Selection Sort Time Complexity Derivation and Exploration

Elliot Wyrick 9/9/2025

Selection Sort definition

Selection sort is a sorting algorithm that works by moving linearly through an array or set of unsorted values, comparing each value with all other unsorted values. As it moves through the set, all values “behind” it are sorted. And example in pseudo code follows:

```
Array values[];
for(i from values[0] to values[max - 1]){
    int min = i;
    for(j from values[i + 1] to values[max]){
        if(arr[j] < arr[min]){
            min_index = j;
        }
    }
    if(arr[min] < arr[i]){
        Swap arr[i] and arr[min]
    }
}
```

While Selection Sort works well for small data sets, it becomes intuitively clear that the algorithm slows down and becomes costly as the number of elements increases. For each iteration of the outer loop, which processes each element in the array, the inner loop must perform all necessary comparisons to find the minimum element. As the input size grows, the total number of comparisons increases significantly.

This relationship can be demonstrated well by deriving an expression for the number of comparisons with respect to the number (n), of elements. This expression is commonly known as T(n), which in a more general definition serves to represent the time complexity of an algorithm, or the total number of basic operations with an input of size n. The derivation for the time complexity of Selection sort is as follows:

Outer loop:

$$\sum_{1}^{n-2} 1$$

Inner loop:

$$\sum_{i=i+1}^{n} 1 = (n - 1) - (i + 1) + 1 = (n - i - 1)$$

Together:

$$T(n) = \sum_{1}^{n-2} \sum_{i=i+1}^{n} 1 = \sum_{1}^{n-2} (n - i - 1)$$

$$T(n) = (n - 1) + (n - 2) + \dots + 1 = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

This outcome is significant, due primarily to the n^2 term. A defining feature of polynomials is that the term with the highest order is the dominant term when describing the general behavior of the expression. Knowing this, and knowing that the highest term in this situation is the n^2 means that we should have a general understanding of this algorithm's complexity with respect to element size. This derivation also places this algorithm into the time complexity category of $O(n^2)$, meaning that the worst case scenario is a time complexity of n^2. For this algorithm specifically, The best and worst case are n^2, as it iterates through terms regardless of their actual value, meaning that it will always take T(n) comparisons to sort a set of values.

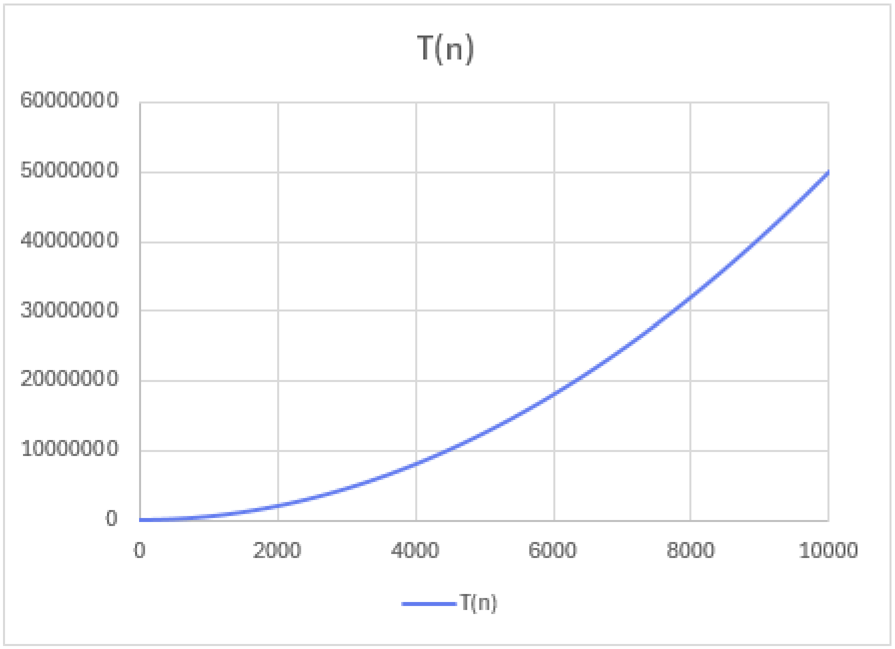

Figure 1 - T(n) = $\frac{n^2-n}{2}$ plotted as a continuous line to demonstrate general behavior

Plotting the derived time complexity as seen in figure 1 reveals what is expected - dominant quadratic behavior. A good pair of input sizes to demonstrate how drastic of a difference this can make with large datasets are 1,000 and 2,000.

$$T(1,000) = \frac{1,000^2 + 1,000}{2} = 499,500$$

$$T(2,000) = \frac{2,000^2 + 2,000}{2} = 1,999,000$$

In this example, it is worth noting that even though the total number of elements to be sorted was doubled, the number of comparisons was nearly quadrupled. This aligns with what we expect with a quadratic system, and raises concerns as to the efficiency of this algorithm as you approach larger datasets, as that divide will only continue to grow.