# **Comprehensive BST activity**

## ReflexAct 3.4

Mariana Esquivel Hernandez A01641244

First, we obtained the IP element from the line of the txt file, and we separated the ports since we only needed Ips addresses.

From that, we create a Heap Tree Structure, and we specify that the Nodes are going to contain elements from a vector that are the IPs and the keys for each element that are the quantity of repetitions of each. We initialize the repetitions value from the Node in “1” since each time we add a element to the Tree we consider it as 1 visualization.  
We Introduce the value as a string and before ordering it into the tree we evaluate if the value already exists in any of the nodes, if that is the case, we add to the key of the Node +1 for each repetition of the IP. If not, we accommodate the value into the Tree.

The functions we used for the solution of the activity and their complexity are:

* Insertion for the maxheap – Complexity O(n)
* top(), ShiftUp & ShiftDown – Complexity O(log n)
* getIP & getIPAccess – Complexity O(n)
* main – Complexity O(n log n)

Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.

Binary search trees are collections that can efficiently maintain a dynamically changing dataset in sorted order, for some "sortable" type.

Having a sorted array is useful for many tasks because it enables binary search to be used to efficiently locate elements. The problem with a sorted array is that elements can't be inserted and removed efficiently.

The binary search tree is a different way of structuring data so that it can still be binary searched (or a very similar procedure can be used), but it's easier to add and remove elements. Instead of just storing the elements contiguously from least to greatest, the data is maintained in many separate chunks, making adding an element a matter of adding a new chunk of memory and linking it to existing chunks.

Binary search trees support everything you can get from a sorted array: efficient search, in-order forward/backwards traversal from any given element, predecessor /successor element search, and max /min queries, with the added benefit of efficient inserts and deletes. With a self-balancing binary search tree (BST), all the above run-in logarithmic time.

It can store any type that has a total order defined on it. A total order means a binary comparison operation between elements has been defined, and the elements can be arranged in a unique sequence from smallest to greatest based on this operation. For example, integers or reals with the typical order.

From other side, a max heap is a complete binary tree in which the value of a node is greater than or equal to the values of its children. Max Heap data structure is useful for sorting data using heap sort.

Balancing a heap is done by sift-up or sift-down operations (swapping elements which are out of order). As we can build a heap from an array without requiring extra memory (for the nodes, for example), heapsort can be used to sort an array in-place.

After an element is inserted into or deleted from a heap, the heap property may be violated, and the heap must be re-balanced by swapping elements within the array.