24/02/2023

Selection Sort

First of all we need to understand what is sorting. Sorting is basically rearranging elements in either increasing or decreasing order.

1, 3, 5, 4, 7, 2 → These elements are not in sorted order as initially it was increasing but 5 to 4 is of decreasing order & hence we can say it is not sorted.

Scanned with Cam

In selection sort, an element is picked & then is placed at correct place. In this we have to place the minimum element at the right place.

Algorithm of Selection Sort

10 1 4 8 5 7

Step-1 Find the minimum element from the array & place at i = 0.

min = 1
swap (arr [0], arr [minIndex]);

1 10 4 8 5 7

Step-2 Find minimum element in subarray start from index = 1 to n-1 & place at 1st index.

min = 4
swap (arr [1], arr [minIndex]);

1 4 10 8 5 7

Step-3 Find minimum in sub-array start from index = 2 to n-1 & place at 2nd index.

min = 5
swap (arr [2], arr [minIndex]);

1 4 5 8 10 7

Step-4 Find minimum in sub-array start from index = 3 to n-1 & place at 3rd index

min = 7
swap (arr [3], arr [minIndex]);

1 4 5 7 10 8

Step-5 Find minimum in sub-array start from index = 4 to n-1 & place at 4th index

min = 8
swap (arr [4], arr [minIndex]);

1 4 5 7 8 10

Now only 1 element remaining, so need to check it as there is no element on its right and hence we get the sorted array.

Note → Every step we do is termed as round or parse.

Step-1 = Round-1 = Parse-1

We can observe that for 6 elements, 5 rounds are required to sort the array. Hence for n elements, n-1 rounds are required to sort the array via Selection Sort.

Code

```
void selectionSort (vector <int> arr){

int n= arr.size();
for (int i=0; i<n-1; i++){
        // Assuming minimum element to be current
                                                element
        int minIndex = i;
        for (int j=i+1; j<n; j++){
        // Comparing with minimum Index.
                if (arr[minIndex] > arr[j]){
                        minIndex = j;
                }       //Updating minimum
        }
    // Placing min element at correct position
    swap (arr[minIndex], arr[i]);
}
}
```

Note → We have started inner loop from i+1 as we have already taken minIndex as i & does not make sense to compare with itself.

Time complexity
$O(n^2)$ as the loops are nested, hence time complexity gets multiplied.

Space complexity
O(1) as only variables have been created & we don't have to consider space of the input.

Bubble Sort

The logic behind this sorting technique is that in the ith round, ith largest element will be placed at the right position.

Algorithm for bubble sort

10 1 7 6 14 9

Round - 1

* 10 > 1 → swap
  1 10 7 6 14 9
* 10 > 7 → swap
  1 7 10 6 14 9
* 10 > 6 → swap
  1 7 6 10 14 9
* 10 > 14 → no swap
* 14 > 9 → swap
  1 7 6 10 9 14

At right place

Round - 2

1 7 6 10 9 | 14

Sorted boundary

* 1 > 7 → no swap
* 7 > 6 → swap
  1 6 7 10 9 14
* 7 > 10 → no swap
* 10 > 9 → swap
  1 6 7 9 10 14

Round - 3

1 6 7 9 10 14

Sorted boundary

* 1 > 6
* 6 > 7 No swap in any case.
* 7 > 9

Round - 4

1 6 7 9 10 14

Sorted boundary

* 1 > 6 No swap in any case
* 6 > 7

Round - 5

1 6 7 9 10 14

Sorted boundary

* 1 > 6 → no swap

Now we don't have to go in round - 6 as only one element is left & hence we can't compare it with any element. Hence we get sorted array in n-1 rounds where n is the no. of elements.

Optimization in Bubble Sort

Also we can observe that in round-3 there was no swap done & hence the array gets already sorted & we don't have to go in further rounds.

Code

```
void bubble Sort (vector <int> & arr) {

        int n = arr.size();
        for (int i=0; i< n-1; i++) {
            bool swapped = false;
            for (int j = 1; j < n-i; j++) {

                if (arr[j-1] > arr[j]) {
                    swapped = true;
                    swap (arr[j-1], arr[j]);
                }
            }
            if (!swapped) {
                break;
            }
    }
}
```

Best case Time Complexity
Best case occurs when the array is already sorted. Here the time complexity is $O(n)$ as only n-1 comparasions will be done.
Normal case Time complexity $O(n^2)$
Worst case Time complexity
Worst case occurs when the array is reverse sorted & hence time complexity here is $O(n^2)$.

By optimization, best case time complexity reduces to O(n).

Space complexity
O(1) as only variables are created.

Note → Use case of selection sort is incase of small arrays.
Use case of bubble sort is when ith largest elements to be put at correct place.

Insertion Sort
This means to insert the element at right place.

$\boxed{2}$ $\boxed{1}$ $\boxed{5}$ $\boxed{4}$

$\boxed{1}$ $\boxed{2}$ $\boxed{5}$
↳ Insert $\boxed{4}$

$\boxed{1}$ $\boxed{2}$ $\boxed{4}$ $\boxed{5}$ → Sorted array

Algorithm for insertion sort

10 1 7 6 14 9

1) 10 is already at right place or will automatically be placed at right place.

2) Pick element at index = 1

1 < 10, shift 10 & place 1 there. There is no swapping involved.

1 10 7 6 14 9

Sorted

3) Pick element at index = 2

7 < 10
7 > 1 } 7 should come in b/w 1 & 10.

So shift 10 & copy 7 at the empty place.

1 7 10 6 14 9

Sorted

4) Pick element at index = 3

6 < 10
6 < 7
6 > 1

Shift 10, then shift 7 & then copy 6 at the empty place

1 6 7 10 14 9

Sorted

5) Pick element at index = 4

14 > 10 → nothing to do.

1 6 7 10 14 9

Sorted

6) Pick element at index = 5

9 < 14
9 < 10
9 > 7

Shift 14, then shift 10 & then copy 9 to the empty place

1 6 7 9 10 14

Hence the array is sorted now.

Code

```
void insertionSort (vector <int> arr) {

        int n = arr.size();
        for (int i=1; i<n; i++) {
            int val = arr [i]; //Pick element
            int j = i-1;
            for( ; j>=0; j--) {
                //Compare element
                if(arr [j] > val) {
    //shifting operation  arr [j+1] = val;
                }
                else {
                    break;
                }
            }
    // Copy step arr [j+1] = val;
        }
}
```

Time complexity
$O(n^2)$ in normal & worst case.
$O(n)$ in the best case i.e already sorted.
Space complexity
$O(1)$ as only variables are created.

classmate

Date

Page 213

Use case of insertion sorted is in case of small arrays or when array is partially sorted.

Inbuilt sort function

sort (arr.begin(), arr.end()); is used to sort the vector.

We need to include algorithm header file to use this inbuilt function.