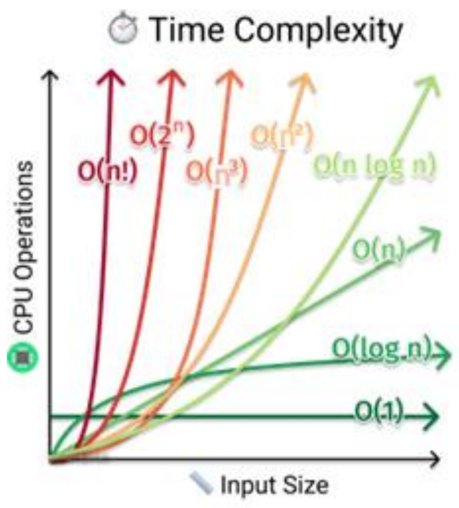
At this lab section, we will practice on algorithm analysis and recursion.

Algorithm Analysis & Recursion

Asst. Prof. Dr. Feriştah DALKILIÇ
Res. Asst. Fatih DİCLE

PART 1 – Algorithm Analysis

Finding out the time complexity of your code can help you develop better programs that run faster. The most common metric is Big O (Big Order) Notation. Big O notation cares about the worst-case scenario. It drops constants and lower order terms. E.g. O(3*n^2 + 10n + 10) becomes O(n^2). Most common Big O notation examples are:



Exercise 1. Specify the Big O notation equivalents for the given growth-rate functions.

Growth-rate function	Big O
$7n^4 + 2n^3 + n^2 + 11$	O(n^4)
$9^n + 2n^4$	O(9^n)
$n^5 + 5n^3 + 1$	O(n^5)
$12n^2 + 6$	O(n^2)
nlogn + n + 3	O(n log n)

Exercise 2. Specify the Big O complexities of the following algorithms.

Algorithm — **Big O**

```java
void floydWarshall(int dist[][]){
       for (int k = 0; k < dist.length; k++) {
            for (int i = 0; i < dist.length; i++) {
                for (int j = 0; j < dist.length; j++) {
                    if (dist[i][k] + dist[k][j] < dist[i][j])
                        dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
```

$O(n^3)$

```java
int binarySearch(int arr[], int x){
      int l = 0, r = arr.length - 1;
      while (l <= r) {
            int m = l + (r - l) / 2;

            if (arr[m] == x)
                  return m;
            if (arr[m] < x)
                  l = m + 1;
            else
                  r = m - 1;
      }
      return -1;
}
```

O(n)

```java
public static int fibonacciRecursion(int n){
      if(n == 0){
            return 0;
      }
      if(n == 1 || n == 2){
            return 1;
      }
      return fibonacciRecursion(n-2) + fibonacciRecursion(n-1);
}
```

O(n)

```java
public static void printArrayLength(int[] array){
      System.out.println("Array length is " + array.length);
}
```

O(1)

```java
public static void printArrayElements(int[] array){
      for (int i = 0; i < array.length; i++) {
            System.out.println(array[i]);
      }
}
```

O(n)

```java
public static void printFactorial(int n){
      for (int i = 1; i < factorial(n); i++) {
            System.out.println(i);
      }
}
```

O(n!)

PART 2 – Recursion

Exercise 3. At this section, you will find some codes written recursively. Test the codes on your computer and try to understand how they work.

recursion1.java

```java
public class recursion1 {

    static void printFun(int test)

    {

        if (test < 1)

            return;

        else {

            System.out.printf("%d ", test);

            printFun(test - 1);

            System.out.printf("%d ", test);

            return;

        }
    }

    public static void main(String[] args)

    {

        int test = 3;

        printFun(test);

    }

}
```

Exercise 4. Examine the code segment given below. What does *recursion2* do?

recursion2.java

```
public static int recursion2(int[] array, int length) {
        if (length == 1)
                return array[0];
        else {
                int x = recursion2(array, length - 1);
                if (x < array[length - 1])
                        return x;
                else
                        return array[length - 1];
        }
}
public static void main(String[] args) {
        int[] array = new int[] { 1, 15, 2, 0 };
        System.out.println(recursion2(array, array.length));
}
```

Your answer:

This recursion function helps us to return smallest member of our array. For doing that in the recursion2 function's else part our function calls itself but decreasing array length with one so our function can compare two components of the array. When the recursion part is done it returns the element which is the smallest.

Exercise 5. Write recursive version of the given iterative function.

Exp: a=132, b=84
a=48, b=84
a=48, b=36
a=12, b=36
a=12, b=24
a=12, b=12 Since a=b, the GCD is 12.

```
public static int greatest_common_divisor (int a, int b)
       {
         while (a != b)
         {
           if (a > b)
           {
             a -= b;
           }
           else if (b > a)
           {
             b -= a;
           }
         }
         return a;
         }
```

Your code
```
public class Main {
   public static int recursion(int a ,int b){
      if (a>b){
         return recursion(a-b,b);
      } else if (b>a) {
         return recursion(a,b-a);
      }
      return a;
   }
   public static void main(String[] args) {
      System.out.print(recursion(72,36));
   }
}
```

```java
public class Main {
   public static int recursion(int a ,int b){
      if (a>b){
         return recursion(a-b,b);
      } else if (b>a) {
         return recursion(a,b-a);
      }
      return a;
   }
   public static void main(String[] args) {
      System.out.print(recursion(72,36));
   }
}
```