Exersice 01: (04M)

A (02 M)	***B (02 M)***
j takes this range of values ***(2,2^2, 2^3,…2^n) → 2^n<=n → n = log_2n*** ***T(n) = n/2 * log_2n*** → *O(nlogn)*	***i takes the values 1, K1, K2, … ,km*** ***km<n ⍰ m=logkn ⍰ O(logmk)***

Exersice 02: (04M)

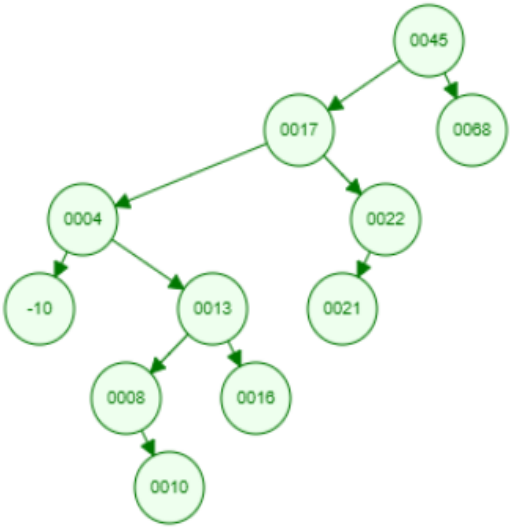
```
int n = len(T), count = 0;
Function inversionsArray (int [] T,int n){
For(int i=0;i<n-1;i++)
   For(int j=i+1;j<n;j++)
      If(T(i) > T(j)) then
         Count++;
Return(Count)
}
```

.

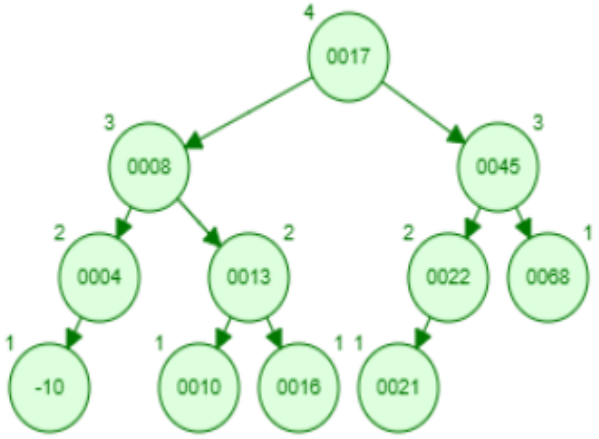
Exersice 03 (06M)

L= {45, 17, 4, 22, 13, ,68, 8, 21, 10, 16, -10}

BST (02M)



AVL (02 M)



preorder (prefixe) : 45 17 4 -10 13 8 10 16 22 21 68 **(0.5M)**
inorder (infixe) : -10 4 8 10 13 16 17 21 22 45 68 **(0.5M)**
postorderorder (Postfixe) : -10 10 8 16 13 4 21 22 17 68 45 **(0.5M)**
breadth-First (Largeur) : 45 17 68 4 22 -10 13 21 8 16 10 **(0.5M)**

Exercise 04: (06 M)

1. **Identify the Algorithm:** What sorting algorithm is implemented in this code? Explain your reasoning.

This code is an implementation of selection sort algorithm because it depends on finding the minimum and then doing the swaps. **(02 M)**

2. What is T (1)? (The number of comparisons when the array has only one element).

T(1) = 1 because we have one element so there is no comparison **(01 M)**

3. **Recursive Case:** Express T(n) in terms of T(n-1). Consider how many comparisons are made in each call to recursiveSort *before* the recursive call.

T(n) = T(n-1) + f(n) f(n) = n **(01 M)**

4. **Write the complete recurrence relation:** Combine the base case and the recursive case into a single recurrence relation. And then deduce the time complexity of this algorithm. **(02 M)**

T(n) = T(n-1) + n = T(n-2) + n + n = ….= T(n-p) + n + n + … p times

The recursion will end when p reach n-1 so T(n) = T(1) + n(n-1)

So the time complexity is $O(n^2)$