CSE 332 Summer 18
Section 04

1 Finding Dominant Terms

When you're looking for dominant terms (to find a big-O/big-Θ), here are some useful principles: In the following list, n is a variable and c is a constant greater than 0.

1. $n^c = 2^{c \log n}$ so every polynomial is less than 2^n. This also means that every polynomial is less than, say $2^{\sqrt{n}}$.
2. n^c dominates $\log(n)$. Even if c is less than 1.
3. Taking the log of a function makes it MUCH smaller.

Some problems

1. Prove or disprove the following claim: If $\log(f(n))$ is $\Theta(\log(g(n)))$ then $f(n)$ is $\Theta(g(n))$. If it is true, right a formal big-O proof. If it is false, provide an explicit counter-example (and explain why it is a counter-example).

2. True or false: $n^{0.001} + \log^{1000000}(n)$ is $\Theta\left(\log^{1000000}(n)\right)$

3. Which is the dominant term: $2^{\sqrt{\log(n)}} + n^5$

2 Analyzing Code

For each of the following code snippets, either give a $\Theta()$ running time, or write a recurrence to describe the running time and solve it.

```
int foo(int n){
      for(int i = 0; i < log(n); i++){
           for(int j = 0; j < n; j++){
                print "hooray"
           }
      }
      for(int k = 0; k < sqrt(n); k++){
           print "hello"
           for(int l=0; l < n; l++){
                print "hi"
                if(l % 3 == 0)
                     break; //out of the inner loop only
           }
     }
}
```

```
int bar(int n){
      if(n < 100){
           for(int i = 0; i < 2^n; i++){
                print ":)";
           }
           return 3;
      }
      else{
           for(int i = 0; i < n; i++){
                print ":("
           }
           return bar(n/2) * bar(n/2) + 8
      }
}
```

3 Heaps and AVL trees

1. Insert 3,1,4,2,5,7,9,6 into an empty binary min-heap (you can draw this either in the array format or as a tree).

2. Insert the same values into an empty binary max-heap

3. Suppose you were given the values 3,1,4,2,5,7,9,6 (in order) in an array. Run Floyd's Build Heap algorithm to turn it into a min-heap.

4. Insert the same values into an empty AVL tree.

4 Recurrences

Find an exact closed form of the following recurrence. Check your answer with the Master Theorem.

$$T(n) = \begin{cases} 4 & \text{if } n \leq 81 \\ 3T(n/9) + n^3 & \text{otherwise} \end{cases}$$

5 Using Data Structures

Professor Dumbledore has finally figured out how to get electricity to work in Hogwarts, and just installed the wizarding world's first computer. Upon hearing of your knowledge of data structures he ~~abducts you~~ apparates you to his office to help modernize the millenia-old institution.

Each night Professor McGonagall and Professor Snape each submit a list of all disciplinary action they took in the last day. Each of them submits a separate minheap to Dumbledore, with priority representing the severity of the incident (more severe incidents have lower-number priorities). Professor McGonagall's list consistently has n elements, while Snape's has 2^n elements.

Dumbledore would like to examine the k most severe incidents (regardless of whether they were in Snape's heap or McGonagall's).

The size of k depends on Dumbledore's mood, and he tells you it could be anything between $k = 1$ and $k = 2^n + n$. His first idea is to use Floyd's Build Heap algorithm to combine the two heaps into one big heap, and then do k `removeMin`s in the new heap.

```
array newHeapArr = new array[S_Heap.size + M_Heap.size]
for(i from 1 to S_Heap.size)
     newHeapArr[i] = S_Heap[i]
for(i from 1 to M_Heap.size)
     newHeapArr[i+S_Heap.size] = M_Heap[i]
Heap Combined_Heap = BuildHeap( newHeapArr )
for (i from 1 to k){
      print Combined_Heap.removeMin()
}
```

What is the (simple) big-O bound for running build heap in Dumbledore's case?

How long will the k `removeMin`s take (you should give the best, simple $O()$ bound you can. Your bound should include both n and k).

What is the overall running time of the algorithm? (again give as simple of an $O()$ bound as you can. In terms of whichever of k and n are necessary).

Some Useful Facts

When we're using the tree method to solve a recurrence, we usually use the following steps:

0. Draw a few levels of the tree.
1. Let the root node be at level 0. Give a formula for the size of the input at level i.
2. What is the number of nodes at level i?
3. What is the work done at the i^{th} recursive level?
4. What is the last level of the tree?
5. What is the work done at the base case?
6. Write an expression for the total work done.
7. Simplify until you have a "closed form" (i.e. no summations or recursion).

Geometric series identities:

$$\sum_{i=0}^{k} c^i = \frac{c^{k+1}-1}{c-1} \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c} \text{ if } |c| < 1$$

Common Summations:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \qquad \sum_{i=0}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

Log identities:

$$a^{\log_b(c)} = c^{\log_b(a)} \qquad \log_b(a) = \frac{\log_d(a)}{\log_d(b)}$$

Master Theorem:
Given a recurrence of the following form:

$$T(n) = \begin{cases} d & \text{if } n \leq \text{ some constant} \\ aT(n/b) + n^c & \text{otherwise} \end{cases}$$

with a, b, c are constants.
If $\log_b(a) < c$ then $T(n)$ is $\Theta(n^c)$
If $\log_b(a) = c$ then $T(n)$ is $\Theta(n^c \log n)$
If $\log_b(a) > c$ then $T(n)$ is $\Theta\left(n^{\log_b(a)}\right)$