CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 08: Analyzing Code

Announcements

- **IF YOU HAVEN'T ACCEPTED THE PA1 ASSIGNMENT, DO IT ASAP!**
  - Testing phase due this Sunday @ 11:59PM
  - Implementation AutoLab will be opened Monday (but you should rely on YOUR tests first)

Recap from Last Class

f and g are in the same complexity class, denoted $g(n) \in \Theta(f(n))$, iff:

$$g(n) \in O(f(n)) \quad \textbf{AND} \quad g(n) \in \Omega(f(n))$$

Recap from Last Class

f and g are in the same complexity class, denoted $g(n) \in \Theta(f(n))$, iff:

$$g(n) \in O(f(n)) \quad \textbf{AND} \quad g(n) \in \Omega(f(n))$$

$\exists\ n_0 \geq 0, c > 0$ s.t.

$\forall\ n \geq n_0, g(n) \leq c \cdot f(n)$

Recap from Last Class

f and ***g*** are in the same complexity class, denoted $g(n) \in \Theta(f(n))$, iff:

$$g(n) \in O(f(n)) \quad \textbf{AND} \quad g(n) \in \Omega(f(n))$$

$\exists\ n_0 \geq 0, c > 0$ s.t.

$\forall\ n \geq n_0, g(n) \leq c \cdot f(n)$

$\exists\ n_0 \geq 0, c > 0$ s.t.

$\forall\ n \geq n_0, g(n) \geq c \cdot f(n)$

In Practice

Most documentation uses Big-O (upper, ‘worst-case’) bounds…

- There’s always a Big-O bound (but Big-Θ might not exist)
- The best case (Big-Ω) usually doesn’t bring down production servers

In Practice

Most documentation uses Big-O (upper, ‘worst-case’) bounds…

- There’s always a Big-O bound (but Big-Θ might not exist)
- The best case (Big-Ω) usually doesn’t bring down production servers

Applying asymptotic analysis to code/algorithms

1. Analyze the code to determine the growth function that describes its runtime

In Practice

Most documentation uses Big-O (upper, ‘worst-case’) bounds…

- There’s always a Big-O bound (but Big-Θ might not exist)
- The best case (Big-Ω) usually doesn’t bring down production servers

Applying asymptotic analysis to code/algorithms

1. Analyze the code to determine the growth function that describes its runtime
2. Find bounds (ideally Big-Θ, but usually tight Big-O) on the growth function

In Practice

Most documentation uses Big-O (upper, ‘worst-case’) bounds…

- There’s always a Big-O bound (but Big-Θ might not exist)
- The best case (Big-Ω) usually doesn’t bring down production servers

Applying asymptotic analysis to code/algorithms

1. Analyze the code to determine the growth function that describes its runtime
2. Find bounds (ideally Big-Θ, but usually tight Big-O) on the growth function
3. More analysis? (amortized runtime, expected runtime, memory/IO bounds…)

Example

```
public void countDuplicates(Data[] data) {
  System.out.println("Counting duplicates");
  int count = 0;
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data[i] == data[j]) {
        count++;
      }
    }
  }
}
```

So what is the complexity and/or tight upper bound on the runtime of this code?

Example

```java
public void countDuplicates(Data[] data) {
  System.out.println("Counting duplicates");
  int count = 0;
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data[i] == data[j]) {
        count++;
      }
    }
  }
}
```

So what is the complexity and/or tight upper bound on the runtime of this code?

First we must determine the runtime growth function

Control Flow

Remember the different types of control flow:

- Sequential (statements executed one after another)
  - Add the number of steps together
  - If I do **A** then **B**, the total cost is the **cost to do A** plus the **cost to do B**

Control Flow

Remember the different types of control flow:

- Sequential (statements executed one after another)
  - Add the number of steps together
  - If I do **A** then **B**, the total cost is the **cost to do A** plus the **cost to do B**
- Selection (conditional execution of statements)
  - Our growth function will be a piecewise function

Control Flow

Remember the different types of control flow:

- Sequential (statements executed one after another)
  - Add the number of steps together
  - If I do **A** then **B**, the total cost is the **cost to do A** plus the **cost to do B**
- Selection (conditional execution of statements)
  - Our growth function will be a piecewise function
- Repetition (repeating execution of one or more statements)
  - Add up the total number of steps…with summations

Example

```java
public void countDuplicates(Data[] data) {
  System.out.println("Counting duplicates");
  int count = 0;
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data[i] == data[j]) {
        count++;
      }
    }
  }
}
```

Let's break this function down into sequential pieces

Example

```
  public void countDuplicates(Data[] data) {
A   System.out.println("Counting duplicates");
B   int count = 0;
    for (int i = 0; i < data.length; i++) {
      for (int j = i+1; j < data.length; j++) {
C       if (data[i] == data[j]) {
          count++;
        }
      }
    }
  }
```

Let's break this function down into sequential pieces

Example

```
public void countDuplicates(Data[] data) {
  System.out.println("Counting duplicates");
  int count = 0;
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data[i] == data[j]) {
        count++;
      }
    }
  }
}
```

Let's break this function down into sequential pieces

$$T(n) = T_A(n) + T_B(n) + T_C(n)$$

Example

```
public void countDuplicates(Data[] data) {
A System.out.println("Counting duplicates");
B int count = 0;
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
C     if (data[i] == data[j]) {
        count++;
      }
    }
  }
}
```

Let's break this function down into sequential pieces

$$T(n) = 1 + 1 + T_C(n)$$

Example

```
public void countDuplicates(Data[] data) {
A System.out.println("Counting duplicates");
B int count = 0;
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
C     if (data[i] == data[j]) {
        count++;
      }
    }
  }
}
```

Now how many steps does C take in total?

Let's isolate it

$$T(n) = 1 + 1 + T_C(n)$$

Example

```
for (int i = 0; i < data.length; i++) {
  for (int j = i+1; j < data.length; j++) {
    if (data[i] == data[j]) {
      count++;
    }
  }
}
```

The outer loop starts at **i = 0**, and goes up to **data.length (n)** stepping by **1**

Each iteration it does a certain amount of work (which may or may not depend on i)

Example

```
for (int i = 0; i < data.length; i++) {
  for (int j = i+1; j < data.length; j++) {
    if (data[i] == data[j]) {
      count++;
    }
  }
}
```

$$\sum_{i=0}^{n}$$

The outer loop starts at **i = 0**, and goes up to **data.length (n)** stepping by **1**

Each iteration it does a certain amount of work (which may or may not depend on i)

Example

```
for (int i = 0; i < data.length; i++) {
  for (int j = i+1; j < data.length; j++) {
    if (data[i] == data[j]) {
      count++;
    }
  }
}
```

D

$$\sum_{i=0}^{n} T_D(n, i)$$

The outer loop starts at **i = 0**, and goes up to **data.length (n)** stepping by **1**

Each iteration it does a certain amount of work (which may or may not depend on i)

Example

```
for (int i = 0; i < data.length; i++) {
  for (int j = i+1; j < data.length; j++) {
    if (data[i] == data[j]) {
      count++;
    }
  }
}
```

D

$$\sum_{i=0}^{n} T_D(n, i)$$

Apply the same approach to determine T_D

Example

```
for (int i = 0; i < data.length; i++) {
  for (int j = i+1; j < data.length; j++) {
    if (data[i] == data[j]) {
      count++;
    }
  }
}
```

$$\sum_{i=0}^{n} T_D(n, i)$$

Apply the same approach to determine T_D

Example

```
for (int i = 0; i < data.length; i++) {
  for (int j = i+1; j < data.length; j++) {
    if (data[i] == data[j]) {
      count++;
    }
  }
}
```

D

E

$$\sum_{i=0}^{n} \sum_{j=i+1}^{n} T_E(n, i, j)$$

Apply the same approach to determine T_D

Example

```
for (int i = 0; i < data.length; i++) {
  for (int j = i+1; j < data.length; j++) {
    if (data[i] == data[j]) {
      count++;
    }
  }
}
```

D

E

$$\sum_{i=0}^{n} \sum_{j=i+1}^{n} T_E(n, i, j)$$

Finally let's isolate **E**

Example

```
if (data[i] == data[j]) {
  count++;
}
```

How many steps does this code take?

Example

```
if (data[i] == data[j]) {
  count++;
}
```

How many steps does this code take?

What if `data[i] == data[j]`?

Example

```
if (data[i] == data[j]) {
  count++;
}
```

How many steps does this code take?

What if `data[i] == data[j]`?

1 step to check the condition plus 1 step to increment count = 2

Example

```
if (data[i] == data[j]) {
  count++;
}
```

How many steps does this code take?

What if `data[i] != data[j]`?

Example

```
    if (data[i] == data[j]) {
      count++;
    }
```

How many steps does this code take?

What if `data[i] != data[j]`?

1 step to check the condition

Example

```
if (data[i] == data[j]) {
  count++;
}
```

How many steps does this code take?

$$T_E(n, i, j) = \begin{cases} 2 & \text{if data[i] == data[j]} \\ 1 & \text{otherwise} \end{cases}$$

Putting It All Together

```java
public void countDuplicates(Data[] data) {
  System.out.println("Counting duplicates");
  int count = 0;
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data[i] == data[j]) {
        count++;
      }
    }
  }
}
```

$$T(n) = 1 + 1 + \sum_{i=0}^{n} \sum_{j=i+1}^{n} T_E(n, i, j)$$

$$T_E(n, i, j) = \begin{cases} 2 & \text{if data[i] == data[j]} \\ 1 & \text{otherwise} \end{cases}$$

Tip #1

Tip: If you know the complexity/bound of a growth function, you can use the complexity/bound instead of the growth function

What is the complexity of T_E?

$$T_E(n, i, j) = \begin{cases} 2 & \text{if data[i] == data[j]} \\ 1 & \text{otherwise} \end{cases}$$

Tip #1

Tip: If you know the complexity/bound of a growth function, you can use the complexity/bound instead of the growth function

What is the complexity of T_E? $\Theta(1)$

$$T_E(n, i, j) = \begin{cases} 2 & \text{if data[i] == data[j]} \\ 1 & \text{otherwise} \end{cases}$$

Tip #1

$$T(n) = 1 + 1 + \sum_{i=0}^{n} \sum_{j=i+1}^{n} T_E(n, i, j)$$

Tip #1

$$T(n) = 1 + 1 + \sum_{i=0}^{n} \sum_{j=i+1}^{n} T_E(n, i, j)$$

$$T(n) = 1 + 1 + \sum_{i=0}^{n} \sum_{j=i+1}^{n} \Theta(1)$$

Tip #1

Now the summations are much easier to deal with

$$T(n) = 1 + 1 + \sum_{i=0}^{n} \sum_{j=i+1}^{n} T_E(n, i, j)$$

$$\Downarrow$$

$$T(n) = 1 + 1 + \sum_{i=0}^{n} \sum_{j=i+1}^{n} \Theta(1)$$

$$\Downarrow$$

$$T(n) = \Theta(1) + \sum_{i=0}^{n} \sum_{j=i+1}^{n} \Theta(1)$$

$$T(n) = \Theta(1) + \sum_{i=0}^{n} \sum_{j=i+1}^{n} \Theta(1)$$

Tip #1

Now the summations are much easier to deal with

Tip #1

Now the summations are much easier to deal with

$$T(n) = \Theta(1) + \sum_{i=0}^{n} \sum_{j=i+1}^{n} \Theta(1)$$

$$\Downarrow$$

$$T(n) = \Theta(1) + \sum_{i=0}^{n} (n - i)\Theta(1)$$

Tip #1

Now the summations are much easier to deal with

$$T(n) = \Theta(1) + \sum_{i=0}^{n} \sum_{j=i+1}^{n} \Theta(1)$$

$$\Downarrow$$

$$T(n) = \Theta(1) + \sum_{i=0}^{n} (n - i)\Theta(1)$$

$$\Downarrow$$

$$T(n) = \Theta(1) + \Theta(n^2)$$

$$
\begin{aligned}
c \cdot \theta(f(N)) &= \theta(f(N)) \\
N \cdot \theta(f(N)) &= \theta(N \cdot f(N)) \\
g(N) \cdot \theta(f(N)) &= \theta(g(N) \cdot f(N)) \quad \text{(if } \theta(g(N)) \text{ exists)} \\
\theta(g(N)) + \theta(f(N)) &= \theta(g(N) + f(N)) \\
&= \text{The greater of } \theta(f(N)) \textbf{ or } \theta(g(N))
\end{aligned}
$$

Algebra with Big-Θ

Tip #1

Finish off the simplification using the rules of Big-Theta algebra

$$T(n) = \Theta(1) + \Theta(n^2)$$

$$\Downarrow$$

$$T(n) = \Theta(n^2)$$

Tip #2

Tip: Start by identifying the parts of the code that have a constant runtime

Example 2

```java
public void updateCells(Data[] data) {
  System.out.println("Updating our data...");
  int num_neighbors = 8;
  for (Data d : data) {
    System.out.println("Processing element " + d);
    for (int i = 0; i < num_neighbors; i++) {
      data.weight += data.neighbor[i] / num_neighbors;
    }
  }
}
```

Example 2

```java
public void updateCells(Data[] data) {
  System.out.println("Updating our data...");
  int num_neighbors = 8;
  for (Data d : data) {
    System.out.println("Processing element " + d);
    for (int i = 0; i < num_neighbors; i++) {
      data.weight += data.neighbor[i] / num_neighbors;
    }
  }
}
```

$\Theta(1)$ (lines 2–3)

$\Theta(1)$ (lines 5–8)

Example 2

```
public void updateCells(Data[] data) {
  Θ(1)
  for (Data d : data) {
    Θ(1)
  }
}
```

Example 2

```
public void updateCells(Data[] data) {
  Θ(1)
  for (Data d : data) {
    Θ(1)
  }
}
```

$$\sum_{i=1}^{n} \Theta(1) = n \cdot \Theta(1) = \Theta(n)$$

Tip #3: If the cost of an iteration does not depend on the loop variable, you can just multiple the cost of an iteration by the number of iterations

(use with caution…)

Example 2

```
public void updateCells(Data[] data) {
  Θ(1)
  Θ(n)
}
```

Example 2

```
public void updateCells(Data[] data) {
  Θ(1 + n) = Θ(n)
}
```

Example 2

```java
public void updateCells(Data[] data) {
  System.out.println("Updating our data...");
  int num_neighbors = 8;
  for (Data d : data) {
    System.out.println("Processing element " + d);
    for (int i = 0; i < num_neighbors; i++) {
      data.weight += data.neighbor[i] / num_neighbors;
    }
  }
}
```

$$\theta(1) + \sum_{d \in data} \theta(1) \in \theta(n)$$

Common Pitfall

Remember: You are not counting "lines of code"

A single line of code does not necessarily mean a single step

Conversely, a loop doesn't guarantee a non-constant runtime

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        data.remove(j--);
      }
    }
  }
}
```

Example 3

```java
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        data.remove(j--);
      }
    }
  }
}
```

Is this single line a single "step"?

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#remove-int-

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        data.remove(j--);
      }
    }
  }
}
```

Is this single line a single "step"?

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#remove-int-

Could have to move all ***n*** elements in the worst case, so upper bound is ***O*(*n*)**

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        O(n)
      }
    }
  }
}
```

Example 3

```java
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        O(n)
      }
    }
  }
}
```

The body is only executed if the condition is true…

Example 3

```java
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        O(n)
      }
    }
  }
}
```

$$\begin{cases} O(1) & \text{if condition is false} \\ O(n) & \text{if condition is true} \end{cases}$$

The body is only executed if the condition is true…

$$
\begin{aligned}
c \cdot O(f(N)) &= O(f(N)) \\
N \cdot O(f(N)) &= O(N \cdot f(N)) \\
g(N) \cdot O(f(N)) &= O(g(N) \cdot f(N)) \\
O(g(N)) + O(f(N)) &= O(g(N) + f(N)) \\
&= \text{the greater of } O(f(N)) \textbf{ or } O(g(N)) \\
\begin{cases} O(g(N)) & \textbf{if} \text{ one thing} \\ O(f(N)) & \textbf{otherwise} \end{cases} &= \text{the greater of } O(f(N)) \textbf{ or } O(g(N)) \\
&= O(g(N) + f(N))
\end{aligned}
$$

Algebra with Big-O

$$c \cdot \Omega(f(N)) = \Omega(f(N))$$

$$N \cdot \Omega(f(N)) = \Omega(N \cdot f(N))$$

$$g(N) \cdot \Omega(f(N)) = \Omega(g(N) \cdot f(N))$$

$$\begin{aligned} \Omega(g(N)) + \Omega(f(N)) &= \Omega(g(N) + f(N)) \\ &= \text{the greater of } \Omega(f(N)) \textbf{ or } \Omega(g(N)) \end{aligned}$$

$$\begin{cases} \Omega(g(N)) & \textbf{if} \text{ one thing} \\ \Omega(f(N)) & \textbf{otherwise} \end{cases} =$$ Smaller of f(N) or g(N)

Algebra with Big-Ω

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        O(n)
      }
    }
  }
}
```

$$\begin{cases} O(1) & \text{if condition is false} \\ O(n) & \text{if condition is true} \end{cases}$$

The body is only executed if the condition is true...

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        O(n)
      }
    }
  }
}
```

$O(n)$

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      O(n)
    }
  }
}
```

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    O(n²)
  }
}
```

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  O(1)
  O(n^3)
}
```

Example 3

```
public void makeUnique(ArrayList<Data> data) {
  O(1 + n^3) = O(n^3)
}
```

Example 3

```java
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        data.remove(j--);
      }
    }
  }
}
```

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} O(n) \in O(n^3)$$

Example 3

```java
public void makeUnique(ArrayList<Data> data) {
  System.out.println("Making all elements in data unique");
  for (int i = 0; i < data.length; i++) {
    for (int j = i+1; j < data.length; j++) {
      if (data.get(i) == data.get(j)) {
        data.remove(j--);
      }
    }
  }
}
```

Remember: Plugging in O(n) here is a shortcut…we may be able to be more specific once we understand how remove depends on j

When in doubt, do the full summation

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} O(n) \in O(n^3)$$

Real Example

```java
public void bubbleSort(List<Integer> list) {
  for (int i = list.size() - 2; i >= 0; i--) {
    for (int j = i; j < list.size() - 1; j++) {
      if (list.get(j) < list.get(j+1)) {
        Integer tmp = list.get(j);
        list.set(j, list.get(j+1));
        list.set(j+1, tmp);
      }
    }
  }
}
```

Real Example

```
public void bubbleSort(List<Integer> list) {
  for (int i = list.size() - 2; i >= 0; i--) {
    for (int j = i; j < list.size() - 1; j++) {
      if (list.get(j) < list.get(j+1)) {
        Integer tmp = list.get(j);
        list.set(j, list.get(j+1));
        list.set(j+1, tmp);
      }
    }
  }
}
```

Is this implementation of bubble sort an **O(*n*^{2})** algorithm?

Real Example

```java
public void bubbleSort(List<Integer> list) {
  for (int i = list.size() - 2; i >= 0; i--) {
    for (int j = i; j < list.size() - 1; j++) {
      if (list.get(j) < list.get(j+1)) {
        Integer tmp = list.get(j);
        list.set(j, list.get(j+1));
        list.set(j+1, tmp);
      }
    }
  }
}
```

Is this implementation of bubble sort an **O(n^2)** algorithm?
What is the runtime of **`get()/set()`**?

Real Example

```
public void bubbleSort(List<Integer> list) {
  for (int i = list.size() - 2; i >= 0; i--) {
    for (int j = i; j < list.size() - 1; j++) {
      O(?)
    }
  }
}
```

Is this implementation of bubble sort an **O(*n*²)** algorithm?

What is the runtime of **`get()/set()`**?

Real Example

```java
public void bubbleSort(List<Integer> list) {
  for (int i = list.size() - 2; i >= 0; i--) {
    for (int j = i; j < list.size() - 1; j++) {
      O(?)
    }
  }
}
```

$$\sum_{i=0}^{n-2}\sum_{j=i}^{n-1} O(?)$$

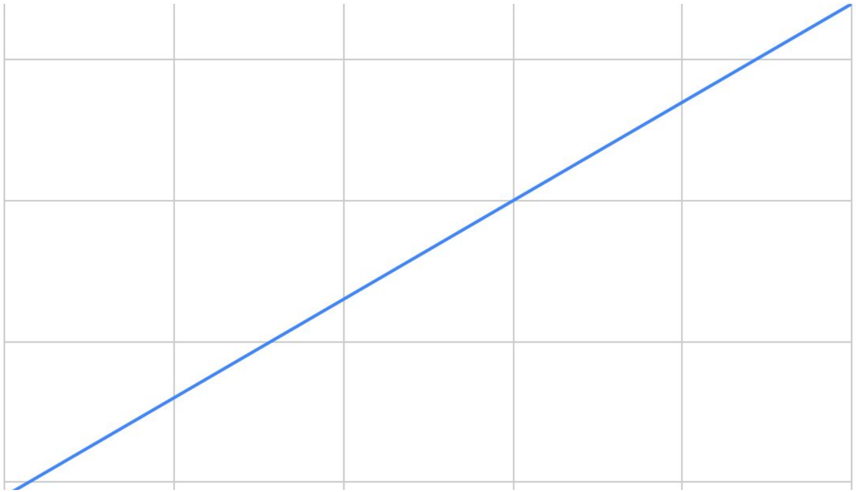
Is this implementation of bubble sort an **O(*n*²)** algorithm?

What is the runtime of **`get()/set()`**?

Comparing Random Access for Array vs List

Array

List



Real Example

```java
public void bubbleSort(List<Integer> list) {
  for (int i = list.size() - 2; i >= 0; i--) {
    for (int j = i; j < list.size() - 1; j++) {
      O(?)
    }
  }
}
```

$$\sum_{i=0}^{n-2} \sum_{j=i}^{n-1} O(?)$$

Is this implementation of bubble sort an **O(n^2)** algorithm?

What is the runtime of **`get()/set()`**?

If our list is an array, ***O*(?) = *O*(1)**, so overall runtime is ***O*(n^2)**

Real Example

```
public void bubbleSort(List<Integer> list) {
  for (int i = list.size() - 2; i >= 0; i--) {
    for (int j = i; j < list.size() - 1; j++) {
      O(?)
    }
  }
}
```

$$\sum_{i=0}^{n-2} \sum_{j=i}^{n-1} O(?)$$

Is this implementation of bubble sort an **O(*n*²)** algorithm?

What is the runtime of **`get()/set()`**?

If our list is an array, ***O*(?) = *O*(1)**, so overall runtime is ***O*(n^2)**

If our list is a LinkedList, ***O*(?) = *O*(*n*)**, so overall runtime is ***O*(n^3)**