CH08-320143: Assignment #3

Harit Krishan
h.krishan@jacobs-university.de

Jacobs University Bremen — February 29, 2020

1 Selection Sort

Selection Sort is similar to Insertion Sort. Given an array of elements, you always take the current element and exchange it with the smallest element that can be found on the right hand side of the current element. In doing so, we gradually build up a sorted sequence on the left side (similar to Insertion Sort). In each iteration, we 'attach' the smallest element from the remaining unsorted right side to it.

1.1 Implementation

We make use of C++ to implement Selection Sort. Below is a snapshot of the Selection Sort algorithm implemented in C++.

```
void selection_sort(int arr[], int size) {
        int i, j, min_index;

        for(i = 0; i < size-1; i++) {
                min_index = i;

                for(j = i+1; j < size; j++) {
                        if(arr[j] < arr[min_index]) {
                                min_index = j;
                        }
                }
                swap(arr[min_index], arr[i]);
        }
}
```

Figure 1: Selection Sort Implementation in C++

1.2 Correctness

To analyze the correctness of an algorithm, it is important to consider the loop invariant. The implemented function in Figure 1 takes an n-element array $arr[n]$ and outputs the same array $arr[n]$ with its elements rearranged into increasing order.

At each iteration of the outer for loop, the subarray $arr[0..j]$ contains the j smallest elements of $arr[n]$ in increasing order. After $size - 1$ iterations of the loop, the $size - 1$ smallest elements of $arr[n]$ are in the first $size - 1$ positions of $arr[n]$ in increasing order. This means, the $size^{\text{th}}$ element is necessarily the largest element. This implies that the loop need not run for the $size^{\text{th}}$ iteration (the final time). The best and worst-case running times of selection sort are $\Theta(n^2)$ because regardless of how the elements are initially arranged, on the $size^{\text{th}}$ iteration of the outer for loop, the algorithm always inspects each of the remaining $size - 1$ elements seeking the smallest remaining one.

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

$= n^2 - n - (n^2 - n)/2 = (n^2 - n)/2$

$= \Theta(n^2)$

1.3 Testing

With the algorithm implemented, we can begin timing and testing for run-time analysis. To create test cases, we make use of the $rand()$ function to generate arrays of random numbers. We then time the algorithm execution process and create a graph. We extract the recorded run-times to a file and use Gnuplot to graph the results. Figure 2 shows the driver function implemented to test the algorithm.

```
int main() {
        srand(time(NULL));
        int size = 1;
        //cout << "Enter size of random array: ";
        //cin >> size;
        //int arr[size];

        ofstream output("sel_sort_data.txt");

        while(size <= 1000){
        int arr[size];
        for(int i = 0; i < size; i++) {         // Random Number Generator
                arr[i] = rand() % 99 + 1;
        }

/*      cout << "Randomly generated array: ";
        print_array(arr, size);
        cout << endl;
*/
        cout << "Applying Selection Sort..." << endl;
        clock_t start = clock();
        selection_sort(arr, size);
        clock_t end = clock();

        output << size << " " << ((double)(end-start)/CLOCKS_PER_SEC)*1000 << endl;
        size++;
        }
/*      cout << "Selection Sort Applied [" << ((double)(end-start)/CLOCKS_PER_SEC)*1000 << "ms]\n\n";

        cout << "Sorted Array: ";
        print_array(arr, size);
*/

        return 0;

}
```

Figure 2: Driver Function of Implemented Variant of Selection Sort

Some lines of code have been commented out in Figure 2. These lines were initially included to ask the user for the size of the array. The file has been modified to carry out the desired tasks.

1.4 Results

The timing results of the algorithm were printed to a file as can be seen in Figure 2. Figure 3 shows the graph of these extracted results. We use Gnuplot to graph the information in the output file of the driver function.

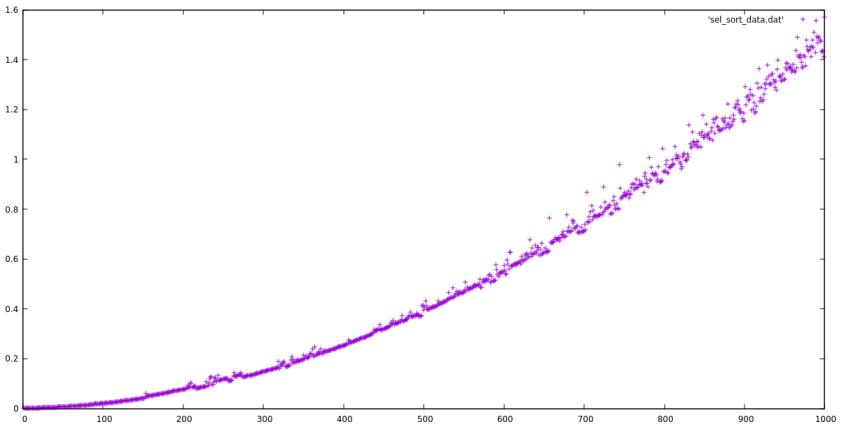

Figure 3: Graph of Running Time vs Input Size

From Figure 3, it is safe to say that the algorithm has a polynomial run-time of degree 2. Once we ignore the machine-dependent constants, we get $\Theta(n^2)$ as the average case running time. All relevant code and data files are included in the root directory of this assignment submission, as instructed.