# Part 1: Growth Rate

$$2^{100} < 3\log_2 n < 2\text{n} < 10^{\log_{10} n} < 10^{\log_2 n} < \text{n}^{90} < \ 2^{n+1} < \text{n}2^{n}$$

# Part 2: Proof and Analysis

$$n^2 \in \Omega(n \log_2 n)$$

The omega function is the best-case scenario such that there exists a number c:

$$n^2 \geq c * n \log_2 n \; for \; all \; n \; \geq n_0$$

Divide both sides by 2:

$$n \;\geq\; \text{c} * \log_2 n$$

This is true for c = 1. This is true for all values of n ≥ 1. So $n_0 = 1$

Since $n^2 \geq c * n \log_2 n \; for \; c = 1 \; when \; n \; \geq 1$ We can conclude $n^2 \in \Omega(n \log_2 n)$

# Part 3: Algorithm Analysis:

## Question 1:

```
static int foo(int[] a) {
  int n = a.length;
  int tot = 0;
  for (int j = 0; j < n; j++)
    if (a[j] > 0) tot = tot + a[j];
}
```

i. The worst-case input for the method foo() is an array with a large size, the bigger the n, the worse runtime.

ii. A summation that represents foo():

$$\sum_{i=0}^{n} 1$$

,

iii. The code goes through the array one time, and each step does something that takes constant time

iv. R(n) $\in O(n)$

# Question 2:

```java
static int bar(int[] z) {
  int x = z.length;
  for (int i = 0; i < x/2; i++) {
    for (int j = 0; j < x; j += 3)
      if (z[i] == 10) {
        System.out.println("Hi");
        break;
      }
    for (int k = 0; k < x; k++) {
      System.out.println("Lo");
      if (k >= i) break;
    }
  }
}
```

i. The worst case input of z is an array with a large number of elements, and no element of z = 10.

ii. Summation of bar():

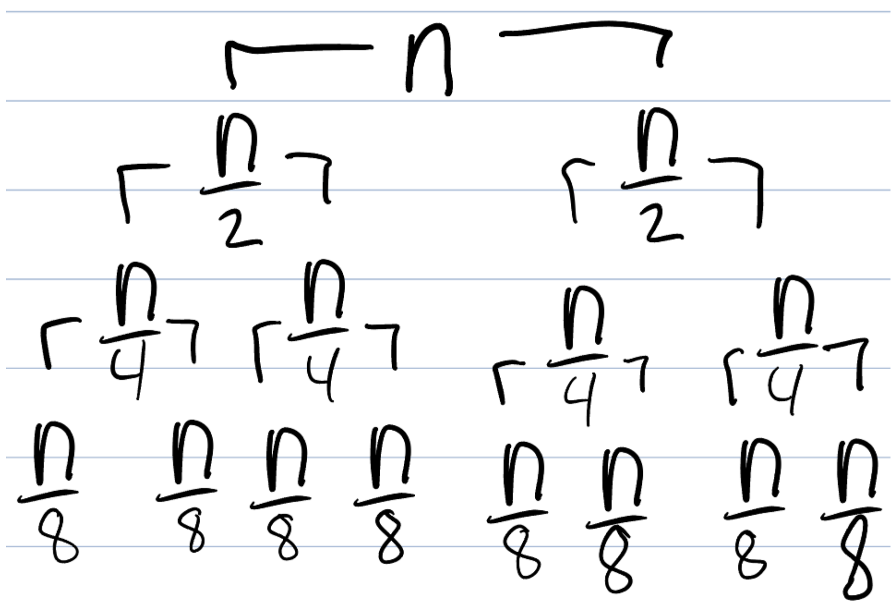
$$\sum_{i=0}^{x/2}(\sum_{j=0}^{x/3} 1 + \sum_{k=0}^{i} 1)$$

iii. The first inner for-loop starts at j=0, incrementing to x by steps of 3.

That's why the upper bound is x/3. Inside that for-loop, there's a constant-time operation, represented by the 1. If the i-th value of z == 19, the loop breaks early, which is why the worst case excludes 19 from the values in z.

The second for-loop starts at k = 0 and looks like it goes up to x, but it breaks when k equals i, so it actually only runs up to i. The runtime in each step of the loop is constant.

## Problem 3: Recursion Chart




There are $\log_2 n$ levels in the tree since each level gets cut in half

## Problem 4:

i. The worst-case runtime of sum(myList) is Θ (n log n) with an input of size n > 1

ii.

$$\sum_{i=0}^{\log_2 n-1} 2^i * \frac{n}{2^i}$$

iii. There are 2^i recursive calls and each sub list is $\frac{n}{2^i}$

iv. $R(n) \in \Theta(n \log_2 n)$

## Problem 5:

i. The worst-case runtime of sum(myList) is when the input has size n > 1.

ii.

$$\sum_{i=0}^{\log_2 n} \frac{n^2}{2^i}$$

iii. The summation is the same as for ArrayList, but with $n^2$ instead of n because LinkedList methods have different runtimes. The constructor and add() are constant time but get() takes O(n), which causes the total runtime to be $O(n^2)$.

iv. $R(n) \in \Theta(n^2 \log_2 n)$