CS 600.226: Data Structures

Michael Schatz

Sept 17 2018

Lecture 8. Sorting

Agenda

1. ***Review HW1***
2. ***Introduce HW 2***
3. ***Recap on complexity***
4. ***Sorting***

Assignment 1: Due Friday Sept 14 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment01/assignment01.md

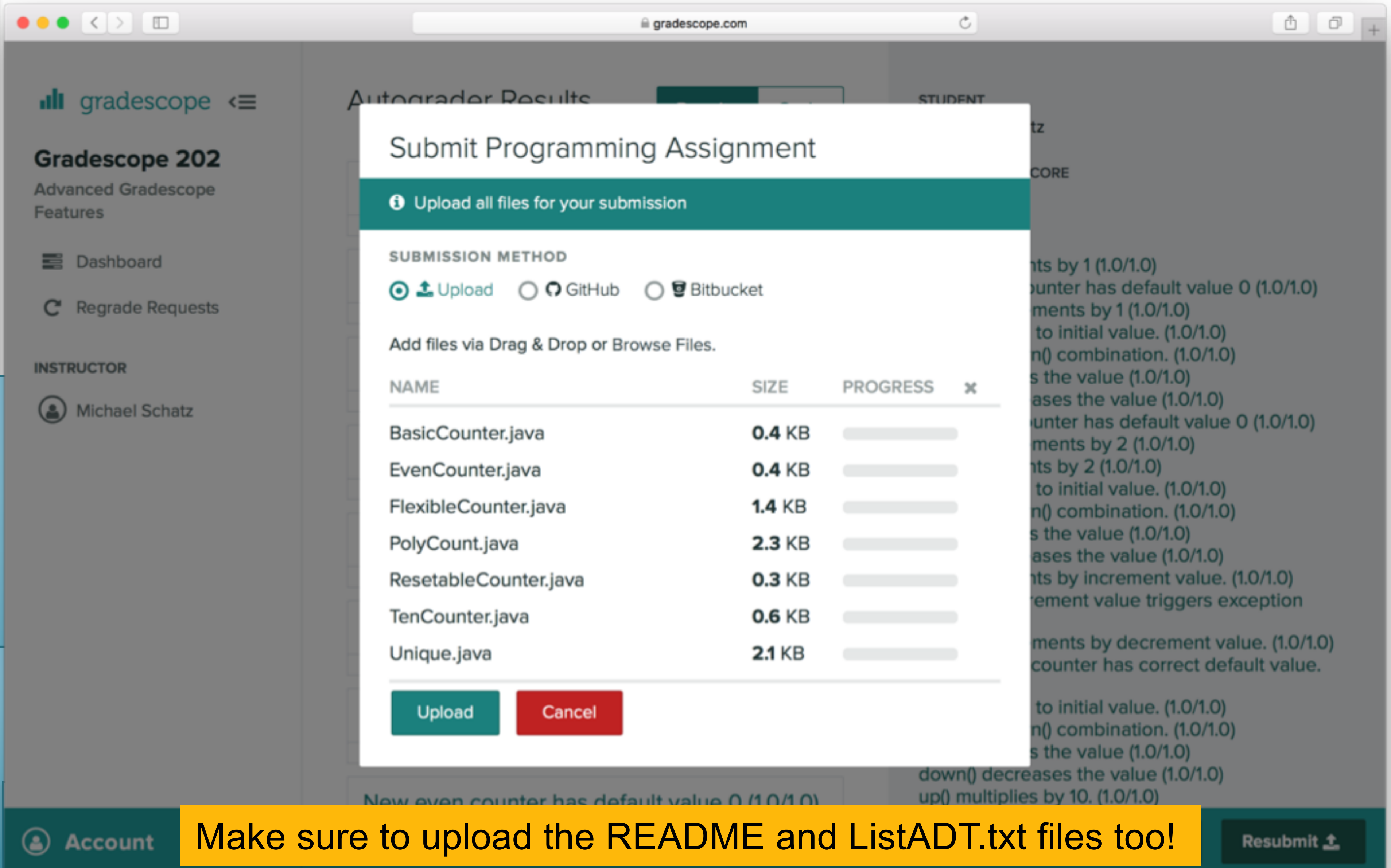
Assignment 1: Warming Up

- **Out on:** September 7, 2016
- **Due by:** September 14, 2016 before 10:00 pm
- **Collaboration:** None
- **Grading:**
  - Functionality 65%
  - ADT Solution 30%
  - Solution Design and READMDE 5%
  - Style 0%

Overview

The first assignment is mostly a warmup exercise to refresh your knowledge of Java and an ADT problem to start you thinking more abstractly about your data.

GradeScope.com
Entry Code: MDJYER



Make sure to upload the README and ListADT.txt files too!

Agenda

1. ***Review HW1***
2. ***Introduce HW 2***
3. ***Recap on complexity***
4. ***Sorting***

Assignment 2: Due Friday Sept 21 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment02/README.md

Assignment 2: Arrays of Doom!

Out on: September 14, 2018
Due by: September 21, 2018 before 10:00 pm
Collaboration: None
Grading:

- Functionality 65%
- ADT Solution 20%
- Solution Design and README 5%
- Style 10%

Overview
The second assignment is mostly about arrays, notably our own array specifications and implementations, not just the built-in Java arrays. Of course we also once again snuck a small ADT problem in there...

Note: The grading criteria now include **10% for programming style**. Make sure you use Checkstyle with the correct configuration file from Github!

Assignment 2: Due Friday Sept 21 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment02/README.md

Problem 1: Revenge of Unique (30%)

You wrote a small Java program called Unique for Assignment 1. The program accepted any number of command line arguments (each of which was supposed to be an integer) and printed each unique integer it received back out once, eliminating duplicates in the process.

For this problem, you will implement a new version of Unique called ***UniqueRevenge*** with two major changes:

- First, you are no longer allowed to use Java arrays (nor any other advanced data structure), but you can use our Array interface and our SimpleArray implementation from lecture (also available on github)

- Second, you're going to modify the program to read the integers from standard input instead of processing the command line.

Assignment 2: Due Friday Sept 21 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment02/README.md

Problem 2: Flexible Arrays (20%)

Develop an algebraic specification for the abstract data type FlexibleArray which works like the existing Array ADT for the most part **except** that both its **lower** and its **upper** index bound are set when the array is created. The lower as well as upper bound can be **any** integer, provided the lower bound is **less than or equal** the upper bound.

Write up the specification for FlexibleArray in the format we used in lecture and **comment** on the design decisions you had to make. Also, tell us what kind of array **you** prefer and why.

Hints

- A FlexibleArray for which the lower bound equals the upper bound has exactly one slot.
- Your FlexibleArray is **not** the Array ADT we did in lecture; it doesn't have to support the exact same set of operations.

Assignment 2: Due Friday Sept 21 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment02/README.md

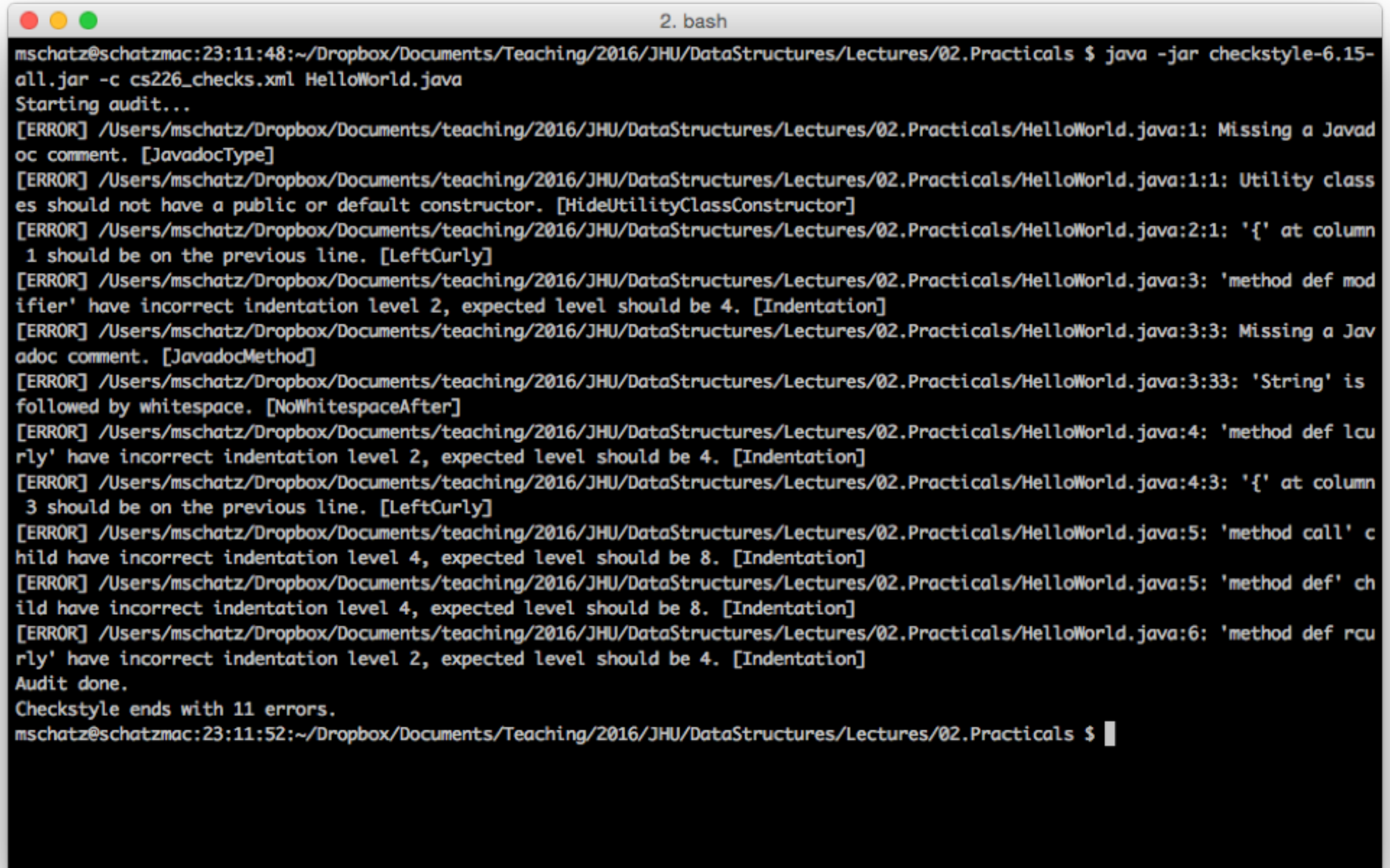
Problem 3: Sparse Arrays (35%)

A **sparse** array is an array in which **relatively few** positions have values that differ from the initial value set when the array was created. For sparse arrays, it is wasteful to store the value of **all** positions explicitly since **most of them never change** and take the default value of the array. Instead, we want to store positions that **have actually been changed**.

For this problem, write a class SparseArray that implements the Array interface we developed in lecture (the same interface you used for Problem 1 above). **Do not modify the Array interface in any way!** Instead of using a plain Java array like we did for SimpleArray, your SparseArray should use a **linked list** of Node objects to store values, similar to the ListArray from lecture (and available in github). However, your nodes no longer store just the **data** at a certain position, they also store **the position itself**!

Introduction to Checkstyle

http://checkstyle.sourceforge.net/




```
$ java -jar datastructures2018/resources/checkstyle-8.12-all.jar \
      -c datastructures2018/resources/cs226_checks.xml HelloWorld.java
```

Agenda

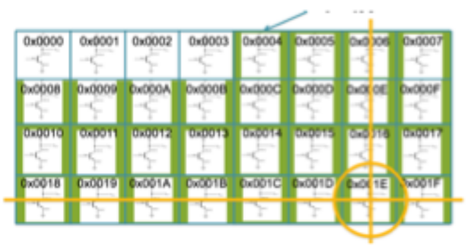
1. ***Review HW1***
2. ***Introduce HW 2***
3. ***Recap on complexity***
4. ***Sorting***

Complexity Analysis

How long will the algorithm take when run on inputs of different sizes:

- If it takes X seconds to process 1000 items, how long will it take to process twice as many (2000 items) or ten times as many (10,000 items)?

Generally looking for an order of magnitude estimate:

Constant time	***Linear time***	***Quadradic time***
		
Accessing 1st or 1 billionth entry from an array takes same amount of time	Takes 10 times longer to scan a list that has 10 times as many values	Nested loops grows with the square of the list length

Also very important for space characterization:

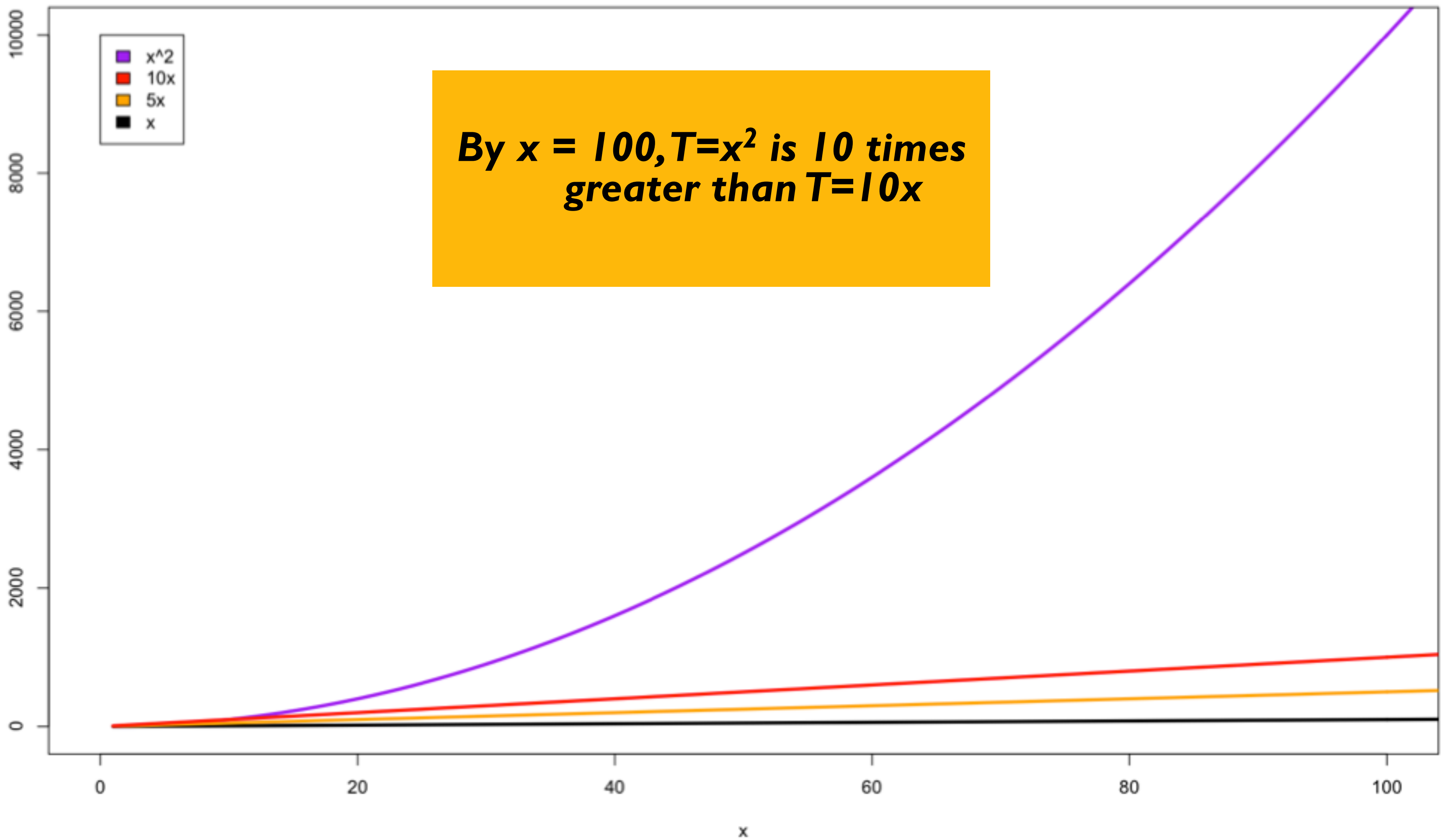
Sometimes doubling the number of elements will more than double the amount of space needed

Big-O Notation

- ***Formally, algorithms that run in O(X) time means that the total number of steps (comparisons and assignments) is a polynomial whose largest term is X, aka asymptotic behavior***
  - ***$f(x) \in O(g(x))$ if there exists $c > 0$ (e.g., $c = 1$) and x_0 (e.g., $x_0 = 5$) such that $f(x) \leq cg(x)$ whenever $x \geq x_0$***
    - *$T(n) = 33$ => $O(1)$*
    - *$T(n) = 5n-2$ => $O(n)$*
    - *$T(n) = 37n^2 + 16n - 8$ => $O(n^2)$*
    - *$T(n) = 99n^3 + 12n^2 + 70000n + 2$ => $O(n^3)$*
    - *$T(n) = 127n \log(n) + \log(n) + 16$ => $O(n \lg n)$*
    - *$T(n) = 33 \log(n) + 8$ => $O(\lg n)$*
    - *$T(n) = 900*2^n + 12n^2 + 33n + 54$ => $O(2^n)$*

- ***Informally, you can read Big-O(X) as "On the order of X"***
    - *$O(1)$ => On the order of constant time*
    - *$O(n)$ => On the order of linear time*
    - *$O(n^2)$ => On the order of quadratic time*
    - *$O(n^3)$ => On the order of cubic time*
    - *$O(\lg n)$ => On the order of logarithmic time*
    - *$O(n \lg n)$ => On the order of n log n time*

Growth of functions



By x = 100, T=x^2 is 10 times greater than T=10x

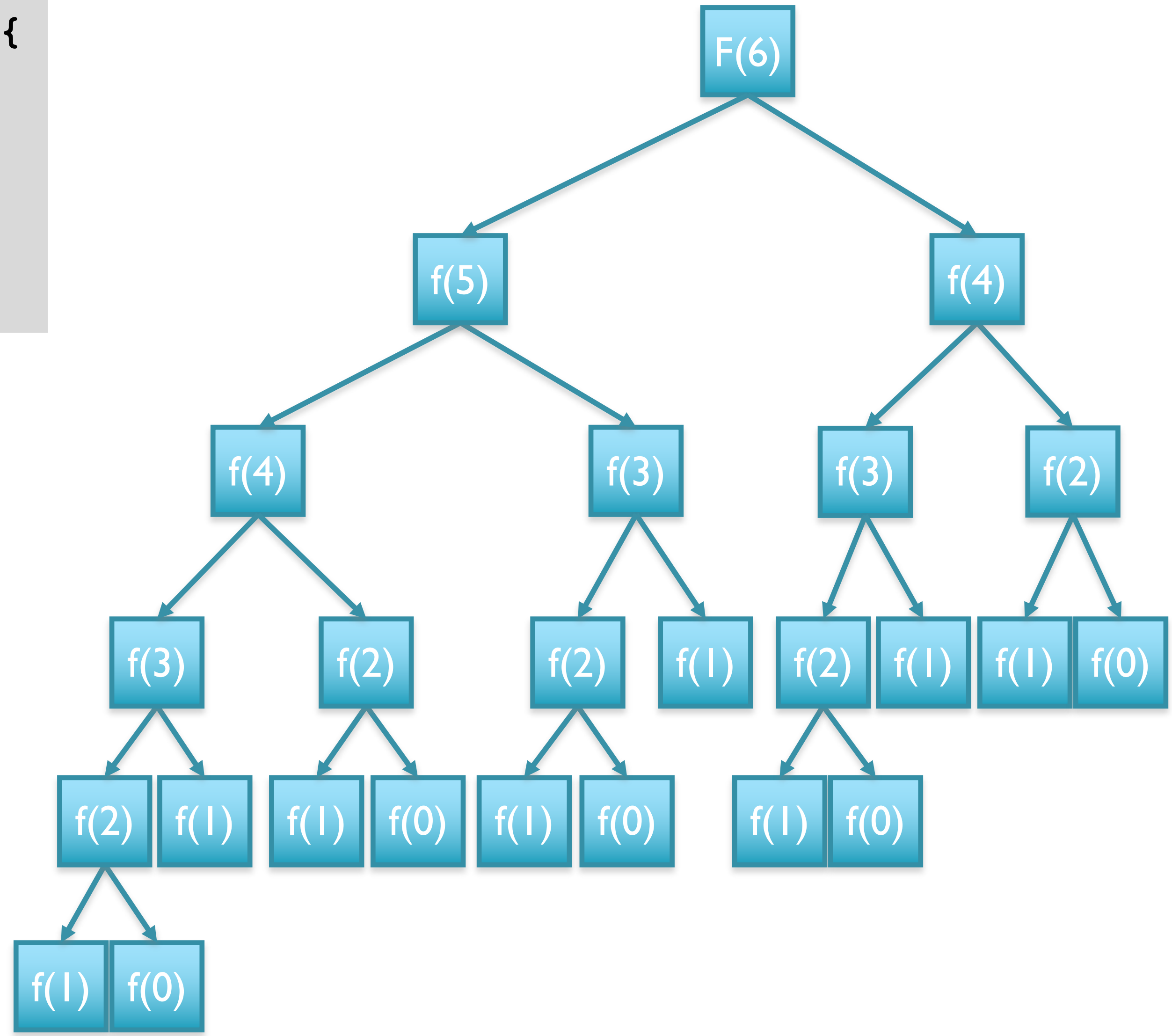
A quadratic function isnt necessarily larger than a linear function for all possible inputs, but eventually will be

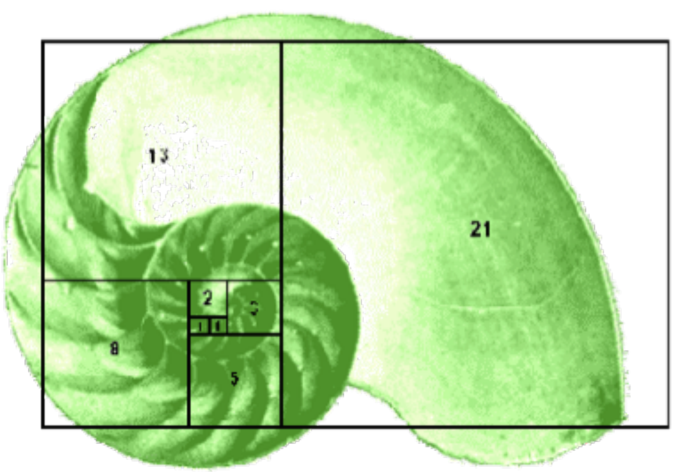
That largest polynomial term defines the Big-O complexity

Fibonacci Sequence

```
public static int fib(int n) {
  if (n <= 1) {
    return 1;
  }

  return buz(n-1) + buz(n-2);
}
```

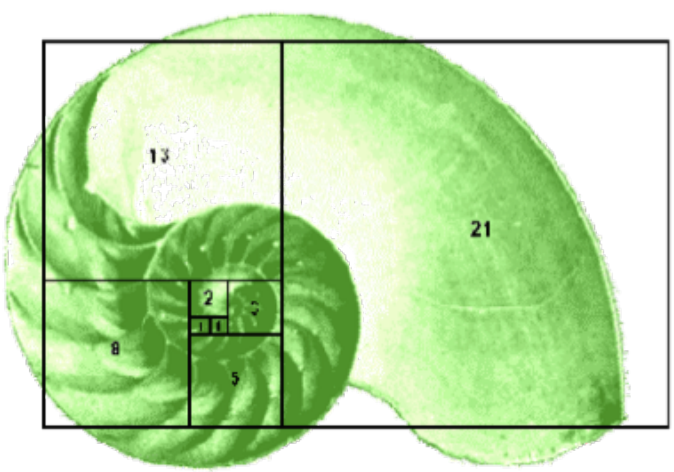


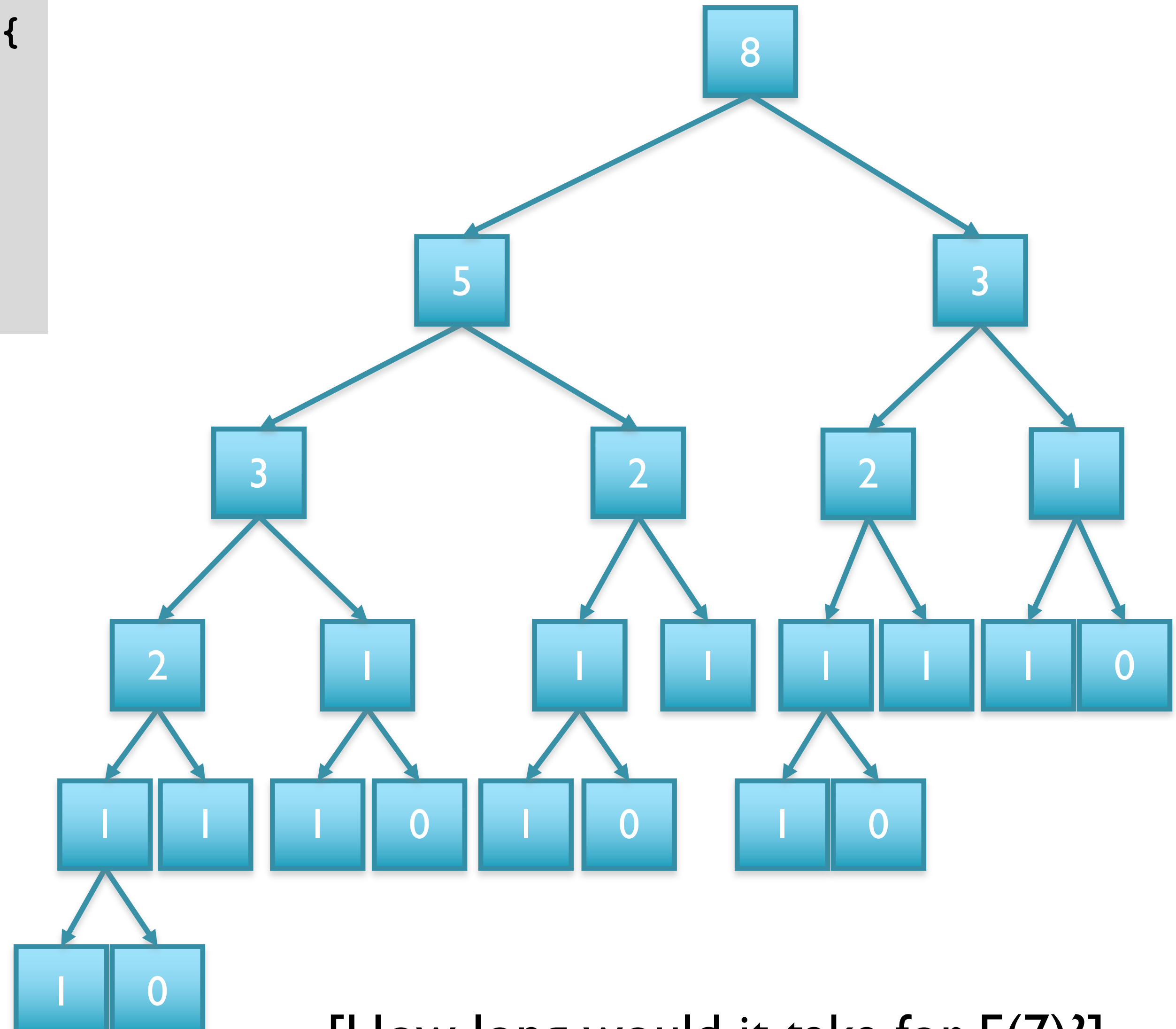


Fibonacci Sequence

```
public static int fib(int n) {
  if (n <= 1) {
    return 1;
  }

  return buz(n-1) + buz(n-2);
}
```





[How long would it take for F(7)?]
[What is the running time?]

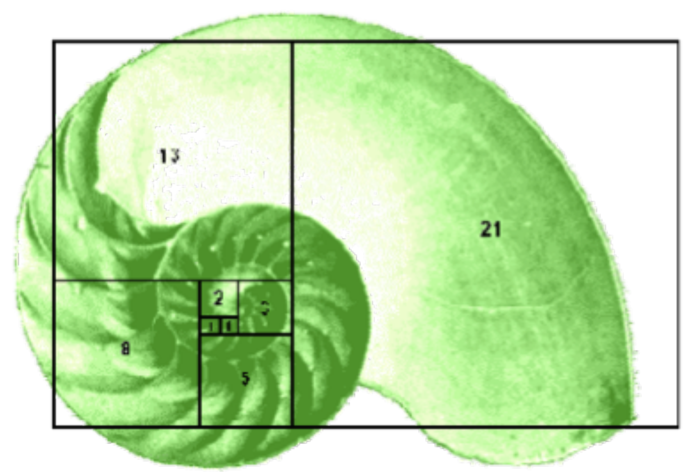
Bottom-up Fibonacci Sequence

```java
public static int fastbuz(int n) {
   int [] s = new int[n+1];
   s[0] = 1; s[1] = 1;

   for (int i = 2; i <= n; i++) {
       s[i] = s[i-1] + s[i-2];
   }

   return s[n];
 }
```

0	1	2	3	4	5	6
0	1	1	2	3	5	8



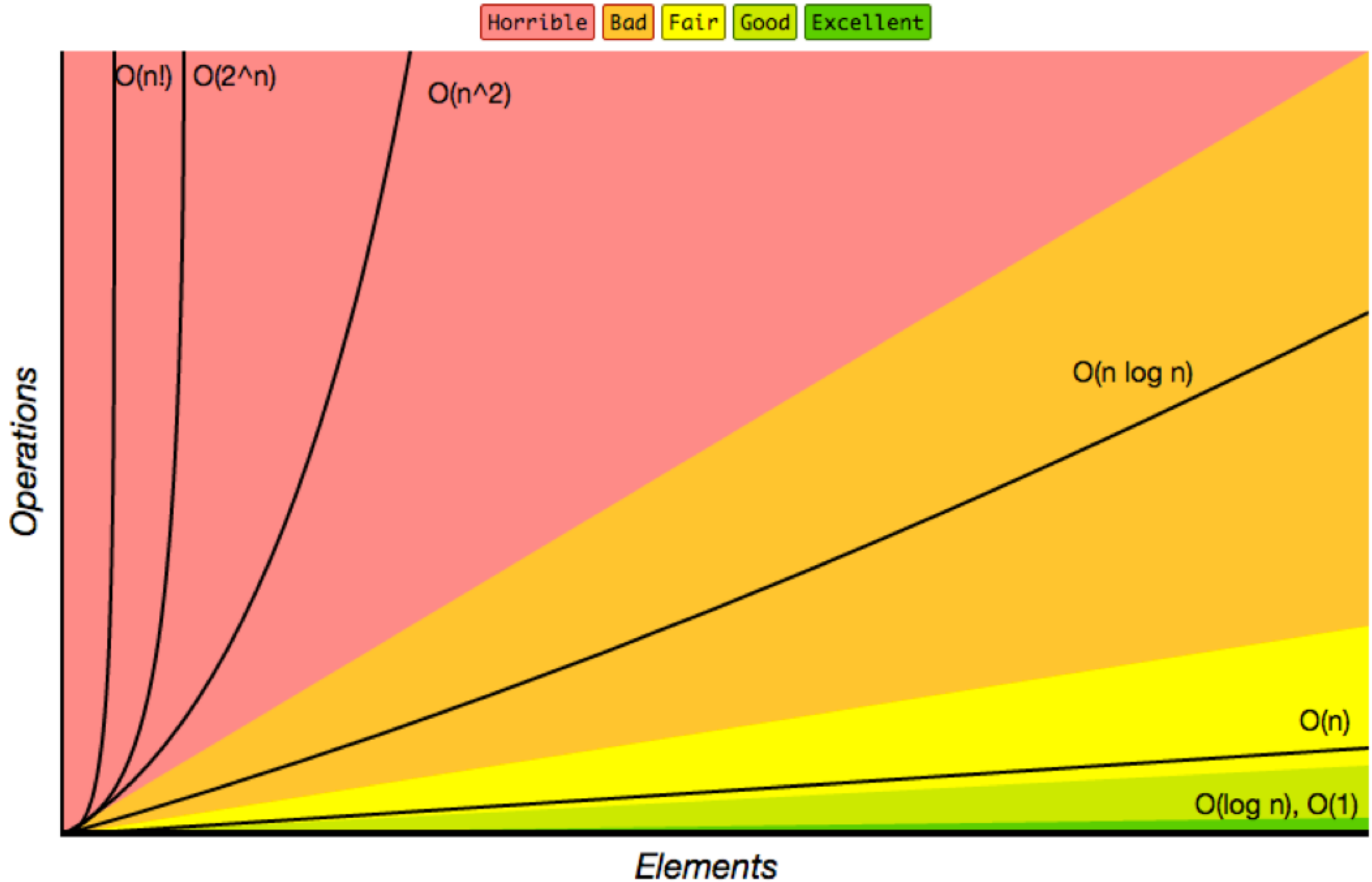
[How long will it take for F(7)?]
[What is the running time?]

Fib vs FastFib

```
$ for i in `seq 1 50`;
  do echo $i; java Buz $i; done
1
Scanning the array of size: 1
The value is: 1
Search took: 3,515 nanoseconds
2
Scanning the array of size: 2
The value is: 2
Search took: 3,849 nanoseconds
3
Scanning the array of size: 3
The value is: 3
Search took: 4,034 nanoseconds
...
47
Scanning the array of size: 47
The value is: 512559680
Search took: 11,723,622,912 nanoseconds
48
Scanning the array of size: 48
The value is: -811192543
Search took: 19,283,637,425 nanoseconds
49
Scanning the array of size: 49
The value is: -298632863
Search took: 33,963,346,264 nanoseconds
50
Scanning the array of size: 50
The value is: -1109825406
Search took: 51,185,363,592 nanoseconds
```

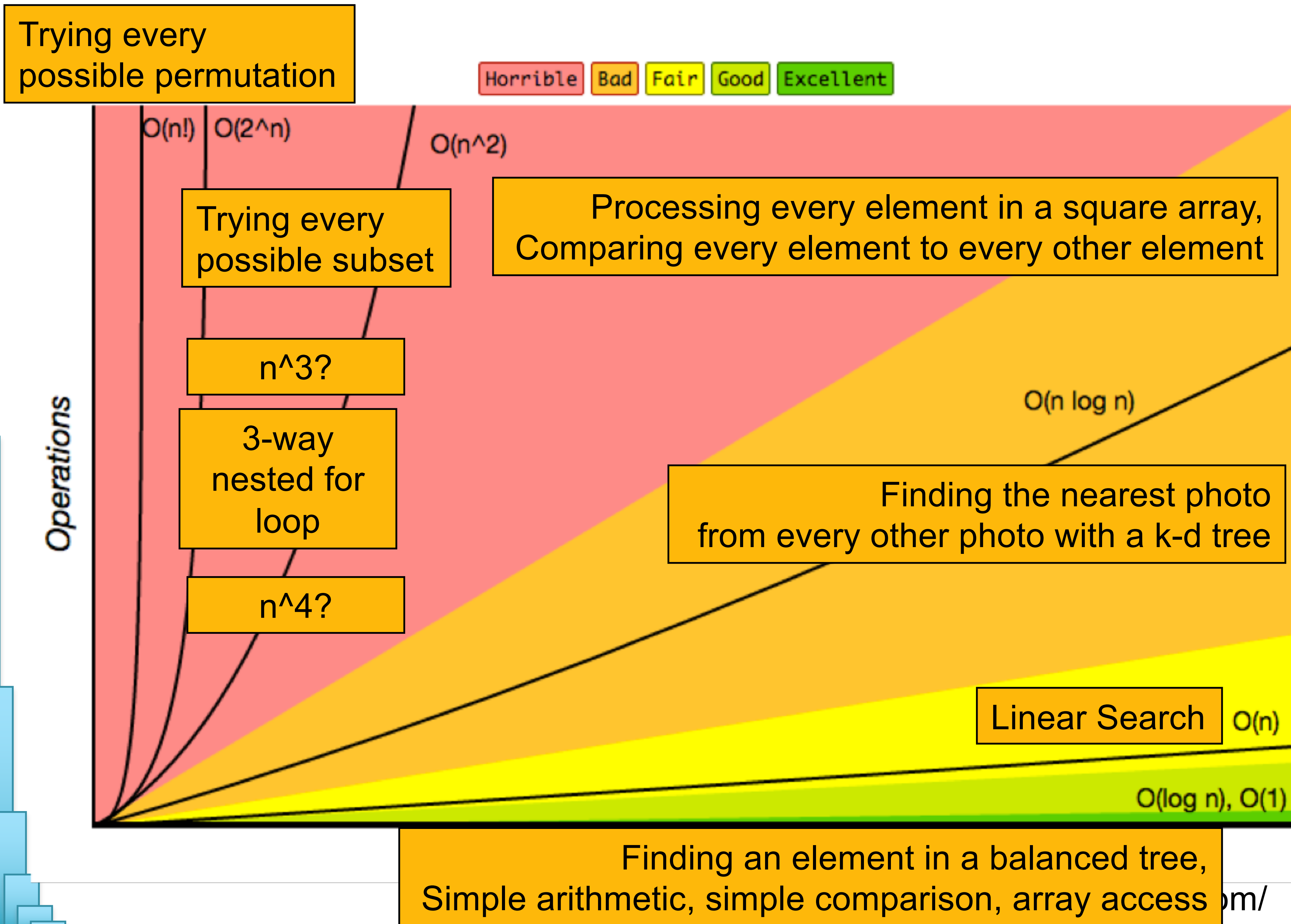
```
$ for i in `seq 1 50`;
  do echo $i; java FastBuz $i; done
1
Scanning the array of size: 1
The value is: 1
Search took: 4,116 nanoseconds
2
Scanning the array of size: 2
The value is: 2
Search took: 4,286 nanoseconds
3
Scanning the array of size: 3
The value is: 3
Search took: 4,600 nanoseconds
...
47
Scanning the array of size: 47
The value is: 512559680
Search took: 9,140 nanoseconds
48
Scanning the array of size: 48
The value is: -811192543
Search took: 10,143 nanoseconds
49
Scanning the array of size: 49
The value is: -298632863
Search took: 9,212 nanoseconds
50
Scanning the array of size: 50
The value is: -1109825406
Search took: 9,662 nanoseconds
```

Growth of functions



http://bigocheatsheet.com/

Growth of functions



Trying every subset

Enumerate every possible subset of N items:

- Encode each subset as a binary vector
  - 0 => not in subset
  - 1 => in the subset

Mike	Peter	Kelly	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

How many distinct subsets are there?

2^n distinct subsets of N items

That doesn't seem too bad, what's 2^{100}

1,267,650,600,228,229,401,496,703,205,376

1.27×10^{30}

Hmm, what's 2^{1000}

1.07×10^{301}

Find the largest subset of 1st year JHU students that <xxx>

Trying every permutation

Enumerate every possible permutation of N items:

- Encode each item as a character
  - Try all possibilities

MPK
MKP
PMK
PKM
KMP
KPM

How many distinct permutations are there?

(N) x (N-1) x (N-2) x (N-3) x … x 3 x 2 x 1

N! => n factorial

That doesn't seem too bad, what's 100!

9.3×10^{157}

Yikes, what's 30!

2.6×10^{32}

Consider every ordering of students in a classroom with 30 students

Trying every permutation

```
public class Permute {
  public static long numtries;

  public static void swap(int [] keys, int x, int y) {
    int temp = keys[x];
    keys[x] = keys[y];
    keys[y] = temp;
  }

  public static void permute(int [] keys, int l, int r) {
     int i;
     if (l == r) {
        if ((numtries < 100) || (numtries % 100000 == 0)) {
          System.out.print("try[" + numtries + "]:");
          for (int x = 0; x < keys.length; x++)
             { System.out.print(" " + keys[x]); }
          System.out.println();
        }
        numtries++;
     } else {
        for (i = l; i <= r; i++) {
          swap(keys,l, i);
          permute(keys, l+1, r);
          swap(keys, l, i);
        }
    }
  }
```

Trying every permutation

```
  public static void main(String [] args) {
    if (args.length == 0) {
      System.out.println("Permute num");
      return;
    }

    int len = Integer.parseInt(args[0]);
    int [] keys = new int[len];
    for (int i = 0; i < len; i++) { keys[i] = i+1; }
    permute(keys, 0, len-1);
    System.err.println("There are " + numtries
                       + " permutations of " + len + " items.");
  }
}
```

```
$ java Permute 1
try[0]: 1
There are 1 permutations of 1 items.
```

```
$ java Permute 2
try[0]: 1 2
try[1]: 2 1
There are 2 permutations of 2 items.
```

```
$ java Permute 3
try[0]: 1 2 3
try[1]: 1 3 2
try[2]: 2 1 3
try[3]: 2 3 1
try[4]: 3 2 1
try[5]: 3 1 2
There are 6 permutations of 3 items.
```

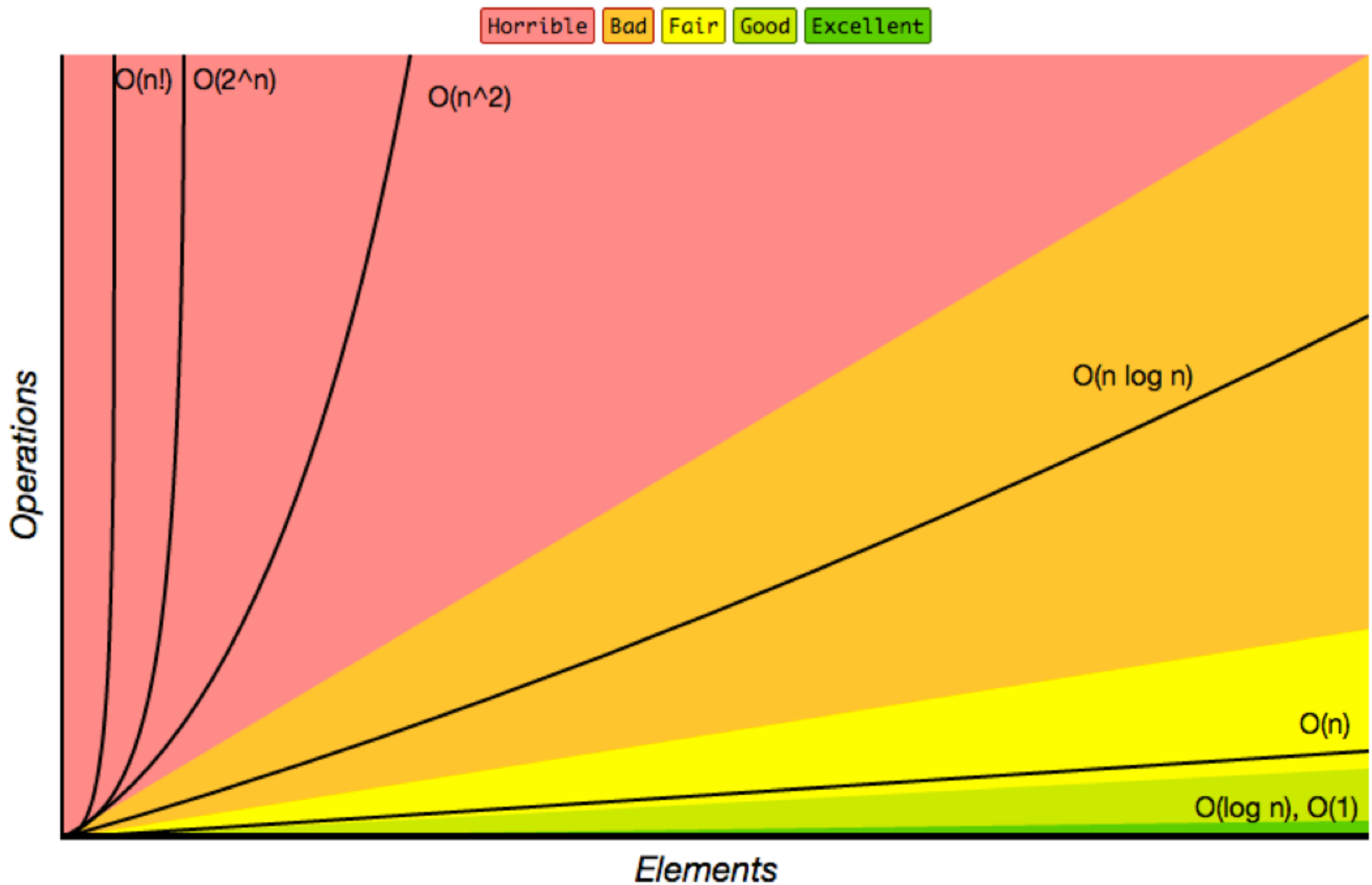
Trying every permutation

```
  public static void main(String [] args) {
    if (args.length == 0) {
      System.out.println("Permute num");
      return;
    }

    int len = Integer.parseInt(args[0]);
    int [] keys = new int[len];
    for (int i = 0; i < len; i++) { keys[i] = i+1; }
    permute(keys, 0, len-1);
    System.err.println("There are " + numtries
                       + " permutations of " + len + " items.");
  }
}
```

```
$ for i in `seq 1 20`;
  do echo $i; java Permute $i > $i.log ; done
1
There are 1 permutations of 1 items.
2
There are 2 permutations of 2 items.
3
There are 6 permutations of 3 items.
4
There are 24 permutations of 4 items.
5
There are 120 permutations of 5 items.
```

Growth of functions



http://bigocheatsheet.com/

Data Structure Complexities

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

http://bigocheatsheet.com/

Agenda

1. ***Review HW1***
2. ***Introduce HW 2***
3. ***Recap on complexity***
4. ***Sorting***

Why Sort?

Sorting is very powerful to organize large amounts of data
Becomes a core routine in many data structures

When data are sorted you can do ***binary search***!
• I'm thinking of a number between 1 and 1,000,000
• ***How many hi/lo guesses will it take to figure it out?***

lg(1,000,000) = 20

How many hi/lo guesses to find my special number?
`26 05 38 28 93 81 71 15 96 33 99 13 58 96 09`

Same Data, Sorted Order
`05 09 13 15 26 28 33 38 58 71 81 93 96 96 99`

Why Sort?

Sorting is very powerful to organize large amounts of data
Becomes a core routine in many data structures

When data are sorted you can do ***binary search***!
- I'm thinking of a number between 1 and 1,000,000
- ***How many hi/lo guesses will it take to figure it out?***

How many hi/lo guesses to find my special number?

`26 05 38 28 93 81 71 15 96 33 99 13 58 96 09`

Same Data, Sorted Order

`05 09 13 15 26 28 33 38 58 71 81 93 96 96 99`

Why Sort?

Sorting is very powerful to organize large amounts of data
Becomes a core routine in many data structures

When data are sorted you can do ***binary search***!
- I'm thinking of a number between 1 and 1,000,000
- ***How many hi/lo guesses will it take to figure it out?***

How many hi/lo guesses to find my special number?
26 05 38 28 93 81 71 15 96 33 99 13 58 96 09

Same Data, Sorted Order
05 09 13 15 26 28 33 38 58 71 81 93 96 96 99

Why Sort?

Sorting is very powerful to organize large amounts of data
Becomes a core routine in many data structures

When data are sorted you can do ***binary search***!

- I'm thinking of a number between 1 and 1,000,000
- ***How many hi/lo guesses will it take to figure it out?***

How many hi/lo guesses to find my special number?

```
26  05  38  28  93  81  71  15  96  33  99  13  58  96  09
```

Same Data, Sorted Order

```
05  09  13  15  26  28  33  38  58  71  81  93  96  96  99
```

Why Sort?

Sorting is very powerful to organize large amounts of data
Becomes a core routine in many data structures

When data are sorted you can do ***binary search***!
- I'm thinking of a number between 1 and 1,000,000
- ***How many hi/lo guesses will it take to figure it out?***

How many hi/lo guesses to find my special number?

`26 05 38 28 93 81 71 15 96 33 99 13 58 96 09`

Same Data, Sorted Order

`05 09 13 15 26 28 33 38 58 71 81 93 96 96 99`

World's worst sorting function

```
public static void permute_sort(int [] keys, int l, int r) {
    Boolean isSorted = checkSorted(keys);
    while (!isSorted) {
        permute_list(keys);
        isSorted = checkSorted(keys)
    }
}
```

Systematically permuting the items will eventually sort them, but will take forever for lists > 30 items

We need a better algorithm!

Sorting

Quickly sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

[How do you do it?]

6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 13, 14, 29, 31, 39, 64, 78, 50, 63, 61, 19
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 50, 64, 78, 63, 61
6, 13, 14, 19, 29, 31, 39, 50, 61, 64, 78, 63
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

To be sorted

Sorted elements

Sorting

Quickly sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

[How do you do it?]

6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 13, 14, 29, 31, 39, 64, 78, 50, 63, 61, 19
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 50, 64, 78, 63, 61
6, 13, 14, 19, 29, 31, 39, 50, 61, 64, 78, 63
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

To be sorted

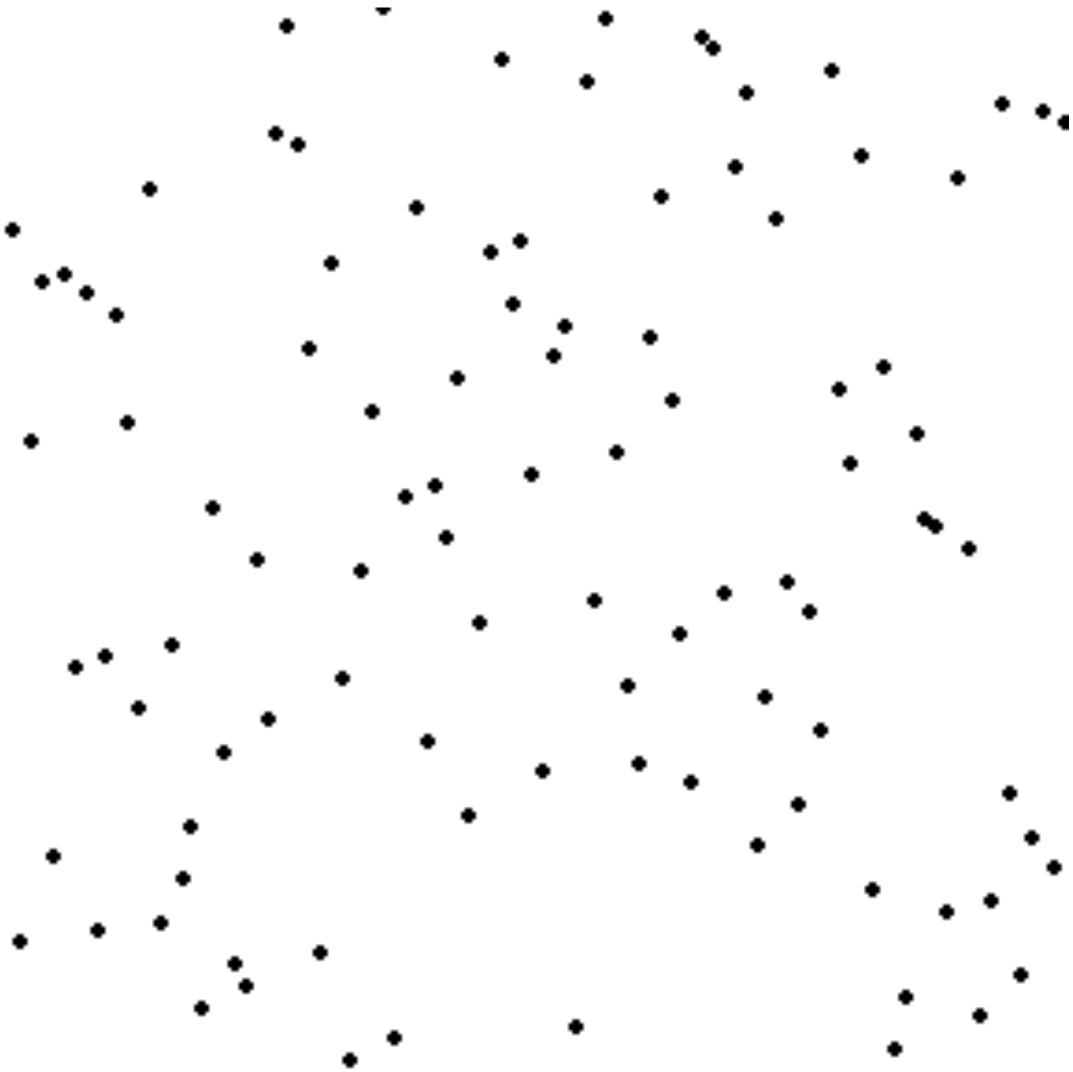
Sorted elements

Here we used your short-term memory to "slide over" the "To be sorted" values to make space for the next smallest

A computer would instead "flip" the next smallest from the "To be sorted" sublist to the end of the "Sorted" sublist

6, 29, 14, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 13, 14, 31, 39, 64, 78, 50, 29, 63, 61, 19
6, 13, 14, 31, 39, 64, 78, 50, 29, 63, 61, 19
6, 13, 14, 19, 39, 64, 78, 50, 29, 63, 61, 31
6, 13, 14, 19, 29, 64, 78, 50, 39, 63, 61, 31
6, 13, 14, 19, 29, 31, 78, 50, 39, 63, 61, 64

Selection Sort



http://en.wikipedia.org/wiki/Selection_sort

Selection Sort Analysis

```
public static void selectionSort(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < a.length; j++) {
            if (a[j] < a[min]) {
                min = j;
            }
        }
        int t = a[i]; a[i] = a[min]; a[min] = t;
    }
}
```

Analysis

- Outer loop: i = 0 to n
- Inner loop: j = i to n
- Total Running time: Outer * Inner = $O(n^2)$

Requires almost no extra space: In-place algorithm

$$T = n + (n-1) + (n-2) + \cdots + 3 + 2 + 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

Selection Sort Analysis

Problem 3: Analysis of Selection Sort (20%)

Your final task for this assignment is to analyze the following selection sort algorithm theoretically (without running it) in detail (without using O-notation). Here's the code, and you must analyze exactly this code (the line numbers are given so you can refer to them in your writeup for this problem):

```
1   public static void selectionSort(int[] a) {
2       for (int i = 0; i < a.length - 1; i++) {
3           int min = i;
4           for (int j = i + 1; j < a.length; j++) {
5               if (a[j] < a[min]) {
6                   min = j;
7               }
8           }
9           int t = a[i]; a[i] = a[min]; a[min] = t;
10      }
11  }
```

The next homework will need a more careful analysis than previous slide ☺

You need to determine exactly how many comparisons C(n) and assignments A(n) are performed by this implementation of selection sort in the worst case. Both of those should be polynomials of degree 2 since you know that the asymptotic complexity of selection sort is O(n^2). (As usual we refer to the size of the problem, which is the length of the array to be sorted here, as "n" above.)

Important: Don't just state the polynomials, your writeup has to explain *how* you derived them! Anyone can google for the answer, but you need to convince us that you actually did the work!

Bubble sort

Sort these values by bubbling up the next largest value

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

[14, 29], 6, 31, 39, 64, 78, 50, 13, 63, 61, 19
[14, 29], 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

14, [29, 6], 31, 39, 64, 78, 50, 13, 63, 61, 19
14, [6, 29], 31, 39, 64, 78, 50, 13, 63, 61, 19

14, 6, [29, 31], 39, 64, 78, 50, 13, 63, 61, 19
14, 6, [29, 31], 39, 64, 78, 50, 13, 63, 61, 19

14, 6, 29, [31, 39], 64, 78, 50, 13, 63, 61, 19
14, 6, 29, [31, 39], 64, 78, 50, 13, 63, 61, 19

14, 6, 29, 31, [39, 64], 78, 50, 13, 63, 61, 19
14, 6, 29, 31, [39, 64], 78, 50, 13, 63, 61, 19

14, 6, 29, 31, 39, [64, 78], 50, 13, 63, 61, 19
14, 6, 29, 31, 39, [64, 78], 50, 13, 63, 61, 19

14, 6, 29, 31, 39, 64, [78, 50], 13, 63, 61, 19
14, 6, 29, 31, 39, 64, [50, 78], 13, 63, 61, 19

14, 6, 29, 31, 39, 64, 50, [78, 13], 63, 61, 19
14, 6, 29, 31, 39, 64, 50, [13, 78], 63, 61, 19

14, 6, 29, 31, 39, 64, 50, 13, [78, 63], 61, 19
14, 6, 29, 31, 39, 64, 50, 13, [63, 78], 61, 19

14, 6, 29, 31, 39, 64, 50, 13, 63, [78, 61], 19
14, 6, 29, 31, 39, 64, 50, 13, 63, [61, 78], 19

14, 6, 29, 31, 39, 64, 50, 13, 63, 61, [78, 19]
14, 6, 29, 31, 39, 64, 50, 13, 63, 61, [19, 78]

14, 6, 29, 31, 39, 64, 50, 13, 63, 61, 19, 78

On the first pass, sweep list
to bubble up the largest element

Bubble sort

Sort these values by bubbling up the next largest value

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

[14, 6], 29, 31, 39, 64, 50, 13, 63, 61, 19, 78
[6, 14], 29, 31, 39, 64, 50, 13, 63, 61, 19, 78

6, [14, 29], 31, 39, 64, 50, 13, 63, 61, 19, 78
6, [14, 29], 31, 39, 64, 50, 13, 63, 61, 19, 78

6, 14, [29, 31], 39, 64, 50, 13, 63, 61, 19, 78
6, 14, [29, 31], 39, 64, 50, 13, 63, 61, 19, 78

6, 14, 29, [31, 39], 64, 50, 13, 63, 61, 19, 78
6, 14, 29, [31, 39], 64, 50, 13, 63, 61, 19, 78

6, 14, 29, 31, [39, 64], 50, 13, 63, 61, 19, 78
6, 14, 29, 31, [39, 64], 50, 13, 63, 61, 19, 78

6, 14, 29, 31, 39, [64, 50], 13, 63, 61, 19, 78
6, 14, 29, 31, 39, [50, 64], 13, 63, 61, 19, 78

6, 14, 29, 31, 39, 50, [64, 13], 63, 61, 19, 78
6, 14, 29, 31, 39, 50, [13, 64], 63, 61, 19, 78

6, 14, 29, 31, 39, 50, 13, [64, 63], 61, 19, 78
6, 14, 29, 31, 39, 50, 13, [63, 64], 61, 19, 78

6, 14, 29, 31, 39, 50, 13, 63, [64, 61], 19, 78
6, 14, 29, 31, 39, 50, 13, 63, [61, 64], 19, 78

6, 14, 29, 31, 39, 50, 13, 63, 61, [64, 19], 78
6, 14, 29, 31, 39, 50, 13, 63, 61, [19, 64], 78

6, 14, 29, 31, 39, 50, 13, 63, 61, 19, 64, 78

On the second pass, sweep list
to bubble up the second largest element

Bubble sort

Sort these values by bubbling up the next largest value

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

[6, 14], 29, 31, 39, 50, 13, 63, 61, 19, 64, 78
6, [14, 29], 31, 39, 50, 13, 63, 61, 19, 64, 78
6, 14, [29, 31], 39, 50, 13, 63, 61, 19, 64, 78
6, 14, 29, [31, 39], 50, 13, 63, 61, 19, 64, 78
6, 14, 29, 31, [39, 50], 13, 63, 61, 19, 64, 78
6, 14, 29, 31, 39, [50, 13], 63, 61, 19, 64, 78
6, 14, 29, 31, 39, [13, 50], 63, 61, 19, 64, 78
6, 14, 29, 31, 39, 13, [50, 63], 61, 19, 64, 78
6, 14, 29, 31, 39, 13, 50, [63, 61], 19, 64, 78
6, 14, 29, 31, 39, 13, 50, [61, 63], 19, 64, 78
6, 14, 29, 31, 39, 13, 50, 61, [63, 19], 64, 78
6, 14, 29, 31, 39, 13, 50, 61, [19, 63], 64, 78

6, 14, 29, 31, 39, 13, 50, 61, 19, 63, 64, 78

On the third pass, sweep list
to bubble up the third largest element

How many passes will we need to do?

O(n)

How much work does each pass take?

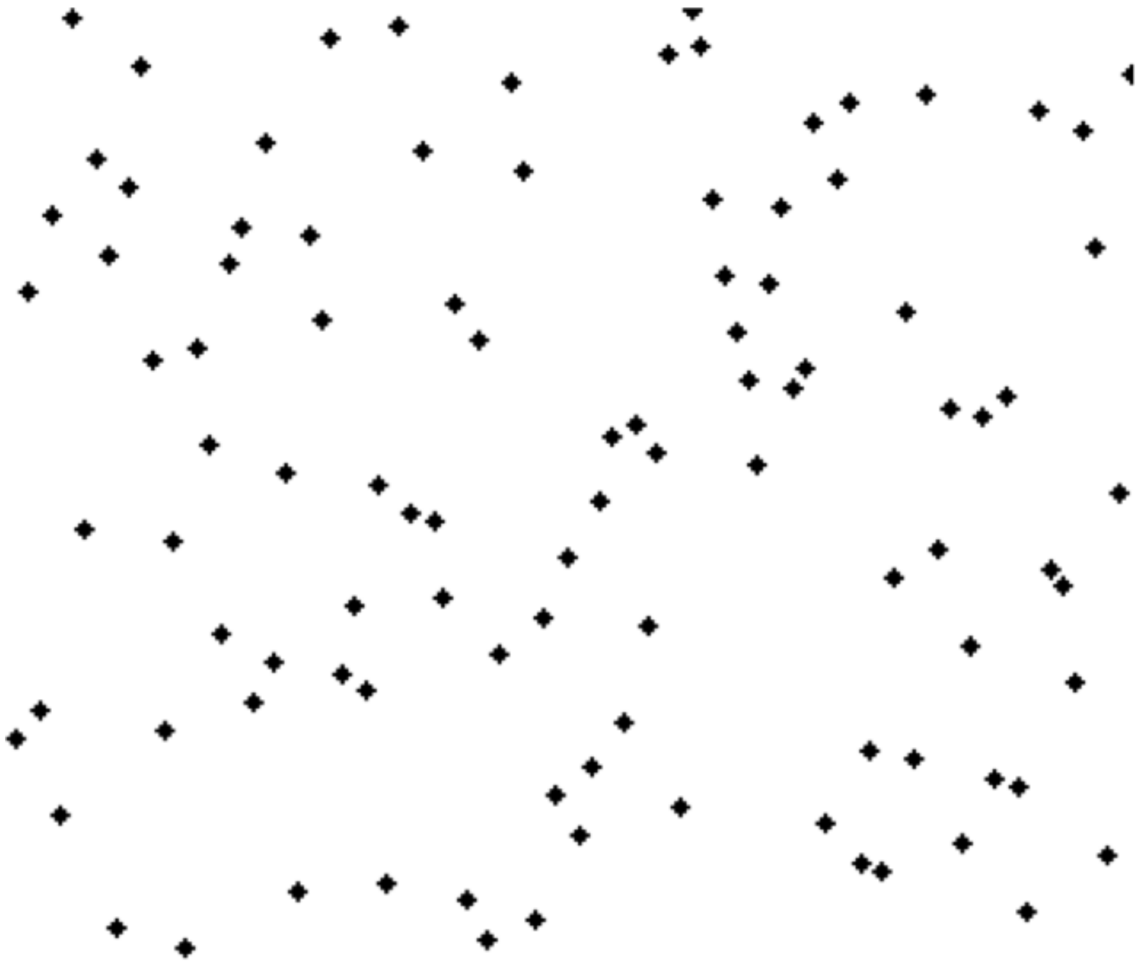
O(n)

What is the total amount of work?

n passes, each requiring O(n) => O(n^2)

Note, you might get lucky and finish much sooner than this

Bubble sort



https://en.wikipedia.org/wiki/Bubble_sort

Insertion Sort

Quickly sort these numbers into ascending order:
14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19
14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 14, 29, 31, 39, 50, 64, 78, 13, 63, 61, 19
6, 13, 14, 29, 31, 39, 50, 64, 78, 63, 61, 19
6, 13, 14, 29, 31, 39, 50, 63, 64, 78, 61, 19
6, 13, 14, 29, 31, 39, 50, 61, 63, 64, 78, 19
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

To be sorted

Sorted elements

Base Case: Declare the first element as a correctly sorted array

Repeat: Iteratively add the next unsorted element to the partially sorted array at the correct position

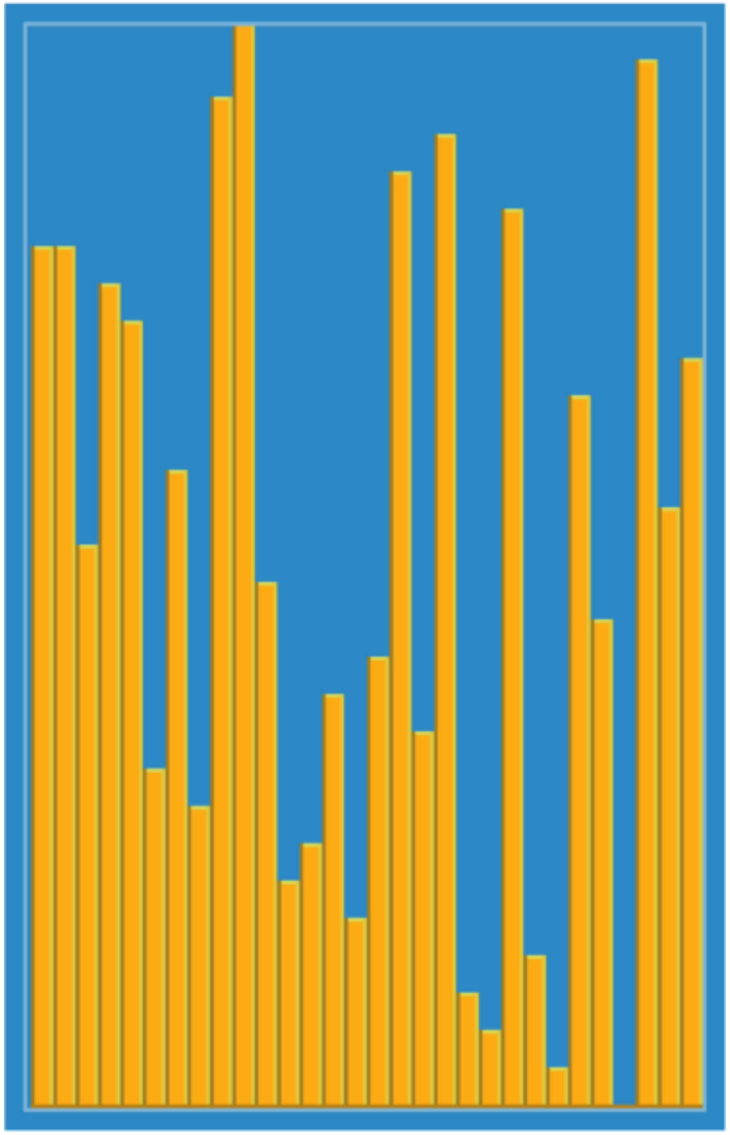
Slide the unsorted element into the correct position:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19
14, 6, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19
6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19

Outer loop: n elements to move into correct position
Inner loop: O(n) work to move element into correct position

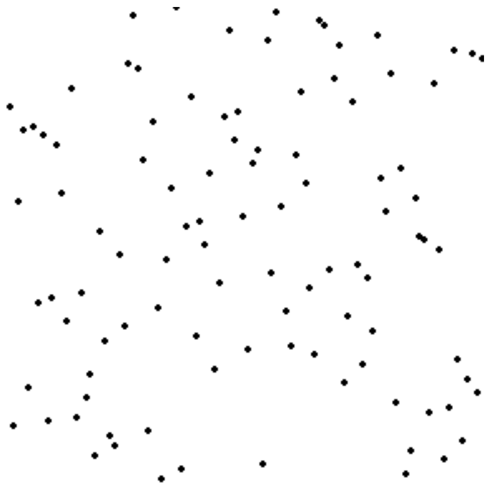
Total Work:
$O(n^2)$

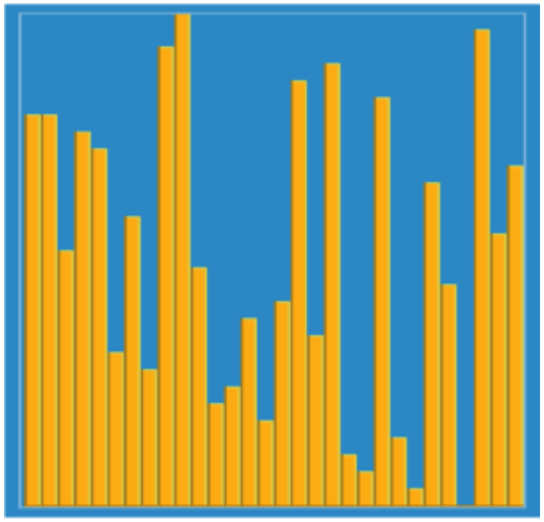
Insertion Sort



https://en.wikipedia.org/wiki/Insertion_sort

Quadratic Sorting Algorithms





Selection Sort
Move next smallest into position

Bubble Sort
Swap up bigger values over smaller

Insertion Sort
Slide next value into correct position

Asymptotically all three have the same performance, but can differ for different types of data. HW 3 will compare them in more detail

Next Steps

1. Work on HW2
2. Check on Piazza for tips & corrections!