# Discussion 7: Asymptotic Analysis

**Discussion**: Wed 4-5pm 120 Wheeler Hall
**Lab**: Thurs 3-5pm 275 Soda Hall
**OH**: Thurs 1-2pm in 220 Jacobs Hall

**email**: asetoodehnia@berkeley.edu
**website**: asetoodehnia.github.io (or find it through the CS 61B staff webpage)

***Please fill out this google form***: http://bit.ly/cs61b-feedback

## Reminders:

- Project 1 (Enigma) due **Monday 10/21**
- HW5 due **Wednesday 10/23**

## Today's Goals:

1. Review ***Asymptotic Analysis***
2. Get through as many questions as possible

## Things to remember:

- when considering asymptotics we will always be concerned with the runtime of the program as the **size of the input grows very large**

$O(\cdot)$ - upper bound $\quad$ $\Omega(\cdot)$ - lower bound

$\Theta(\cdot)$ - tight bound

## Let's start with Big-Oh

Let $f(x)$ be any arbitrary function. We say that $f(x) = O(g(x))$ as $x \to \infty$ ***IF AND ONLY IF*** the following is satisfied:

there exists some number $M$, such that $|f(x)| \leq Mg(x)$ for any $x > x_0$.

**In other words**, if some multiple of $g(x)$ is an upper bound for $f(x)$ for sufficiently large $x$, then we can say that $f(x) = O(g(x))$.

## We see a similar definition for Big-Omega

Let $f(x)$ be any arbitrary function. We say that $f(x) = \Omega(g(x))$ as $x \to \infty$ ***IF AND ONLY IF*** the following is satisfied:

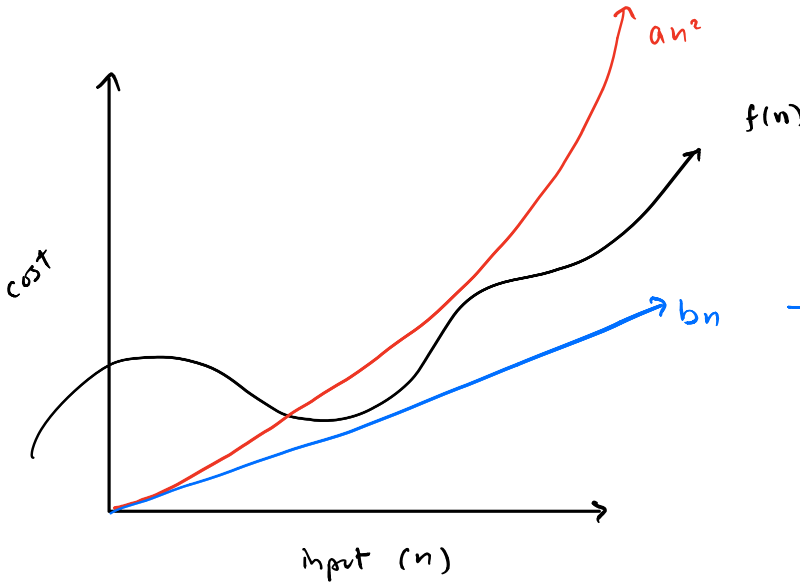
there exists some number $M$, such that $|f(x)| \geq Mg(x)$ for any $x > x_0$.

**In other words**, if some multiple of $g(x)$ is a lower bound for $f(x)$ for sufficiently large $x$, then we can say that $f(x) = \Omega(g(x))$.

## Now for Big-Theta

If we grasp the previous two definitions, then this one won't be too bad. If there exists a $g(x)$that satisfies both of the above conditions, then we say that $f(x) = \Theta(g(x))$.

## Let's look at some graphs now...

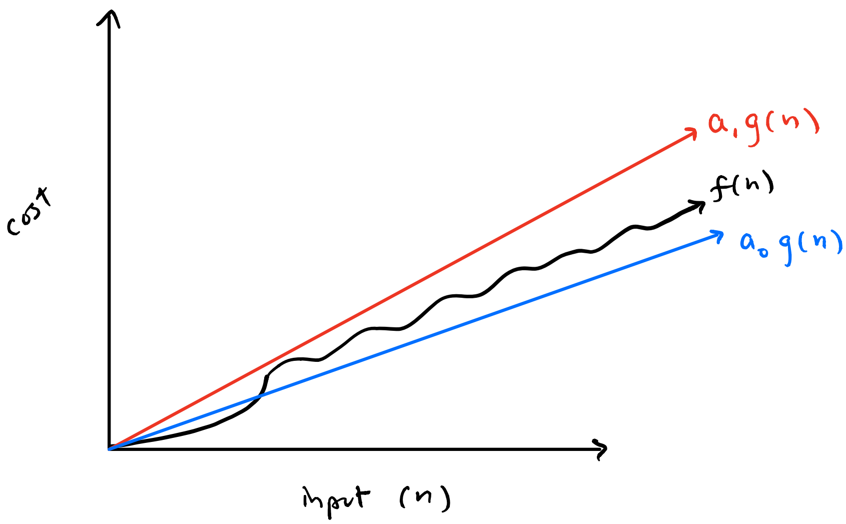



→ $O(n^2)$

→ $\Omega(n)$

∴ we have

$f(n) = O(n^2)$ & $f(n) = \Omega(n)$




now... $g(n) = n$

so... we see

$f(x) = O(n) = \Omega(n) = \Theta(n)$

## Other important stuff to remember

1. $$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} = \frac{1}{2}(N^2 + N) \in \Theta(N^2)$$

2. $$\sum_{i=0}^{\log_2(N)} 2^i = 1 + 2 + 4 + \cdots + 2^{log_2(N)-1} + N = 2N - 1 \in \Theta(N)$$

# Recursive Runtime tips...

**The most helpful** thing to do to help understand the runtime of a recursive problem is to
**consider a tree which represents all of the function calls.**

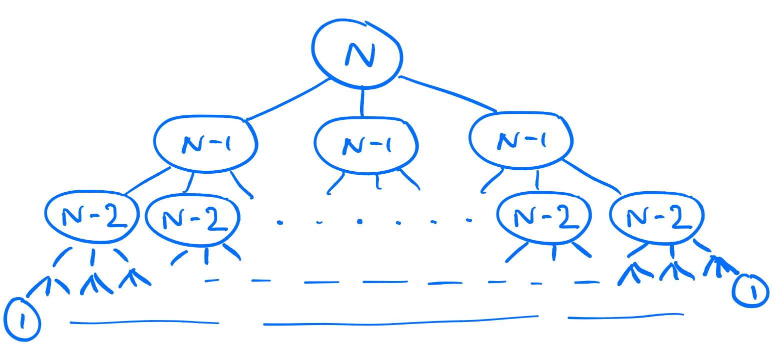
1. Determine the height of the tree. There are various ways in which to do this which we be shown throughout the problems below.
2. Determine the branching factor. This is typically the **number of recursive function calls** that are made from **each function call**. You can also use the branching factor in determining the number of nodes at any given layer of the tree.
3. Determine the **amount of work done at each node relative to the input size.**
4. Calculate the entire amount of work being done in the entire function call by:

$$\sum_{\text{layers in tree}} \frac{\#\text{ nodes}}{\text{layer}} \cdot \frac{\text{amount of work}}{1 \text{ node}}$$

Let's look at an example to try and make sense of this...

(c) Give the running time in terms of $N$.

```
public int tothe(int N) {
    if (N <= 1) {
        return N;
    }
    return tothe(N - 1) + tothe(N - 1) + tothe(N - 1);
}
```




each node does $\Theta(1)$ amount of work.

height is $N$.

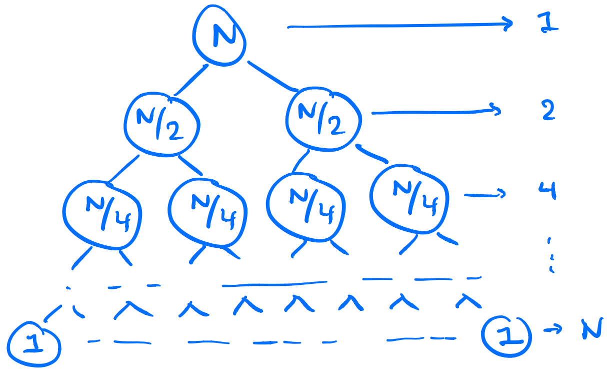
branching factor is 3.

so... $\sim 3^N$ nodes ...

$\therefore\ \Theta(3^N)$ ◻

Another example...

```
public int f(int n){
    if (n <= 1){
        return n;
    }
    return f(n/2) + f(n/2);
}
```




height is log(N)

branching factor is 2

$$\Rightarrow \sum_{i=0}^{\log(N)} 2^i = 1 + 2 + 4 + \cdots + N = 2N - 1$$

$\therefore \ \Theta(N)$ $\square$

# 1. More Running Time

Give the worst case and best case running time in $\Theta(\cdot)$ notation in terms of $M$ and $N$.

(a) Assume that `slam()` $\in \Theta(1)$ and returns a boolean.

```
public void comeon(int M, int N) {
    int j = 0;
    for (int i = 0; i < N; i += 1) {
        for (; j < M; j += 1) {
            if (slam(i, j))
                break;
        }
    }

    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```

(b) *Exam Practice:* Give the worst case and best case running time in $\Theta(\cdot)$ notation in terms of $N$ for `find`.

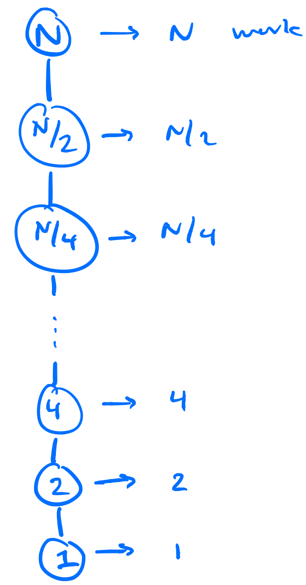
```
public static boolean find(int tgt, int[] arr) {
    int N = arr.length;
    return find(tgt, arr, 0, N);
}
private static boolean find(int tgt, int[] arr, int lo, int hi) {
    if (lo == hi || lo + 1 == hi) {
        return arr[lo] == tgt;
    }
    int mid = (lo + hi) / 2;
    for (int i = 0; i < mid; i += 1) {
        System.out.println(arr[i]);
    }
    return arr[mid] == tgt || find(tgt, arr, lo, mid)
                           || find(tgt, arr, mid, hi);
}
```

## 2. Recursive Running Time

For the following recursive functions, give the worst case and best case running time in $\Theta(\cdot)$ notation.

(a) Give the running time in terms of $N$.

```java
public void andslam(int N) {
    if (N > 0) {
        for (int i = 0; i < N; i += 1) {
            System.out.println("bigballer.jpg");
        }
        andslam(N / 2);
    }
}
```




$\Rightarrow 1 + 2 + 4 + \cdots + \frac{N}{4} + \frac{N}{2} + N$ total work

$= 2N - 1 \in \Theta(N)$ □

(b) Give the running time for `andwelcome(arr, 0, N)` where $N$ is the length of the input array `arr`.

```java
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) {
         System.out.print("loyal  ");
    }
    System.out.println("]");
    if (high - low > 1) {
        double coin = Math.random();
        if (coin > 0.5) {
            andwelcome(arr, low, low + (high - low) / 2);
        } else {
            andwelcome(arr, low, low + (high - low) / 2);
            andwelcome(arr, low + (high - low) / 2, high);
        }
    }
}
```

(d) Give the running time in terms of $N$

```
public static void spacejam(int N) {
    if (N == 1) {
        return;
    }
    for (int i = 0; i < N; i += 1) {
        spacejam(N-1);
    }
}
```