# Studying Datastructures and algorithms

- Roughly following the articles of : programiz.com/dsa
- Also following the book Introduction to Algorithms, third edition from Ronald L. Rivest.
- Also using youtube videos for resources where i might need them.

## Algorithms

### Linear Search

- The simplest Search algorithm that just takes an array *arr* and looks linearly through every index to see if the target *key* matches up with the given value of arr[index].

```
public static int linearSearch(int[] arr, int key){
    int out = -1;
    for (int i = 0; i < arr.length; i++) {
        if(key == arr[i]){
            out = i;
        }

    }
    System.out.println( (out != -1) ? arr[out] +" is at index: "+ out : key + "
Not found");
    return out;
}
```

**Time Commplexity**

Worst case: O(n)

> The worst case gets reached when the target *key* is found in the last index n denotes the length of the array

BestCase: O(1)

> Best case is reached when the target *key* is found in the first index of the array

---

### Binary Search

- Binary Search works with the idea that you take any given array, put a pointer towards the head and tail of the array and see if the Middle of those two numbers, used as an index is the given target.
  If this is not the case then either the head or tail pointer will be moved to i +/- 1. Effectively cutting the array left to be searched in half.
  This continues until the target is found.
  For this method to work the array **needs** to be sorted first.

- This can be Achieved with 2 methods:

*Itterative*

```
static int BinarySearch(int[] arr, int target){
int lowP = 0;
int highP = arr.length;

while(lowP <= highP) {
    int mid = (lowP + highP) / 2;
    if(target == arr[mid]) {
        return mid;
    } else if (target > arr[mid]){
        lowP = mid + 1 ;

    } else {
        highP = mid - 1;

    }
}
return -1;
}
```

> In the Itterative approache we put either the low or high pointer in the new middle of the cut up array depending on if it is smaller or bigger then the given target. When high and low Pointer meet the result *mid* is returned this is the index of the target that was searched for.

**recursive**

```
static int BinarySeach(int[] arr, int target){
int low = 0;
int high = arr.length;
return search(arr,target,low,high);

}

static int search(int[] arr, int target, int low,int high){
    int mid = (low + high) / 2;
    if(low > high) {
        return -1;
    }
    if(target == arr[mid]){
        return mid;
    } else if ( target > arr[mid]) {
        return search(arr, target,mid +1, high);
    } else {
        return search(arr,target,low,mid -1);
    }
}
```

> The recursive approach looks quite similar to the itterative one but by using the recursive method you could have better performance in certiant situations

**Time Complexity**

WorstCase: O(log(n))

> worst case is reached if the target is either at the ends of the array or not in the array.

BestCase: O(1)

> Best case is reached if the index of the target is directly in the middle of the given array.

## Bubble Sort

> Bubble Sort is one of the easiest methods to implement a searching algorithm.
> You take a given array *arr* start at the beginning and check if the element arr[i] is smaller, or bigger then arr[i + 1] if this is the case then the two values get swapped by use of a tempoary variable. this gets repeated until every element got itterated over and then agian for every element besides the ones already swapped

```
private static int[] bubbleSort(int[] arr) {
int swaps = 0;
for (int i = 0; i < arr.length; i++) {
    boolean swapped = false;
    for (int j = 0; j < arr.length - i - 1; j++) {

        if(arr[j] > arr[j +1]) {
            swaps++;
            int temp = arr[j];
            arr[j] = arr[j +1];
            arr[j +1] = temp;
            swapped = true;
        }

    }
    if(!swapped) {
        break;
    }
}
System.out.println(swaps);
return arr;
}
```

**Time Complexity**

WorstCase:O(n2)

Space:O(2)

> 0(1) if no bool:swapped variable is used