

UNIVERSITY OF HELSINKI

---

# PreAssignment 2016

---

*Author:*

Andres Medina

(UAF 1608168)

February 13, 2016

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Materials and Methods</b>	<b>1</b>
2.1	Programming Tools . . . . .	2
2.2	Programming Logic . . . . .	2
2.2.1	Creating a Random Binary Tree . . . . .	2
2.2.2	Traversing a Binary Tree in inorder . . . . .	3
<b>3</b>	<b>Results</b>	<b>4</b>

# PREASSIGNMENT - TASK 1

## 1 Introduction

The first task of the preassignment consists in experimentally test that the amortized complexity of  $(n - 1)$  calls to **inorder\_next** in a binary tree is  $2(n - 1)$ .

To achieve this task, the following strategy was followed:

1. First, we create a program that generates random Binary Trees. This allows us to had a test set in which we prove the current hypothesis.
2. After creating the random Binary Tree generator, we create a second program that allows us to transverse the binary tree in inorder (*inorder means, first visiting the left node, then the root and finally the right node*). The program also returns the amount of steps involved in this operation.
3. By using both programs, we made a table that compare the amount of steps taken to transverse every binary tree in comparison with the number of nodes of each tree.
4. Finally, we made a plot with the data of the table and by using linear regression, we get the equation corresponding to that line plot. We expect that the slope of that line plot should be something bigger or equal to  $2(n - 1)$  (*it should be bigger or equal because the amortized complexity correspond to an upper bound of the actual complexity of an algorithm*).

The results obtained in the analysis are described in the results section of this paper.

## 2 Materials and Methods

In this section, we will describe the tools and the logic used to solve every single step that composed the task 1 of this preassignment. This section include topics like the programming framework used to solve this task and also what was the logic applied to develop those programs.

## 2.1 Programming Tools

For developing the required programs we choose to use the following tools:

- Python 3.5.1 as the programming language
- Github as a repository tool
- Sublime Text as the code editor

In addition, the random library of python was used for accomplish the tasks related with randomness.

## 2.2 Programming Logic

The two programs involved in this task were programmed by using OOP paradigm. Considering the fact that the task 1 of the preassignment involves the creation of two different programs, we choose to explain the logic of each program in a different subsection.

### 2.2.1 Creating a Random Binary Tree

The creation of the binary tree was implemented by developing two different classes: a binary node class and a binary tree class. The binary node class contains 3 different attributes, an Id, a reference to a left child node and a reference to a right child node while the binary tree class contains only two attributes, a list of binary nodes and the total cost of traversing the tree in inorder. By using this two classes, creating a random binary tree was possible. Also when every binary tree is created, a random tree setting method is called so that every tree created has a different and also random structure.

The random tree setting method work as follows:

- First, a integer number between 50 and 100 is choose randomly. This number will represent the number of nodes of the corresponding binary tree
- Second, a left-right flag is set by choosing a random integer number between 0 and 1

- Then, after the left-right flag is set, a random node of the tree is chosen. Only the nodes that doesn't already have the corresponding child are candidates for this random choice. *For example, if the left-right flag is set to left, only nodes that doesn't have left childs are possible candidates of this random choice. Conversely, if the left-right flag is set to right, only nodes that doesn't have right childs are possible candidates*
- Finally, the corresponding child is created in the randomly chosen node of the tree

The described logic repeats until the number of nodes of the binary tree is reached.

### 2.2.2 Traversing a Binary Tree in inorder

The traversing of a binary tree in inorder can be made mainly in two different ways: by using recursion and by not using recursion. In this case, we choose to implement this method using recursion as this method allow to achieve a smaller and cleaner code and also allow us to avoid the implementation of a link between a child node and it's parent node.

The idea of this implementation is mainly the following:

- We start by applying this method to the root node of the tree
- If the root node has a left child, then we call again the `traverse_inorder` method but this time we pass the left child node as an argument to the method
- When the corresponding node has no left child node, then we ask if the node has a right child. If the node has a right child, then we call again the `traverse_inorder` method but this time we pass the right child node as an argument to the method
- After no more left and right child are found, the `traverse_inorder` method returns. Of course we should consider a logic for counting the corresponding steps in each call of the method so that when the method ends, we have the total amount of steps already calculated.

In the next section we will show and analyze the results of the above implementations.

### 3 Results