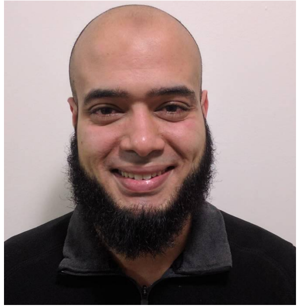
Algorithms
Selection Sort

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Selection sort

- Given an array of numbers, order them from small to large
  - Input: [9,2, 10, 0, 5, 3, 90, 85]
  - Output: [0, 2, 3, 5, 9, 10, 85, 90]
- In the previous sessions, we learned insertion sort
  - It is based on the **incremental thinking** approach
- Selection sort is another simple but slow approach
- It is based on a simple **observation**
  - The **first** element in the sorted array is the **minimum** in the whole array
  - The **second element** in the sorted array is the **second minimum** in the whole array
  - And so on
  - Can you sketch the whole idea?

The procedure

- Keep repeating:
  - Find the minimum in the (reminder of the) array
  - Make it the first (or the next) element

Selection Sort

0	1	2	3	4	5	6	7
9	85	10	0	5	3	90	2

- What is the minimum in the above array?
- 0 from index 3
- Swap it with the first number!

Selection Sort

0	1	2	3	4	5	6	7
0	85	10	9	5	3	90	2

- Now we have 2 subarrays
  - Sorted [0-0]
  - Unsorted [1-7]
- What is the minimum in the unsorted array? 2 in index 7
- Swap with index 1 so that we expect the sorted array

Selection Sort

0	1	2	3	4	5	6	7
0	2	10	9	5	3	90	85

- Now we have 2 subarrays: Sorted [0-1] and unsorted [2-7]
- What is the minimum in the unsorted array? 3 in index 5
- Swap with index 2

Selection Sort

0	1	2	3	4	5	6	7
0	2	3	9	5	10	90	85

- Now we have 2 subarrays: Sorted [0-2] and unsorted [3-7]
- What is the minimum in the unsorted array? 5 in index 4
- Swap with index 3

Selection Sort

0	1	2	3	4	5	6	7
0	2	3	5	9	10	90	85

- 9 and 10 are in the right place
- Observe: the sorted subarray never changes once determined

Selection Sort

0	1	2	3	4	5	6	7
0	2	3	5	9	10	90	85

- Now we have 2 subarrays: Sorted [0-5] and unsorted [6-7]
- What is the minimum in the unsorted array? 85 in index 7
- Swap with index 6

Selection Sort

0	1	2	3	4	5	6	7
0	2	3	5	9	10	85	90

- Completely sorted!
- Code it. Analyze it. Think about correctness.
- Think about its properties!
  - Find their answers in the quiz.

O(n^2) time and O(1) memory

- 9,85,10,0,5,3,9,0,2

```cpp
void selection_sort(vector<int> &nums) {
    int n = nums.size();

    for (int i = 0; i < n - 1; i++) {
        // Find the minimum element in unsorted array
        int mn_idx = i;
        for (int j = i + 1; j < n; j++)
            if (nums[j] < nums[mn_idx])
                mn_idx = j;

        // Put the min in its right place
        swap(nums[mn_idx], nums[i]);
    }
}
```

Worst vs Best analysis

- We can see how the algorithm doesn’t depend on its number of operations based on the data
- This makes its best, average and worst case complexity the same
- So keep this general property in your mind: To what degree do the data values themselves affect the total number of operations?

Correctness

- Similar to insertion sort, it is trivial to see weather the algorithm is correct
- Don’t waste time on (in)formal proofs for trivial algorithms

Selection Sort vs Heap Sort

- During a data-structure course, we study min/max heap data-structure which can be used in sorting values ⇒ Heap Sort
- Heapsort is similar to selection sort: it locates the **largest** value and places it in the **final** array position. Then it locates the next **largest** value and places it in the **next-to-last** array position and so forth.
  - This is similar to the selection sort process where we find the **smallest** and place it in the **first** array position (the opposite).
  - Selection sort takes O(n) to locate the smallest.
  - Heap sort locates the minimum in O(1) but takes O(logn) to remove it ⇒ O(nlogn)
- If you did not study the **heap** data structure so far, review this slide later

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."