Selection Sort

24 February 2022 12:16

What is sorting ?

arr[ ] : {1, 7, 9, 2, 3, 0}

↓ sort

arr[ ] : {0, 1, 2, 3, 7, 9}

Arranging elements in a non-decreasing order is called sorting. Sorting can also be done in a non-increasing order.

What is Selection Sort ?

1. You will make rounds/passes through the array.
2. In each pass, you have to bring the smallest element to its correct place in the array.
3. You will then only consider the unsorted array excluding the smallest element you dealt with.
4. Repeat this until your array becomes sorted. (The array to sort will have only 1 element left).

Example : arr[ ] : {64, 25, 12, 22, 11}

1\.

	0	1	2	3	4
	{64	25	12	22	11}

i = 0.

Search for the smallest element in this array and put it at i = 0.

2\.

	0	1	2	3	4
	{11	25	12	22	64}

i = 1

Sorted

Search for the smallest element in this array and put it at $i = 1$.

3. $\{ \underbrace{\overset{0}{11} \quad \overset{1}{12}}_{\text{Sorted}} \quad \underbrace{\overset{2}{25} \quad \overset{3}{22} \quad \overset{4}{64}} \}$

Sorted

Search for the smallest element in this array and put it at $i = 2$.

4. $\{ \underbrace{\overset{0}{11} \quad \overset{1}{12} \quad \overset{2}{22}}_{\text{Sorted}} \quad \underbrace{\overset{3}{25} \quad \overset{4}{64}} \}$

Search for the smallest element in this array and put it at $i = 3$.

5. $\{ \underbrace{\overset{0}{11} \quad \overset{1}{12} \quad \overset{2}{22} \quad \overset{3}{25}}_{\text{Sorted}} \quad \underbrace{\overset{4}{64}} \}$

1 element left so it is definitely sorted.

Note: I made 4 passes for an array of length 5.

For each pass we will have to run a for loop from i = no. of elements put in their correct places to $n-1$ to get the smallest element from the unsorted array.

There will be $n-1$ passes

Code:

```
void selectionSort(vector<int>& arr, int n)
{
    for(int i = 0; i < n-1; i++ ) {
        int minIndex = i;

        for(int j = i+1; j<n; j++) {

            if(arr[j] < arr[minIndex])
                minIndex = j;

        }
        swap(arr[minIndex], arr[i]);
    }
}
```

Time Complexity :

$(n-1)$ passes :

Each pass has $(n-i-1)$ comparisons.
and 1 swap.

(So $(n-i)$ operations that are $O(1)$)

For $i=0$: n
$i=1$: $n-1$
$i=2$: $n-2$
⋮
$i=n-2$: 2

$$\Rightarrow f(n) = n + (n-1) + (n-2) + \dots + 2$$

$$\Rightarrow f(n) = 2 + 3 + \dots + (n-1) + n.$$

$$\Rightarrow f(n) = \frac{n(n+1)}{2} - 1$$

$$\Rightarrow f(n) = \frac{n^2}{2} + \frac{n}{2} - 1$$

$$\Rightarrow f(n) = \boxed{O(n^2)}$$

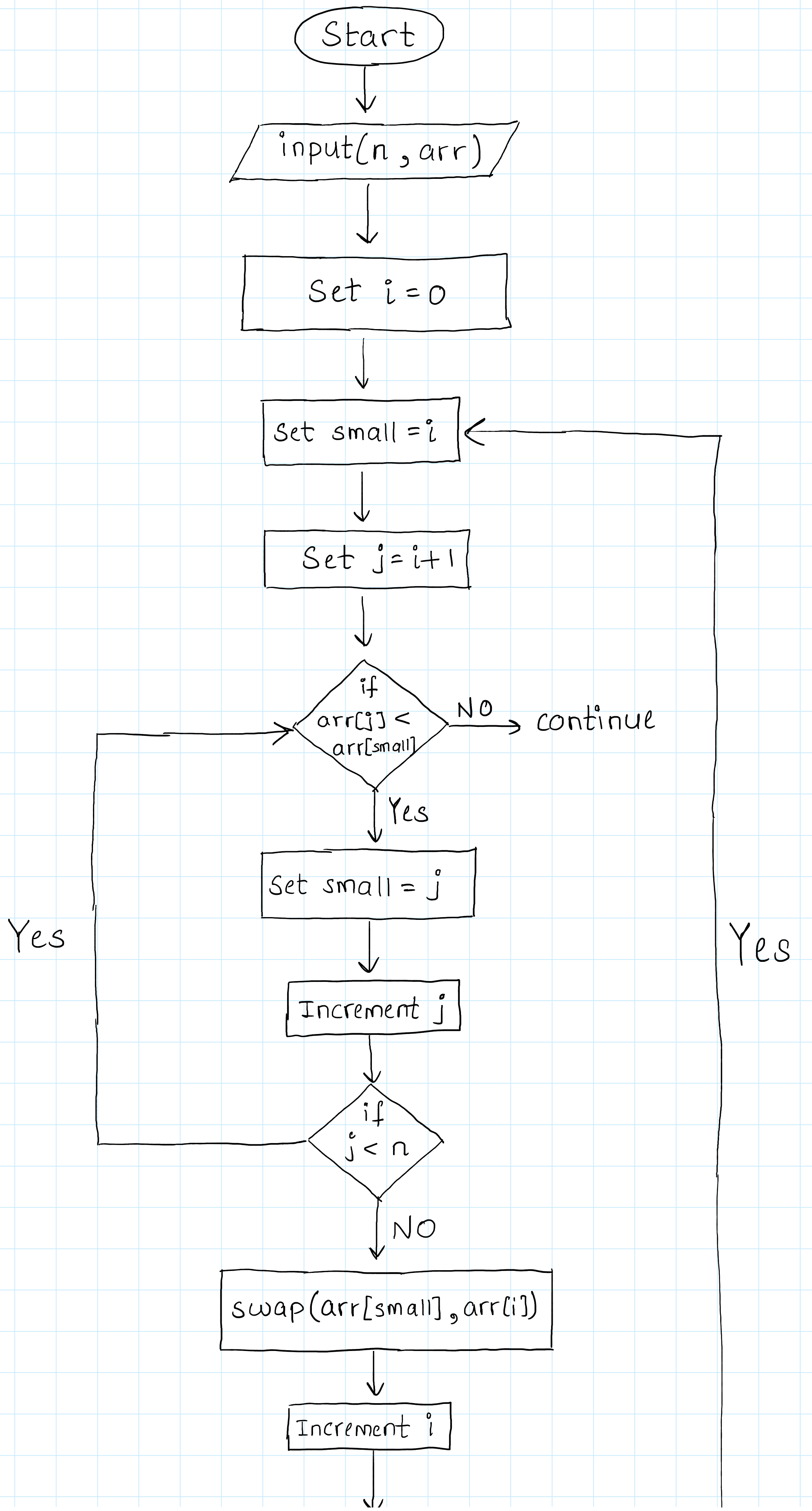
Space Complexity :

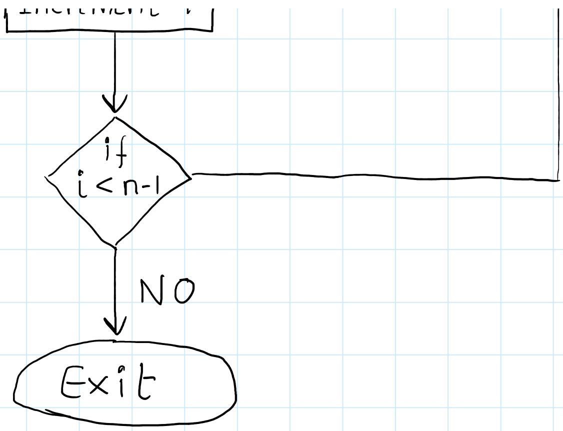
No extra memory/space has been used. Thus $\boxed{O(1)}$

Use Cases :

1. Works nicely with smaller arrays.
2. When there are strict memory constraints.

Homework: Flow Chart for Selection Sort.





Homework: Is selection sort stable?

Ans: Since we make swaps after each pass in the unsorted array, we can have an array like

$arr[\,] = \{\underset{0}{4}, 2, 3, 1, \underset{1}{4}\}$ where the two 4's have an order which might get changed after sorting i.e. the (zero) 4 might appear after the (one) 4 in the sorted array.

Link: [Is selection sort stable?]