CS 61B
Spring 2018

Asymptotics II

Exam Prep 8: March 5, 2018

1 Warmup

Given the following method on a sorted array, what is the worst-case runtime? There is an approach to make this algorithm faster. What is that approach and what is the worst-case runtime of the faster algorithm?

```
public static int f1(int i, int[] numList) {
    for (int j = 0; j < numList.length; j++) {
        if (numList[j] == i) {
            return j;
        }
    }
    return -1;
}
```

In the worst case we iterate through the entire numList, resulting in a worst-cast runtime of $\Theta(N)$, where N is the number of elements in numList. Since it is sorted, we can use binary search to cut the runtime down to $\Theta(\log N)$ in the worst case.

2 You wanna hang out this Spring '15? Asymptotes!

For each of the pieces of code below, give the runtime in Θ notation as a function of the given parameters. Let $f(x)$ be a function that runs in time linear to the size of its input x.

```
public static void f1(int n) {
    if (n == 0) {return;}
    f1(n/2);
    f(n);
    f1(n/2);
}
```

$\Theta(n \log n)$. There are $\log n$ levels and each level does n work, for a runtime of $\Theta(n \log n)$ in total.

```
public static void f2(int n) {
    if (n == 0) {return;}
    f2(n-1);
    f(17);
    f2(n-1);
}
```

$\Theta(2^n)$. The runtime is dominated by the work done at the leaves, i.e. the bottom of the recursion, with 2^N leaves.

```java
public static void f3(int n, int m) {
    if (m <= 0) {
        return;
    } else {
        for (int i = 0; i < n; i +=1) {
            f3(n, m - 1);
        }
    }
}
```

$\Theta(n^m)$. This is a generalization of f2.

3 It's Fall '16 And I'm Still Doing Asymptotics

1. Give best- and worst-case runtime bounds for the call foo2(N,N) as a function of N. Assume that cnst() is some function that runs in constant time.

```java
public static void foo2(int i, int N) {
    if (i==0) {return;}
    for (int j = 0; j < i; j = j+1) {
        cnst();
    }
    if (i > N/2) {
        foo2(i-1, N);
    } else {
        foo2(i/2, N) + foo2(i/2, N);
    }
}
```

   Best case: $\Theta(N^2)$ Worst case: $\Theta(N^2)$

2. True or false: if $f(N) \in O(N)$ and $g(N) \in O(N^2)$, and both functions are non-negative, then $|g(N) - f(N)| \in \Omega(N)$. if true, explain why; otherwise, give a counterexample.
   False. Let $f(N) = g(N) = N$. Then the difference of the two is not bounded below by N.

3. True or false: if $f(N) \in \Theta(N)$ and $g(N) \in \Theta(N^2)$, and both functions are non-negative, then $|g(N) - f(N)| \in \Omega(N)$. If true, explain why; otherwise give a counterexample.
   True. Because $g(N)$ is bounded below by N^2, and $f(N)$ is negligible compared to N^2, then their difference is bounded below by N.

4. What is a tight big-O bound for the worst case running time of the following algorithm, as a function of the parameter r?

```
/** Assumes that VALS is a square array, and that 0 <= R, C < vals.length. */
double best(double vals[][], int r, int c) {
    if (r == 0) {
        return vals[r][c];
    }
    double v = best(vals, r-1, c);
    if (c > 0) {
        v = Math.max(v, best(vals, r-1, c-1));
    }
    if (c < vals[r].length - 1) {
        v = Math.max(v, best(vals, r-1, c+1));
    }
    return v + vals[r][c];
}
```

Worst case: $O(3^r)$

4 Fall '16: I'm more than just a runtime

1. Your friend, a budding politician, meets several hundred people a day and places their names onto the front of an ArrayList. Once there, he never removes a name, but he sometimes looks through the list to see the order in which he met people. At least one aspect of his procedures is slower than it could be. Describe and justify a small change in your friend's use of the data structure that would improve runtime without changing the data structure involved.

   Adding to the front of an ArrayList is costly because you have to shift all the elements over by 1, making it a linear time operation. Adding to the back can be done much faster.

   Now describe and justify a change in the data structure that would improve runtime without requiring a change in actions taken.

   Use a Linked List to do constant time additions to the front.

5 (Most 61B Problems) $\in$ O(These Problems)

1. Give a tight Θ bound on the running time.

```
public int f1(n):
    if (n == 1){return 0;}
    if (n is even){
        return f3(n/2);
    } else {
        return f3(n+1);
    }
```

Answer: $\Theta(\log N)$

One can come up with O and Ω bounds for different cases, and see that they're the same, since any odd number incremented by 1 is even.

2. Give a tight Θ bound on the running time, where process() is a method that runs in $\Theta(n \log n)$

```
function f2(n):
    if (n = 1){return 1;}
    int a = f2(n/2);
    int b = f2(n/2);
    x = process(a, b);
    return x;
```

Answer: $\Theta(n \log^2 n)$ The recursion tree should give you a summation that looks like this:

$$
\begin{aligned}
n \log n + n \log(\frac{n}{2}) + n \log(\frac{n}{4}) + \dots &= n(\log n + \log(\frac{n}{2}) + \log(\frac{n}{4}) + \dots) \\
&= n(\log n + (\log n - \log 2) + (\log n - \log 4) + \dots) \\
&= n((\log n + \log n + \cdots + \log n) - (\log 2 + \log 4 + \cdots + \log n)) \\
&= n((\log n)^2 - (1 + 2 + \cdots + \log n)) \\
&\approx n(\log^2 n - \frac{\log^2 n}{2}) \\
&= \frac{n \log^2 n}{2} \\
&= \Theta(n \log^2 n)
\end{aligned}
$$

3. True or False: If $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$

False; Consider $f(n) = n$ and $g(n) = n/2$, then use what you know about exponents to mess around with the base.