Assignment 2 Pt. 1

10/9/22

2. $O((n(n^2 + \log n \cdot (\frac{3n + 3n^2}{\log n}))))$

$O(n(n^2 + 3n * \log n + 3n^2)$

$O(n(4n^2 + 3n\log n)$

$O(4n^3 + 3n^2 \log n)$

Simplifying we keep the biggest degree and take out constants so this is simplified to $O(n^3)$

1. The Big-Oh estimate of this code is $O(n \log n)$ this is because in the first loop we are looping through n times and then the inner loop divides n by 1 each time so that operation is on average $\log n$ so it simplifies to $O(n \log n)$

3. If we assume that a full binary tree has 2^k nodes at each level of k.

The total # of nodes, $N = 2^0 + 2^1 + 2^2 + \ldots + 2^h$ Where h is the height

- If we assume $h = 2$ then $N = 1 + 2 + 4$

Then we can see that the last term 4 is the # of leaves and $1+2$ are the non-leaves

- If we do the same for $h = 3$, $N = 1 + 2 + 4 + 8$

Then 8 is the number of leaves and $1+2+4$ is the # of non

These two examples that the non leaves are one less than the number of leaves. Together it's $2N - 1$.

So both take time $O(\frac{N}{2})$ which has to simplify to $O(N)$.

4. Algorithm: Wildcard (String one, String two)

```
1. if both strings are null
       return true
   if (one.length > 1 & one.charAt(0) == '*') {
       int i = 0;
       while (i + 1 < one.length & one.charAt(i+1) = '*')
           i++;
       one = one.substring(i);
   }
   if (one.length > 1 & one.charAt(0) = '*' & two.length = 0)
       return false;
   if (one.length > 0 & first.charAt(0) = '*')
       return wildcard(one.substring(1), two) ||
              wildcard(one, two.substring(1));
   return false;
```

Algorithm: Non-wildcard (String one, String two, int ind1, int ind2)

```
1. if (ind1 = one.length & ind2 != two.length)
       return false
   if (ind2 = two.length)
       return true
   if (one.charAt(ind1) = two.charAt(ind2)
       return Non-wildcard(one, two, ind1+1, ind2+1);
   return false
```

5. Algorithm: findSection (A, i, j B)

1.
```
int c = 0;
int bcount = 0;
while ( i != j  &  c != B.length)
{
    if (A[i] = B[c])
    {
        j++;
        bcount ++;
    }
    else
      bcount = 0;
    c++;
}
if ( i = j  &  bcount =  j - i)
      return true;
return false;
```