CS 600.226: Data Structures

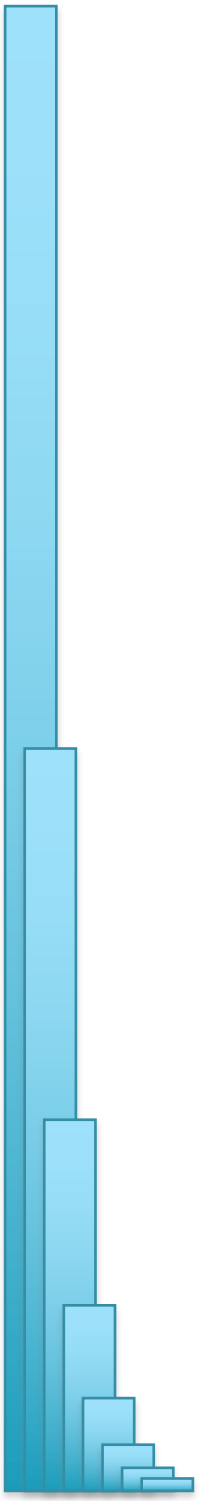
Michael Schatz

Sept 11 2018

Lecture 6. Introduction to Complexity

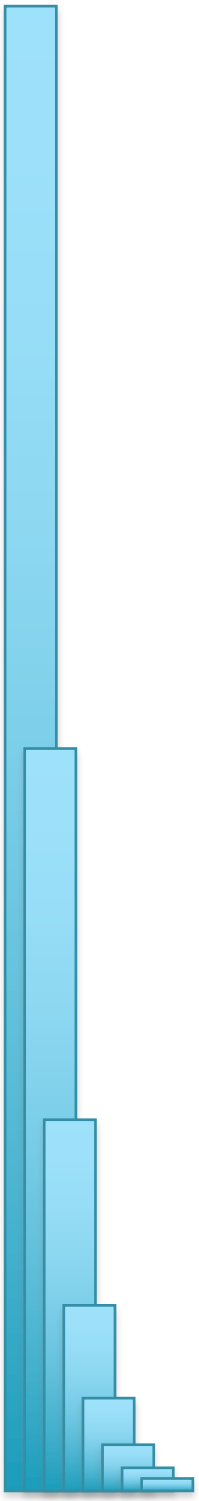
Agenda

1. *Review HW1*
2. *Recap*
3. *Introduction to complexity*



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.

Array ADT

```
adt Array
  uses Any, Integer
  defines Array<T: Any>

  operations
      new: Integer x T ---> Array<T>
      get: Array<T> x Integer ---> T
      put: Array<T> x Integer x T ---> Array<T>
      length: Array<T> ---> Integer

  axioms
      get(new(n, t), i) = t
      get(put(a, i, t), j) = (if i = j then t else get(a, j))
      length(new(n, t)) = n
      length(put(a, i, t)) = length(a)

  preconditions
      new(n, t): 0 < n
      get(a, i): 0 <= i < length(a)
      put(a, i, t): 0 <= i < length(a)
```

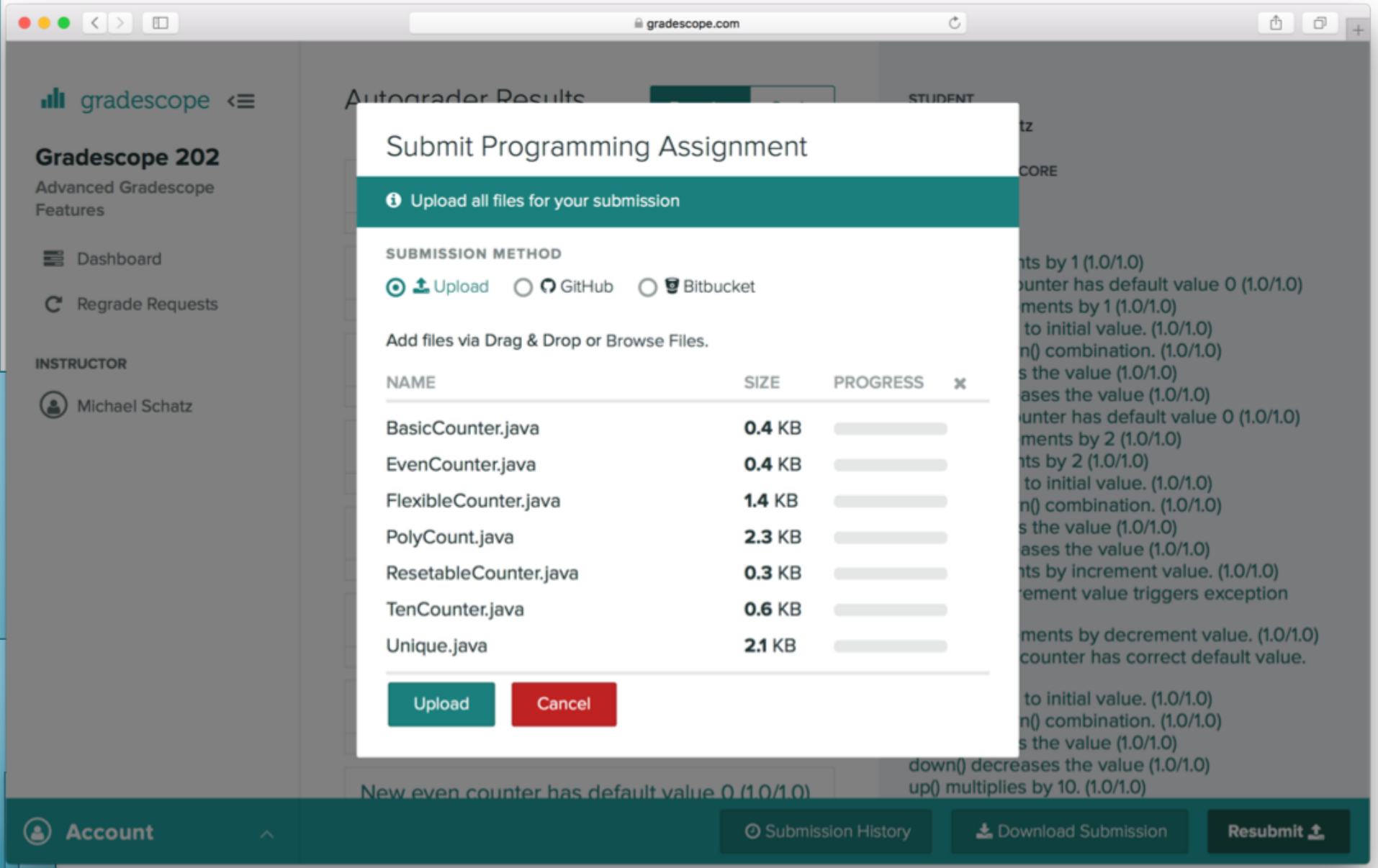
Uses two related ADTs

Defines method signatures

Enforced by asserts

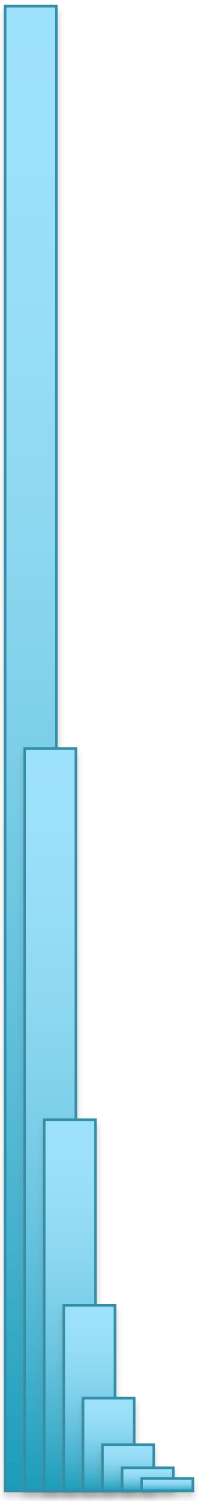
Enforced by exceptions

GradeScope.com
Entry Code: MDJYER

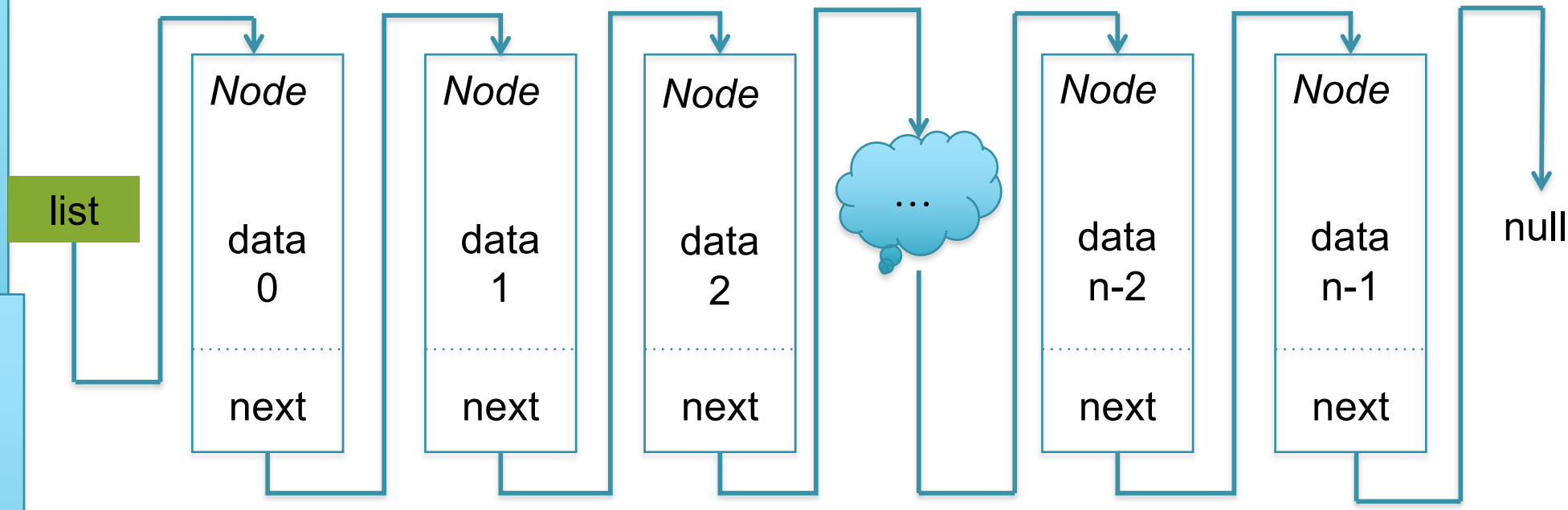


Agenda

1. *Review HW1*
2. *Recap*
3. *Introduction to complexity*



ADT: Linked List
(Singly Linked List)



- Variable length data structure
- Constant time to head of list
- Linear time seek, easy to add/remove to middle once found

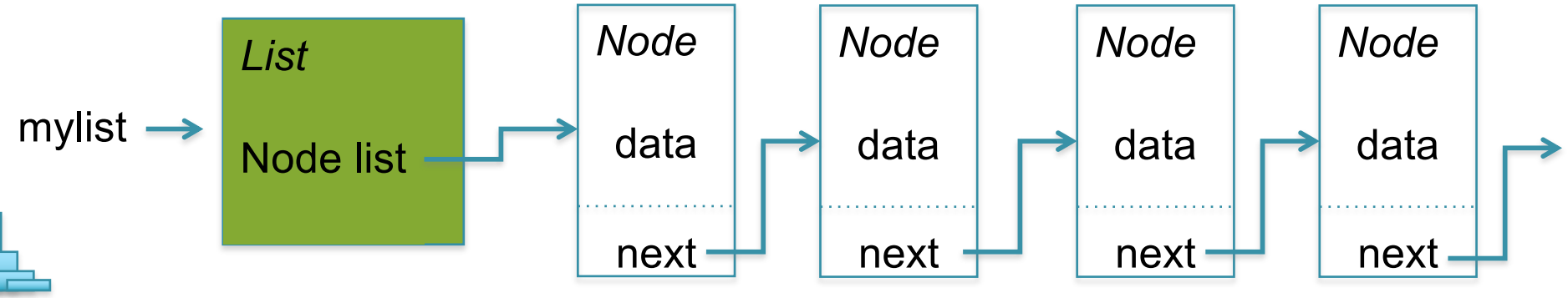
List Class

StringNode.java

```
public class StringNode {
  private String data;
  private StringNode next;

  public StringNode(String d) {
    this.data = d;
    this.next = null; // not necessary
  }

  public void setNext(StringNode n) {
    this.next = n;
  }

  public StringNode getNext() {
    return this.next;
  }

  public String getData() {
    return this.data;
  }
};
```

List.java

```
public class List {
  private StringNode list;

  public List() {
    this.list = null; // not necessary
  }

  public static void main(String[] args) {
    List mylist = new List();
  }
};
```



Searching

StringNode.java

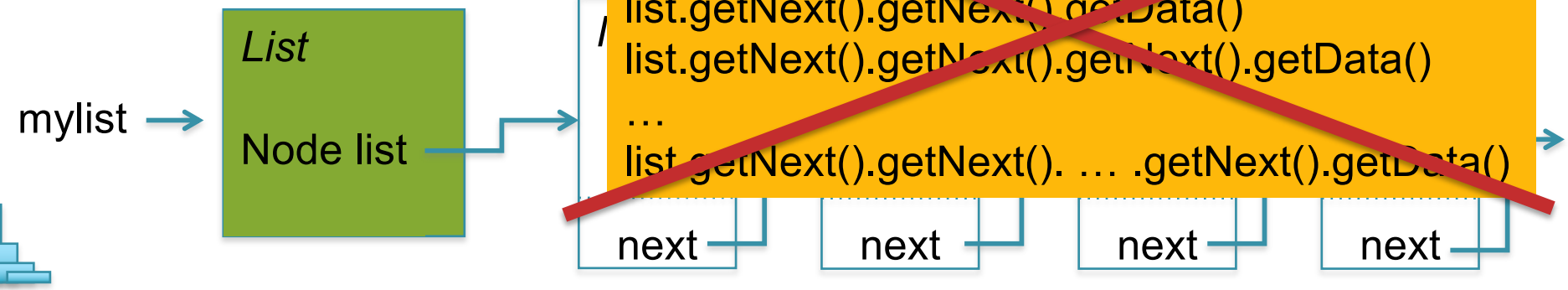
```
public class StringNode {
  private String data;
  private StringNode next;

  public StringNode(String d) {
    this.data = d;
    this.next = null; // not necessary
  }

  public void setNext(StringNode n) {
    this.next = n;
  }

  public StringNode getNext() {
    return this.next;
  }

  public String getData() {
    return this.data;
  }
};
```

List.java

```
public class List {
  private StringNode list;

  public List() {
    this.list = null; // not necessary
  }

  public static void main(String[] args) {
    List mylist = new List();
  }
};
```

If you have a linked list of items (photos in Instagram), how would you search?

~~list.getData()~~
~~list.getNext().getData()~~
~~list.getNext().getNext().getData()~~
~~list.getNext().getNext().getNext().getData()~~
~~…~~
~~list.getNext().getNext(). … .getNext().getData()~~



Linked List Searching

List.java

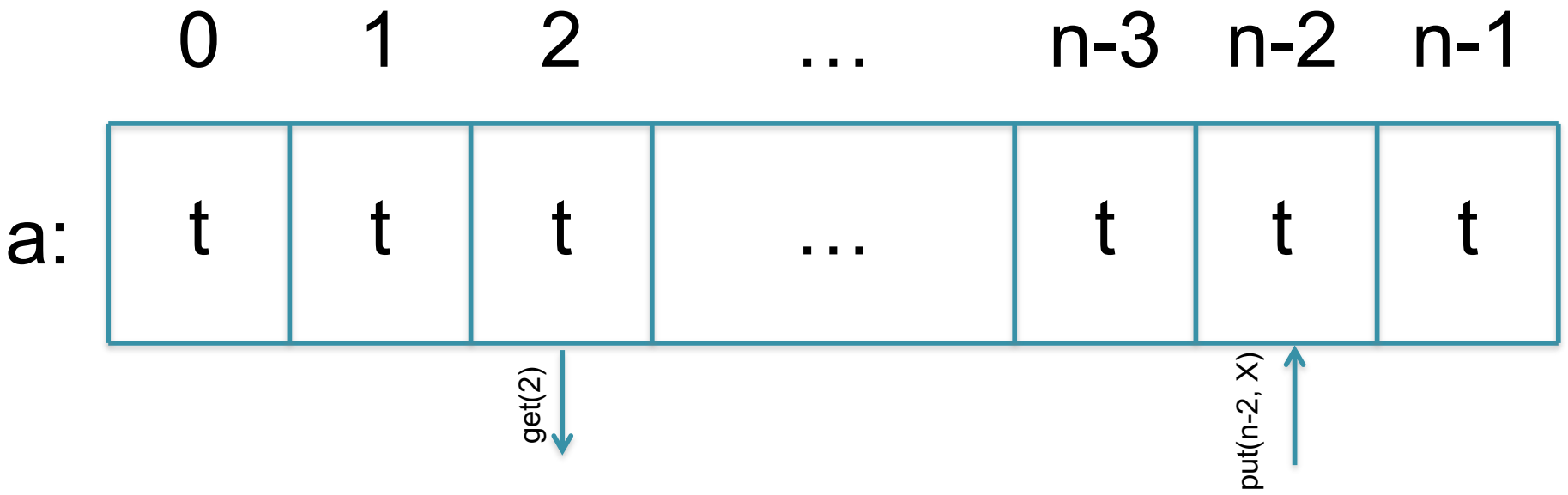
```
public StringNode find(String value) {
  StringNode cur = list;
  while (cur != null) {
    System.out.println("checking: " + value + " at " + cur.getData());
    if (cur.getData().equals(value)) {
      return cur;
    }

    cur = cur.getNext();
  }

  return cur; // must be null
}
```

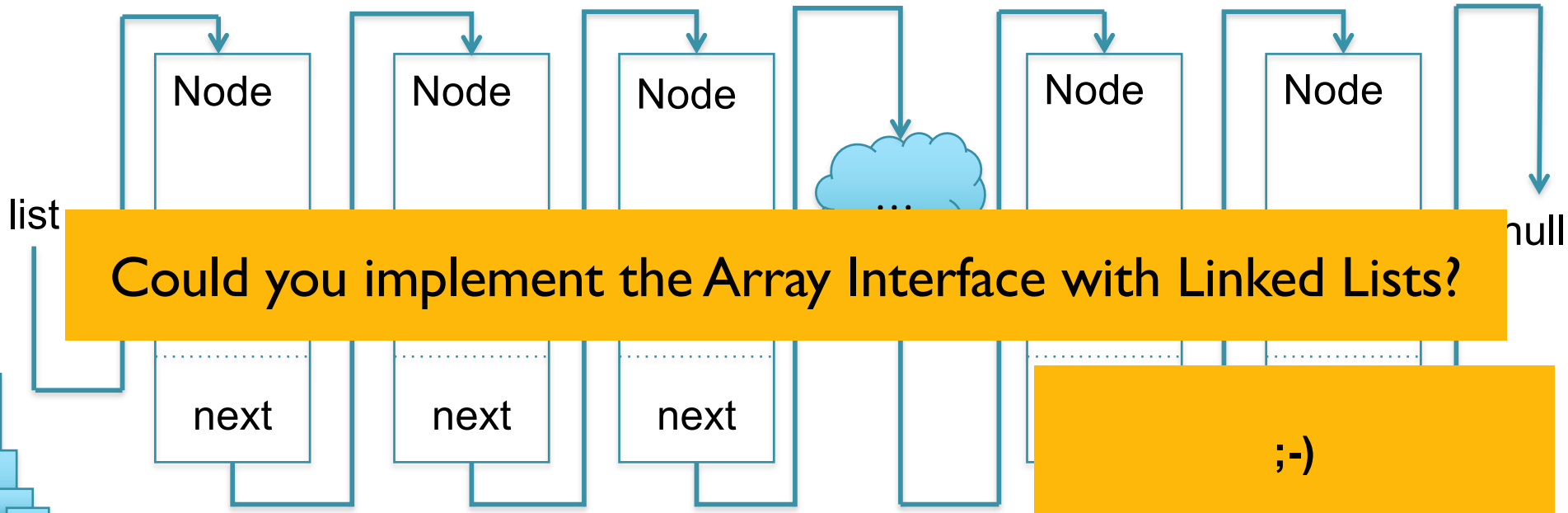
```
List mylist = new List();
mylist.add("Mike");
mylist.add("Peter");
mylist.add("Kelly");
```

```
StringNode f1 = mylist.find("Mike");
System.out.println("f1 finished");
System.out.println(f1.getData());
```

```
checking: Mike at Kelly
checking: Mike at Peter
checking: Mike at Mike
f1 finished
Mike
```

Arrays versus Linked Lists

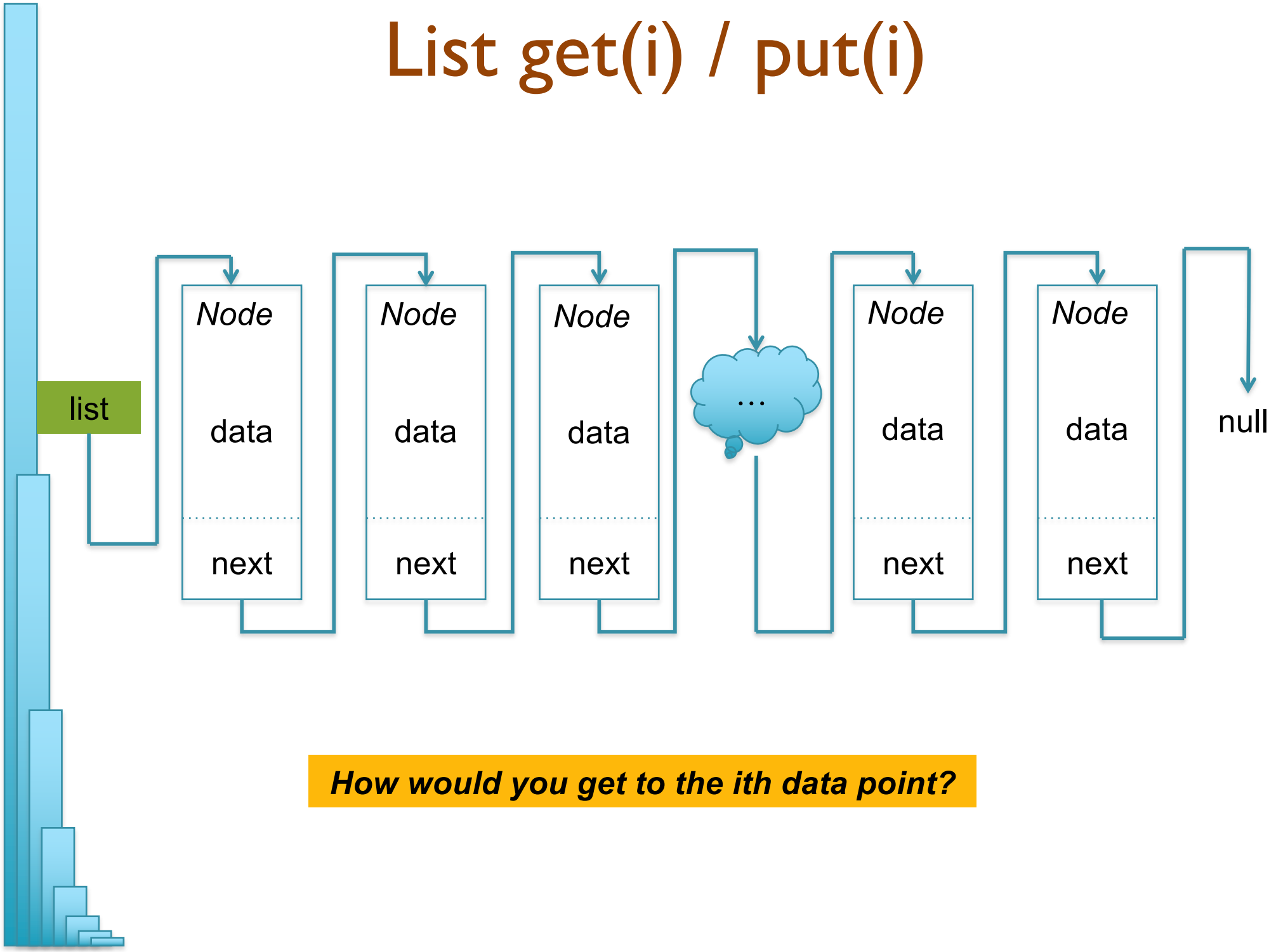




Could you implement the Array Interface with Linked Lists?

;-)

List get(i) / put(i)



How would you get to the ith data point?

Java Iterators

http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html

java.util

Interface Iterator<E>

All Known Subinterfaces:
ListIterator<E>, XMLEventReader

All Known Implementing Classes:
BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

```
public interface Iterator<E>
```

An iterator over a collection. Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the Java Collections Framework.

Since:
1.2

See Also:
`Collection`, `ListIterator`, `Enumeration`

Method Summary	
boolean	**hasNext()** Returns `true` if the iteration has more elements.
E	**next()** Returns the next element in the iteration.
void	**remove()** Removes from the underlying collection the last element returned by the iterator (optional operation).

Java Iterators

http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html

java.util

Interface Iterator<E>

All Known Subinterfaces:

ListIterator<E>, XMLEventReader

All Known Implementing Classes:

BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

```
for (MyType obj : list) {
    System.out.print(obj);
}
```

ffer from

fined

Since:

1.2

See Also:

Collection, ListIterator, Enumeration

Method Summary	
boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

Nested and Inner Classes

Nested classes are divided into two categories: static and non-static.

- Nested classes that are declared static are called ***static nested classes***.
- Non-static nested classes are called ***inner classes.***

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

A static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference.

An inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.

ListArray (1)

ListArray.java

```
/**
    Array implementation using a linked list.
    @param <T> Element type.
*/
public class ListArray<T> implements Array<T> {
    // A nested Node<T> class to build our linked list out of. We use a
    // static nested class (instead of an inner class) here since we don't need
    // access to the ListArray object we're part of.
    private static class Node<T> {
        T data;
        Node<T> next;
    }

    // The not-so-obvious representation of our abstract Array: A linked
    // list with "length" nodes instead of an array of "length" slots.
    private Node<T> list;
    private int length;
```

Why length?

Static Nested Class

Private static class embedded inside a parent class

One less file to checkstyle ☺

No leakage of implementation details, also doesn't need getters & setters

ListArray (2)

```
/**
     Constructs a new ListArray.

     @param n Length of array, must be n &gt; 0.
     @param t Default value to store in each slot.
     @throws LengthException if n &le; 0.
 */
public ListArray(int n, T t) throws LengthException {
     if (n <= 0) {
         throw new LengthException();
     }

     // Initialize all positions as we promise in the specification.
     // Unlike in SimpleArray we cannot avoid the initialization even
     // if t == null since the nodes still have to be created. On the
     // upside we don't need a cast anywhere.
     for (int i = 0; i < n; i++) {
         this.prepend(t);
     }

     // Remember the length!
     this.length = n;
 }

 // Insert a node at the beginning of the linked list.
 private void prepend(T t) {
     Node<T> n = new Node<T>();
     n.data = t;
     n.next = this.list;
     this.list = n;
 }
```

Constructor
Make sure to follow the interface by initializing N empty nodes!

prepend()
Not in interface, but needed for implementation

ListArray (3)

```
    // Find the node for a given index. Assumes valid index.
    private Node<T> find(int index) {
        Node<T> n = this.list;
        int i = 0;
        while (n != null && i < index) {
            n = n.next;
            i = i + 1;
        }
        return n;
    }

    @Override
    public T get(int i) throws IndexException {
        if (i < 0 || i >= this.length) {
            throw new IndexException();
        }
        Node<T> n = this.find(i);
        return n.data;
    }

    @Override
    public void put(int i, T t) throws IndexException {
        if (i < 0 || i >= this.length) {
            throw new IndexException();
        }
        Node<T> n = this.find(i);
        n.data = t;
    }

    @Override
    public int length() {
        return this.length;

    }
}
```

find()
Get the i^{th} value in list
Be careful to not walk off the end of the list

get()
From the Array Interface
Return what you find()

put()
From the Array Interface
Update what you find()

length()
Why is this needed?

ListArray Iterator

Array.java

```
public interface Array<T> extends Iterable<T> {
```

ListArray.java

```
import java.util.Iterator;

    // An iterator to traverse the array from front to back.
    private class ArrayIterator implements Iterator<T> {
        // Current position in the linked list.
        Node<T> current;

        ArrayIterator() {
            this.current = ListArray.this.list;
        }

        public T next() {
            T t = this.current.data;
            this.current = this.current.next;
            return t;
        }

        public boolean hasNext() {
            return this.current != null;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    public Iterator<T> iterator() {
        return new ArrayIterator();
    }
```

Extend Iterable<T>

Inner class

Maintains a reference to something in the parent class

Simplifies use by client: a regular nested class would need a reference to list for every method

ListArray Iterator

Array.java

```
public interface Array<T> extends Iterable<T> {
```

ListArray.java

```
import java.util.Iterator;

    // An iterator to traverse the array from front to back.
    private class ArrayIterator implements Iterator<T> {
        // Current position in the linked list.
        Node<T> current;

        ArrayIterator() {
            this.current = ListArray.this.list;
        }

        public T next() {
            T t = this.current.data;
            this.current = this.current.next;
            return t;
        }

        public boolean hasNext() {
            return this.current != null;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    public Iterator<T> iterator() {
        return new ArrayIterator();
    }
```

Extend Iterable<T>

Implement Iterator<T>

Why current?

ListArray.this.what???

What happens if you call next() without checking hasNext()?

Required by interface

Lets the constructor access "this" list

Printing the list

```
public static void main(String[] args) {
  ListArray<String> mylist = new ListArray(5, "Mike");

  System.out.println("Created array of length: " + mylist.length());
  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }

  mylist.put(3, "Peter");

  System.out.println("Printing Array after setting mylist[3]=Peter");

  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }

   System.out.println("Printing Array with Iterator");
   Iterator<String> it = mylist.iterator();
   while (it.hasNext()) {
     String val = it.next();
     System.out.println(val);
   }
}
```

Notice: the iterator must be generated from an existing List object (non-static method). Trying to call List.iterator() would cause an error

Printing the list

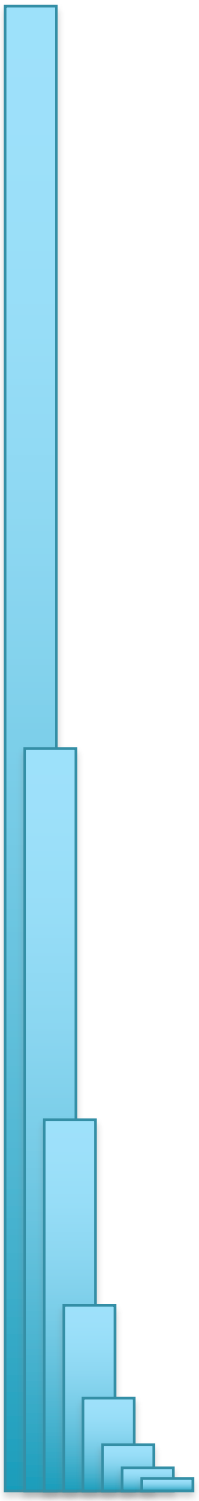
```
        ...

        System.out.println("Printing Array after setting mylist[3]=Peter");

        for (int i = 0; i < mylist.length(); i++) {
            String val = mylist.get(i);
            System.out.println("mylist[" + i + "]: " + val);
        }

        System.out.println("Printing Array with Iterator");
        Iterator<String> it = mylist.iterator();
        while (it.hasNext()) {
           String val = it.next();
           System.out.println(val);
        }

        System.out.println("Printing Array with foreach");
        for (String val : mylist) {
            System.out.println(val);
        }
    }
```

Is this better than explicitly using the iterator?

…
Printing Array with foreach
Mike
Mike
Mike
Peter
Mike

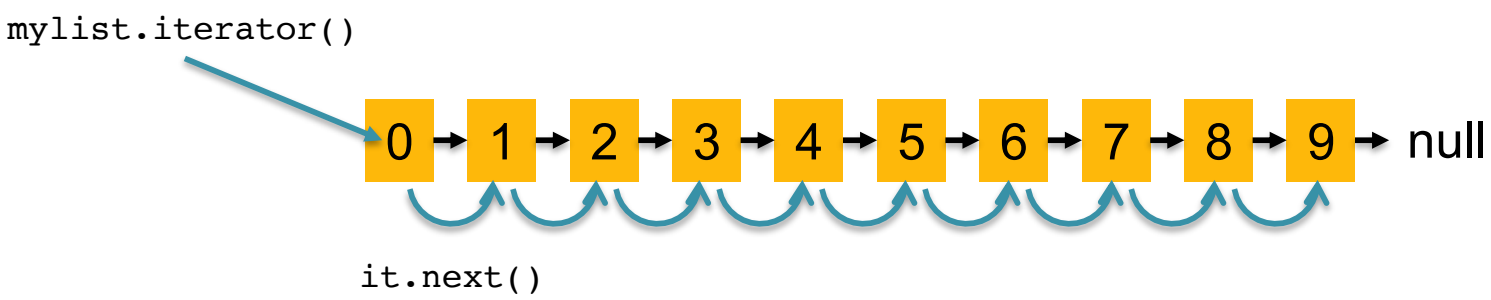
Agenda

1. *Review HW1*
2. *Recap*
3. *Introduction to complexity*



Iterator vs get()

```
Iterator<String> it = mylist.iterator();
while (it.hasNext()) {
  String val = it.next();
  System.out.println(val);
}
```



Total number of “hops”: 10

If the list had 100 elements, how many hops would it need?

100

If the list had 1000 elements, how many hops would it need?

1000

The number of hops scales exactly with the number of elements:
We call this a linear runtime

Iterator vs get()

```
for (int i = 0; i < mylist.length(); i++) {
  String val = mylist.get(i);
  System.out.println("mylist[" + i + "]: " + val);
}
```

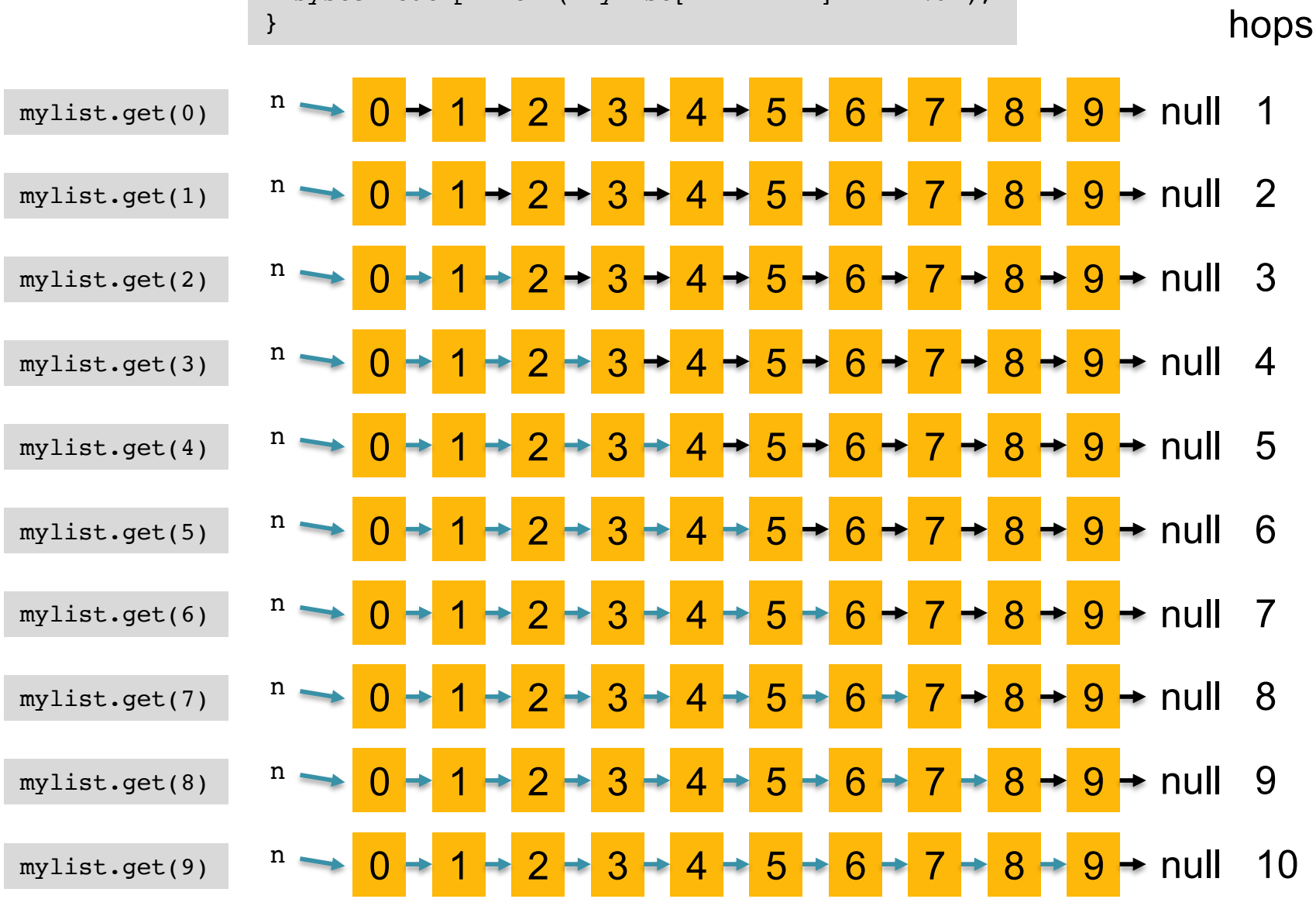

```
mylist.get(0)
mylist.get(1)
mylist.get(2)
mylist.get(3)
mylist.get(4)
mylist.get(5)
mylist.get(6)
mylist.get(7)
mylist.get(8)
mylist.get(9)
```

```
// Find the node for a given index. Assumes valid index.
private Node<T> find(int index) {
    Node<T> n = this.list;
    int i = 0;
    while (n != null && i < index) {
        n = n.next;
        i = i + 1;
    }
    return n;
}

@Override
public T get(int i) throws IndexException {
    if (i < 0 || i >= this.length) {
        throw new IndexException();
    }
    Node<T> n = this.find(i);
    return n.data;
}
```

Notice get(i) requires walking i steps

Iterator vs get()

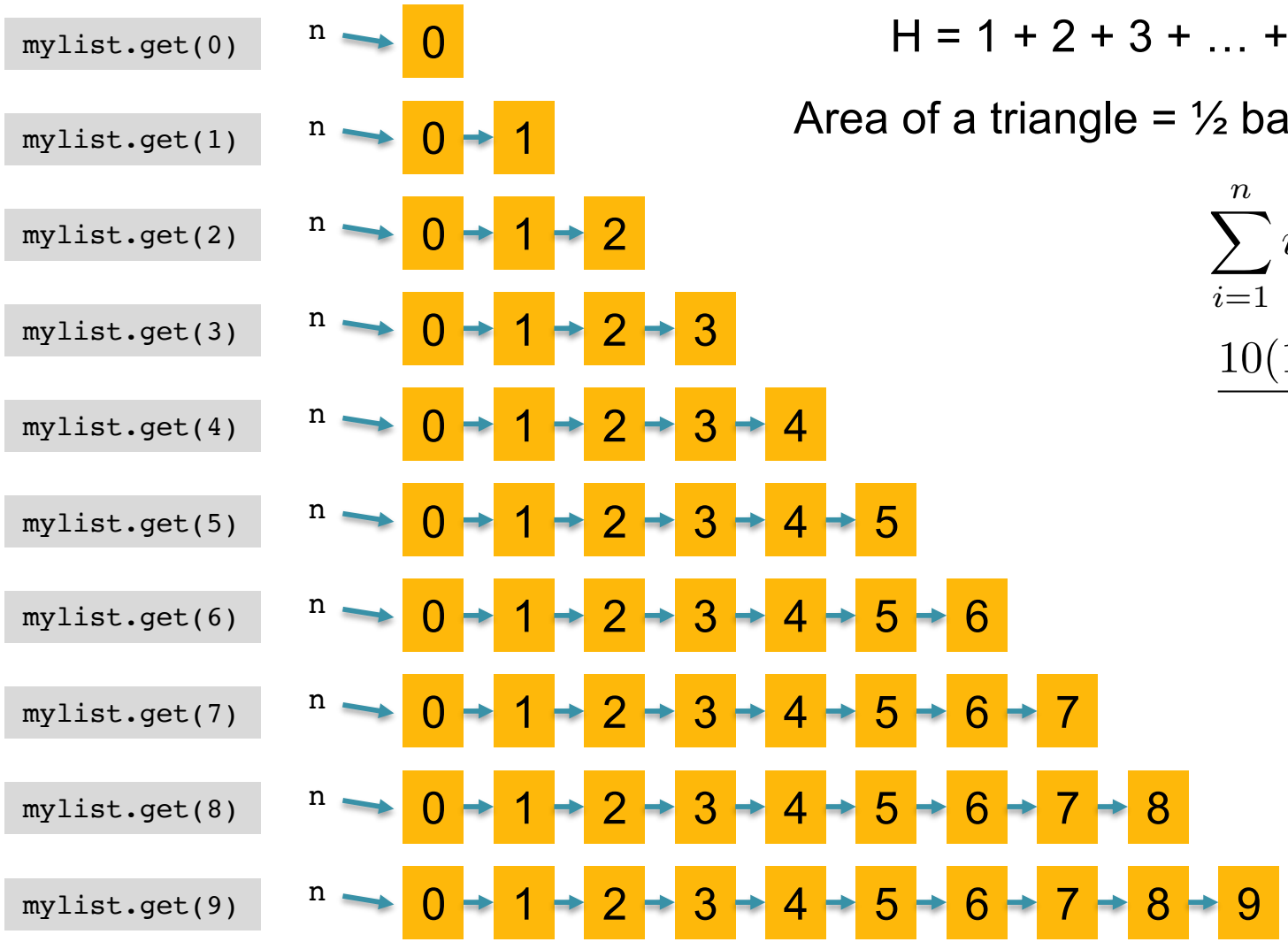
```
for (int i = 0; i < mylist.length(); i++) {
  String val = mylist.get(i);
  System.out.println("mylist[" + i + "]: " + val);
}
```



How many total hops do we make?

Iterator vs get()

```
for (int i = 0; i < mylist.length(); i++) {
  String val = mylist.get(i);
  System.out.println("mylist[" + i + "]: " + val);
}
```



H = 1 + 2 + 3 + … + 8 + 9 + 10

Area of a triangle = ½ base * height

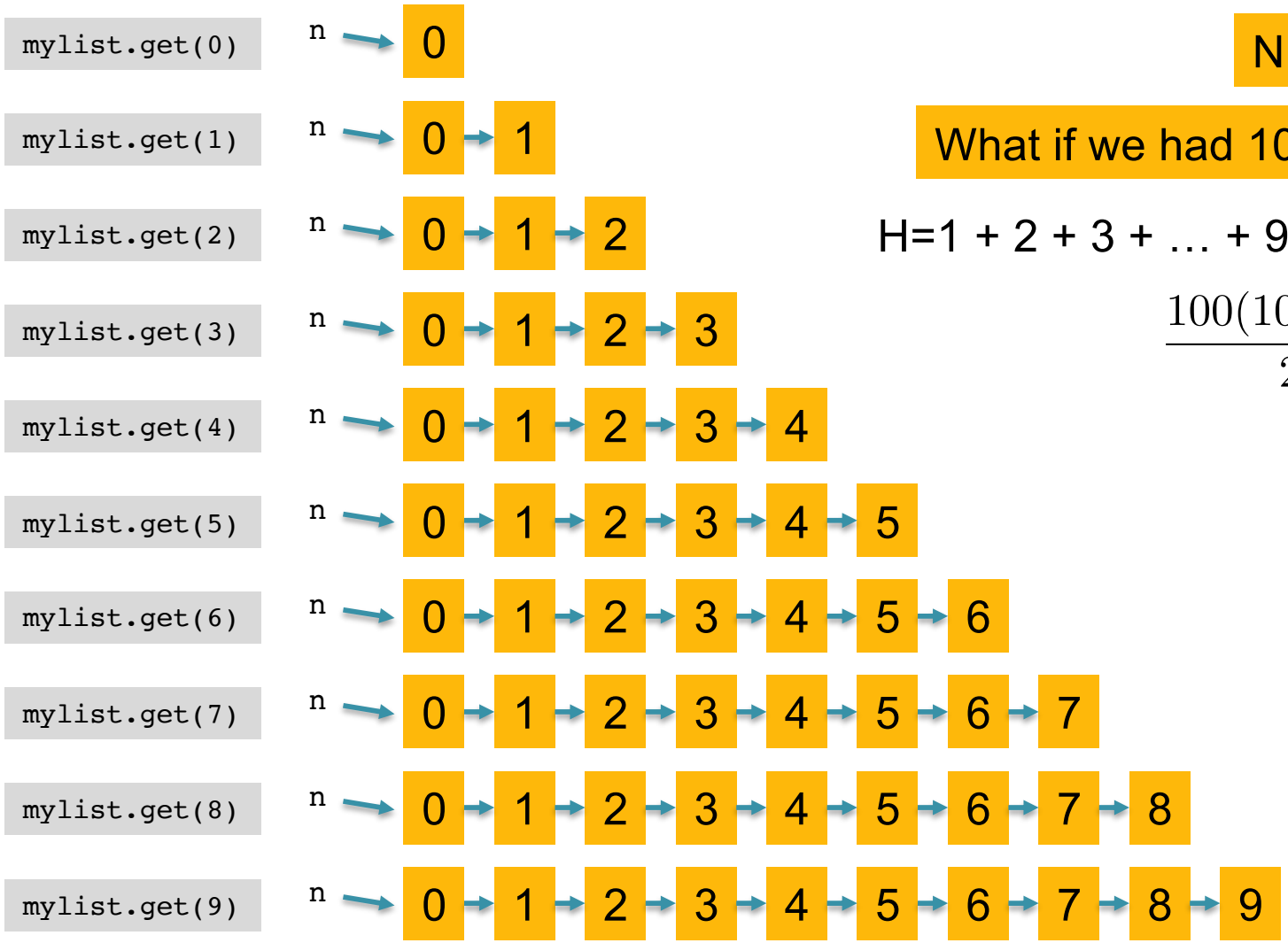
$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\frac{10(10+1)}{2} = 55$$

How many total hops do we make?

Iterator vs get()

```
for (int i = 0; i < mylist.length(); i++) {
  String val = mylist.get(i);
  System.out.println("mylist[" + i + "]: " + val);
}
```



N = 10; H = 55

What if we had 100 elements?

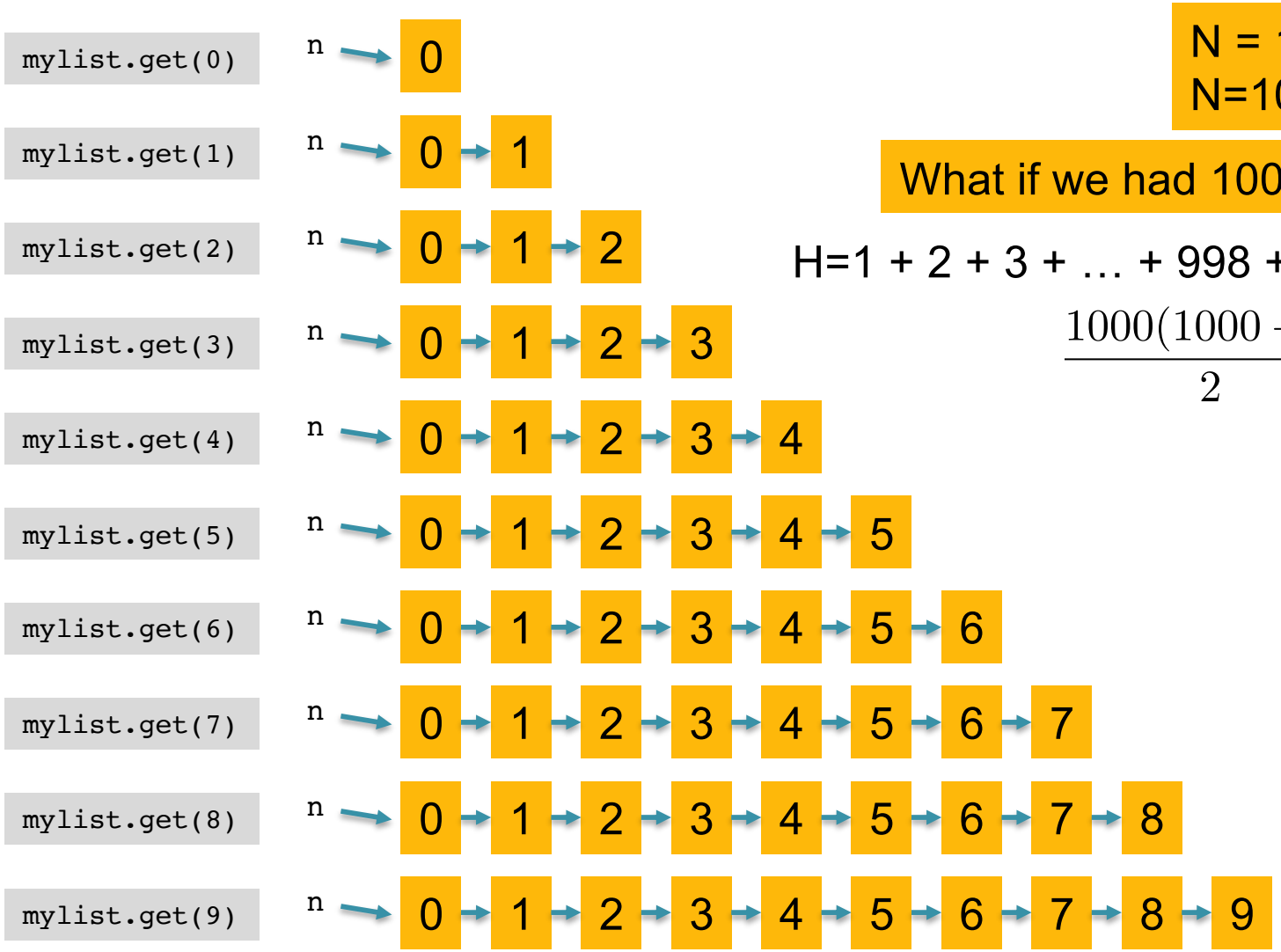
H=1 + 2 + 3 + … + 98 + 99 + 100

$$\frac{100(100+1)}{2} = 5050$$

How many total hops do we make?

Iterator vs get()

```
for (int i = 0; i < mylist.length(); i++) {
  String val = mylist.get(i);
  System.out.println("mylist[" + i + "]: " + val);
}
```



N = 10; H = 55
N=100; H = 5050

What if we had 1000 elements?

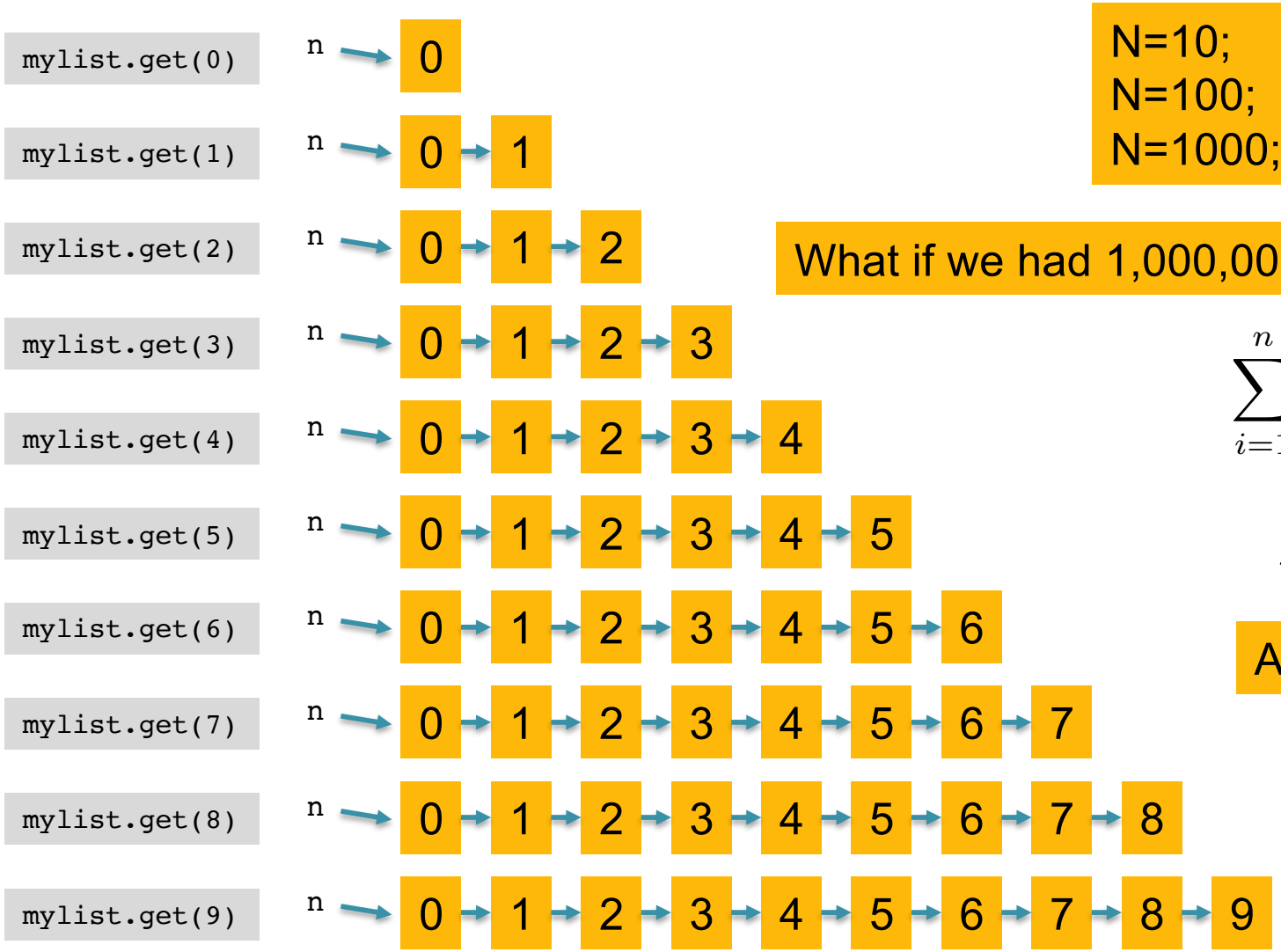
H=1 + 2 + 3 + … + 998 + 999 + 1000

$$\frac{1000(1000+1)}{2} = 500500$$

How many total hops do we make?

Iterator vs get()

```
for (int i = 0; i < mylist.length(); i++) {
  String val = mylist.get(i);
  System.out.println("mylist[" + i + "]: " + val);
}
```



N=10; H = 55
N=100; H = 5050
N=1000; H = 500500

What if we had 1,000,000 elements?

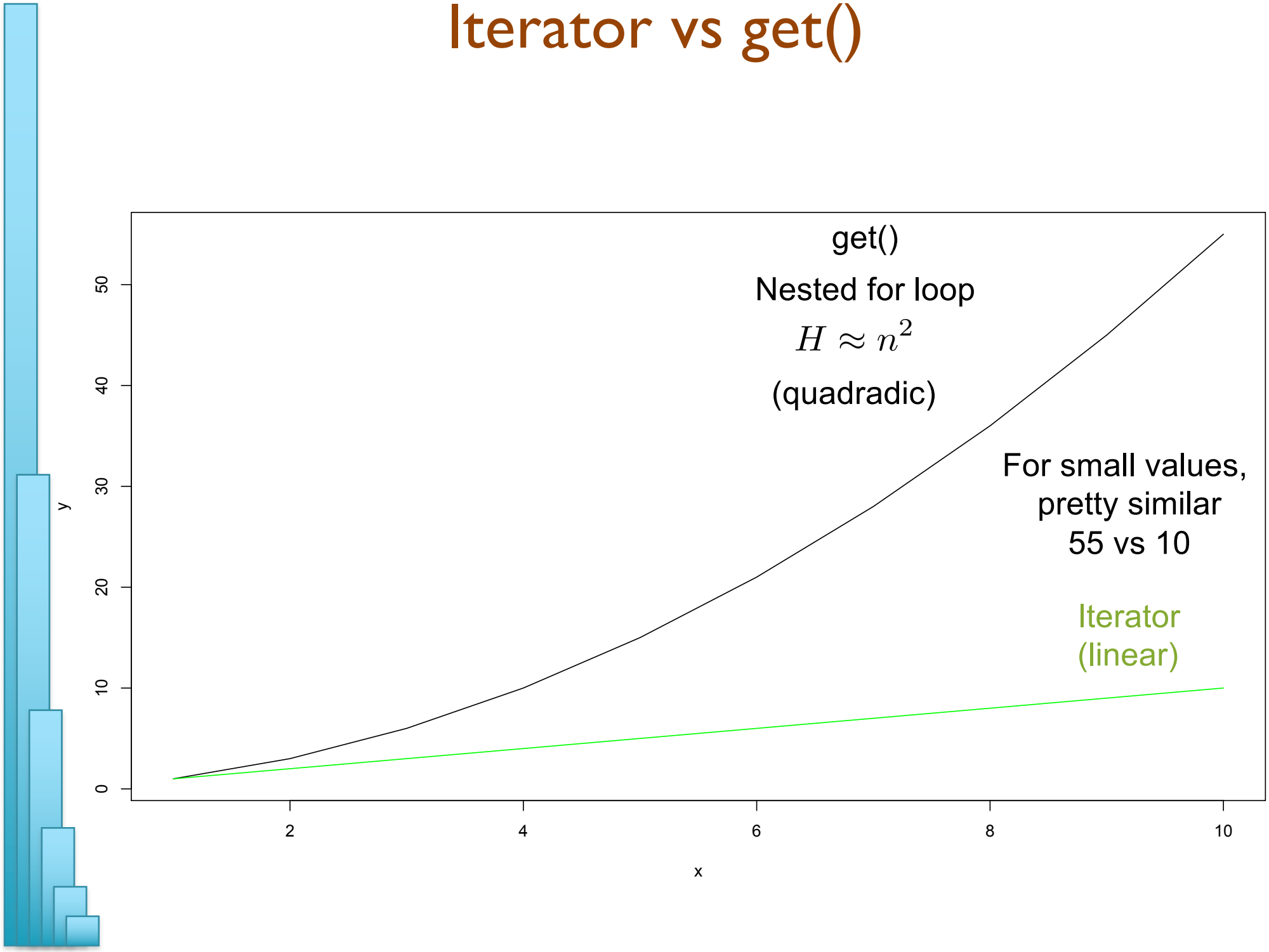
$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$H \approx n^2$$

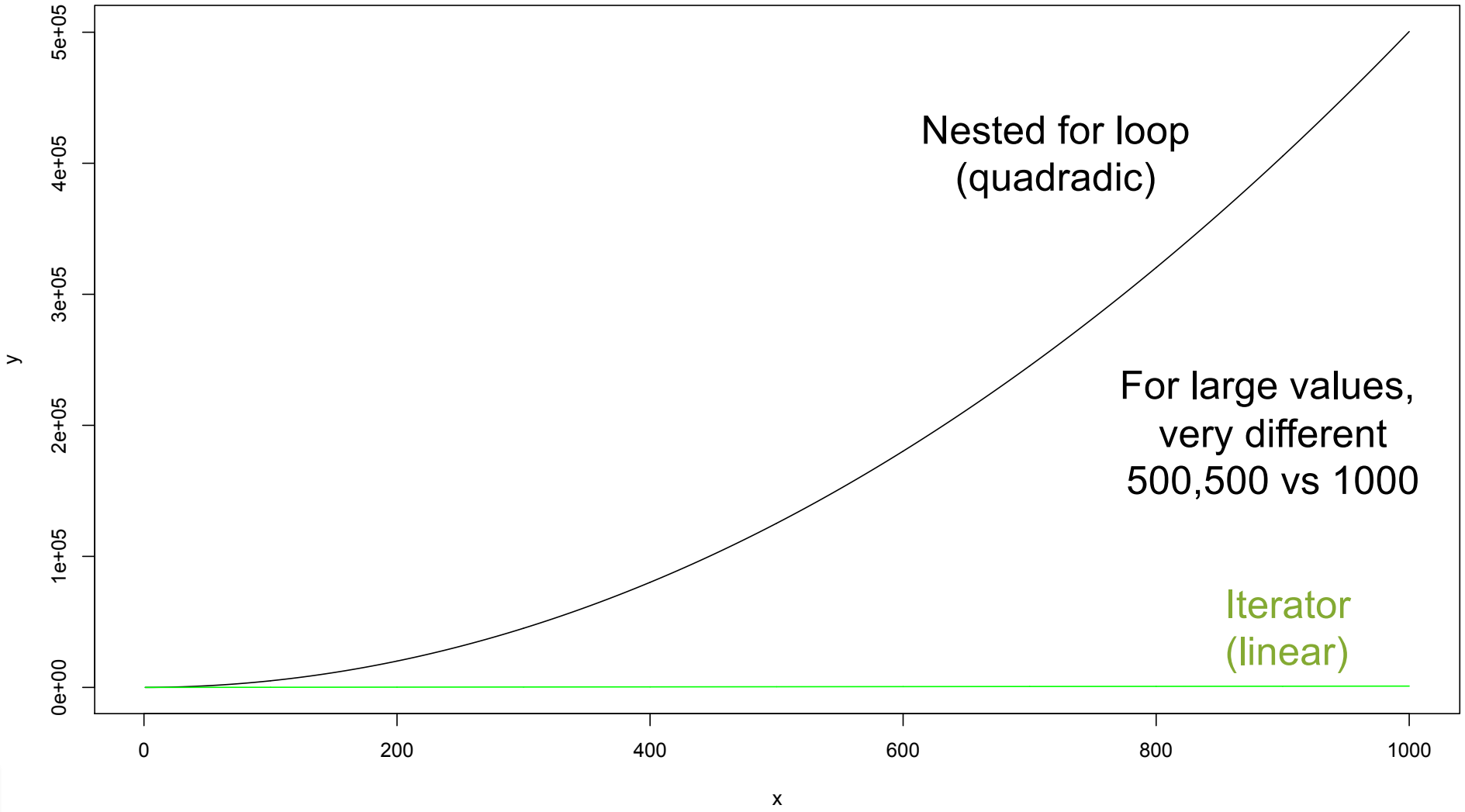
About a trillion

How many total hops do we make?

Iterator vs get()



Iterator vs get()



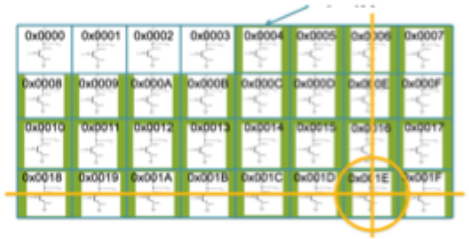
Complexity Analysis

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

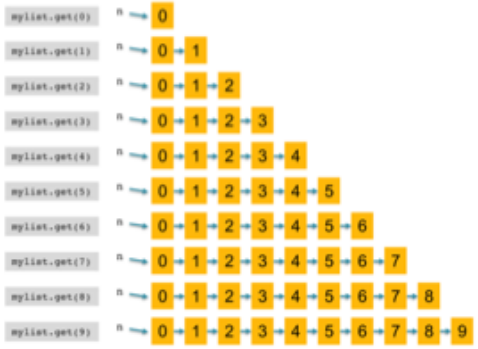


Accessing 1st or 1 billionth entry from an array takes same amount of time

Linear time

Takes 10 times longer to scan a list that has 10 times as many values

Quadradic time



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

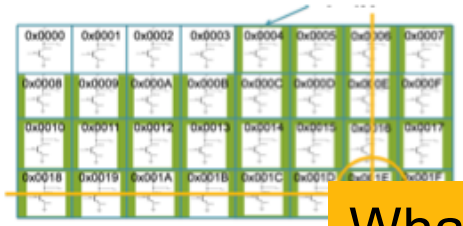
Complexity Analysis

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

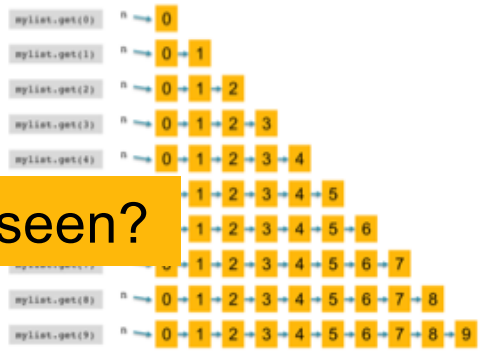


Accessing 1st or 1 billionth entry from an array takes same amount of time

Linear time

Takes 10 times longer to scan a list that has 10 times as many values

Quadradic time



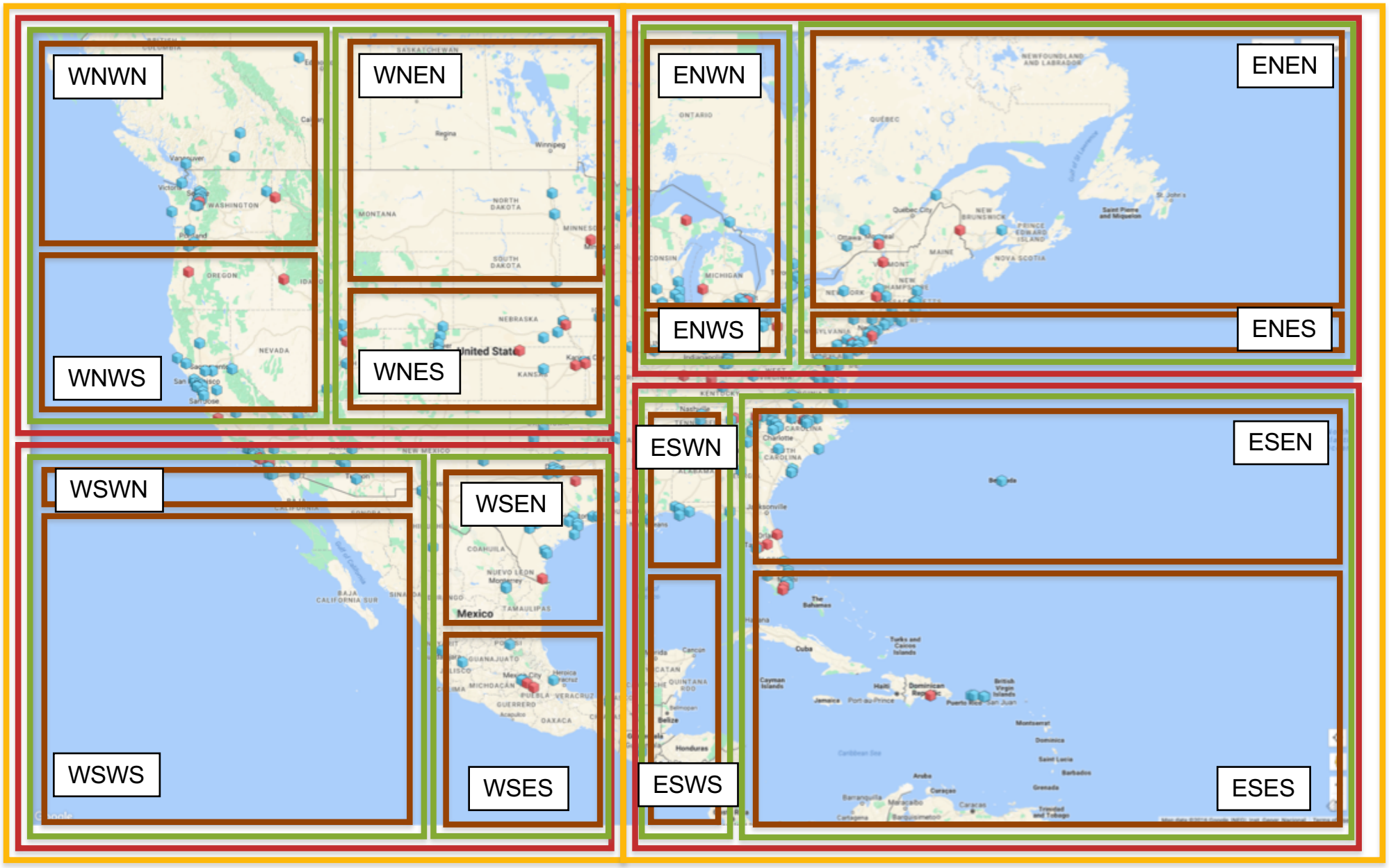
Nested loops grows with the square of the list length

What's another complexity we have seen?

Also very important for space characterization:

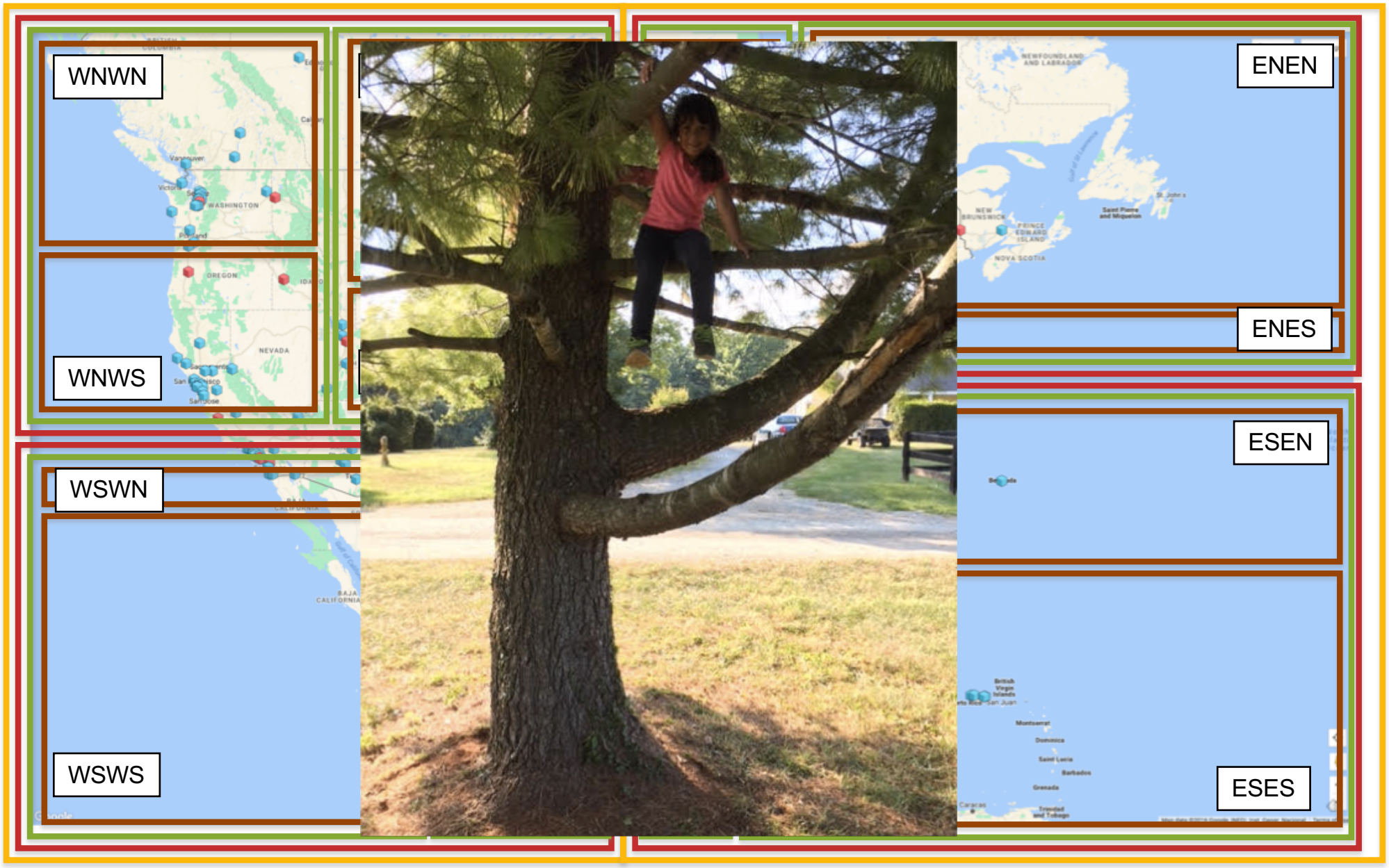
Sometimes doubling the number of elements will more than double the amount of space needed

Show me the photos!



Each sublist has N/16 elements & Balanced in both dimensions

Show me the photos!



Each sublist has N/16 elements & Balanced in both dimensions

Dividing N in half: 20 Billion

- Step 0: 20,000,000,000 possible elements ($N/1 = N/2^0$)
- Step 1: 10,000,000,000 possible elements ($N/2 = N/2^1$)
- Step 2: 5,000,000,000 possible elements ($N/4 = N/2^2$)
- …
- Step X: 1 possible element ($N/N = N/2^X$)

Find X such that: $2^X \geq N$

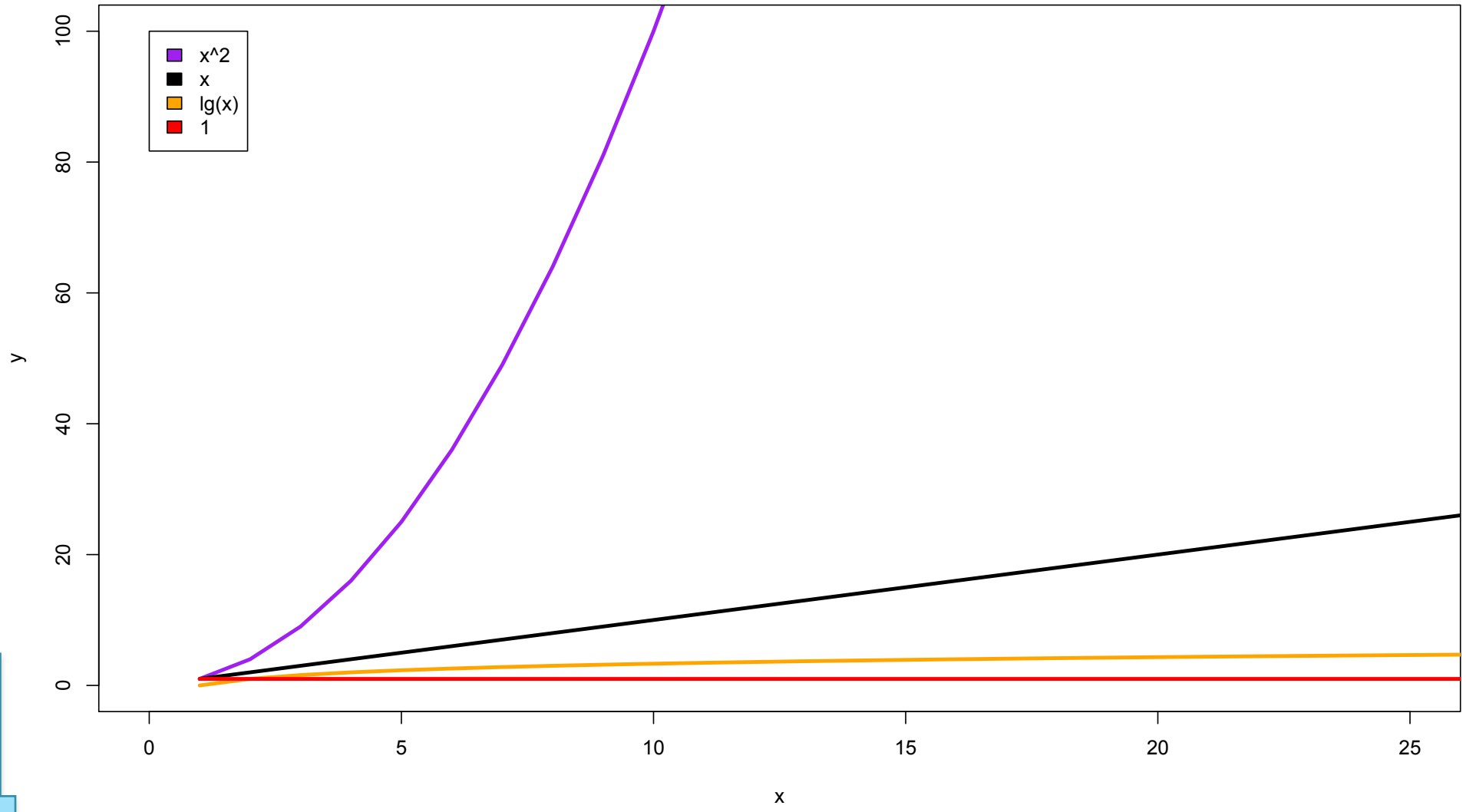
$\lg(2^X) \geq \lg(N)$

$X \geq \lg(N)$

X = 35

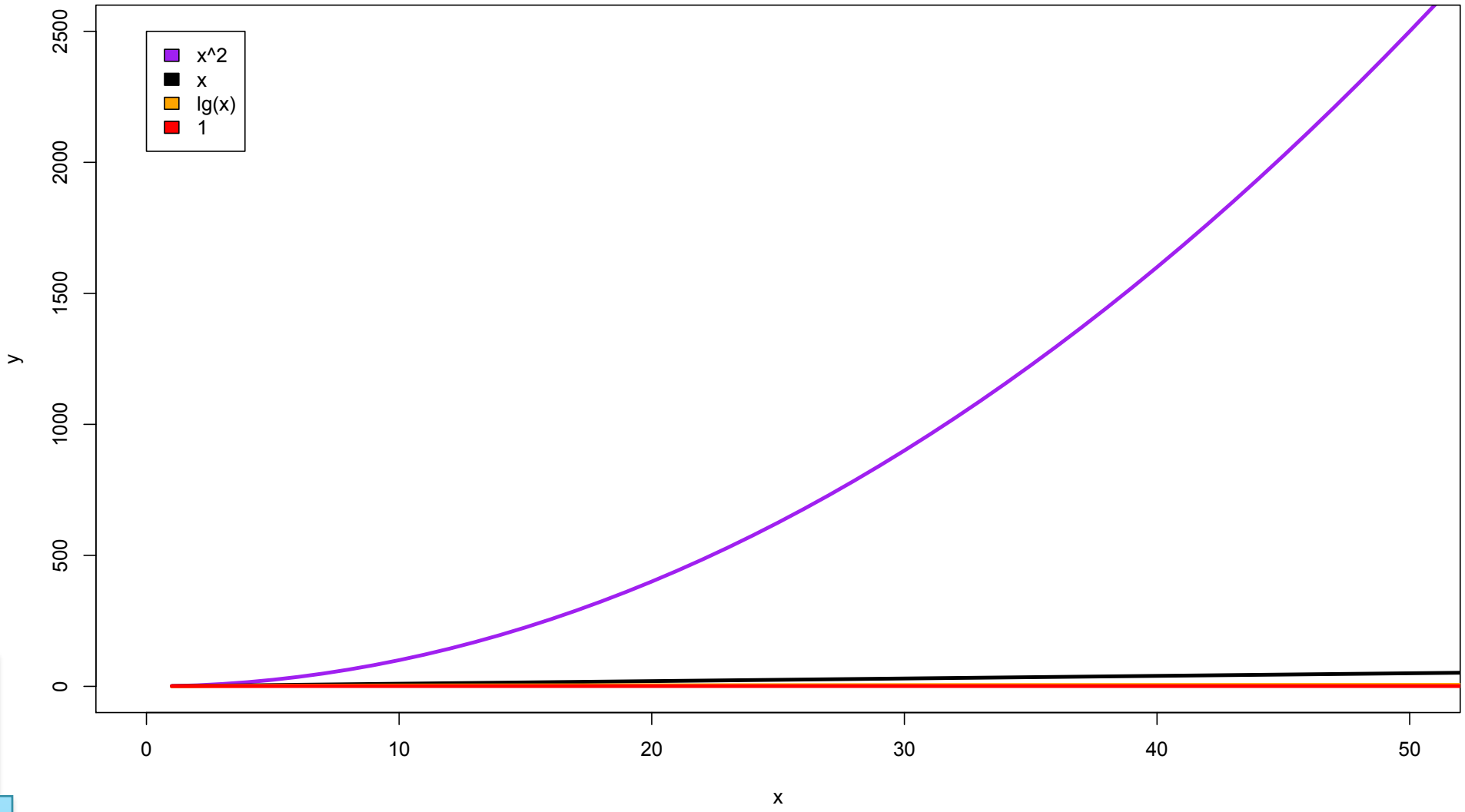
571.4 million times faster than brute force!

Growth of functions



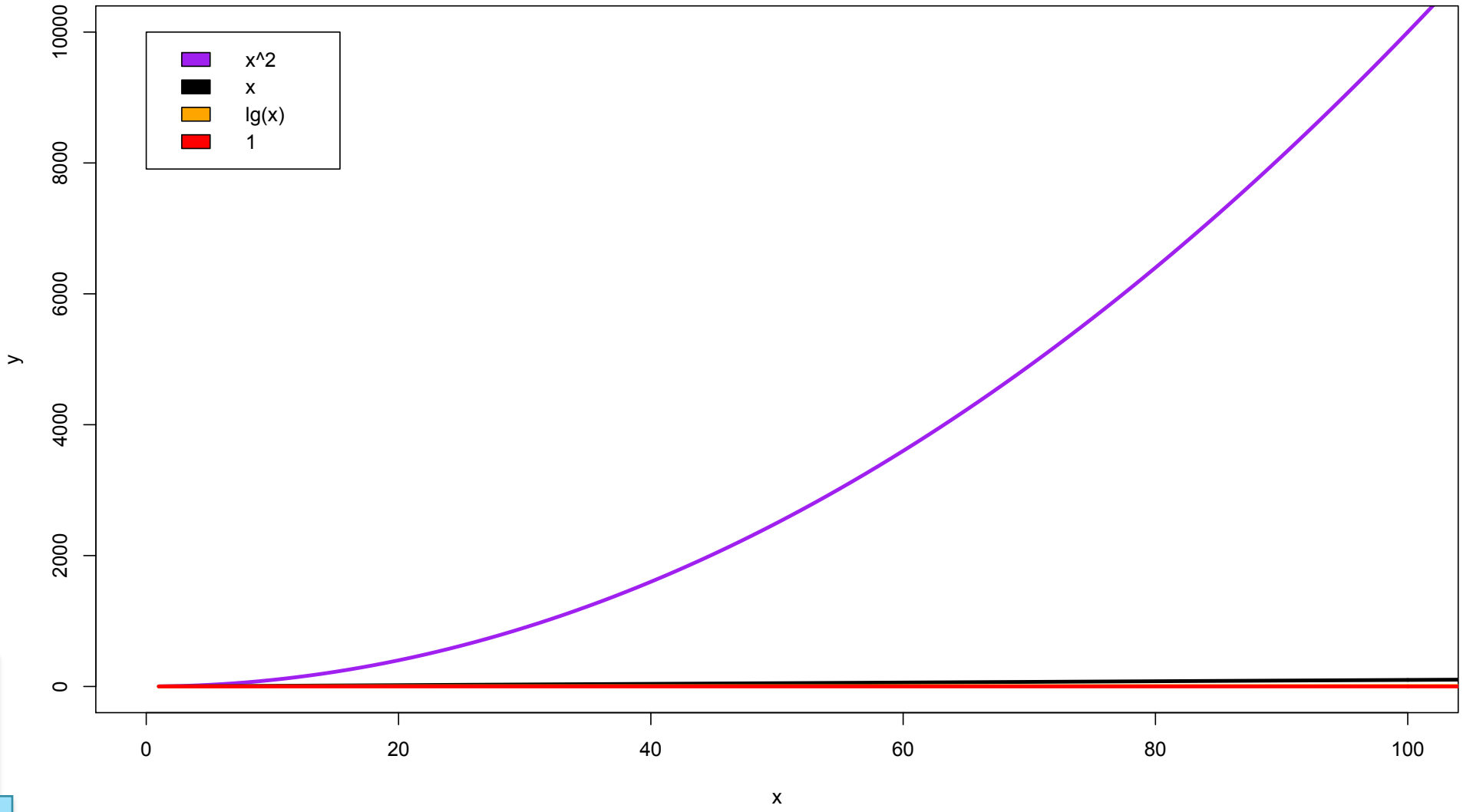
x^2 is clearly the slowest

Growth of functions



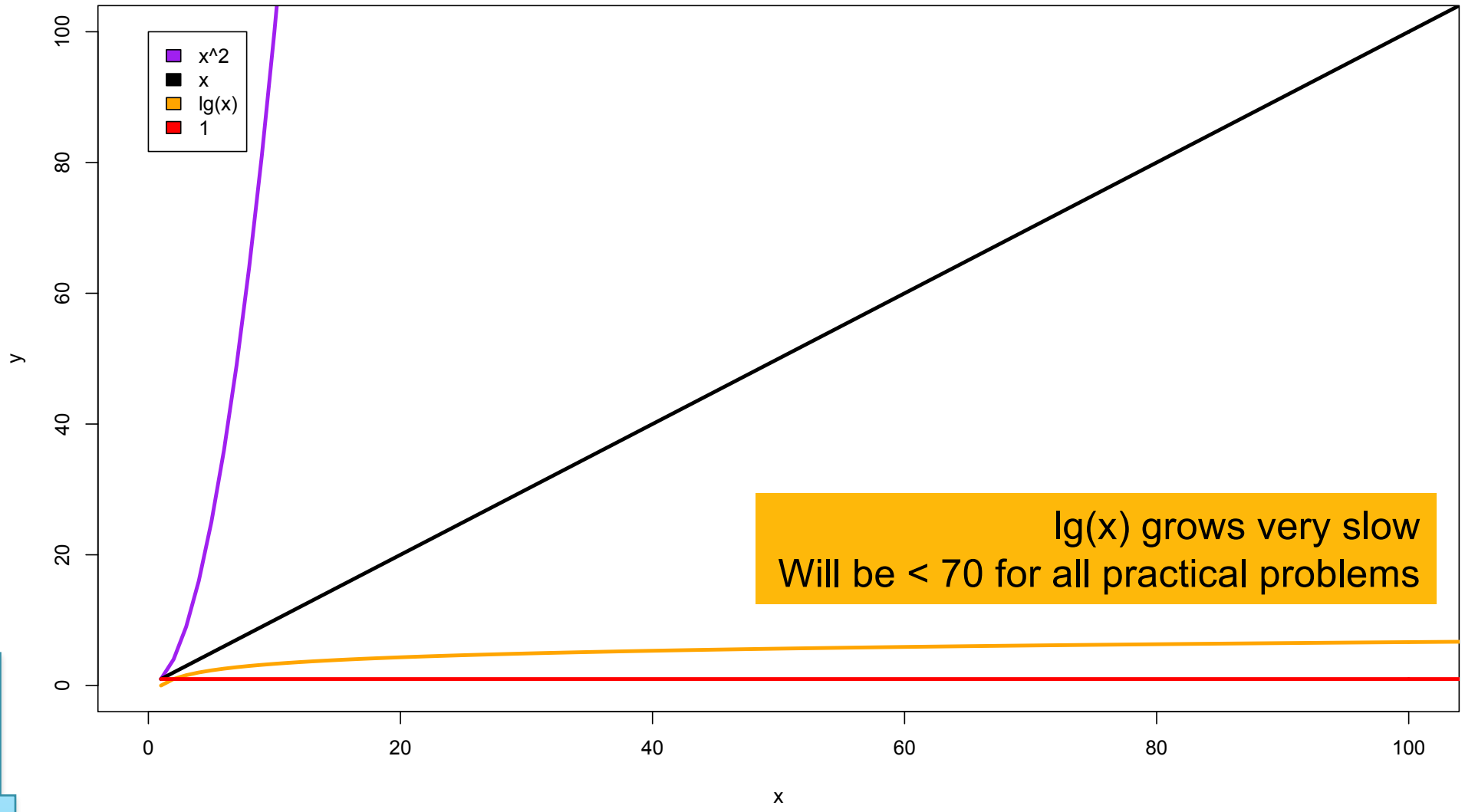
With larger values of x, slowness of x^2 is even more apparent

Growth of functions



Notice gap between constant and linear appears to shrink

Growth of functions



lg(x) grows very slow
Will be < 70 for all practical problems

Zooming in to focus on linear time

Find Max: Linear Search (1)

```
import java.text.NumberFormat;

public class ArrayFind {
  final static int MAXINT = 100000000;

  // return the biggest int in the array
  public static int findMaximum(int [] myarray){
   …
  }

  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println("USAGE: ArrayFind <array size>");
      return;
    }

    int arraysize = Integer.parseInt(args[0]);

    System.out.println("Scanning the array of size: ” +
                        NumberFormat.getInstance().format(arraysize));

    int [] myarray = new int[arraysize];

    ...
```

Find Max: Linear Search (2)

```
...

    int [] myarray = new int[arraysize];

    // initialize with random valuas
    for (int i = 0; i < myarray.length; i++) {
      int random = (int)(Math.random() * MAXINT);
      myarray[i] = random;
    }

    long startTime = System.nanoTime();
    int max = findMaximum(myarray);
    long endTime = System.nanoTime();
    long duration = endTime - startTime;

    System.out.println("The max is: " + max);
    System.out.println("Search took: " +
        NumberFormat.getInstance().format(duration) + " nanoseconds");
  }
}
```

FindMax Analysis

```
public static int findMaximum(int [] myarray) {
   int max = myarray[0];
   for (int i = 1; i < myarray.length; i++) {
     if (myarray[i] > max) {
       max = myarray[i];
     }
   }

   return max;
 }
```

```
$ java ArrayFind 10000000
Scanning the array of size: 10,000,000
The max is: 99999989
Search took: 11,666,963 nanoseconds
```

```
$ java ArrayFind 100000000
Scanning the array of size: 100,000,000
The max is: 99999999
Search took: 71,270,945 nanoseconds
```

Why isnt ArrayFind 100M exactly 10 times longer than ArrayFind 10M?

FindMax Analysis

```
public static int findMaximum(int [] myarray) {
   int max = myarray[0];
   for (int i = 1; i < myarray.length; i++) {
     if (myarray[i] > max) {
       max = myarray[i];
     }
   }

   return max;
 }
```

How many comparisons are done?

i < myarray.length	*n*
myarray[i] > max	*n*

C(n) = 2n

How many assignments are done (worst case)?

max = myarray[0]	*1*
for i =1; i < myarray.length; i++	*n*
val = myarray[i]	*n-1*
max = myarray[i]	*n-1*

A(n) = 1+ n + 2(n-1) = 3n-1

FindMax Analysis

```
public static int findMaximum(int [] myarray) {
   int max = myarray[0];
   for (int i = 1; i < myarray.length; i++) {
     if (myarray[i] > max) {
       max = myarray[i];
     }
   }

   return max;
 }
```

What is the total amount of work done?

T(n) = C(n) + A(n) = (2n) + (3n – 1) = 5n-1

Should we worry about the "-1"?

Nah, for sufficiently large inputs will make a tiny difference

Should we worry about the 5n?

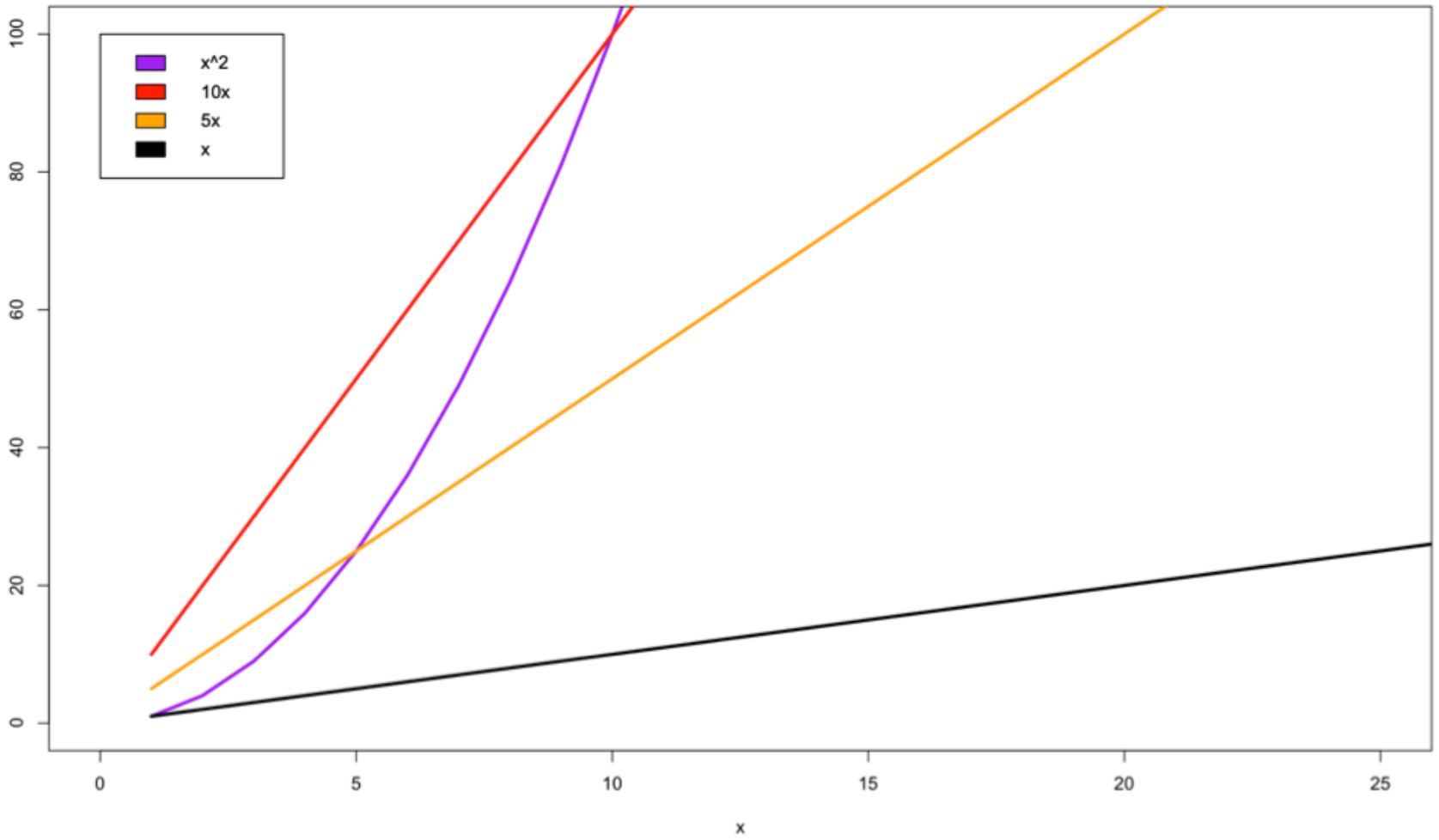
Nah, the runtime is linearly proportional to the length of the array

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) ∈ O(g(x)) if there exists c > 0 (e.g., c = 1) and x_0 (e.g., x_0 = 5) such that f(x) ≤ cg(x) whenever x ≥ 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*

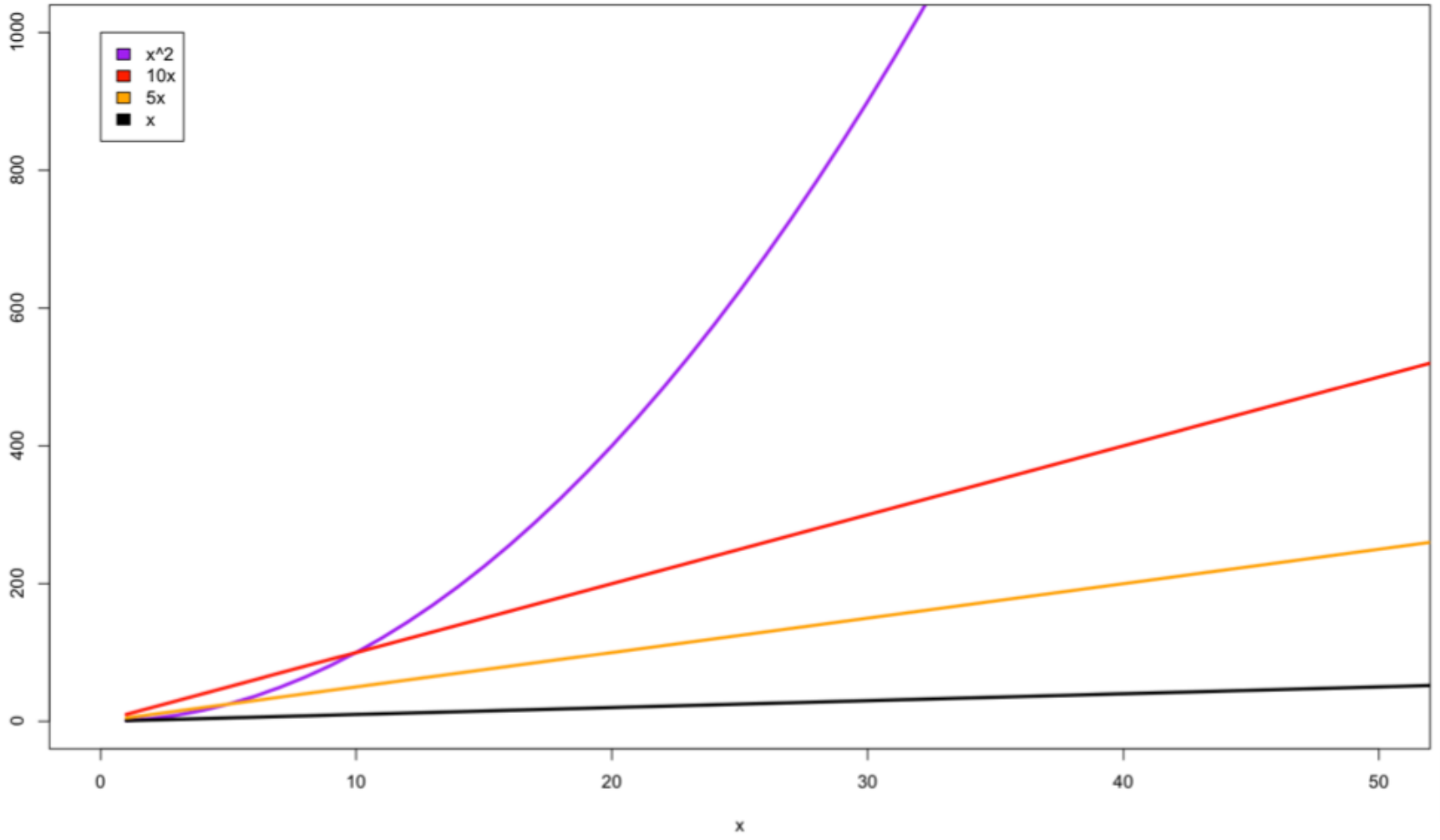
Growth of functions



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

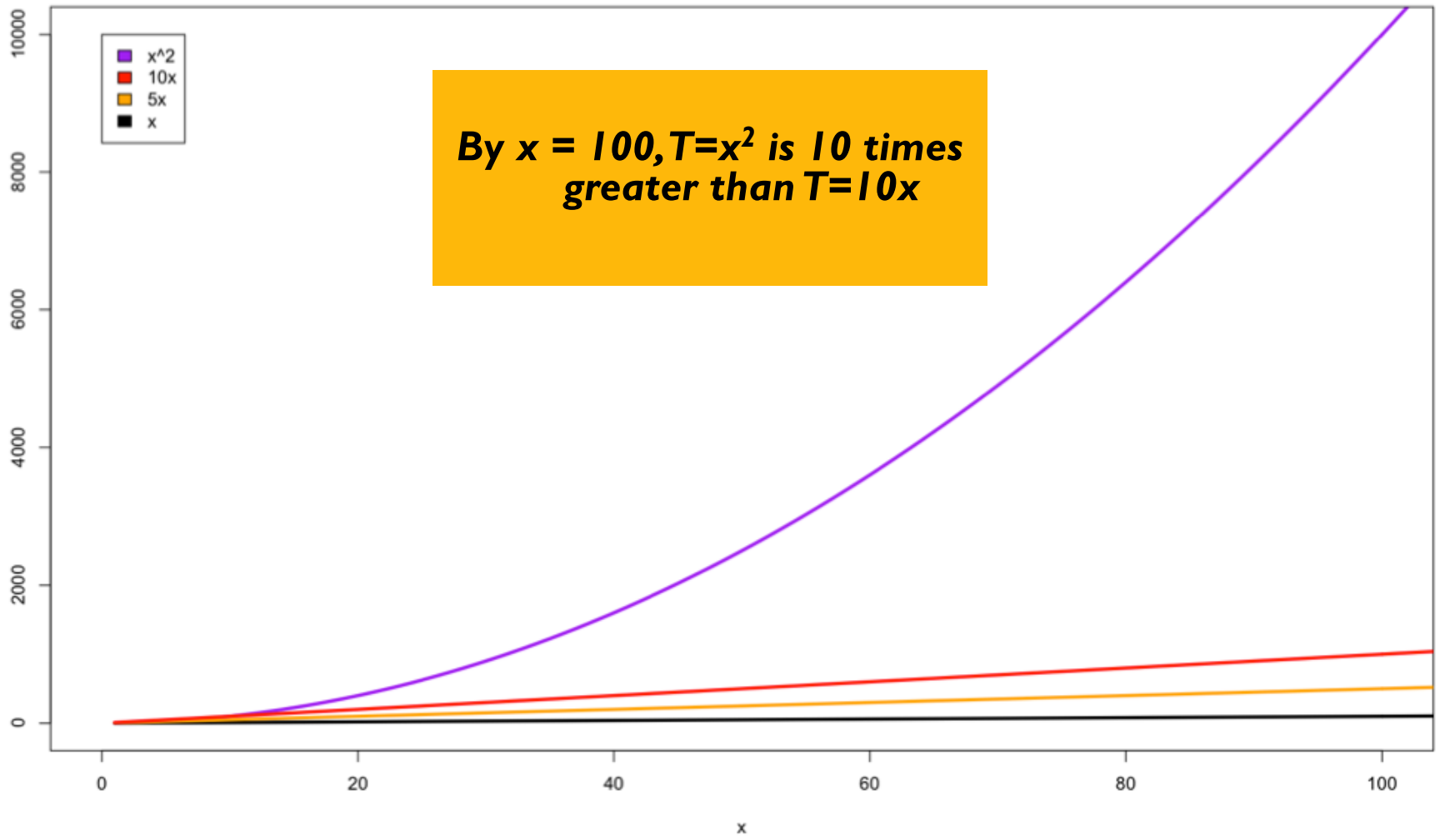
Growth of functions



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

Growth of functions

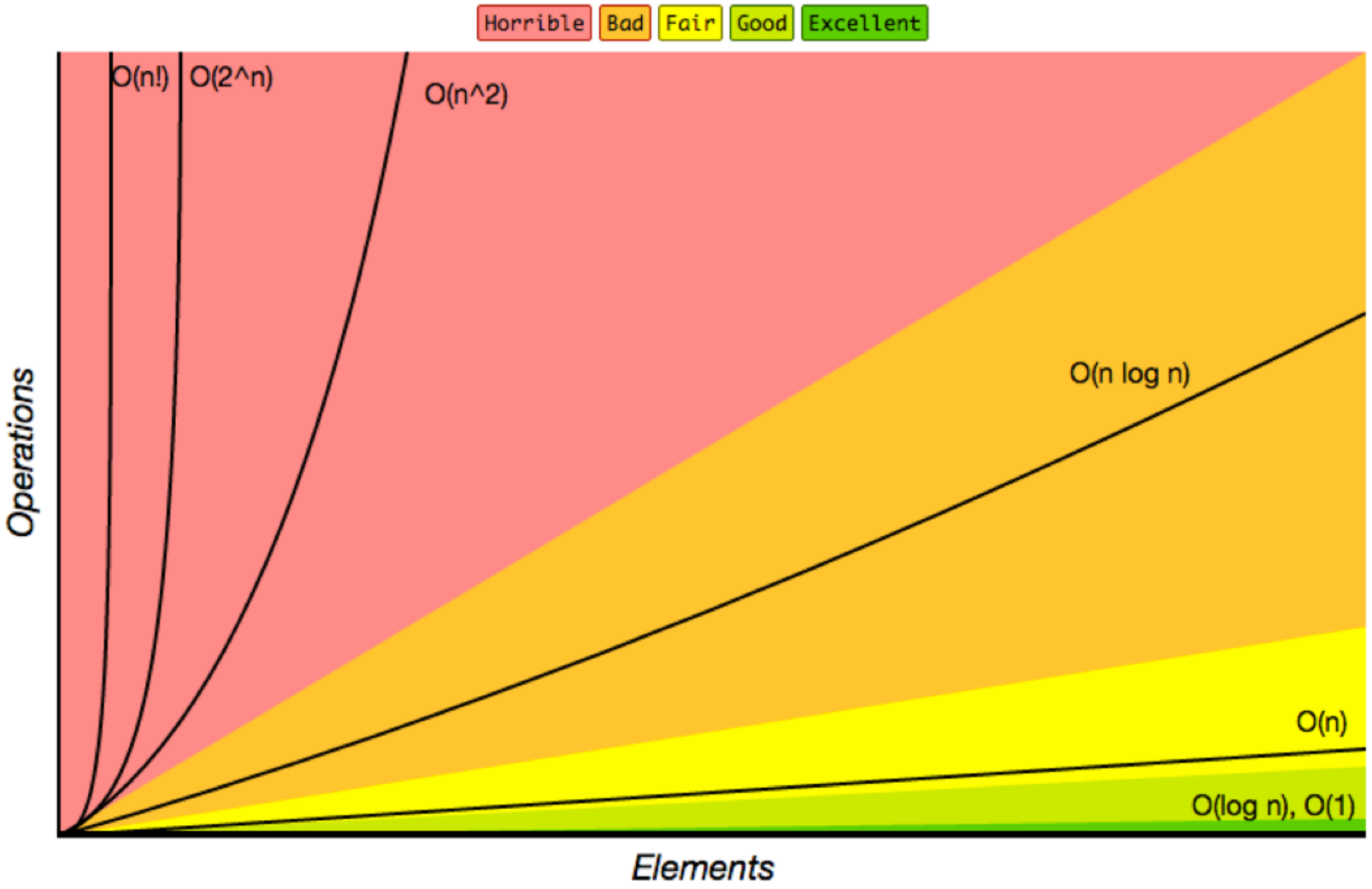


By x = 100, $T=x^2$ is 10 times greater than T=10x

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

Growth of functions



http://bigocheatsheet.com/

Next Steps

1. Work on HW1

2. Check on Piazza for tips & corrections!