Searching :

- To search a key ele in a given collection/list of elements.

Linear Search:

1. Also called as "sequential search".
2. It sequentially checks each element of the list until the match is found or the whole list has been searched.

```
  Algorithm LinearSearch(A, n, key)
  {
    for( int index = 1 ; index <= SIZE ; index++ )
    {
        if( A[ index ] == key )
            return true;
    }
    return false;
  }
```

```
* Best case
   - when key is found at first pos then algo does only one comparison,
     time complexity of an algo in this case = O(1).

      Big Omega(1)


* Worst case
   - when either key is exists at last position or key does not exists
     algo does max "n" no. of comparisons whereas "n" size of an array,
     in this case time complexity of an algo = O(n).

     Big Oh(n)

* Average case
   - if key is exists at in between position the algo takes neither
     minimum nor max time to complete its execution, in this case time
     complexity of an algo = O(n/2) => O(n).

     Big Theta(n)
```

Binary Search:

1. Also called as "logarithmic search" or "half interval search"
2. This algo follows "divide-and-conquer" stratergy.
3. To apply binary search prerequisite is collection/list of elements must be in a sorted manner.
4. In the first iteration -- mid position gets calculated and key ele gets compared
5. With ele at mid position, if key ele is found then it will be the best case, otherwise array gets divided logically into two sub array's left subarray and right sub array.
6. If key ele is smaller than mid position ele then key ele gets searched into the left sub array only, by skipping the whole right sub array checking, or, if key ele is greater than mid position ele then key ele gets searched into the right sub
7. Array only by skipping whole left sub array.
8. The logic repeats either till key ele is not found or till size of an array is less than one.
9. If key ele is found at mid position in the very first iteration then no. of comparisons are "1" and it is considered as a best case, in this algo takes O(1) time, otherwise it takes O(log n) time.

```
Algorithm BinarySearch(A, n, key)
{
 left = 0;
 right = n-1;

 //till array/subarray is valid
 while( left <= right )
 {
     mid = (left+right)/2;

     if( key == A[mid] )
         return true;

     if( key < A[mid] )
          right = mid-1;
     else
         left = mid+1
 }
 return false;
}
```

10. As in every iteration this algo does 1 comparison and divides array into two sub arrays and key ele gets searched either one of the subarray, i.e. after every iteration it search space is divide almost by half, and hence we can write following recurrent equation for binary search, and can use substitution method to calculate its time complexity:

- Recurrent Equation:

```
for n = 1
T(n) = T(1) + 1
i.e. running time of an algo is O(1). --- trivial case
```

```
    for n > 1
    T(n) = T(n/2)+ 1 ..... (I)
    to get the value of T(n/2) put  n = n/2 in eq-I we get,
    => T(n/2) = T( n/2 / 2 ) + 1
    => T(n/2) = T(n/4) + 1 .....(II)

    substitute the value of T(n/2) in eq-I we get,
    => T(n) = [ T(n/4) + 1 ] + 1
    => T(n) = T(n/4) + 2 .....(III)


    to get the value of T(n/2) put n = n/2 in eq-II we get,
    => T( (n/2) / 2 ) = T( (n/4) / 2 ) + 1
    => T(n/4) = T(n/8) + 1 .... (IV)

    substitute the value of T(n/4) in eq-III we get,
    => T(n/4) = [ T(n/8) + 1 ] + 2
    => T(n/4) = T(n/8) + 3 ......(V)

    .
    .
    after k iterations:

    T(n) = T(n/2^k) + k

    for n = 2^k
    log n = log 2^k .... by taking log on both side
    log n = k log 2
    therefore, k = log n
    => T(n) = T(2^k/2^k) + log n
    => T(n) = T(1) + log n
    => T(n) = log n
```

```
and hence, T(n) = O(log n).
```

Difference between Linear Search and Binary Search:

- In linear search after every iteration search space is reduced by 1 i.e. (n-1) and in binary search search space is reduced by half of elements i.e. by (n/2).
- Worst case time complexity of linear search is O(n) and of binary search is O(log n) hence binary search is efficient than linear search.
- Binary search cannot be applied on linked list.

==========

Sorting:

Features of sorting algo's:

- Adaptive
  - When a sorting algo works efficiently for already sorted input sequence then it is reffered as an adaptive.
- Inplace
  - If a sorting algo do not takes extra space then it is reffered as inplace.
- Stable
  - If the relative order of two ele's having same key remains same after sorting then such sorting algo is reffered as stable.

Selection Sort:

- Inplace comparison sort
- This algo divides the list logically into two sublists, first list contains all elements and another list is empty.
- In the first iteration -- first element from the first list is selected and gets compared with remaining all ele's in that list, and the smallest ele can be added into the another list, so after first iteration second list contains the smallest ele in it.
- In the second iteration -- second element from the first list is selected and gets compared with remaining all ele's in that list and the smallest amongst them can be added into the another list at next position, so in second iteration the second smallest element gets added into the another list next to the smallest one, and so on.....
- So in max (n-1) no. of iterations all elements from first list gets added into the another list (which was initially empty) in a sorted manner and we will get all elements in a collection/list in a sorted manner.
- In every iteration one element gets selected and gets compared with remaining
- best case, worst case and average case time complexity of selection sort algo is O(n^2).

advantages:

1. Simple to implement
2. Inplace

disadvantages:

1. Not efficient for larger input size collection of ele's array/list.
2. Not adaptive i.e. not efficient for already sorted input sequence.

Bubble Sort:

- Sometimes reffered to as "sinking sort".

- This algo divides the list logically into two sublists, initially first list contains all elements and another list is empty.
- In the first iteration -- the largest element from first list gets selected and gets added into the another list at last position.
- In the second iteration -- largest element from the ele's left in a first list is selected and gets added into the second list at second last position and so on....
- So in max (n-1) no. of iterations all elements from first list gets added into the another list (which was initially empty) in a sorted manner from last position to first position and we will get all elements in a collection/list in a sorted manner. OR
- Ele's at consecutive locations gets compared with each other of they are not in order then they gets swapped otherwise their position remains same.

best case

- If array ele's are already sorted then this algo takes O(n) time

worst case and average case

- Time complexity of bubble sort algo is O(n^2).

advantages:

- Simple to implement
- Inplace - do not takes extra space for sorting ele's
- Can be implement as an adaptive
- Highly stable

disadvantages:

- Not efficient for larger input size collection of ele's array/list.
- Not adaptive in nature but can be implement as an adaptive