Complex Runtimes and Strategies

1 Complicated Runtimes

Not all runtimes are straightforward. Recursive steps and nested for loops are the prevalent manners in which runtimes can become more tedious.

For most problems, where the runtime relies on a different factor, we can use the following summation.

$$\sum_{i=0}^{\#oflayers} \frac{work}{node} * \frac{\#ofnodes}{layer}$$

Or in English, the sum of the work per node times the number of nodes per layer. Let's step through this recursive function as practice.

```
1 public void stepper(int n){
2       int q = 0;
3       for(int i = 0; i < n; i++){
4            q++;
5       }
6       stepper(n/2);
7 }
```

Let's use the sum formula that we established earlier. It seems that the recursive call makes n $\frac{1}{2}$ of what it was; however, the recursive call does not increase the amount of nodes. Because of this, the number of nodes per layer is 1. The for loop that occurs at each iteration does work proportional to $\frac{n}{2^i}$ where is the current layer. Thus the formula for the sum is:

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

. We can then rewrite this as $n + \frac{n}{2} + \frac{n}{4} + + ... \frac{n}{2^{log(n)}} = \Theta(2n) = \Theta(n)$

```
1 public void(int n){
2          int q = 0;
3      for(int i = 0; i < n; i++){
4           for(int j =0; j < i; j++){
5               q++;
6            }
7      }
8 }
```

The running time for this method would be n^2 this is because the outer loop works a total of n times and the inner loop works in proportion to what iteration the outer loop is on. This results in the inner loop working 1 time on the first iteration, 2 times on the second etc. This make the sum 1+2+3+4+... n = $\Theta(n^2)$

Right now you're probably wondering "how can I solve these, is there a universal method?". Well, I'm sorry to be the bearer of bad news, but really, the only way to get good at asymptotics is to practice over and over again. You will begin to recognize patterns in the way asymptotics are formed and eventually you'll be able to master them. In the next chapter, we will discuss runtimes and their relation to data structures.

2 Revision of Strategies for Solving Asymptotics

Now that we have gone over strategies for solving asymptotics informally, let's list when to use which method. Using the wrong method at the wrong time may result in the wrong answer or a misunderstanding.

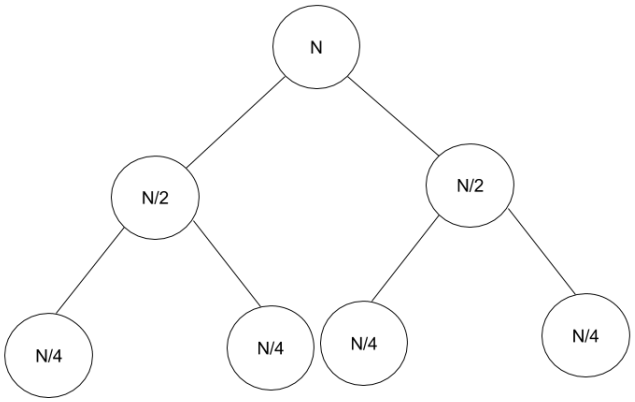
If the code is relatively straightforward, such as a basic while loop, with no recursion or loops that rely on it, you can simply just "logic" it out and count. We have an example here:

```
1 int i = 0;
2 while(int i < N){
3         i++;
4 }
```

The asymptotic runtime of this function is trivially $\Theta(N)$, no complex math is needed, just logic. These asymptotics are nice and easy; however, these are not too common.

One strategy that helps those who are more visual is the "recursion tree". For each iteration, create one more layer. Once you get a good image of it in your mind, you can stop drawing it. Let's draw a recursion tree for the following code.

```
1 public void grewt(int N){
2       for(int i = 0; i < N ; i++){
3             System.out.println("I am grewt");
4       }
5       grewt(N/2); grewt(N/2);
6 }
```



We could have gone done more layers, but for the sake of space, let's just do 3 layers. Now we can see from the following tree that we made, that each layer we divide by 2, and thus our work divides by two. This leads to the work per node being $\frac{N}{2^i}$ where i is the current layer. Now, let's look at the nodes per layer- we see that each node has 2 children because of the 2 recursive calls. So the first layer has 1 node, the second layer has 2 nodes, the third has 4 and so forth. The pattern that we can see is that there are 2^i nodes per layer where is the current layer. Now let's try to plug all the information that we gathered into our equation:

$$\sum_{i=0}^{log(n)} \frac{n}{2^i} * 2^i \rightarrow \sum_{i=0}^{log(n)} n \rightarrow \Theta(nlog(n))$$

If you go step by step, these problems become much easier to solve

2.1 Some Closing Words on Asymptotics

When thinking about "N" and best case/worst case, make sure you do not think of a specific number. For example, if the code resembles the following:

```
1  if(n ==1){
2      return;
3   }
```

you CANNOT assume that the input will just be 1 and call it a day. What you can do, when thinking about best case is a quality of the number such as:

```
1  if(n %2 == 0){
2       return;
```

This is because you are now looking at a quality of the input and assuming something about it, not the actual numeric quantity.

Additionally, when writing out your summations make sure you simplify as much as possible, (cancel out all terms that repeat) otherwise you will end up with weird numbers. Don't get demoralized if you find asymptotics hard, it is difficult to internalize the rules and situations; however, with enough practice (trust me I'll put enough practice on this document), you will be able to solve them.