CMSC 351: SelectionSort

Justin Wyss-Gallifent

February 3, 2022

1 What it Does

Sorts a list of elements on which there is a total order. Think of integers or real numbers.

2 How it Works

We identify the smallest element in `A[0,...,n-1]` and swap it with the element at index 0. At this point `A[0]` is (trivially) sorted and we can ignore it. We then identify the smallest element in `A[1,...n-1]` and swap it with the element at index 1. At this point `A[0,1]` is sorted and we can ignore it. We then identify the smallest element in `A[2,...n-1]` and swap it with the element at index 2. At this point `A[0,...,2]` is sorted and we can ignore it. We continue until `A[0,...,n-2]` is sorted, at which point the entire list is sorted.

3 Visualization

Consider the following list:

4	3	1	8	3

We first find the smallest element in `A[0,...,4]`:

4	3	1	8	3

We swap it with index 0:

1	3	4	8	3

We then find the smallest element in `A[1,...,4]`:

1	3	4	8	3

We swap it with index 1, which does not change the result.
We then find the smallest element in `A[2,...,4]`:

1	3	4	8	3

We swap it with index 2:

1	3	3	8	4

We then find the smallest element in `A[3,...,4]`:

1	3	3	8	4

We swap it with index 3:

1	3	3	4	8

Then we are done.

4 Pseudocode with Time Complexity

Here is the pseudocode with time assignments:

```
\\ PRE: A is a list of length n.
for i = 0 to n-2                      $n-1$ times
    minindex = i                      $c_1$
    for j = i+1 to n-1                $n-1-(i+1)+1 = n-i-1$ times.
       if A[j] < A[minindex]          ⎫
          minindex = j                ⎬ $c_2$
       end                            ⎭
    end
    swap A[i] with A[minindex]        $c_3$
end
\\ POST: A is sorted.
```

The time required is always the same in the best-, worst-, and average-cases:

$$\begin{aligned}
T(n) &= \sum_{i=0}^{n-2}\left[c_1 + \left[\sum_{j=i+1}^{n-1} c_2\right] + c_3\right] \\
&= \sum_{i=0}^{n-2}\left[c_1 + (n-1-(i+1)+1)c_2 + c_3\right] \\
&= \sum_{i=0}^{n-2}\left[c_1 + (n-1-i)c_2 + c_3\right] \\
&= \sum_{i=0}^{n-2}\left[c_1 + (n-1)c_2 + c_3\right] - c_2\sum_{i=0}^{n-2} i \\
&= (n-2+1)\left[c_1 + (n-1)c_2 + c_3\right] - c_2\left(\frac{(n-2)(n-1)}{2}\right) \\
&= \frac{1}{2}c_2 n^2 + \left(c_1 - \frac{1}{2}c_2 + c_3\right) n - (c_1 + c_3) \\
&= \Theta(n^2)
\end{aligned}$$

The same quick note from BubbleSort applies here with regards to labeling of the conditional.

5 Auxiliary Space

The auxiliary space is $\mathcal{O}(1)$ which includes two indices, `minindex` and potentially a swap variable.

6 Stability

SelectionSort is not stable. The reason for this is when SelectionSort finds a smallest element at index `minindex` and swaps it with the element at index `i`, the element at index `i` could be moved to the right of an equal element that is sitting between them.

7 In-Place

SelectionSort is in-place.

8 Notes

After k iterations the first k elements are correctly placed and sorted. If SelectionSort were running on a very long list and it was forced to stop early this means that some amount of the beginning of the list would be sorted but the end would not. This is the opposite of BubbleSort, as we saw.

9 Thoughts, Problems, Ideas

1. Fill in the summation details of the time calculations. In other words, get an exact answer before the $\mathcal{O}$ step.

2. Modify SelectionSort so that after k iterations the last k elements are correct.

3. Modify SelectionSort so that after $2k$ iterations the first k and last k elements are correct. Hint: Alternate!

4. Suppose the algorithm does this instead: It first scans for any element smaller than `A[0]` and when it finds one it immediately exchanges it with `A[0]`. Then it tries again, repeatedly until it fails to find one. It then repeats the process with `A[1]`, `A[2]`, and so on. Write the pseudocode for this and find its worst-case $\mathcal{O}$ time complexity.

5. The basic premise of SelectionSort is that the `for` loop is selecting (hence the name) the index of the next smallest element at each iteration, then that smallest element is swapped into its proper position. That locating step which takes $\mathcal{O}(n)$. Suppose that step is replaced by a single function `indexofsmallest(a,b)` which finds the index of the smallest element between indices `i` and `j`. Suppose this new function takes time $S(n)$.

   ```
   for i = 0 to n-2                           $n-1$ times
       minindex = indexofsmallest(i,n-1)      $S(n)$
       swapindices(i,minindex)                $c_1$
   end
   ```

   What would need to be true about the time requirement $S(n)$ to make the time complexity faster than the standard SelectionSort?

6. Modify the pseudocode so that it finds both the `minindex` and `maxindex` during each pass and swaps those into their appropriate positions. This is the *Cocktail Sort*.

7. In a worst-case scenario how many assignments does the pseudocode make?

8. Suppose a list A has an odd number of elements, so $n = 2k + 1$. Modify `SelectionSort` to create a new function `median` which finds the median of the list. How many comparisons will be necessary to find the median of the list $\{5, 3, 0, 16, 8, 3, 7\}$?

9. Rewrite your `median` function from the previous question so that it will work if A has either an even or an odd number of elements. You should not break the code into even/odd cases and you don't even need to check if n is even or odd but you can use the ceiling command `ceil` as well as simple arithmetic. It is enough!

10 Python Test with Output

Code

```
import random
A = []
for i in range(0,10):
    A.append(random.randint(0,100))
n = len(A)
print(A)

for i in range(0,n-1):
    minindex = i
    for j in range(i+1,n):
       if A[j] < A[minindex]:
          minindex = j
    print(’In A[’+str(i)+’,’+str(n-1)+’] the index of the
        minimum is ’+str(minindex))
    temp = A[i]
    A[i] = A[minindex]
    A[minindex] = temp
    print(’Swap indices ’ + str(minindex) + ’ and ’ + str(i)
        )
    print(’Now: ’ + str(A))

print(A)
```

Output:

```
[37, 89, 47, 59, 87, 23, 43, 98, 68, 30]
In A[0,9] the index of the minimum is 5
Swap indices 5 and 0
Now: [23, 89, 47, 59, 87, 37, 43, 98, 68, 30]
In A[1,9] the index of the minimum is 9
Swap indices 9 and 1
Now: [23, 30, 47, 59, 87, 37, 43, 98, 68, 89]
In A[2,9] the index of the minimum is 5
Swap indices 5 and 2
Now: [23, 30, 37, 59, 87, 47, 43, 98, 68, 89]
In A[3,9] the index of the minimum is 6
Swap indices 6 and 3
Now: [23, 30, 37, 43, 87, 47, 59, 98, 68, 89]
In A[4,9] the index of the minimum is 5
Swap indices 5 and 4
Now: [23, 30, 37, 43, 47, 87, 59, 98, 68, 89]
In A[5,9] the index of the minimum is 6
Swap indices 6 and 5
Now: [23, 30, 37, 43, 47, 59, 87, 98, 68, 89]
In A[6,9] the index of the minimum is 8
Swap indices 8 and 6
Now: [23, 30, 37, 43, 47, 59, 68, 98, 87, 89]
In A[7,9] the index of the minimum is 8
Swap indices 8 and 7
Now: [23, 30, 37, 43, 47, 59, 68, 87, 98, 89]
In A[8,9] the index of the minimum is 9
Swap indices 9 and 8
Now: [23, 30, 37, 43, 47, 59, 68, 87, 89, 98]
[23, 30, 37, 43, 47, 59, 68, 87, 89, 98]
```