Data Structures and Algorithms

Workshop – Algorithm Analysis

# Objectives

* Analyze the time efficiency of algorithms using Big-Oh notation.

# Introduction

In theoretical analysis of algorithms, it is common to estimate their complexity in the asymptotic sense, i.e., to estimate the complexity function for arbitrarily large input.

Big Oh notation characterizes functions according to their growth rates: different functions with the same growth rate may be represented using the same Big-Oh notation. The growth rate of a function is also referred to as the order of the function.

In this workshop, you are asked to analyze some algorithms using the Big-Oh notation.

# Exercises

1. What are the Big Oh of the following snippets?

a.

sum = 0;

for (int i = 1; i <= N; i += 4)

{

sum++;

}

b.

sum = 0;

for (int i = 1; i <= N; i \*= 2)

{

sum++;

}

c.

sum = 0;

for (int i = 1; i <= N \* N; i++)

{

for (int j = 1; j <= N \* N \* N; j++)

{

sum++;

}

}

1. What is the order of the following algorithm, given that the order of *SomeOperation()* is O(n)?

static void Method2(int n)

{

for (int i = 100; i >= 1; i--)

{

for (int j = 0; j < 2 \* n; j++)

{

for (int k = 1000; k > 2; k = k/2)

{

SomeOperation(n);

}

}

}

}

1. What is the Big Oh of the following method? Is there a best case or a worse case?

static void Method3(int[] arr, int n)

{

for (int i = 0; i < n - 1; i++)

{

int mark = PrivateMethod3(arr, i, n - 1);

int tmp = arr[i];

arr[i] = arr[mark];

arr[mark] = tmp;

}

}

static int PrivateMethod3(int[] arr, int first, int last)

{

int max = arr[first];

int indexOfMax = first;

for (int index = last; index > first; index--)

{

if (arr[index] > max) {

max = arr[index];

indexOfMax = index;

}

}

return indexOfMax;

}