Selection Sort

- Let A be an array of n elements, and we wish to sort these elements in non-decreasing order.
- Basic Algorithm:
  - Set i = 0.
  - While i < n do the following:
    - Find j, i ≤ j ≤ n-1, such that A[j] < A[k], ∀k, i ≤ k ≤ n-1.
    - Swap A[j] with A[i]
    - Add 1 to i
- This algorithm works in place, meaning it uses its own storage to perform the sort.
Selection Sort

- General Idea:

```
+-------+-------+
|       |       |
| i     | j     |
+-------+-------+
```

```
A
```
```
SORTED
```
```
UNSORTED
```
```
min
```

Loop invariant: A[0..i-1] are sorted in non-decreasing order.

Selection Sort Example

```
66  44  99  55  11  88  22  77  33
11  44  99  55  66  88  22  77  33
11  22  99  55  66  88  44  77  33
11  22  33  55  66  88  44  77  99
11  22  33  44  66  88  55  77  99
11  22  33  44  55  88  66  77  99
11  22  33  44  55  66  88  77  99
11  22  33  44  55  66  77  88  99
11  22  33  44  55  66  77  88  99
```
Run time analysis

- Worst Case:
  - Search for 1st min: \( n-1 \) comparisons
  - Search for 2nd min: \( n-2 \) comparisons
  - ... 
  - Search for 2nd-to-last min: 1 comparison
  - Total comparisons: \( (n-1) + (n-2) + ... + 2 + 1 \) = \( O(______) \)

- Average Case: = \( O(______) \)

- Best Case: = \( O(______) \)

Insertion Sort

- Let A be an array of n elements, and we wish to sort these elements in non-decreasing order.

- Basic Algorithm:
  - Set \( i = 1 \)
  - While \( i < n \) do the following:
    - Set \( item = A[i] \).
    - Let \( k \) be the smallest \( j \) above, or \( k=i \) if no shifts.
    - Set \( A[k] = item \).
    - Add 1 to \( i \).

- This algorithm also works in place.
Insertion Sort

- General Idea:

```
  k     i
  SORTED   UNSORTED
    insert
```

Loop invariant: A[0..i-1] are sorted in non-decreasing order.

Insertion Sort Example

```
 66  44  99  55  11  88  22  77  33
 44  66  99  55  11  88  22  77  33
 44  66  99  55  11  88  22  77  33
 44  55  66  99  11  88  22  77  33
 11  44  55  66  99  88  22  77  33
 11  44  55  66  88  99  22  77  33
 11  22  44  55  66  88  99  77  33
 11  22  44  55  66  77  88  99  33
 11  22  33  44  55  66  77  88  99
```
Run time analysis

- Worst Case (when does this occur?):
  Insert 2\textsuperscript{nd} element: 1 comparison
  Insert 3\textsuperscript{rd} element: 2 comparisons
  ...
  Insert last element: n-1 comparisons
  Total comparisons:
  \[1 + 2 + \ldots + (n-1)\]
  = O(\_\_\_\_\_\_)

- Average Case:
  = O(\_\_\_\_\_\_)

- Best Case:
  = O(\_\_\_\_\_\_)

Stable Sorts

- A sort is \underline{stable} if two elements with the same value maintain their same relative order before and after the sort is performed.

\begin{array}{cccc}
  & & \times & \times \\
\end{array}

After stable sort:

\begin{array}{cccc}
  \times & \times & & \\
\end{array}

- Is selection sort stable?
- Is insertion sort stable?
Quadratic Sorts

- Quadratic sorts have a worst-case order of complexity of $O(n^2)$
- Selection sort always performs poorly, even on a sequence of sorted elements!
- Insertion sort performs much better if the elements are sorted or nearly sorted.
- Another famous quadratic sort: “Bubble sort”

Tree Sort

- Build a binary search tree out of the elements.
- Traverse the tree using an inorder traversal to get the elements in increasing order.
- Worst case order of complexity:
  - $O(n^2)$ to build the binary search tree (Why?)
  - $O(n)$ to traverse the binary tree. (Why?)
  - Total: $O(n^2) + O(n) = \_\_\_\_$