Selection Sort

Selection Sort Algorithm (ascending)

1. Find smallest element (of remaining elements).
2. Swap smallest element with current element (starting at index 0).
3. Finished if at the end of the array. Otherwise, repeat 1 and 2 for the next index.

Selection Sort Example(ascending)

- 70 75 89 61 37
  - Smallest is 37
  - Swap with index 0
- 37 75 89 61 70
  - Smallest is 61
  - Swap with index 1
- 37 61 89 75 70
  - Smallest is 70
  - Swap with index 2
- 37 61 70 75 89
  - Smallest is 75
  - Swap with index 3
    - Swap with itself
- 37 61 70 75 89
  - Don't need to do last element because there's only one left
- 37 61 70 75 89

Selection Sort Example(ascending)

- Write out each step as you sort this array of 7 numbers (in ascending order)
- 72 4 17 5 5 64 55
- 4 72 17 5 5 64 55
- 4 5 17 72 5 64 55
- 4 5 5 72 17 64 55
- 4 5 5 17 72 64 55
- 4 5 5 17 55 64 72
- 4 5 5 17 55 64 72
- 4 5 5 17 55 64 72

Swapping

- a = b; b = a; //Does this work?
  - a gets overwritten with b's data
  - b get overwritten with the new data in a (same data now as b)
- Need a temporary variable to store a value while we swap.

  temp = a;

  a = b;

  b = temp;

Selection Sort Code (ascending)

```
public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int minIndex = i;
        int min = arr[minIndex];
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[j] < min) {
                minIndex = j;
                min = arr[minIndex];
            }
        }
        int temp = arr[minIndex];    // swap
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}
```

Selection Sort Algorithm (descending)

1. Find largest element (of remaining elements).
2. Swap largest element with current element (starting at index 0).
3. Finished if at the end of the array. Otherwise, repeat 1 and 2 for the next index.

Selection Sort Example(descending)

- 84 98 35 1 67
  - Largest is 98
  - Swap with index 0
- 98 84 35 1 67
  - Largest is 84
  - Swap with index 1
    - Swap with itself
- 98 84 35 1 67
  - Largest is 67
  - Swap with index 2
- 98 84 67 1 35
  - Largest is 35
  - Swap with index 3
- 98 84 67 35 1
  - Don't need to do last element because there's only one left
- 98 84 67 35 1

Selection Sort Example(descending)

- Write out each step as you sort this array of 7 numbers (in descending order)
- 72 4 17 5 5 64 55
- 72 4 17 5 5 64 55
- 72 64 17 5 5 4 55
- 72 64 55 5 5 4 17
- 72 64 55 17 5 4 5
- 72 64 55 17 5 4 5
- 72 64 55 17 5 5 4
- 72 64 55 17 5 5 4

Selection Sort Code (descending)

```
public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int maxIndex = i;
        int max = arr[maxIndex];
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[j] > max) {
                maxIndex = j;
                max = arr[maxIndex];
            }
        }
        int temp = arr[maxIndex];   // swap
        arr[maxIndex] = arr[i];
        arr[i] = temp;
    }
}
```

Selection Sort Efficiency

- n^2 $comparisons$
  - n is the number of elements in array
- $O(n^2)$ $time\ complexity$
  - Big O notation, will talk about this later
- Inefficient for large arrays

Why use it?

- Memory required is small
  - Size of array (you're using this anyway)
  - Size of one variable (temp variable for swap)
- Selection sort is useful when you have limited memory available
  - Inefficient otherwise when you have lots of extra memory
- Relatively efficient for small arrays