Overview

• Complexity and complexity analysis
Complexity
Complexity

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

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

```c
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”

```c
int foo(int* array) {
    return array[2] * array[3];  // cheap(?)
}

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

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

```c
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.
$\mathcal{O}(N)$

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

![Graph showing a linear relationship between input size and time taken.](image)
\( O(n^2) \)

- The amount of time increases quadratically with input size
Determining Big O

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

```c
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** $c_1$.  

---

Thursday, July 24, 14
Determining Big O

```c
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 $c_2$. 

Thursday, July 24, 14
Determining Big O

```c
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** $c_3$.  

Thursday, July 24, 14
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

```c
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 $c_5$.  

Thursday, July 24, 14
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:

\[ c_1 + c_2 + (c_3 \times \text{length}) + (c_4 \times \text{length}) + (c_5 \times \text{length}) + c_6 \]
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

\[ c_1 + c_2 + (c_3 \times \text{length}) + (c_4 \times \text{length}) + (c_5 \times \text{length}) + c_6 \]
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

\[ 1 + 1 + (1 \times \text{length}) + (1 \times \text{length}) + (1 \times \text{length}) + 1 \]
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

\[ 3 + \text{length} + \text{length} + \text{length} \]
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

\[3 + 3 \cdot \text{(length)}\]
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

\[ 1 + \text{length} \]
Determining Big O

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

\[ 1 + \text{length} \]
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
Determining Big O

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

\[ O(N) \]
Another Example

```c
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

```c
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

```c
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

```c
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 $\text{length}$ times. Call this $c_4$. 
Another Example

```c
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

```c
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* \(length \times length\) *times.*

Call this \(c_6\).
Another Example

```c
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 C7.*
Another Example

```c
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** \( \text{length} \times \text{length} \) 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 \( C_9 \).
Putting it Together

• We are left with the following formula:

\[ c_1 + c_2 + (\text{length} \times c_3) + (\text{length} \times c_4) + \\
(\text{length} \times c_5) + (\text{length} \times \text{length} \times c_6) + \\
(\text{length} \times \text{length} \times c_7) + \\
(\text{length} \times \text{length} \times c_8) + c_9 \]
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.

\[ c_1 + c_2 + (\text{length} \times c_3) + (\text{length} \times c_4) + (\text{length} \times c_5) + (\text{length} \times \text{length} \times c_6) + (\text{length} \times \text{length} \times c_7) + (\text{length} \times \text{length} \times c_8) + c_9 \]
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

\[ 1 + 1 + (\text{length} \times 1) + (\text{length} \times 1) + (\text{length} \times 1) + (\text{length} \times \text{length} \times 1) + (\text{length} \times \text{length} \times 1) + 1 \]
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 + \text{length} + \text{length} + \text{length} + (\text{length} \times \text{length}) + (\text{length} \times \text{length}) + (\text{length} \times \text{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

\[ 3 + 3 \text{(length)} + 3 \text{(length} \times \text{length}) \]
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.

\[ 1 + \text{length} + (\text{length} \times \text{length}) \]
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

\[ 1 + \text{length} + \text{length}^2 \]
Putting it Together

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

\[ 1 + \text{length} + \text{length}^2 \]
Putting it Together

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

\[\text{length} + \text{length}^2\]
Putting it Together

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

\[ \text{length}^2 \]
Putting it Together

- Observe that $\text{length}$ is really $N$, the input size
- For this example, we are done

$\text{length}^2$
Putting it Together

- Observe that \texttt{length} is really \( N \), the input size
- For this example, we are done

\( O(N^2) \)
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