CS24 Week 6 Lecture 1

Kyle Dewey

Overview

- Complexity and complexity analysis

Complexity

Complexity

- Up until this point, we have used terms like "efficiency", "expensive", and "cheap"

```
int foo(int* array) {
  return array[2] * array[3];
}
```

```
int bar(int* array) {
  int x;
  for(x = 0; x < MAX_SIZE; x++) {
    if (array[x] == 7) return x;
  }
  return -1;
}
```

Complexity

- Up until this point, we have used terms like "efficiency", "expensive", and "cheap"

```
int foo(int* array) {
  return array[2] * array[3];
}
```

cheap(?)

```
int bar(int* array) {
  int x;
  for(x = 0; x < MAX_SIZE; x++) {
    if (array[x] == 7) return x;
  }
  return -1;
}
```

expensive(?)

Ambiguous Terms

- Under what circumstances is this cheap?
- When is it expensive?

```
int bar(int* array) {
  int x;
  for(x = 0; x < MAX_SIZE; x++) {
    if (array[x] == 7) return x;
  }
  return -1;
}
```

“Expensive”, “Cheap”, “Efficient”

- What is good about these terms?
- What is bad about these terms?

"Expensive", "Cheap" "Efficient"

- What is good about these terms?
  - Easy to understand
- What is bad about these terms?
  - Imprecise
  - Binary in nature (either cheap or expensive)
  - Program efficiency is often dependent on **input size**

Measuring Efficiency

- How might we determine the efficiency of a program?

Measuring Efficiency

- How might we determine the efficiency of a program?
  - Benchmarks tend to be too specific (new hardware? How big of inputs do we test?)
  - Better approach: define a formula in terms of the input size

Big O Notation

- A formula that gives an upper bound of how expensive something is *in the worst case*, in terms of an input size N
- Which is most efficient below?

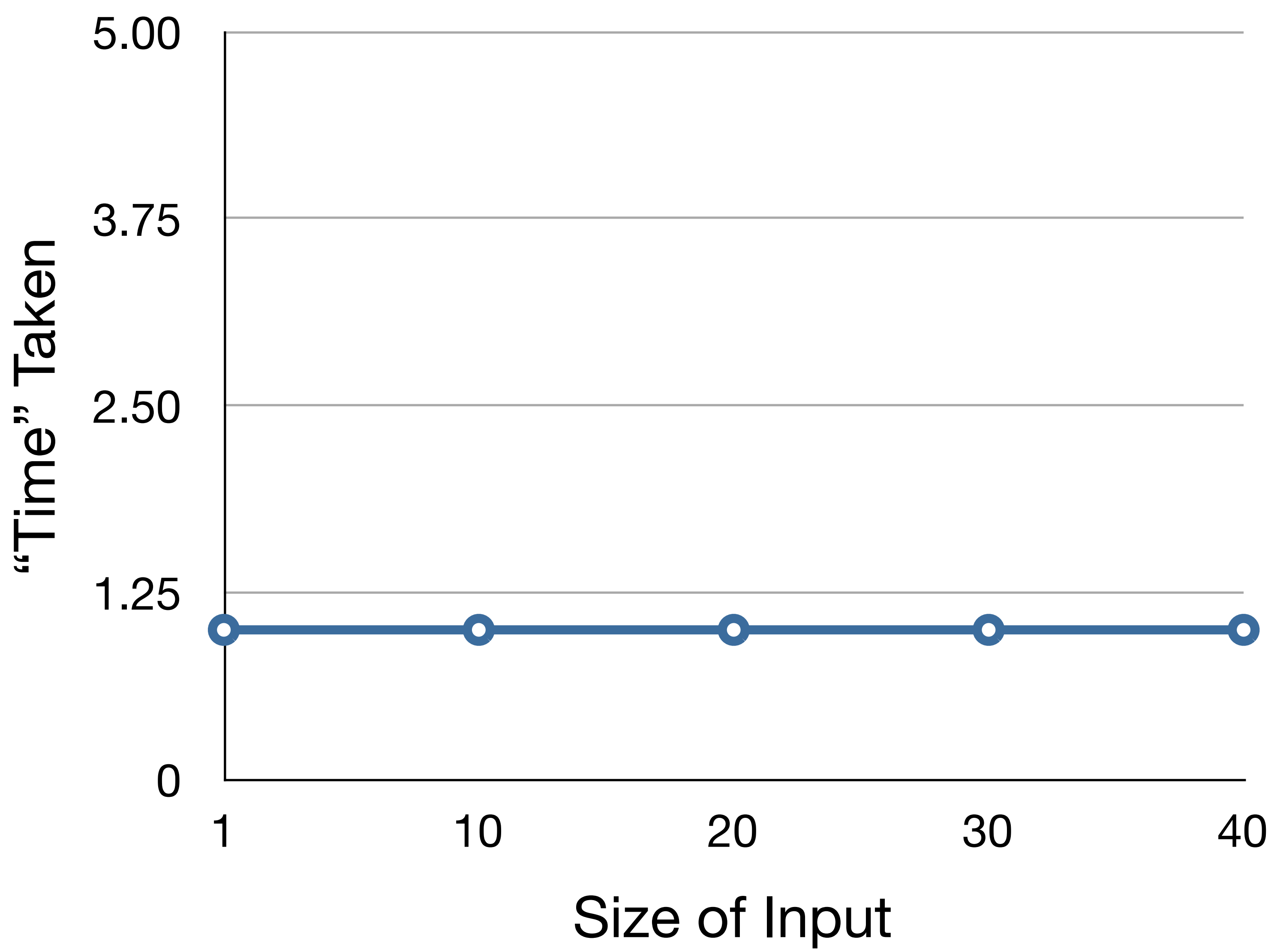
```
O(1) // constant time

O(n) // linear time
```

$O(n^2)$ `// quadratic time`

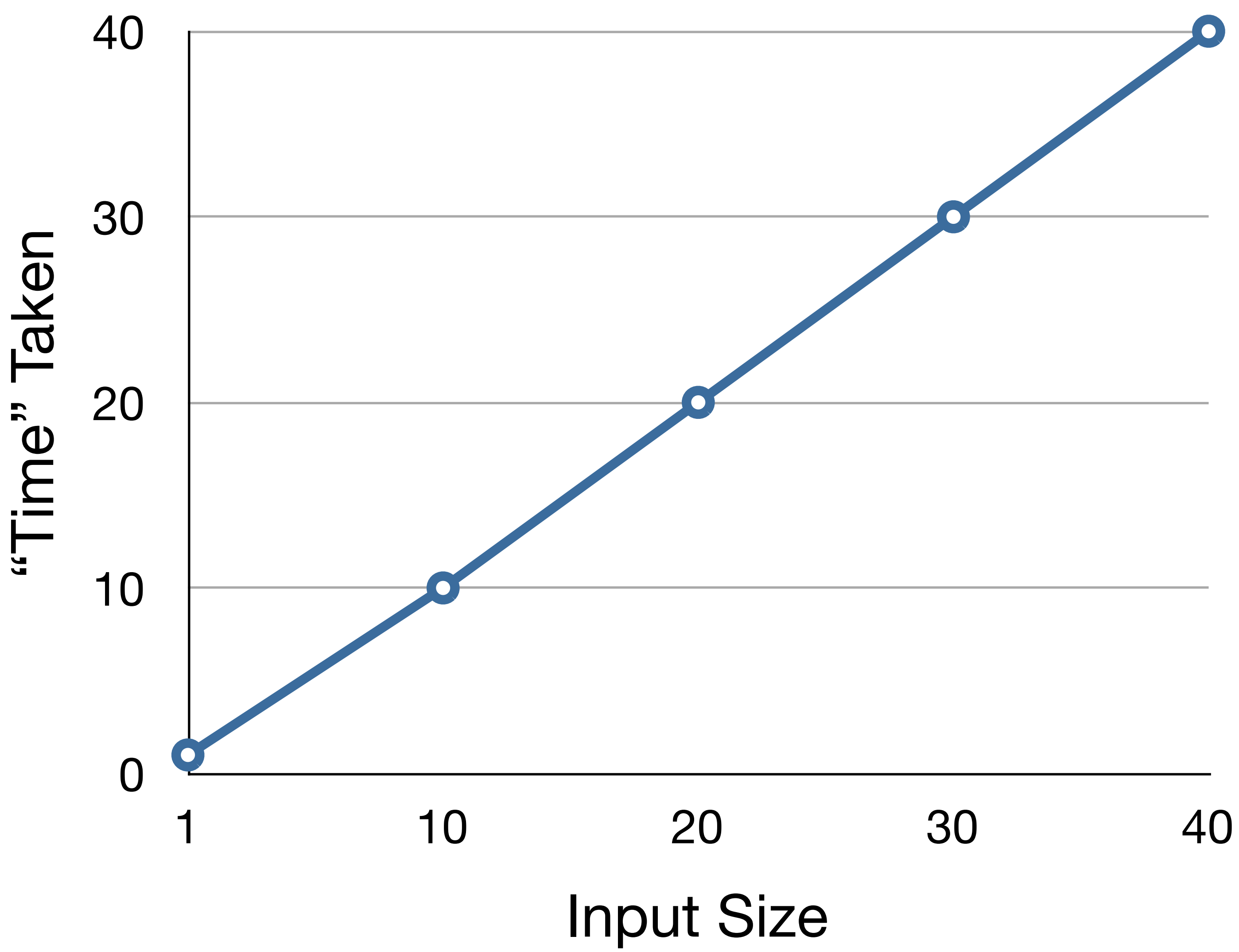
O(1)

- Regardless of the size of the input, it takes the same amount of time



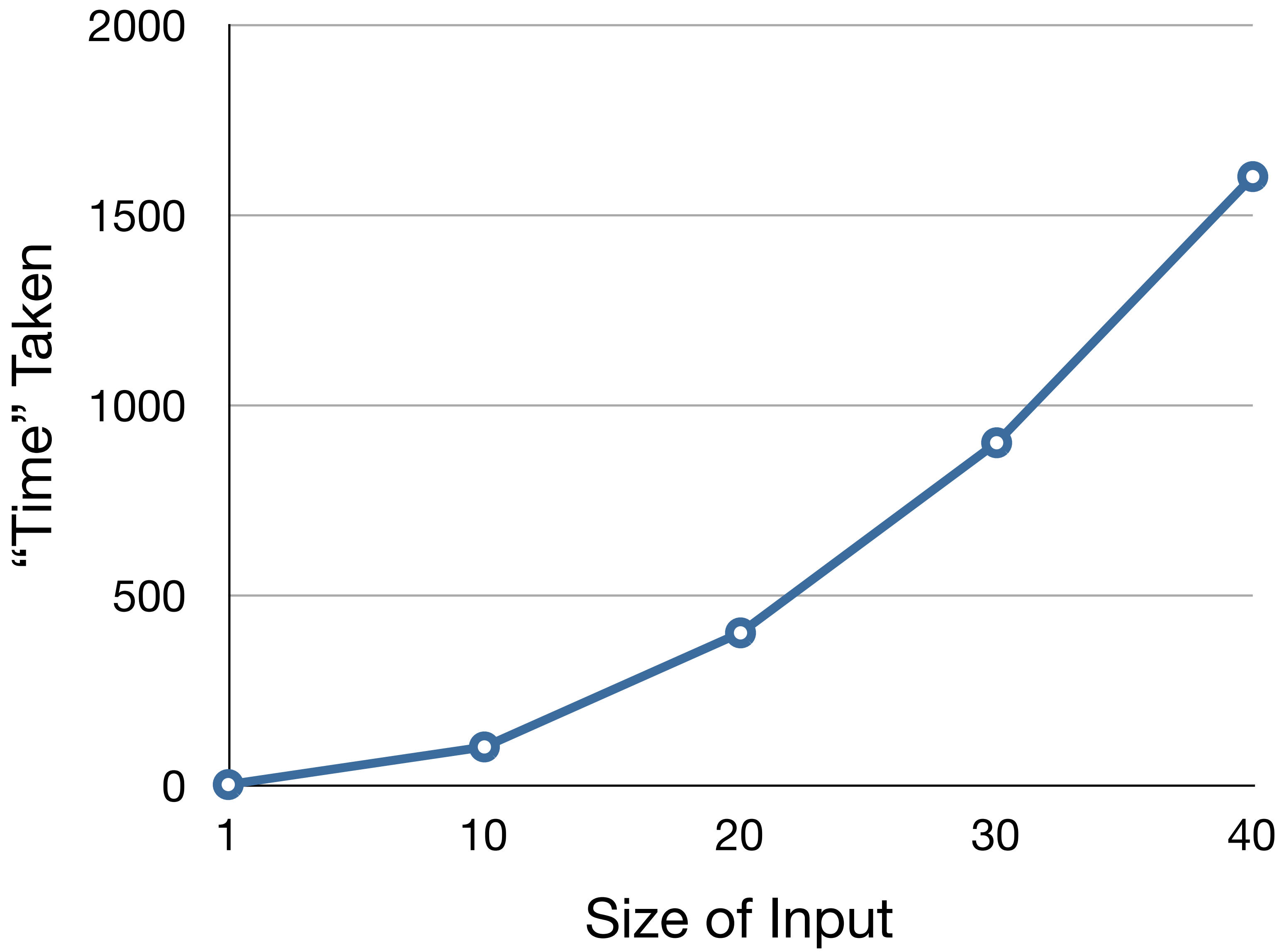
O(N)

- The amount of time taken increases linearly with the input size



O(n^2)

- The amount of time increases quadratically with input size



Determining Big O

```
int sum(int* arr, int length) {
  int s = 0, x;
  for (x = 0; x < length; x++) {
    s += arr[x];
  }
  return s;
}
```

Determining Big O

```
int sum(int* arr, int length) {
  int s = 0, x;
  for (x = 0; x < length; x++) {
    s += arr[x];
  }
  return s;
}
```

Constant time, done once. Call this c_1.

Determining Big O

```
int sum(int* arr, int length) {
   int s = 0, x;
   for (x = 0; x < length; x++) {
      s += arr[x];
   }
   return s;
}
```

Constant time, done once. Call this c_2.

Determining Big O

```
int sum(int* arr, int length) {
   int s = 0, x;
   for (x = 0; x < length; x++) {
      s += arr[x];
   }
   return s;
}
```

Constant time, done `length` times. Call this c_3.

Determining Big O

```
int sum(int* arr, int length) {
   int s = 0, x;
   for (x = 0; x < length; x++) {
      s += arr[x];
   }
   return s;
}
```

Constant time, done `length` times. Call this c_4.

Determining Big O

```
int sum(int* arr, int length) {
   int s = 0, x;
   for (x = 0; x < length; x++) {
      s += arr[x];
   }
   return s;
}
```

Constant time, done `length` times. Call this c_5.

Determining Big O

```
int sum(int* arr, int length) {
   int s = 0, x;
   for (x = 0; x < length; x++) {
      s += arr[x];
   }
   return s;
}
```

Constant time, done once. Call this c_6.

Determining Big O

- Putting it together, we get the formula:

```
c1 + c2 + (c3 * length) + (c4 * length)
        + (c5 * length) + c6
```

Determining Big O

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
c1 + c2 + (c3 * length) + (c4 * length)
        + (c5 * length) + c6
```

Determining Big O

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
1 + 1 + (1 * length) + (1 * length) +
(1 * length) + 1
```

Determining Big O

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
3 + length + length + length
```

Determining Big O

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
3 + 3(length)
```

Determining Big O

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
1 + length
```

Determining Big O

- With sums, we always choose the larger sum
- A variable is always larger than a constant

```
1 + length
```

Determining Big O

- With sums, we always choose the larger sum
- A variable is always larger than a constant

`length`

Determining Big O

- Observe that `length` is really N, the input size
- For this example, we are done

```
length
```

Determining Big O

- Observe that `length` is really N, the input size
- For this example, we are done

`O(N)`

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Constant time, done once. Call this c_1.

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Constant time, done once. Call this c_2.

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Constant time, done `length` times. Call this `c3`.

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Constant time, done `length` times. Call this c_4.

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Constant time, done `length` times. Call this `c5`.

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Constant time, done `length * length` times.

Call this c_6.

Another Example

```
int sum2(int* arr, int length) {
   int s = 0, x, y;
   for (x = 0; x < length; x++) {
      for (y = 0; y < length; y++) {
         s += arr[x] + arr[y];
      }
   }
   return s;
}
```

Constant time, done `length * length` times.

Call this c_7.

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Constant time, done `length * length` times.

Call this c_8.

Another Example

```
int sum2(int* arr, int length) {
  int s = 0, x, y;
  for (x = 0; x < length; x++) {
    for (y = 0; y < length; y++) {
      s += arr[x] + arr[y];
    }
  }
  return s;
}
```

Constant time, done once. Call this `c9`.

Putting it Together

- We are left with the following formula:

$$c_1 + c_2 + (\texttt{length} * c_3) + (\texttt{length} * c_4) + (\texttt{length} * c_5) + (\texttt{length} * \texttt{length} * c_6) + (\texttt{length} * \texttt{length} * c_7) + (\texttt{length} * \texttt{length} * c_8) + c_9$$

Putting it Together

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

$$c_1 + c_2 + (\texttt{length} * c_3) + (\texttt{length} * c_4) + (\texttt{length} * c_5) + (\texttt{length} * \texttt{length} * c_6) + (\texttt{length} * \texttt{length} * c_7) + (\texttt{length} * \texttt{length} * c_8) + c_9$$

Putting it Together

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
1 + 1 + (length * 1) + (length * 1) +
(length * 1) + (length * length * 1) +
(length * length * 1) +
(length * length * 1) + 1
```

Putting it Together

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
3 + length + length + length + (length *
     length) + (length * length) +
          (length * length)
```

Putting it Together

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
3 + 3(length) + 3(length * length)
```

Putting it Together

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

```
1+ length + (length * length)
```

Putting it Together

- The specific values of constants are unimportant as long as they are positive
- We can replace all these with the value 1 as far as Big O notation is concerned

$$1 + \texttt{length} + \texttt{length}^2$$

Putting it Together

- With sums, we always choose the larger sum
- A variable is always larger than a constant

$$\texttt{1+ length + length}^2$$

Putting it Together

- With sums, we always choose the larger sum
- A variable is always larger than a constant

$$\texttt{length} + \texttt{length}^2$$

Putting it Together

- With sums, we always choose the larger sum
- A variable is always larger than a constant

`length`^{2}

Putting it Together

- Observe that `length` is really N, the input size
- For this example, we are done

`length`^{2}

Putting it Together

- Observe that `length` is really N, the input size
- For this example, we are done

$\mathtt{O(N^2)}$

Big O Heuristics

- A non-loop is often `O(1)`
- A single loop is often `O(N)`
- A singly nested loop is often $O(N^2)$
- Not always true though - we will see exceptions later in this class
  - Determining time complexity can be quite difficult in general