Selection Sort

Selection sort is a sorting algorithm, specifically an in-place comparison sort. Selection sort is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.

It works as follows:

1. Find the minimum value in the list.

2. Swap it with the value in the first position.

3. Repeat the steps above for remainder of the list (starting at the second position).

Selection Sort: Implementation

```
void selectionSort(int a[], int n) {
  for (int i = n-1; i >= 1; i--) {
    int maxIdx = i;
    for (int j = 0; j < i; j++)
      if (a[j] >= a[maxIdx])
        maxIdx = j;
    // swap routine is in STL <algorithm>
    swap(a[i], a[maxIdx]);
  }
}
```

Step 1: Search for maximum element

Step 2: Swap maximum element with the last item i

Example: sort this number (45,20,40,5,15,25,50,35,30,10) using selection sort? (ascending)

Trace of a Selection Sort

Passes →	I	II	III	IV	V	VI	VII	VIII	IX	Sorted Array
A[0] = 45	05	05	05	05	05	05	05	05	05	05
A[1] = 20	20	10	10	10	10	10	10	10	10	10
A[2] = 40	40	40	15	15	15	15	15	15	15	15
A[3] = 05	45	45	45	20	20	20	20	20	20	20
A[4] = 15	15	15	40	40	25	25	25	25	25	25
A[5] = 25	25	25	25	25	40	30	30	30	30	30
A[6] = 50	50	50	50	50	50	50	35	35	35	35
A[7] = 35	35	35	35	35	35	35	50	40	40	40
A[8] = 30	30	30	30	30	30	40	40	50	45	45
A[9] = 10	10	20	20	45	45	45	45	45	50	50

Selection sort is very easy to analyze since none of the loops depend on the data in the array.

Selecting the lowest element requires scanning all n elements (this takes n - 1 comparisons) and then swapping it into the first position.

Finding the next lowest element requires scanning the remaining n - 1 elements and so on, for a total of (n - 1) + (n - 2) + ... + 2 + 1 = O(n2) comparisons.

Each of these scans requires one swap for a total of n - 1 swaps (the final element is already in place).

Thus, the comparisons dominate the running time, which is O(n2).