# Course: ESO207A – Data Structures and Algorithms

Indian Institute of Technology Kanpur

**Programming Assignment 4 :** *Nearly Balanced Binary Search Tree ?*

## Most Important guidelines

- It is only through the assignments that one learns the most about the algorithms and data structures. You are advised to refrain from searching for a solution on the net or from a notebook or from other fellow students. Remember - **Before cheating the instructor, you are cheating yourself**. The onus of learning from a course lies first on you. So act wisely while working on this assignment.
- Refrain from collaborating with the students of other groups. If any evidence is found that confirms copying, the penalty will be very harsh. Refer to the website at the link: https://cse.iitk.ac.in/pages/AntiCheatingPolicy.html regarding the departmental policy on cheating.
- This assignment has to be done in groups of 2 only. It is your responsibility to find a partner.
- Submit 2 files. First file is a pdf report containing the 2 plots as described in point 4 on the following page. The second file is a .c file that has the implementation of the nearly balanced binary search tree. Please follow the naming convention <RollNo1>_<RollNo2>_A4.pdf and <RollNo1>_<RollNo2>_A4.c while submitting your assignmment on Moodle. For example - 210449_210733_A4.pdf and 210449_210733_A4.c. You may use any word processor (latex, word, ...) to prepare the report file. However, no scanned copy of a handwritten submission is allowed as report.
- Both the students in a group must submit the same files on Moodle.
- In case of any issue related to this assignment, send an email at anukal@cse.iitk.ac.in (and **not to the instructor**).

## The Objective of the Assignment

In this course, we (re)-invented the novel data structure called the Binary Search Tree (BST). Along with that, the concepts of perfectly-balanced binary search trees and nearly-balanced binary search trees were also introduced. It was shown that in case of a normal BST, if the data is inserted in increasing/decreasing order, the tree becomes skewed, and the time to search an element in this case can be as bad as searching in a linked list. Moreover, it was claimed that if we can maintain a nearly-balanced BST *efficiently*, then we can answer each search query in $O(\log n)$ time, where $n$ is the number of elements present in the tree at that time. We also discussed a very simple and quite intuitive algorithm to maintain nearly-balanced BST. The idea was to transform a subtree into a perfectly balanced binary search tree as soon as it ceases to be nearly-balanced. Though this idea seems to be quite nice and looks promising, the formal analysis of the algorithm is not at all obvious. The formal analysis of this algorithm is deferred towards the end of this course. However, this is the right time for you to verify this claim experimentally. This goal will be achieved through this assignment:

## Tasks to be done

1. You need to implement the nearly-balanced binary search tree using the algorithm discussed in the lecture. If there are multiple nodes whose subtrees (the subtree rooted at them) cease to be nearly balanced, do we need to rebuild each of them. Can't we just rebuild only one of them ? Think along these lines and implement the nearly-balanced binary search tree accordingly.

2. You need to insert numbers from 1 to $10^7$ in the increasing order of their values. When you are inserting a number, say $i$, then you need to find out the time it took to insert the number $i$ (including the time to carry out the rebuilding of any subtree during this process, if required). Store this time in an array. The time required to insert numbers from 1 to $i$ is the sum of first $i$ elements of this array.

3. In the C program that you have written, print the total time required for the insertion of values from $i = 1$ to $i = n$ where $n$ is a multiple of $5\times10^4$ and $n \leq 10^7$ ($n = 5\times10^4, 10\times10^4, 15\times10^4, ...1000\times10^4$). Print the values in a .csv file using File output. In another .csv file, print the time taken for the $i^{th}$ insertion, where $10^5 \leq i \leq 10^6$.

4. Now plot the following graphs
   (a) Total time required for insertion of values from $i = 1$ to $i = n$ as a function of $n \log_2 n$ where $n$ is a multiple of $5 \times 10^4$ and $n \leq 10^7$.
   (b) Time taken for the $i^{th}$ insertion as a function of $i$ where $10^5 \leq i \leq 10^6$ .

**Note:** You will require a lot of computation time for steps 3 and 4. So please start working on the assignment well in time. Otherwise, you will not be able to submit the assignment on time.

## Useful pointers and Suggestions

1. For calculating the actual running time in executing a function, you may use clock() function in C. For this you need to include the time.h library. The following code shows how to use it to find the time taken in executing a part of the code:

```
#include<stdio.h>
#include<time.h>        // for clock_t, clock()
void func(){
    printf("Hello World");
}
int main()
{
    clock_t start_t, end_t;
    double total_t;
    start_t = clock();
    func();
    end_t = clock();
    total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
    // CLOCKS_PER_SEC is a constant defined in time.h and its value is 10^6.
    printf("Total time taken: %f\n", total_t );
    return 0;
}
```

   clock() returns the actual time in microseconds.

2. You can plot using any tool like Matplotlib, gnuplot, Microsoft Excel.

3. When using arrays, make sure that they are declared **globally**. Declaration of array locally inside main or any other function might lead to segmentation fault.

4. While transforming a subtree, say $T'$, into a perfectly balanced BST, you should ideally store (the pointers to) all the nodes of $T'$ in the increasing order of their key values. This is a better and more efficient approach than storing only the keys in the array (in increasing order of their values). This is because, in the latter case, you would have to waste time in freeing the memory allocated for old nodes and allocating memory while creating the new nodes.

5. Please **refrain** from doing unnecessary optimization (like rotation of tree). The objective of this assignment, as mentioned clearly on top of Page 2, is not to test whether you can implement a red-black tree. Please do the tasks in the way you have been asked to do explicitly. If you have free time, utilize that in coming with a theoretical explanation for the experimental results. That is more useful, interesting, deeper, and more challenging. For any query, please contact the corresponding tutor responsible for this assignment.