Algorithm performance

Sorting analysis

By the end of this video you will be able to…

- State and justify the asymptotic performance for
  - selection sort
  - insertion sort

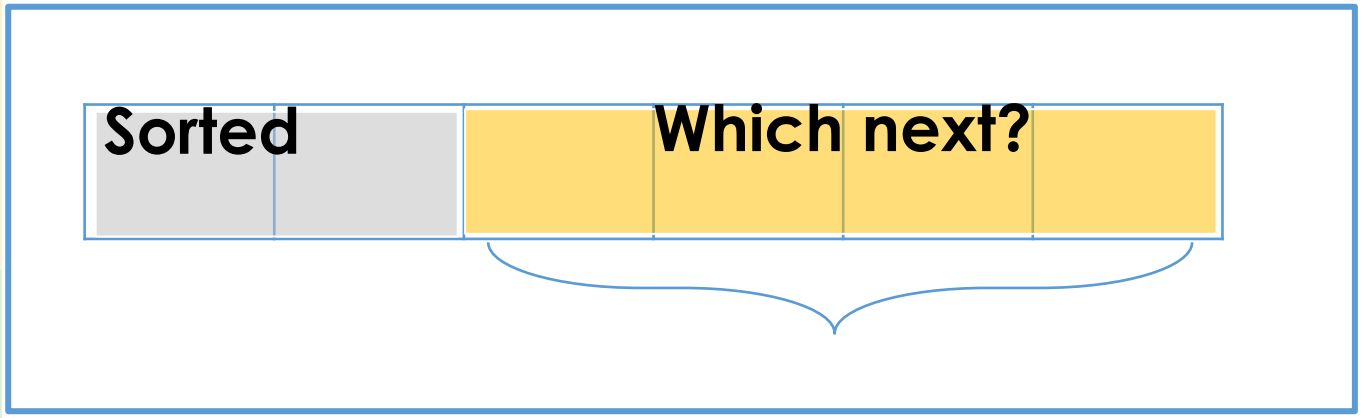
in the best case and in the worst case

	Best case	Worst case
Selection Sort		
Insertion Sort		

Selection Sort: Basic Algorithm

For each **position `i`** from `0` to **`length-2`**

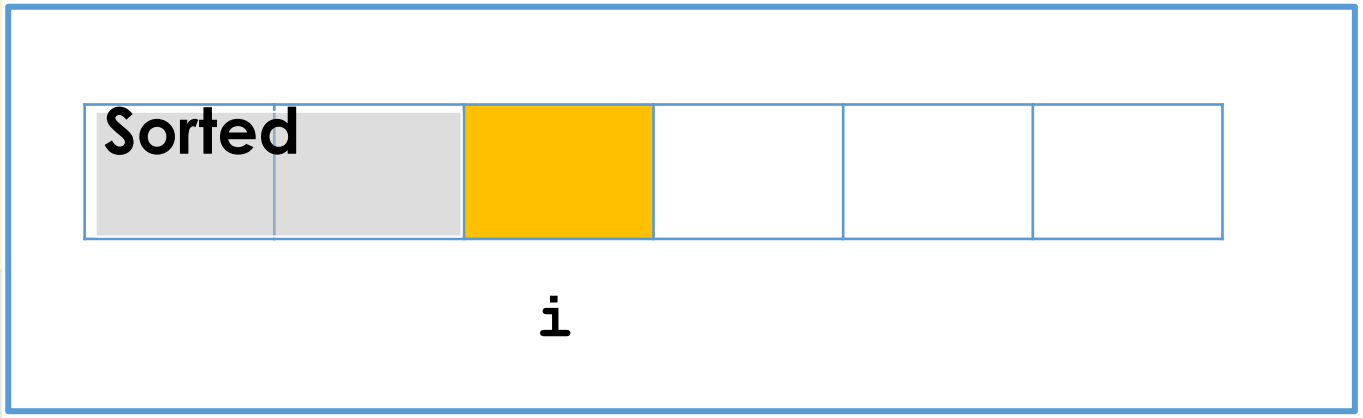
Find smallest element in **positions `i` to `length-1`**
Swap it with element in **position `i`**



Insertion Sort: Basic Algorithm

For each **position `i`** from **`1`** to **`length-1`**

Swap successive pairs to put value in position **`i`** in correct location relative to earlier values



	Best case	Worst case
Selection Sort		
Insertion Sort		

Selection sort $O(n^2)$

```
public static void selectionSort( int[] vals )    {

  int indexMin;

  for ( int i=0; i < vals.length-1 ; i++ ) {

       indexMin = i ;
       for ( int j=i+1; j < vals.length; j++ ) {
         if ( vals[j] < vals[indexMin] ) {
              indexMin = j ;
         }
       }

       swap ( vals, indexMin , i );
  }
}
```

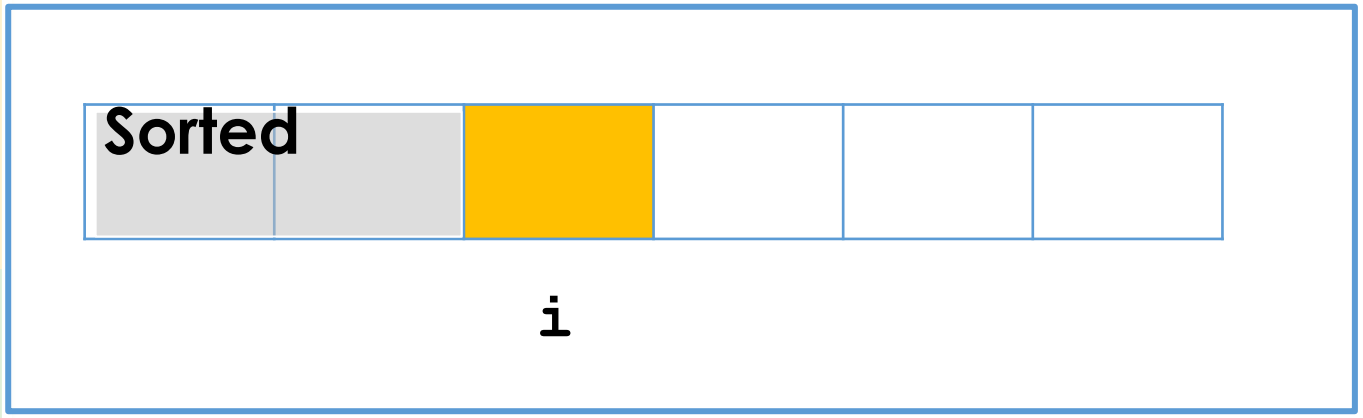
	Best case	Worst case
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort		

	Best case	Worst case
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	?	?

Insertion Sort: Basic Algorithm

For each **position `i`** from **`1`** to **`length-1`**

Swap successive pairs to put value in position **`i`** in correct location relative to earlier values



```
public static void insertionSort( int[] vals )    {
   int currInd;
   for ( int pos=1; pos < vals.length ; pos++ ) {
      currInd = pos ;
      while ( currInd > 0 &&
            vals[currInd] < vals[currInd-1] ) {
        swap(vals, currInd, currInd-1);
        currInd = currInd - 1;
      }
   }
}
```

```
public static void insertionSort( int[] vals )    {
   int currInd;
   for ( int pos=1; pos < vals.length ; pos++ ) {
      currInd = pos ;
      while ( currInd > 0 &&
            vals[currInd] < vals[currInd-1] ) {
        swap(vals, currInd, currInd-1);
        currInd = currInd - 1;
      }
   }
}
```

	Best case	Worst case
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	?	?

```
public static void insertionSort( int[] vals )    {
   int currInd;
   for ( int pos=1; pos < vals.length ; pos++ ) {
      currInd = pos ;
      while ( currInd > 0 &&
            vals[currInd] < vals[currInd-1] ) {
        swap(vals, currInd, currInd-1);
        currInd = currInd - 1;
      }
   }
}
```

```
public static void insertionSort( int[] vals )     {
   int currInd;
   for ( int pos=1; pos < vals.length ; pos++ ) {
      currInd = pos ;
      while ( currInd > 0 &&
            vals[currInd] < vals[currInd-1] ) {
         swap(vals, currInd, currInd-1);
         currInd = currInd - 1;
      }
   }
}
```

```
public static void insertionSort( int[] vals )    {
   int currInd;
   for ( int pos=1; pos < vals.length ; pos++ ) {
      currInd = pos ;
      while ( currInd > 0 &&
            vals[currInd] < vals[currInd-1] ) {
         swap(vals, currInd, currInd-1);
         currInd = currInd - 1;
      }
   }
}
```

1	2	3	4	5

	Best case	Worst case
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	?

when already sorted!

	Best case	Worst case
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	?

```
public static void insertionSort( int[] vals )     {
   int currInd;
   for ( int pos=1; pos < vals.length ; pos++ ) {
      currInd = pos ;
      while ( currInd > 0 &&
            vals[currInd] < vals[currInd-1] ) {
         swap(vals, currInd, currInd-1);
         currInd = currInd - 1;
      }
   }
}
```

```
public static void insertionSort( int[] vals )     {
   int currInd;
   for ( int pos=1; pos < vals.length ; pos++ ) {
      currInd = pos ;
      while ( currInd > 0 &&
            vals[currInd] < vals[currInd-1] ) {
         swap(vals, currInd, currInd-1);
         currInd = currInd - 1;
      }
   }
}
```

5	4	3	2	1

	Best case	Worst case
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$

when in reverse order