Write a recurrence relation that models the following code. It should count the maximum number of calls to any when values has N elements.

```python
def any(f, values):
    if not values:
        return False
    elif f(values[0]):
        return True
    else:
        return any(f, values[1:]
```

\[ T(0) = 1 \]

\[ T(N) = 1 + T(N-1) \]
How many times does the platypus quack?

Theoretical tools: code → math

platypus.quack();
Theoretical tools: code → math

How many times does the platypus quack?

```java
for (int j=0; j<N; j++) {
    platypus.quack();
}
```
Theoretical tools: code $\rightarrow$ math

summations

$$\sum_{i=1}^{5} i = 1 + 2 + 3 + 4 + 5$$
Theoretical tools: code $\rightarrow$ math

summations

\[
\sum_{j=0}^{N-1} 1
\]
Theoretical tools: code $\rightarrow$ math

Summations

$$\sum_{j=0}^{N-1} 1$$

<table>
<thead>
<tr>
<th>$j$</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>N-2</td>
<td>1</td>
</tr>
<tr>
<td>N-1</td>
<td>1</td>
</tr>
</tbody>
</table>

$N \in O(N)$
Theoretical tools: code $\rightarrow$ math

summations

$$\sum_{i=1}^{N} 1$$

<table>
<thead>
<tr>
<th>$i$</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N-1</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
</tr>
</tbody>
</table>

$N \in O(N)$
Theoretical tools: code $\rightarrow$ math

summations

$$\sum_{i=1}^{N} N$$

<table>
<thead>
<tr>
<th>$i$</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>N</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N-1</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

$N^2 \in O(N^2)$
Theoretical tools: code $\rightarrow$ math

Summations

$$\sum_{i=1}^{N} i$$

<table>
<thead>
<tr>
<th>i</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>N-1</td>
<td>N-1</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \in O(N^2)$$
Theoretical tools: code $\rightarrow$ math

How many times does the platypus quack?

code

```java
for (int j=0; j<N; j++) {
    platypus.quack();
}
```

math

$$\sum_{j=0}^{N-1} 1$$

Wolfram Alpha

sum $1, j=0$ to $N-1$

closed form

$$N$$

asymptotic notation

$$O(N)$$
Theoretical tools: code → math

How many times does the platypus quack?

```java
for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
        platypus.quack();
    }
}
```
Theoretical tools: code $\rightarrow$ math

How many times does the platypus quack?

```java
for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
        platypus.quack();
    }
}
```

Math

$$
\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} 1
$$

Wolfram Alpha

sum (sum 1, j=0 to N-1), i=0 to N-1

Closed form

$$N^2$$

Asymptotic notation

$$O(N^2)$$
Theoretical tools: code → math

How many times does the platypus quack?

```java
for (int i=0; i<N; i++) {
    for (int j=i; j<N; j++) {
        platypus.quack();
    }
}
```
Theoretical tools: code $\rightarrow$ math

How many times does the platypus quack?

```java
for (int i=0; i<N; i++) {
    for (int j=i; j<N; j++) {
        platypus.quack();
    }
}
```

\[
\sum_{i=0}^{N-1} \sum_{j=i}^{N-1} 1
\]

Wolfram Alpha

sum (sum 1, j=i to N-1), i=0 to N-1

closed form

\[
\frac{N(N + 1)}{2}
\]

asymptotic notation

\(O(N^2)\)
Sorting algorithms

Things to consider

Theory vs Practice — Algorithms vs Implementations
Theoretical best-case performance on worst-case input: $n \log n$

Is the algorithm in-place?
Does it use space efficiently?

Is the algorithm adaptive?
Does it perform well when the data is already sorted?

What are we measuring / modeling / optimizing for?
comparisons vs swaps • time vs space vs energy vs codability
Results

vote here: tinyurl.com/cs42sortdetective
<table>
<thead>
<tr>
<th>Alg.</th>
<th>Input</th>
<th>Math</th>
<th>Closed Form</th>
<th>Asymptotic</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Sorted</td>
<td>( \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} 1 )</td>
<td>( \frac{N(N-1)}{2} )</td>
<td>( O(N^2) )</td>
</tr>
<tr>
<td></td>
<td>Antisorted</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Selection sort (2)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Sorted</td>
<td>( T(1) = 0 )</td>
<td>( N \log_2(N) )</td>
<td>( O(N \log N) )</td>
</tr>
<tr>
<td></td>
<td>Antisorted</td>
<td>( T(N) = N + 2T\left(\frac{N}{2}\right) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Merge sort (1)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Sorted</td>
<td>( \sum_{i=1}^{N} \sum_{j=0}^{N-2} 1 )</td>
<td>( N(N-1) )</td>
<td>( O(N^2) )</td>
</tr>
<tr>
<td></td>
<td>Antisorted</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bubble sort (4)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Sorted</td>
<td>( \sum_{i=1}^{N-1} 1 )</td>
<td>( N - 1 )</td>
<td>( O(N) )</td>
</tr>
<tr>
<td></td>
<td>Antisorted</td>
<td>( \sum_{i=1}^{N-1} \sum_{j=1}^{i} 1 )</td>
<td>( \frac{N(N-1)}{2} )</td>
<td>( O(N^2) )</td>
</tr>
<tr>
<td><strong>Insertion sort (3)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More fun with sorting

Here are some more ways to learn about sorting algs:

- On Wikipedia
- Using visualizations
- Using sonifications
- Using folk-dancification