Discussion 7: Asymptotics and Bits

Discussion: Wed 2-3pm 3111 Etcheverry Hall
Lab: Fri 1-3pm 275 Soda Hall
OH: Tues 3-4pm in 109 Morgan Hall

email: asetoodehnia@berkeley.edu
website: asetoodehnia.github.io (or find it through the CS 61B staff webpage)
(I post my discussion materials here if you are interested in referring to them after section)

Reminders:

1. HW 4 is due Friday
2. Proj 1 Checkpoint is due Tuesday
3. Project Party this Saturday
4. Lab this week is Project OH
5. Post Midterm Advising

Today's Goals:

1. Review ***Asymptotic Analysis*** and ***Bitwise Operations***
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

$\Omega(\cdot)$: lower bound

$\Theta(\cdot)$: tight bound

$f(x) = O(g(x))$

$f(x) < g(x) \cdot k \quad \forall x > x_0$

$f(x) = x \qquad g(x) = x^2$

$f(x) = O(g(x))$

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 $f(x) = \Omega(g(x))$.

$g(x) = x + 2 \quad f(x) = x^2$

$f(x) = \Omega(g(x))$ ✓

Now for Big-Theta

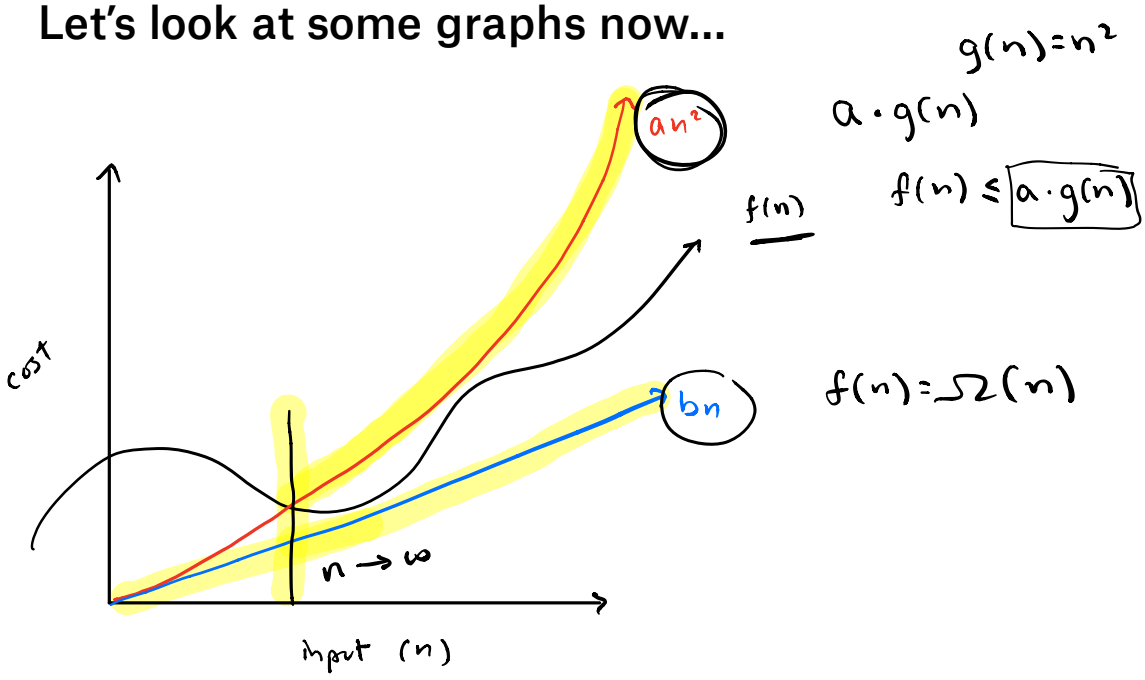
If we grasp the previous two definitions, then this one won't be too bad. If there exists a $g(x)$that satisfies both of the above conditions, then we say that $f(x) = \Theta(g(x))$.

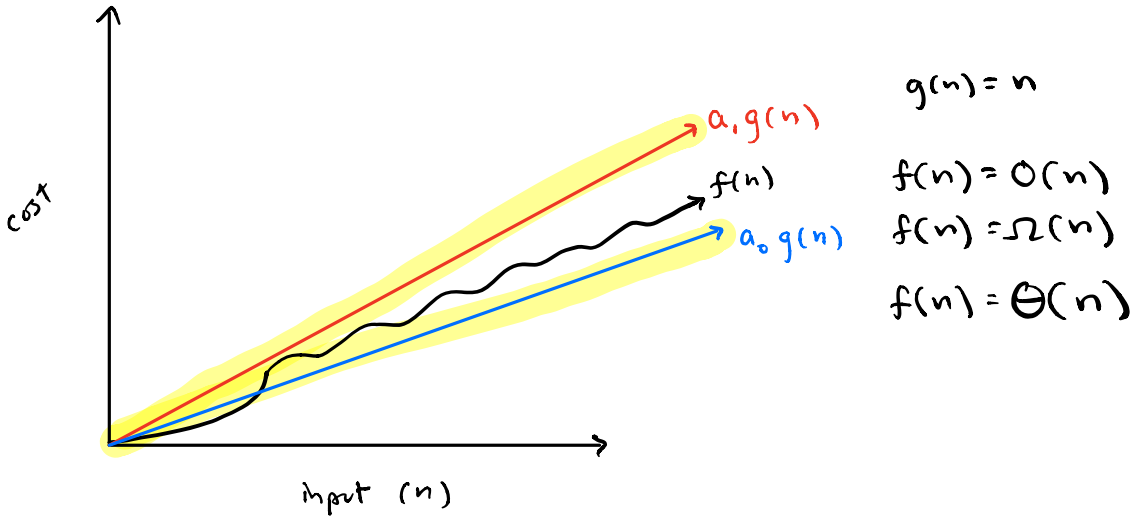
$f(x) = x^2 + x + 1$

$\rightarrow g(x) = x^2$

$f(x) = \Theta(g(x))$

Let's look at some graphs now...





Other important stuff to remember

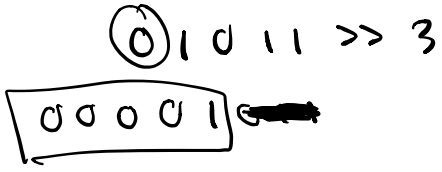
1. $\sum_{i=1}^{N} i = \frac{N(N+1)}{2} = \frac{1}{2}(N^2+N) \in \Theta(N^2)$

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

Bitwise Operations

Mask (And)	Set (Or)	Flip (Xor)	Flip All (Neg)
01101011	01101011	01101011	
& 10100101	| 10100101	^ 10100101	~ 10100101
00100001	11101111	11001110	01011010

Shift Left	Shift Logical Right	Shift Arithmetic Right
11101011	11101011	11101011
<<< 3	>>> 3	>> 3
01011000	00011101	11111101



CS 61B Asymptotics and Bits Spring 2020

1 Basic Algorithmic Analysis

For each of the following function pairs f and g, list out the Θ, Ω, O relationships between f and g, if any such relationship exists. For example, $f(x) \in O(g(x))$.

1. $f(x) = x^2, g(x) = x^2 + x$ — $f(x) \in \Theta(g(x))$
2. $f(x) = 5000000x^3, g(x) = x^5$ — $f(x) \in O(g(x))$
3. $f(x) = \log(x), g(x) = 5x$ — $f(x) \in O(g(x))$
4. $f(x) = e^x, g(x) = x^5$ — $f(x) \in \Omega(g(x))$
5. $f(x) = \log(5^x), g(x) = x$ — $f(x) \in \Theta(g(x))$

$\log(5^x) = x\log(5)$

2 Practice with Runtime

For each of the following functions, find the Big-Theta expression for the runtime of the function in terms of the input variable n.

You may find the following relations helpful:

$$1+2+3+4+\cdots+N = \Theta(N^2)$$

$$1+2+4+8+\cdots+N = \Theta(N)$$

1. For this problem, you may assume that the static method `constant` runs in $\Theta(1)$ time.

```
1  public static void bars(int n) {
2      for (int i = 0; i < n; i += 1) {
3          for (int j = 0; j < i; j += 1) {
4              System.out.println(i + j);
5          }
6      }
7
8      for (int k = 0; k < n; k += 1) {
9          constant(k);
10     }
11 }
```

$i = 2$

$\rightarrow 0 + 1 + 2 + 3 + 4 + \cdots + n$

$\rightarrow \Theta(n^2)$

$\rightarrow \Theta(n)$

$\Theta(n^2)$

2. Determine the runtime for `barsRearranged`.

```
1  public static void cowsGo(int n) {
2      for (int i = 0; i < 100; i += 1) {
3          for (int j = 0; j < i; j += 1) {
4              for (int k = 0; k < j; k += 1) {
5                  System.out.println("moove");
6              }
7          }
8      }
9  }
10
11 public static void barsRearranged(int n) {
12     for (int i = 1; i <= n; i *= 2) {
13         for (int j = 0; j < i; j += 1) {
14             cowsGo(j);
15         }
16     }
17 }
```

→ $\Theta(1)$

$1 + 2 + 4 + 8 + \cdots + N$
$= 2N - 1$
$\Theta(n)$

→ i = 1 2 4 8 ... n

→ $1 + 2 + 4 + 8 + \cdots + n$

$= 2n - 1$

$\Theta(n)$

3 A Bit on Bits

Recall the following bit operations and shifts:

1. Mask (x & y): yields 1 only if both bits are 1.
   01110 & 10110 = 00110

2. Set (x | y): yields 1 if at least one of the bits is 1.
   01110 | 10110 = 11110

3. Flip (x ^ y): yields 1 only if the bits are different.
   01110 ^ 10110 = 11000

4. Flip all (~ x): turns all 1's to 0 and all 0's to 1.
   ~ 01110 = 10001

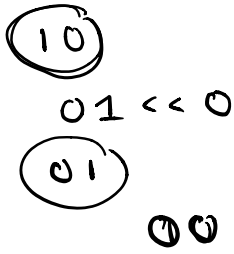
5. Left shift (x << left_shift): shifts the bits to the left by left_shift places, filling in the right with zeros.
   10110111 << 3 = 10111000

6. Arithmetic right shift (x >> right_shift): shifts the bits to the right by right_shift places, filling in the left bits with the current existing leftmost bit.
   10110111 >> 3 = 11110110
   00110111 >> 3 = 00000110

7. Logical right shift (x >>> right_shift): shifts the bits to the right by right_shift places, filling in the left with zeros.
   10110111 >>> 3 = 00010110

Implement the following two methods. For both problems, `i=0` represents the least significant bit, `i=1` represents the bit to the left of that, and so on.

1. Implement `isBitIOn` so that it returns a boolean indicating if the `ith` bit of `num` has a value of 1. For example, `isBitIOn(2, 0)` should return `false` (the 0th bit is 0), but `isBitIOn(2, 1)` should return `true` (the 1st bit is 1).




```
/** Returns whether the ith bit of num is a 1 or not. */
public static boolean isBitIOn(int num, int i) {

    int mask = 1 << i;

    return (mask & num) != 0;
}
```

2. Implement `turnBitIOn` so that it returns the input number but with its `ith` significant bit set to a value of 1. For example, if `num` is 1 (1 in binary is 01), then calling `turnBitIOn(1, 1)` should return the binary number 11 (aka 3).

```
/** Returns the input number but with its ith bit changed to a 1. */
public static int turnBitIOn(int num, int i) {

    int mask = 1 << i;

    return num | mask;
}
```