Let’s make life easier by only worrying about asymptotic costs.

**Definition.** Given two functions $f$ and $g$, both $\mathbb{N} \to \mathbb{R}$, we say $f(n)$ is $O(g(n))$ if

$$\exists \kappa > 0 \text{ and } n_0 \in \mathbb{N} \text{ such that } \forall n \geq n_0, |f(n)| \leq \kappa |g(n)|$$

and we say $f(n)$ is $\Omega(g(n))$ if

$$\exists \delta > 0 \text{ and } n_0 \in \mathbb{N} \text{ such that } \forall n \geq n_0, |f(n)| \geq \delta |g(n)|.$$ 

If $f(n)$ is $O(g(n))$ and also $\Omega(g(n))$ we say that $f(n)$ is $\Theta(g(n))$.

Let $f(n) = \frac{1}{2} k_1 n (n-1) + k_2 (n-1)$, $k_1, k_2$ constants.

Then $f(n)$ is $O(n^2)$ since $f(n) \leq \frac{1}{2} k_1 n^2 + k_2 n = n^2 \left( \frac{\frac{k_1}{2n} + \frac{k_2}{n}}{n} \right) \leq 2n^3$ for $n \geq \max(\frac{k_1}{2}, k_2)$.

But also $f(n)$ is $O(n^2)$ by similar reasoning. And $O(n^3)$. And... 

Also, $f(n)$ is $\Omega(n^2)$, and $O(\log n)$, and $\Omega(1)$...

by similar reasoning.

Since $f(n)$ is $O(n^2)$, and $\Omega(n^2)$, it is $\Theta(n^2)$. 

Last time: the InsertSort algorithm, on an array of length $n$, has running time $\leq \frac{1}{2} k_1 n (n-1) + k_2 (n-1)$. 
In this course, we’re typically interested in an algorithm’s worst-case running time as a function of input size.

We’ve shown that for every input $x$ of size $n$, the cost is $\leq \kappa n^2$ (for some $\kappa > 0$, and sufficiently large $n$). In other words, all the blue dots are $\leq \kappa n^2$.

In other words, the purple circles are $\leq \kappa n^2$.

In other words, if we define the worst-case cost to be $h(n) = \max_{x: \text{size}(x)=n} \text{cost}(x)$, then $h(n)$ is $O(n^2)$.

Can we find a matching $\Omega$ bound, i.e. show that $h(n)$ is $\Omega(n^2)$?

In other words, can we show that the purple circles are $\geq \delta n^2$ (for some $\delta > 0$, and sufficiently large $n$)?

In other words, can we find for each $n$ a specific input $x$ whose cost is $\geq \delta n^2$?
In this course, we’re typically interested in an algorithm’s worst-case running time as a function of input size.

```python
def insert_sort(x):
    for i in 1..(len(x)-1):
        j = i - 1
        while j >= 0 and x[j] > x[j+1]:
            swap x[j] with x[j+1]
            j = j - 1
```

Q. Given an arbitrary $n$, what is an input of size $n$ that gives the worst possible running time?

For input $[n, n-1, \ldots, 1]$ cost is $\sum 2(n^2)$

ChatGPT struggles with $\exists$ problems. For example, see the “vulnerability report” at https://hackerone.com/reports/2298307
After we show that our algorithm is $O(n^2)$, it’s good manners to also demonstrate that the worst case is $\Omega(n^2)$.

There is some $\delta > 0$ for which $\forall$ sufficiently large $n$, $\exists$ a problem of size $n$ with cost $\geq \delta n^2$.

There is some $\kappa > 0$ for which $\forall$ sufficiently large $n$ and $\forall$ problems of size $n$ cost $\leq \kappa n^2$. 
Simple sorting algorithms compared

Two optimal algorithms

Better than optimal!?
2.5 Minimum cost of sorting

Can we do better than InsertSort’s $\Theta(n^2)$ worst-case running time?

Complexity of Comparison Sort?

- typically count the number of comparisons $C(n)$
- there are $n!$ permutations of $n$ elements
- each comparison eliminates half of the permutations $2^{C(n)} \geq n!$
- therefore $C(n) \geq \log(n!) \approx n \log n - 1.44n$
- The lower bound of comparison is $O(n \log n)$

**Properly-stated theorem**

Given any sorting alg. A

Let $g_A(x) = \#$ comparisons when we run A on input $x$

Let $f_A(n) = \max_{x:s_i(x) = n} g_A(x)$

Then $f_A(n)$ is $\leq (n \log n)$. 

*Alert! We don’t expect to see “lower bound” and “$O$” in the same sentence!*
§2.7 Binary InsertSort
Can we sort using only $O(n \log n)$ comparisons?

```python
def insert_sort(x):
    for i in 1..(len(x)-1):
        do a linear search for where $x[i]$ should go, and insert it there

def binary_insert_sort(x):
    for i in 1..(len(x)-1):
        do a binary search for where $x[i]$ should go, and insert it there
```
QUESTION
What’s a big-$O$ bound on the number of comparisons for BinaryInsertSort?

$x = \lfloor x \rfloor < 1+x$

$\log n! \text{ is } \Theta(n \log n)$.

```python
def binary_insert_sort(x):
    for i in 1..(len(x)-1):
        do a binary search for where $x[i]$ should go, and insert it there
```

# comparisons to place $x[i] \leq \lceil \log_2 (i+1) \rceil$
So total # comparisons

\[ \leq \sum_{i=1}^{n-1} \lceil \log_2 (i+1) \rceil + \sum_{i=1}^{n-1} (1 + \log_2 (i+1)) \]

\[ = n-1 + \sum_{i=1}^{n-1} \log_2 (i+1) \]

\[ = n-1 + \log_2 \left[ n^2(n-1) \cdots 2 \right] \geq n-1 + \log_2 n! \]

We used $\leq$ right at the beginning. This contaminates all the rest of the working, and means we can only end up with a $O(\cdot)$ conclusion.

So total # comparisons is $O(n \log n)$. for some $K$ for $n$ sufficiently large.
QUESTION
What’s the asymptotic worst-case number of swaps?

Recall: sum of arithmetic series.
\[ 1 + 2 + \ldots + n = \frac{1}{2} n(n+1) \]

- To place \( x[i] \) we might need \( i \) swaps.
  - Total \# swaps = \( \sum_{i=1}^{n-1} i \)
  - So worst-case total \# swaps is \( \mathcal{O}(n^2) \)
- Thinking of the input \( [n, n-1, \ldots, 1] \),
  - Worst-case total \# swaps is \( \Omega(n^2) \)

```python
def binary_insert_sort(x):
    for i in 1..(len(x)-1):
        do a binary search for where \( x[i] \) should go, and insert it there
```
§2.6 SelectSort

What’s a lower bound for the worst-case number of swaps to sort an array of length $n$?

**Theorem.** For any sorting algorithm, the worst-case number of swaps is $\Omega(n)$.

**Proof.** Given arbitrary $n$, consider the input $x = [2, 3, ..., n, 1]$.

Every item starts in the wrong place, so every item needs to be “touched” by a swap.

Each swap touches two items.

Thus $\#\text{swaps} \geq \lceil \frac{n}{2} \rceil$, which is $\Omega(n)$.

Can we sort using only $O(n)$ swaps?

```python
def select_sort(x):
    for i in range(len(x)-2):
        # Find what belongs in x[i]
        j = arg min x[k]
        swap x[i] with x[j]
```

**QUESTION**

What’s the asymptotic worst-case number of comparisons?

Total # comparisons = $\sum_{i=0}^{n-2} (n-i-1) = \Theta(n^2)$.
<table>
<thead>
<tr>
<th></th>
<th>comparisons</th>
<th>swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>any algorithm</td>
<td>worst case is $\Omega(n \log n)$</td>
<td>worst case is $\Omega(n)$</td>
</tr>
<tr>
<td>InsertSort</td>
<td>worst case is $O(n^2)$ worst case is $\Omega(n^2)$</td>
<td>worst case is $O(n^2)$ worst case is $\Omega(n^2)$</td>
</tr>
<tr>
<td>BinaryInsertSort</td>
<td>worst case is $O(n \log n)$</td>
<td></td>
</tr>
<tr>
<td>SelectSort</td>
<td>every case is $\Theta(n^2)$</td>
<td>worst case is $O(n)$</td>
</tr>
</tbody>
</table>
Here is a concrete example input. It demonstrates a lower bound on worst-case running time.

Here is a universal argument about the worst that can happen. It demonstrates an upper bound on running time.

If our bounds don’t agree, we should think harder!
- Can we find a better example, one that hits our upper bound?
- Or maybe the algorithm isn’t as bad as we thought: can we find a tighter upper bound?
As well as $O$ and $\Omega$ and $\Theta$, we also use $o$ and $\omega$ [see notes]

$O$ is pronounced “big-O”
$o$ is pronounced “little-o”
$\Omega$ is pronounced “big-Omega”
$\omega$ is pronounced “little-omega”

literally means big $o$!