Analyzing code ("worst case")… let's count!

Assume basic operations take "some amount of" constant time
- Arithmetic
- Assignment
- Access one Java field or array index
- Etc.

This is an approximation of reality: a very useful "lie"

Consecutive statements	Sum of time of each statement
Loops	Num iterations * time for loop body
Conditionals	Time of condition plus time of *slower* branch
Function Calls	Time of function's body
Recursion	Solve *recurrence equation*

Examples

```
b = b + 5
c = b / a
b = c + 100
```

```
for (i = 0; i < n; i++) {
  sum++;
}
```

```
if (j < 5) {
  sum++;
} else {
  for (i = 0; i < n; i++) {
    sum++;
  }
}
```


What is the number of operations in this code? What is the big Oh?

```
int coolFunction(int n, int sum) {
  int i, j;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
      sum++;
    }
  }
  print "This program is great!"
  for (i = 0; i < n; i++) {
    sum++;
  }
  return sum
}
```

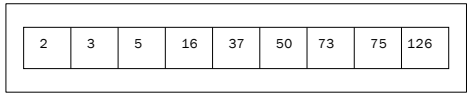

Using Summations for Loops

```
for (i = 0; i < n; i++) {
  sum++;
}
```

When math is helpful

```
for (i = 0; i < n; i++) {
  for (j = 0; j < i; j++) {
    sum++
  }
}
```


Linear search – Best Case & Worst Case



Find an integer in a *sorted* array

```
// requires array is sorted
// returns whether k is in array
boolean find(int[]arr, int k){
  for(int i=0; i < arr.length; ++i)
    if(arr[i] == k)
      return true;
  return false;
}
```

Best case:

Worst case:

Review: Properties of logarithms

- $\log(A*B) = \log A + \log B$
  - So $\log(N^k) = k \log N$
- $\log(A/B) = \log A - \log B$
- $\log_2 2^x = x$

Asymptotic Analysis

About to show formal definition, which amounts to saying:

1. Eliminate low-order terms
2. Eliminate constant coefficients

Examples:

- $4n + 5$
- $0.5n \log n + 2n + 7$
- $n^3 + 2^n + 3n$
- $n \log (10n^2)$