 4 + \cdots + N = \frac{N(N+1)}{2} = \mathbf{\frac{N^2+N}{2}}$
- $\sum_{i=0}^{N-1} 2^i = 1 + 2 + 4 + 8 + \cdots + 2^{N-1} = 2 \cdot 2^{N-1} - 1 = \mathbf{2^N - 1}$

1 Dumpling Time!

For each problem below, give the tighest possible O runtime of the code snippet

(a)
```java
public void wrapWonton(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 1; j < n; j*=2) {
            System.out.println("Wrapping");
        }
        System.out.println("Wonton Wrapped!");
    }
}
```

(b)
```java
public void wrapDumpling(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            System.out.println("Wrapping");
        }
        System.out.println("Dumpling Wrapped!");
    }
}
```

(c)
```java
public void wrapBigDumpling(int n) {
    wrapDumpling(n);
    wrapBigDumpling(n/2);
}
```

(d)
```java
public void letsEat(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i; i < n; i++) {
            System.out.println("Eating");
        }
    }
    System.out.println("Done eating!");
}
```

2 I am Speed

For each example below, there are two algorithms solving the same problem. Given the asymptotic runtimes for each, is one of the algorithms **guaranteed** to be faster? If so, which? And if neither is always faster, explain why.

(a) Algorithm 1: $\Theta(N)$, Algorithm 2: $\Theta(N^2)$

(b) Algorithm 1: $\Omega(N)$, Algorithm 2: $\Omega(N^2)$

(c) Algorithm 1: $O(N)$, Algorithm 2: $O(N^2)$

(d) Algorithm 1: $\Theta(N^2)$, Algorithm 2: $O(\log N)$

(e) Algorithm 1: $O(N \log N)$, Algorithm 2: $\Omega(N \log N)$

3 Getting A Little Loopy

Give the runtime for each method in $\Theta(\cdot)$ notation in terms of the inputs. You may assume that `System.out.println` is a constant time operation.

(a) *Hint:* We cannot multiply over the two iterations of the for loop to find the runtime. *Why?*

```
public static void liftHill(int N) {
    for (int i = 1; i < N * N; i *= 2) {
        for (int j = 0; j <= i; j++) {
            System.out.println("-_-");
        }
    }
}
```

(b) Assume that `Math.pow` $\in \Theta(1)$ and returns an int.

```
public static void doubleDip(int N) {
    for (int i = 0; i < N; i += 1) {
        int numJ = Math.pow(2, i + 1) - 1;
        for (int j = 0; j <= numJ; j += 1) {
            System.out.println("AHHHH");
        }
    }
}
```

(c) *Hint:* When do we return "WHOA"?

```
public static String corkscrew(int N) {
    for (int i = 0; i <= N; i += 1) {
        for (int j = 1; j <= N; j *= 2) {
            if (j >= N/2) {
                return "WHOA";
            }
        }
    }
}
```

(d) *Hint:* Draw the recursive tree!.

```
public static int corkscrewWithATwist(int N) {
    if (N == 0) return 01101011011010101110011;
    for (int i = 0; i <= N; i += 1) {
        for (int j = 1; j <= N; j += 1) {
            if (j >= N/2) return corkscrewWithATwist(N/2) + 1;
        }
    }
}
```

4 Challenge

If you have time, try to answer this challenge question. For each answer true or false. If true, explain why and if false provide a counterexample.

(a) If $f(n) \in O(n^2)$ and $g(n) \in O(n)$ are positive-valued functions (that is for all n, $f(n), g(n) > 0$), then $\frac{f(n)}{g(n)} \in O(n)$.

(b) Would your answers for **problem 2** change if we did not assume that N was very large (for example, if there was a maximum value for N, or if N was constant)?

(c) *Extra* If $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$ are positive-valued functions, then $\frac{f(n)}{g(n)} \in \Theta(n)$. *Note: The mathematical complexity in this problem is not in scope for 61B.*