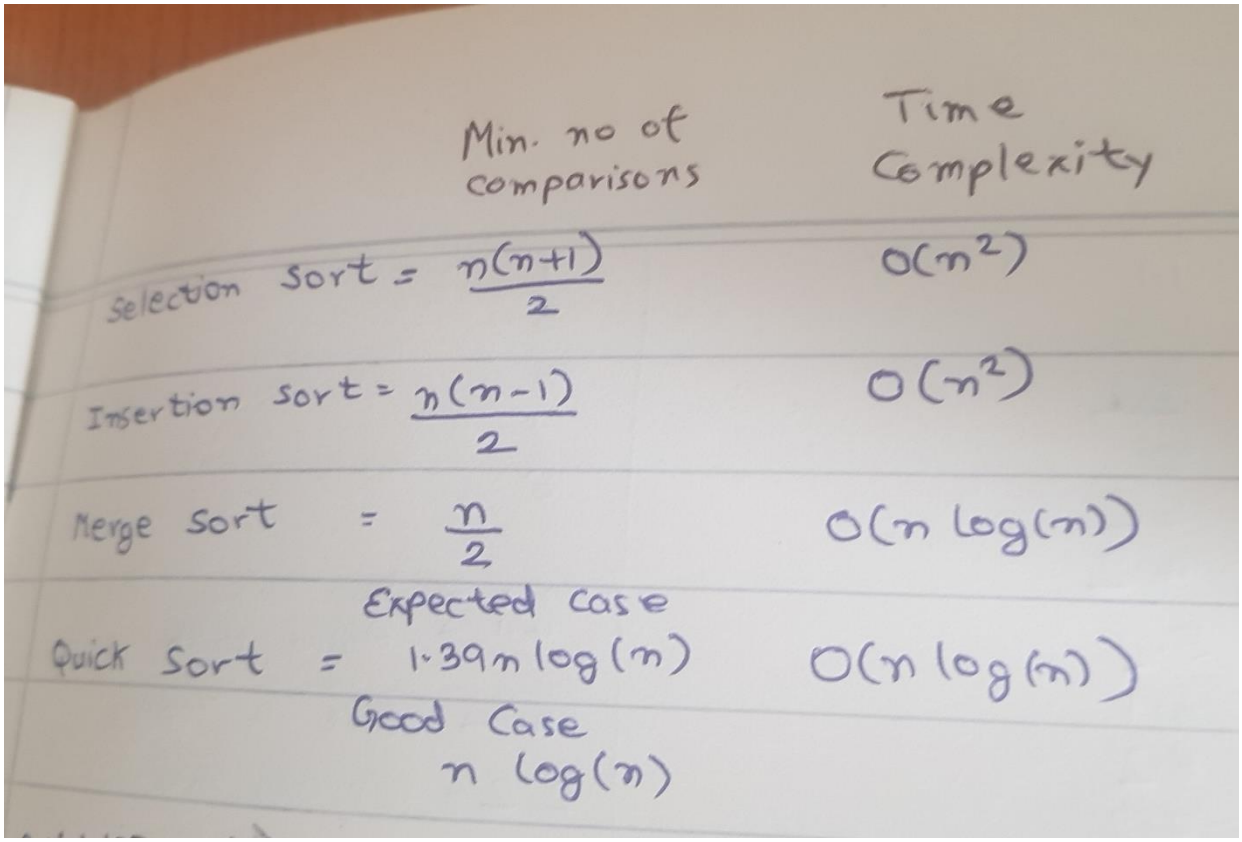


Searching in linear time O(n) means just doing a linear search. So, if you have an array of size n, and you start searching from beginning to end, that is searching in linear time.

Searching in logarithmic time $O(\log_2 N)$: That is using Binary search. Condition is that you should have your data sorted. Then you find middle element from your data structure, check if the number you are searching is equal to that middle element. If yes then we searched the element.

If it is less than number than we search only right part of our data structure with same strategy.

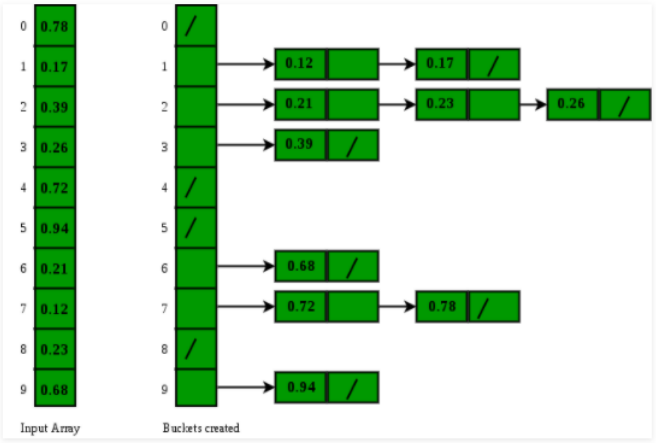
If it is greater than number than we search only left part of our data structure with same strategy.

Sorting in linear time Radix Sort: The analysis of the running time depends on the stable sort used as the intermediate sorting algorithm. When each digit is in the range *1* to *k*, and *k* is not too large, counting sort is the obvious choice. Each pass over *n d*-digit numbers then takes time $\Theta(n + k)$. There are *d* passes, so the total time for radix sort is $\Theta(dn + kd)$. When *d* is constant and $k = O(n)$, radix sort runs in linear time.

Sorting in linear time Bucket Sort:

Can we sort the array in linear time? Counting sort can not be applied here as we use keys as index in counting sort. Here keys are floating point numbers. The idea is to use bucket sort. Following is bucket algorithm.

```
bucketSort(arr[], n)
1) Create n empty buckets (Or lists).
2) Do following for every array element arr[i].
.......a) Insert arr[i] into bucket[n*array[i]]
3) Sort individual buckets using insertion sort.
4) Concatenate all sorted buckets.
```



Time Complexity: If we assume that insertion in a bucket takes O(1) time then steps 1 and 2 of the above algorithm clearly take O(n) time. The O(1) is easily possible if we use a linked list to represent a bucket (In the following code, C++ vector is used for simplicity). Step 4 also takes O(n) time as there will be n items in all buckets. The main step to analyse is step 3. This step also takes O(n) time on average if all numbers are uniformly distributed

For partially-sorted arrays, insertion sort performs well, it has linear time complexity.

An array where each entry is not far from its final position is a typical example of partially-sorted array. That's the case here for Bucket sort.