

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

# Algorithm Analysis

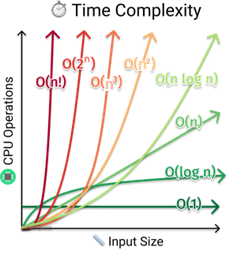
# &

# 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\*n2 + 10n + 10) becomes O(n2). 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** |
| 7n4 + 2n3 + n2 + 11 |  |
| 9n + 2n4 |  |
| n5 + 5n3 + 1 |  |
| 12n2 + 6 |  |
| nlogn + n + 3 |  |

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

|  |  |
| --- | --- |
| **Algorithm** | **Big O** |
| **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];  }  }  }  } |  |
| **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;  } |  |
| **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); } |  |
| **public** **static** **void** printArrayLength(**int**[] array){  System.***out***.println("Array length is " + array.length);  } |  |
| **public** **static** **void** printArrayElements(**int**[] array){  **for** (**int** i = 0; i < array.length; i++) {  System.***out***.println(array[i]);  }  } |  |
| **public** **static** **void** printFactorial(**int** n){  **for** (**int** i = 1; i < *factorial*(n); i++) {  System.***out***.println(i);  }  } |  |

## 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

**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:

|  |
| --- |
|  |

**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 |
|  |