Assignment Two

Josh Seligman

joshua.seligman1@marist.edu

September 22, 2022

1 Selection Sort

1.1 The Algorithm

Selection sort is a sorting algorithm that, for each iteration of the array, selects the smallest (or largest) element of the unsorted part of the array and places the element into its sorted position. As shown in the pseudocode for the sort in Algorithm 1, selection sort works with the subset of the array in the range $[i,\ n)$ in each iteration because each element in an index less than i is already sorted and does not have to be checked. Thus, as more elements get sorted, the quicker each iteration becomes because a smaller portion of the array is compared until $i = n - 1$, which will terminate the algorithm.

Algorithm 1 Selection Sort Algorithm

```
1:  procedure SelectionSort(arr)
2:      for i ← 1, n − 2 do    // Iterate through the second to last element as an array of size 1 is sorted
3:          smallestIndex ← i
4:          for j ← i + 1, n − 1 do    // Iterate through the remainder of the array
5:              if arr[j] < arr[smallestIndex] then
6:                  smallestIndex ← j    // Set the new smallest index if a smaller element is found
7:              end if
8:          end for
9:          swap(arr, i, smallestIndex)    // Place the smallest item in the subarray into its sorted place
10:     end for
11: end procedure
```

1.2 Asymptotic Analysis

Listing 1 contains the C++ code implementing selection sort on lines 6 - 25. The outer loop iterates through all but the very last element of the array, which, by default, makes it run in $O(n)$ time. The outer loop contains 3 main parts inside: the initialization of the smallest index variable (line 8), the inner loop (lines 11-19), and the swap to put the next element in place (lines 22-24). First, the time used to declare a variable is independent of the length of the array and, therefore, runs in constant time $O(1)$. Next, the inner loop

iterates through the array beginning at index $i + 1$. The worst case for the inner loop is when $i = 0$, which makes the inner loop iterate through all but the last element, making the loop run in at least $O(n)$ time. The body of the inner loop contains an if statement and 2 variable assignments, all of which will run in constant time regardless of the size of the array or the current indices being checked. Therefore, the final runtime for the inner loop is $O(1 * n)$, which simplifies to $O(n)$. Lastly, the swap to put the smallest element in its proper place contains 3 variable assignments that will always run for each iteration of the outer loop, which makes the swap $O(1)$. Overall, the runtime of selection sort is $O((1 + n + 1) * n)$, which is $O(n^2)$.

2 Appendix

2.1 Selection Sort

```
int selectionSort(StringArr* data) {
    // Start comparisons at 0
    int comparisons = 0;

    // Iterate through the second to last element because the last element will already be
        sorted as is
    for (int i = 0; i < data->length - 1; i++) {
        // The smallest index is going to start as the start of the subset of the list
        int smallestIndex = i;

        // Iterate through the rest of the list
        for (int j = i + 1; j < data->length; j++) {
            // Compare the current element to the current smallest element in the subset
            if (data->arr[smallestIndex].compare(data->arr[j]) > 0) {
                // If the current element comes first, make it the new smallest element
                smallestIndex = j;
            }
            // Increment comparisons
            comparisons++;
        }

        // Put the smallest index in its respective place
        std::string temp = data->arr[i];
        data->arr[i] = data->arr[smallestIndex];
        data->arr[smallestIndex] = temp;
    }

    // Return the number of comparisons
    return comparisons;
}
```

Listing 1: Selection Sort (C++)