Geeks Man

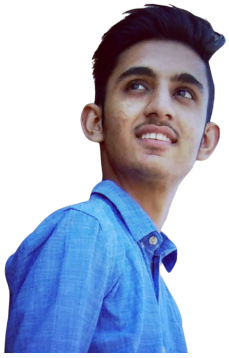
Algorithms

Lesson 5

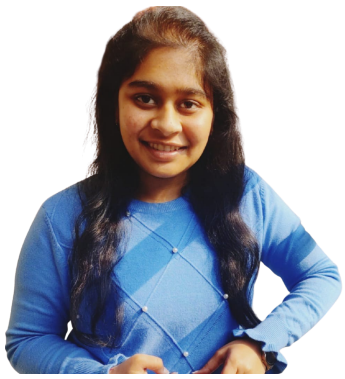
Trilok Kaushik

Founder of Geeksman

Aditya Aggarwal



Aditya Gupta

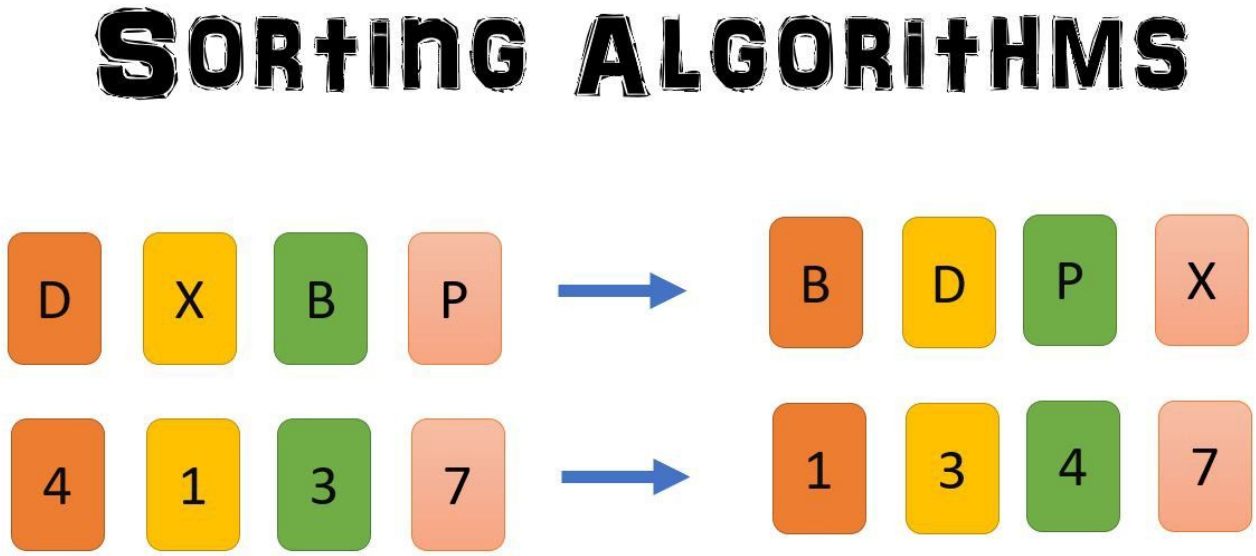


Suhani Mittal

Team Coordinators

Sorting (Part-I)

Sorting is arranging the elements either in ascending or descending order.



Types of Sorting Algorithms :

Algorithm	Time Complexity		
	Best	Average	Worst
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)
Heap Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Quick Sort	Ω(n log(n))	θ(n log(n))	O(n^2)
Merge Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)
Radix Sort	Ω(nk)	θ(nk)	O(nk)

Sorting Basic Approach :- Try out all possible combinations of the input array and select the ascending order

Time Complexity would be :- O(n!) or $O(n^n)$

1. Bubble Sort

Bubble sort is an algorithm that compares the adjacent elements and swaps their positions if they are not in the intended order

Characteristic :

1. At every iteration the largest element is at the end .
2. Best case = O(n)
3. Worst case = $O(n^2)$
4. Space complexity is `O(1)` because an extra variable `temp` is used for swapping.

Algorithm :-

1. Input the array.
2. I^{th} loop iterate over the array i.e n-times
3. J^{th} loop compares the element pairwise at every value of i
4. i tells how many largest element has occupied its location
5. j compares the element with element at j+1 position

Code :

```
1.// A function to implement bubble sort
2.void bubbleSort(int arr[], int n)
3.{
4.    int i, j;
```

```
5.    for (i = 0; i < n; i++)
6.
7.    // Last i elements are already in place
8.    for (j = 0; j < n-i-1; j++)
9.        if (arr[j] > arr[j+1])  {
10.            int temp=arr[j];
11.            arr[j]=arr[j+1];
12.            arr[j+1]=temp;
13.        }
14. }
```

Here is the complete code of bubble sort
https://sapphireengine.com/@/z6mofk

Bubble sort is used in the following cases where

1. the complexity of the code does not matter.
2. a short code is preferred.

Optimized Bubble Sort :- As the i^{th} loop runs for n-times even if the array is sorted which takes O(n) so reduce the time to O(1) by taking the swap variable , if Swap doesn't takes place in first iteration of i^{th} loop then it means the array is sorted due to this we can break the loop .

2. Selection Sort

Selection sort is an algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

Characteristics :-

1. *At every iteration min. Element is placed at the beginning of an unsorted list .*
2. *Best Case = $O(n^2)$*
3. *Worst Case = $O(n^2)$*
4. *Space complexity is `O(1)` because an extra variable `temp` is used.*

Algorithm :-

1. Set the first element of unsorted list as minimum
2. From the second element onwards search for value less than min. Till the array size n.
3. Swap the value of 1st element with the min.value.
4. After each iteration, `minimum` is placed in the front of the unsorted list.
5. Continue this process until the unsorted list gets sorted.

Code :

```
1.void selectionSort(int arr[], int n)
2.{
3.    int i, j, min_idx;
4.
5.    // One by one move boundary of unsorted
  subarray
6.    for (i = 0; i < n-1; i++)
7.    {
8.        // Find the minimum element in unsorted
  array
9.        min_idx = i;
10.         for (j = i+1; j < n; j++)
11.         if (arr[j] < arr[min_idx])
```

```
            min_idx = j;

        // Swap the found minimum element with
  the first element
        int temp=arr[min_idx];
        arr[min_idx]=arr[i];
        arr[i]=temp;
    }
}
```

Here is the complete code of selection sort
https://sapphireengine.com/@/3p4wsv

The selection sort is used when:

- a small list is to be sorted
- cost of swapping does not matter
- checking of all the elements is compulsory
- cost of writing to a memory matters like in flash memory (number of writes/swaps is `O(n)` as compared to `O(n2)` of bubble sort)

3. Insertion Sort

Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.

Characteristics :-

1. The array gets separated into 2 parts: one is sorted part and other is unsorted part.
2. Best Case = O(n)
3. Worst Case = $O(n^2)$
4. Space complexity is `O(1)` because an extra variable `key` is used.

Algorithm :-

1. The first element in the array is assumed to be sorted.
2. Take the second element and store it separately in `key`
3. Compare the key with the first element. If the first element is greater than `key`, then `key` is placed in front of the first element.
4. Now, the first two elements are sorted.
5. Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.
6. Continue the process till the size of left part of the sorted array gets equal to the size of the array

Code :

```
/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
         greater than key, to one position ahead
         of their current position */
        while (j >= 0 && arr[j] > key)
```

```
14.         {
15.             arr[j + 1] = arr[j];
16.             j = j - 1;
17.         }
18.         arr[j + 1] = key;
19.     }
20. }
```

Here is the complete code of insertion sort
https://sapphireengine.com/@/w2bksp

The insertion sort is used when:

- the array is has a small number of elements
- there are only a few elements left to be sorted

In above algorithms, Time Complexity is measured in terms of 2 factors :-

1. Comparing elements
2. Shifting the elements

Optimized Insertion Sort :- As i^{th} element is comparing with elements from i-1 to j>=0 which takes searching of element linearly which is Linear Search Whose time Complexity is O(n), We can reduce the following searching in O(logn) as the array from j=0 to i-1 is already sorted !!

Questions

Ques 1 : Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced).

Ques 2 :Consider a situation where swap operation is very costly. Which of the following sorting algorithms should be preferred so that the number of swap operations are minimized in general?

Ques 3 : What is the worst case time complexity of insertion sort where position of the data to be inserted is calculated using binary search?

Ques 4 : Which sorting algorithm will take the least time when all elements of the input array are identical? Consider typical implementations of sorting algorithms.

Ques 5 : Which of the algorithms takes the minimum no. of Auxiliary Space ?