Solutions for Exercise Sheet 1

Handout: September 12th — Deadline: 4pm September 19th

SelectionSort sorts by repeatedly finding the smallest element amongst those not yet sorted, and swaps it with the first number in the unsorted part of the array.

Selection-Sort(A)

```
1: n = A.length
2: for j = 1 to n − 1 do
3:     smallest = j
4:     for i = j + 1 to n do
5:         if A[i] < A[smallest] then smallest = i
6:     exchange A[j] with A[smallest]
```

Question 1.1 (0.25 marks)

To get a feeling of how SelectionSort works, assume it is run on the array

24	5	6	23	42	45	2	1	8

Write down the contents of the array after every iteration of the for loop in line 2.

Solution:

24	5	6	23	42	45	2	1	8

1	5	6	23	42	45	2	24	8

1	2	6	23	42	45	5	24	8

1	2	5	23	42	45	6	24	8

1	2	5	6	42	45	23	24	8

1	2	5	6	8	45	23	24	42

1	2	5	6	8	23	45	24	42

1	2	5	6	8	23	24	45	42

1	2	5	6	8	23	24	42	45

Question 1.2 (0.5 marks)

Show the correctness of SelectionSort when run on an array of n different elements (any array, not just the instance from Question 1.1). Find a loop invariant for the loop in line 2 that implies that at termination the array is sorted. Show that this invariant holds at initialisation, and that if it is true before an iteration of the loop, it remains true before the next iteration. Show that the loop invariant at termination implies that the array is sorted.

Solution: We use the following loop invariant:

"at the start of each iteration of the outer for loop, the subarray $A[1 \dots j-1]$ consists of the $j-1$ smallest elements in the array $A[1 \dots n]$, and this subarray is in sorted order."

Now we prove that the three conditions hold:

Initialisation: The condition is true at initialisation, when $j = 1$, as the empty subarray consists of the 0 smallest elements, and these are in sorted order.

Maintenance: If the condition is true before the iteration for j starts, SelectionSort finds the smallest element in the subarray $A[j \dots n]$, and swaps it so that it ends up in $A[j]$. The loop invariant states that all elements in $A[1 \dots j-1]$ are smaller than $A[j]$, and they are sorted: $A[1] \leq A[2] \leq \dots A[j-1]$. Hence at the end of the iteration $A[1 \dots j]$ contains the j smallest elements in sorted order.

Termination: After the first $n-1$ elements, the subarray $A[1 \dots n-1]$ contains the smallest $n-1$ elements, sorted, and therefore element $A[n]$ must be the largest element.

Hence, the algorithm is correct.

Question 1.3 (0.5 marks)

Assume for simplicity that one execution of each line of the algorithm takes time 1 (that is, in the notation for analysing InsertionSort, $c_1 = c_2 = \dots = 1$). Give the best-case and the worst-case running time of SelectionSort. How does this compare to best-case and worst-case times of InsertionSort.

Solution: We tabulate costs and numbers of times a line is executed. Note that the head of a for loop is executed one more time than its body. This means line 2 is executed n times and lines 3 and 6 are only run $n-1$ times.

For any given value of j (where j runs from 1 to $n-1$), the body of the inner for loop is executed for values $j+1, j+2, \dots, n$, which is $n-j$ times. The head of the inner for loop is executed one more time, which gives $n-j+1$. This means that line 4 is executed a total of $\sum_{j=1}^{n-1}(n-j+1)$ times. There are several ways of simplifying this formula. Here's one of them: We can write this as $\sum_{j=1}^{n-1}(n-j) + \sum_{j=1}^{n-1} 1$ and note that $\sum_{j=1}^{n-1} 1 = n-1$. The sum $\sum_{j=1}^{n-1}(n-j)$ can be written as $(n-1) + (n-2) + \dots + 1$ and writing these terms down in reverse order yields $1 + 2 + \dots + n - 1 = \sum_{j=1}^{n-1} j$. This is now in a shape to which we can apply the formula $\sum_{j=1}^{m} j = m(m+1)/2$ for $m := n-1$, which gives a term of $\frac{n(n-1)}{2}$. Hence, line 4 is executed $n - 1 + \frac{n(n-1)}{2}$ times. For line 5 we get $\sum_{j=1}^{n-1}(n-j) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$ in the same way (the only difference is the absence of the $n-1$ term).

		cost	times
1:	$n = A$.length	1	1
2:	**for** $j = 1$ to $n-1$ **do**	1	n
3:	smallest $= j$	1	$n-1$
4:	**for** $i = j+1$ to n **do**	1	$\sum_{j=1}^{n-1}(n-j+1) = n-1+\sum_{j=1}^{n-1} j = n-1+(n-1)n/2$
5:	**if** $A[i] < A$[smallest] ...	1	$\sum_{j=1}^{n-1}(n-j) = \sum_{j=1}^{n-1} j = (n-1)n/2$
6:	exchange $A[j]$ with A[smallest]	1	$n-1$

Summing up all costs, the overall running time is always

$$\begin{aligned} & 1 + n + 3(n-1) + 2 \cdot \frac{(n-1)n}{2} \\ = \; & n^2 + 3n - 2. \end{aligned}$$

In particular, the best case equals the worst case.

To compare this with InsertionSort, the worst-case running time of InsertionSort is $\frac{3}{2} \cdot n^2 + \frac{7}{2} \cdot n - 4$, which is similar to SelectionSort, apart from constant factors being better for SelectionSort, most notably n^2 versus $\frac{3}{2} \cdot n^2$. But the best-case running time of InsertionSort is only $5n - 4$. For the best case we get a quadratic function for SelectionSort, but only a linear function for InsertionSort. For large n, the best case is a lot better for InsertionSort.

Programming Question 1.4 (0.25 marks)

Implement InsertionSort and SelectionSort.