CS 445: Lab 4

Algorithm Analysis

1. Sort the following growth rates in ascending order:

   - n^3
   - $4n + 7$
   - $n!$
   - 3^n
   - $2nlog(n)$
   - $log(n)$

   Answer:

   - $log(n)$
   - $4n + 7$
   - $2nlog(n)$
   - n^3
   - 3^n
   - $n!$

   - $\frac{1}{2}n$
   - n^2
   - 256
   - n^4
   - $log(n)^2$

   Answer:

   - 256
   - $log(n)^2$
   - $\frac{1}{2}n$
   - n^2

- n^4

2. Determine the growth rates of these functions:

   Remember to remove lower order terms and multiplicative constants

   (a) $F(n) = n^2 - 3^n + 28$ *Answer:* 3^n

   (b) $F(n) = 100 + 81n \log(n) + 123n$ *Answer:* $nlog(n)$

   (c) $F(n) = 4n + \frac{n^3}{4}$ *Answer:* $\frac{n^3}{4}$

3. Determine the growth rates of each code block. The problems they solve are irrelevant; simply look at how they'll scale as n increases.

```
//Count how many even numbers are in an array of length n
int counter = 0;
for(int i = 0; i < myArray.length; i++)
{
    if(myArray[i] % 2 == 0)
        counter++;
}
```

   Growth rate is $O(n)$.

```
//print a square checkerboard of size n
for(int i = 0; i <= n; i++)
{
    for(int j = 0; j <= n; j++)
    {
        if((i+j)%2 == 0)
        {
            System.out.print("O");
        }
        else
        {
            System.out.print("X");
        }
    }
    System.out.println();
}
```

Growth rate is $O(n^2)$.

```
//Prints out a -fun- ramp (with base length of n)
//for this stick figure to skateboard down
System.out.println("   O    ");
System.out.println("  /|\\/ ");
System.out.println(" / |\\   ");
System.out.println("_/__|_ ");
System.out.println("  o  o  ");

for(int i = 0; i <= n; i++)
{
    for(int j = 0; j <= i; j++)
    {
        System.out.print("O");
    }
    System.out.println();
}
```

Growth rate is $O(n^2)$ because these loops would be $\frac{1}{2}n^2$, note that the output looks like half of a SQUARE.

```
int counter = 0;
while(n > 1)
{
    for(int i = 0; i < n; i++)
    {
        counter++;
    }
    n = n / 2;
}
```

Growth rate is $O(n)$ because it's technically $2n - \epsilon$, where it runs for n, then $\frac{n}{2}$, then $\frac{n}{4}$, $\frac{n}{8}$ getting half of the way to $2n$ each time but not quite all the way to $2n$ in the end.

```
int binarySearch(int[] a, int e)
{
    int begin = 0, end = a.length, mid;
    while (begin < end)
    {
        mid = (begin + end) / 2;
        if (e > a[mid])
        {
            begin = mid + 1;
        }
        else if ( e < a[mid])
        {
            end = mid;
        }
        else
        {
            return mid;
        }
    }
    return -1;
}
```

Growth rate is $O(log(n))$ because it halves the input size, n, on each pass.