# Algorithms

# Linear search

This is the simplest technique for searching and the most obvious one as well. Time complexity of the algorithm is O(n), this means that our algorithm will take *n* operations to get the result. It goes like this:

1. Start from the left most element of the array. Compare each element with the key that we are searching for.
2. If the element matches the key. Return true/index if the element.
3. Else return false/-1.

# Binary search

Searching a sorted array by repeatedly dividing the search interval in half. If the search key is less than the item in the middle narrow the search interval to the lower half. Otherwise narrow the interval to the upper half. The basic idea behind binary searching algorithm is to use the information that the array is sorted and to reduce the time complexity to O(log n).

1. Compare search key (x) with the middle element. We can find the middle element with the formula L + (R – L ) / 2 where L is start (Left) index and R is the end(Right) index of the current interval.
2. If x matches the middle element, we return the middle index.
3. Else if x is less than the middle element, we take the left side as the next interval of the mid element.
4. Else if x is greater than the middle element, we take the right side as the next interval of the mid element.

# Jump search

Jump search is also a search for sorted arrays. The basic idea of jump search is to check fewer elements by jumping ahead by fixed steps or skipping some elements instead of checking all elements.

Now the question arises that what is the optimal jump that we should go for?

As in a worst-case scenario, we will have to make n/m jumps, where m is the block size. And if the last checked value is greater than the element to be searched for, we perform m-1 comparisons more for linear search. Therefore, the total number of comparisons in the worst case will be ((n/m) + m-1). The value of the function ((n/m) + m-1) will be minimum when m = √n. Therefore, the best step size is m = **√n.**

Time complexity for this search algorithm is O(√n).

1. Find value for one jump.
2. Jump from index 0 to index 0 + m.
3. Jump from index m to index 2m.
4. If element at index 2m is greater than the search key, go back to index m and start linear searching for the key.

# Fibonacci search

Fibonacci search is a comparison base technique that uses Fibonacci numbers to search and element in a sorted array. The idea is to first find the smallest Fibonacci number that is greater than or equal to the length of given array. Let the found Fibonacci number be fib (m’th Fibonacci number). We use (m-2)’th Fibonacci number as the index (If it is a valid index). Let (m-2)’th Fibonacci Number be i, we compare arr[i] with x, if x is same, we return i. Else if x is greater, we recur for sub array after i, else we recur for sub array before i.

# Selection sort

This algorithm sorts an array by continuously looking for the smallest element in the array. In each iteration it finds the smallest element and puts it in the beginning of the array. The algorithm maintains two sub arrays in the array. One is sorted and other is unsorted.

1. Start from index 0. Find the smallest value in the array and swap it with the element at the starting index that is 0.
2. Now start with next index that is 1. Repeat step ‘a’.
3. Keep repeating steps ‘a’ and ‘b’ until complete array is sorted.

# Bubble sort

Bubble sort is the simplest sorting algorithm which repeatedly swaps the adjacent elements if they are in the wrong order.