

Algorithm Complexity

Tutorial 2

Exercise 1

Consider an algorithm SELECTION SORT to sort n numbers stored in a table A . SELECTION SORT starts by finding the smallest element in A and swapping it with the first element of A . Then, the algorithm finds the second smallest element and swaps it with the second position. It continues this process for the remaining elements.

1. Write the algorithm in pseudo-code.
2. State the loop invariant of the algorithm and prove its correctness.
3. Find the best-case and worst-case time complexity.
4. Calculate the algorithm's complexity based on fundamental operations.

Solution

1. Pseudocode for Selection-Sort

The algorithm repeatedly finds the smallest element in the unsorted part of the table and places it at the start in this part.

Algorithm 1: SELECTION-SORT(A)

```
1 for  $i \leftarrow 1$  to  $\text{length}(A) - 1$  do
2    $m \leftarrow i$ ;
3   for  $j \leftarrow i + 1$  to  $\text{length}(A)$  do
4     if  $A[j] < A[m]$  then
5        $m \leftarrow j$ ;
6   end
7 end
8 if  $m \neq i$  then
9    $\text{temp} \leftarrow A[i]$ ;  $A[i] \leftarrow A[m]$ ;  $A[m] \leftarrow \text{temp}$ ;
10 end
11 end
```

The condition $m \neq i$ avoids an unnecessary swap when the smallest element is already in position i .

2. Loop invariant and correctness demonstration

At the start of each iteration of the outer loop, the following invariant holds:

$A[1..i - 1]$ contains the $i - 1$ smallest elements of the original table in sorted order.

Initialization:

Before the first iteration,

$$i = 1.$$

The subtable $A[1..0]$ is empty, so it trivially contains the 0 smallest elements in sorted order. Therefore, the invariant holds.

Maintenance:

Assume that the invariant holds at the start of an iteration with index i .

The inner loop searches the subtable

$$A[i..n]$$

and identifies the position m of its smallest element.

This element is then placed at position i . Since all elements in

$$A[1..i-1]$$

are already the $i-1$ smallest elements in sorted order, placing the smallest element from $A[i..n]$ at position i gives

$$A[1..i]$$

containing the i smallest elements in sorted order.

Thus, the invariant remains valid for the next iteration.

Termination:

The outer loop terminates after the iteration $i = n-1$.

At this point,

$$A[1..n-1]$$

contains the $n-1$ smallest elements in sorted order. The remaining element $A[n]$ must therefore be the largest element.

Hence,

$$A[1] \leq A[2] \leq \dots \leq A[n],$$

and the table is sorted.

SELECTION-SORT correctly sorts the input table.

3. Best-case and worst-case time complexity

The fundamental operation is the comparison

$$A[j] < A[m].$$

For every value of i , the inner loop performs exactly

$$n-i$$

comparisons.

Therefore, the total number of comparisons is

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

Using

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2},$$

we obtain

$$C(n) = \frac{n(n-1)}{2}.$$

Thus,

$$C(n) = \frac{n(n-1)}{2} = \Theta(n^2).$$

This number is exactly the same regardless of the initial ordering of the table.

Consequently,

$$T_{\text{best}}(n) = \Theta(n^2)$$

and

$$T_{\text{worst}}(n) = \Theta(n^2).$$

In fact, the average case also has the same asymptotic complexity:

$$T_{\text{average}}(n) = \Theta(n^2).$$

The reason is that Selection Sort always scans the complete unsorted part of the table to identify its smallest element.

4. Complexity based on fundamental operations

Let $C(n)$ denote the number of executions of the fundamental operation.

We have

$$C(n) = (n - 1) + (n - 2) + \dots + 2 + 1.$$

Therefore,

$$C(n) = \frac{n(n - 1)}{2} = \frac{n^2 - n}{2}.$$

Hence,

$$C(n) = \Theta(n^2).$$

The number of assignments and swaps is smaller:

at most $n - 1$ swaps.

Thus,

number of swaps = $O(n)$,

while

number of comparisons = $\Theta(n^2)$.

Since the quadratic number of comparisons dominates the linear number of swaps and other constant-time operations, the overall running time is

$$\Theta(n^2).$$

The complete complexity summary is therefore:

Case	Time Complexity
Best case	$\Theta(n^2)$
Average case	$\Theta(n^2)$
Worst case	$\Theta(n^2)$
Additional space	$\Theta(1)$

Important observation

Unlike algorithms such as INSERTION-SORT, Selection Sort does not significantly take advantage from an already sorted table. Even when the table is already sorted, it must still perform

$$\frac{n(n-1)}{2}$$

comparisons.

Therefore, the best, average, and worst cases all have the same asymptotic order:

$$\boxed{\Theta(n^2)}.$$