Level: 2nd year "Computer Science"	**Module:** Algorithmic and Data Structures 3	14 Janvier 2025 **Duration**: 1h30m

Typical correction

Exercise n° 1 (5 pts):

1. Determine the asymptotic complexity of both algorithms in Big-O notation. Which algorithm has the better asymptotic complexity? 1.5 pts

- The asymptotic complexity of $T_1(n)= 18n +84$ is: $T_1(n)\in \mathbf{O(n)}$
- The asymptotic complexity of $T_2(n)$ is: $T_2(n)= n^2 + 12n+ 12 \in \mathbf{O(n^2)}$

Comparison:

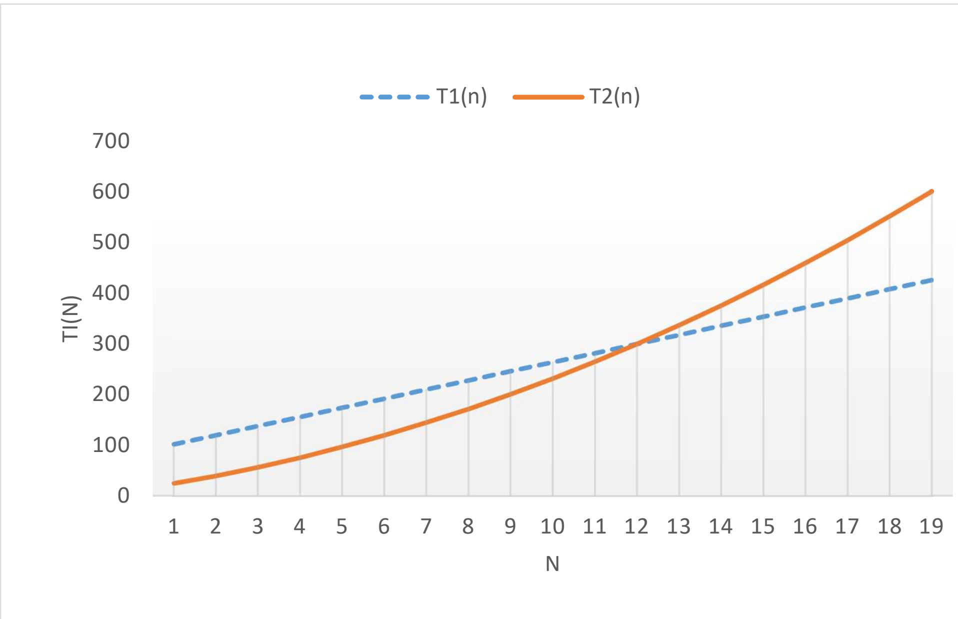
- $O(n)$ grows slower than $O(n^2)$, so algorithm A_1 has better asymptotic complexity than algorithm A_2.

2. Show that your solutions are correct by specifying a constant c and n_0 such that the following relation is satisfied: $O(f) = \{g \mid \exists\, c > 0, \exists\, n_0 > 0 : \forall n > n_0 : g(n) \leq c * f(n)\}$ **1pt**

For $T_1(n) \in O(n)$: Let's choose c=19 and n_0=84 : $18n+84 \leq 19n$ for $n > 84$.

For $T_2(n) \in O(n^2)$: Let's choose c=2 and n_0=12 : $n^2+12n+12 \leq 2n^2$ for $n >12$
OR c=3 and n_0=6

3. Sketch the curves of both functions $T_1(n)$ and $T_2(n)$ on the same graph. 1.5 pts

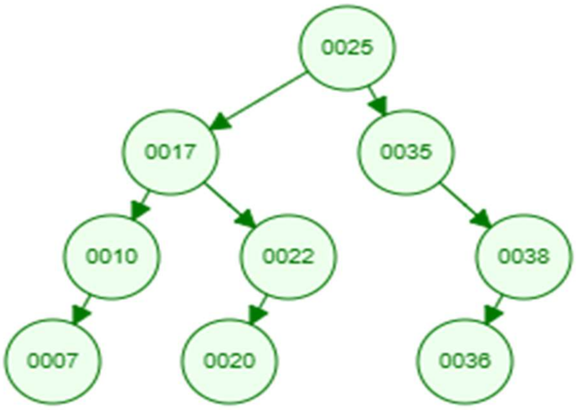


4. For what lengths of data n is each of the algorithms most efficient? 1pt

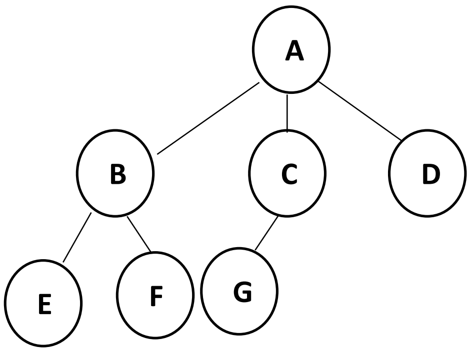
- From the previous curves, we observe:
  - For $n \in [\, 0, 12[$ A_2 more efficient than A_2.
  - For $n > 12$ A_1 is more efficient than A_1.

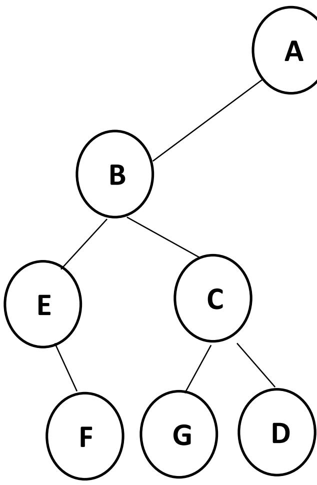
Exercise n° 2 (7.5 pts):

1. **T is** a Binary Search Tree (BST). Its level-order traversal (from left to right) yields the following result: 25, 17, 35, 10, 22, 30, 38, 7, 20, 36.
   - Provide the graphical representation of this tree T. **1pt**

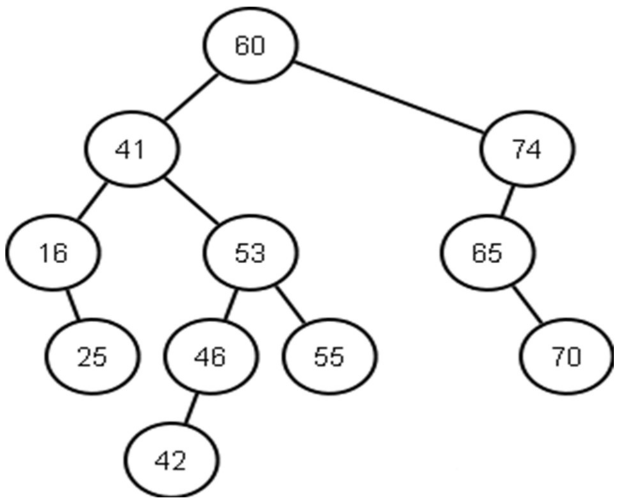


2. Let's consider this N-ary tree with N=3
   - Convert it to a binary tree. **1pt**





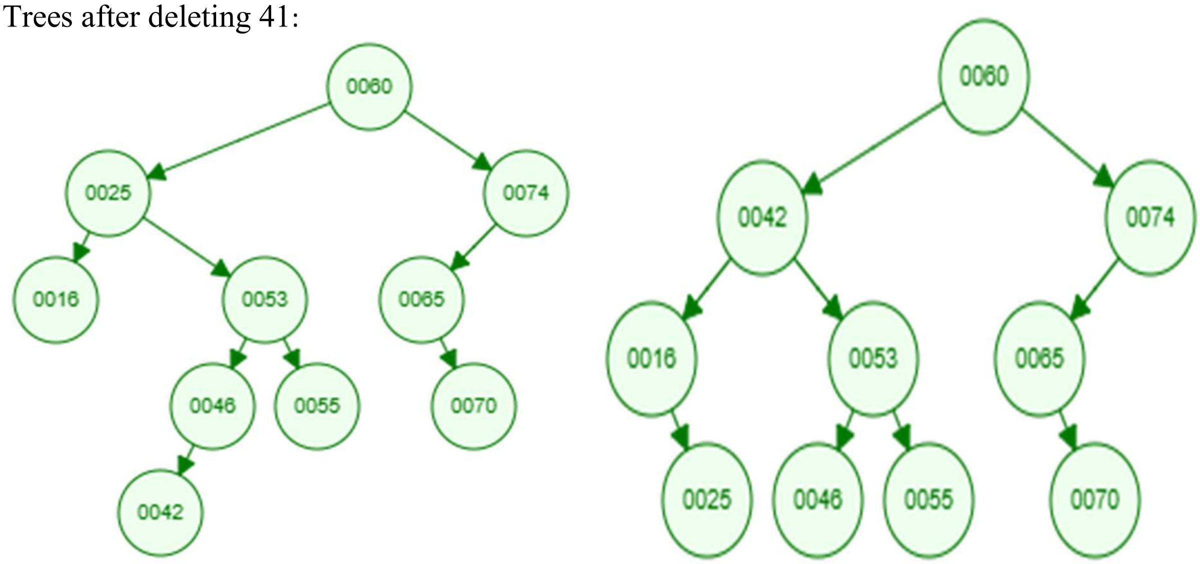
3.
   a. Give the result of the three depth-first traversals of the following binary search tree. **1.5 pts**



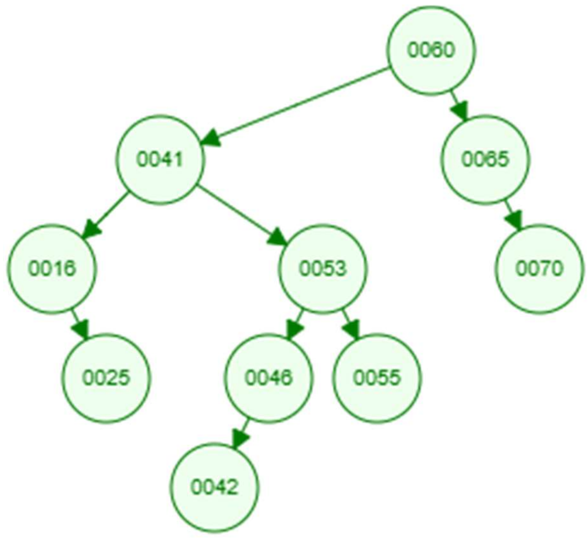
- **In-Order Traversal** (Left, Root, Right): 16, 25, 41, 42, 46, 53, 55, 60, 65, 70, 74
- **Pre-Order Traversal** (Root, Left, Right): 60, 41, 16, 25, 53, 46, 42, 55, 74, 65, 70
- **Post-Order Traversal** (Left, Right, Root): 25, 16, 42, 46, 55, 53, 41, 70, 65, 74, 60

b. Show the new trees obtained after deleting these nods 41,74 and 55. **2 pts**

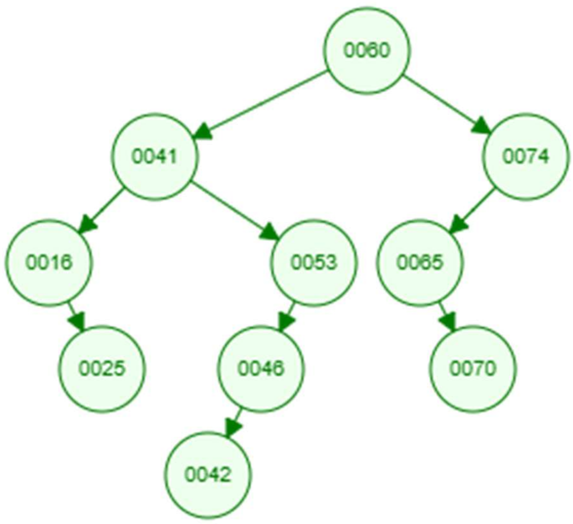
Trees after deleting 41:



Tree after deleting 74



Tree after deleting 55



4. Write a function in C to check whether a binary search tree T2 is contained within another binary search tree T1. **2 pts**

bool areIdentical(Node* T1, Node* T2) {

if (T1 == NULL && T2 == NULL) return true;

if (T1 == NULL || T2 == NULL) return false;

return (T1->data == T2->data) && areIdentical(T1->left, T2->left) &&

areIdentical(T1->right, T2->right); }

bool isSubtree(Node* T1, Node* T2) {

if (T2 == NULL) return true;

if (T1 == NULL) return false;

if (areIdentical(T1, T2)) return true;

return isSubtree(T1->left, T2) || isSubtree(T1->right, T2); }

Exercise n° 3 (7.5 pts): Let’s recall the bubble sort algorithm

```
void BubbleSort (char* t, int n) {

   int i, j;

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

        for (j= n-1; j > i; j--)

             if (t[k] < t[k - 1])   swap(&t[k], &t[k - 1]);

}
```

Given the following character array: I N F O R M A T

1. Provide the successive states of the array at the end of each step of the internal loop when i = 0.   **1.5 pts**

   j=7: INFORMAT   j=3 : INAFORMT

   j=6: INFORAMT   j=2 : IANFORMT

   j=5: INFOARMT   j=1 : **A**INFORMT

   j=4 : INFAORMT

2. Provide the successive states of the array at the end of each step of the external loop. **1.5 pts**

   i=1: **AF**INMORT   i=4: **AFIMN**ORT

   i=2: **AFI**MNORT   i=5: **AFIMNO**RT

   i=3: **AFIM**NORT   i=6: **AFIMNOR**T

3. Write the recursive version of the bubble sort. **1.5 pts**

```
void bubbleSortRecursive(int arr[], int n) {

   if (n == 1)     return;

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

      if (arr[i] > arr[i + 1])

           swap(&arr[i],&arr[i + 1])

   }

   bubbleSortRecursive(arr, n - 1);

}
```

4. Write a function in C that merges two arrays T1 and T2 of respective sizes N1 and N2, whose values are sorted in ascending order, into an array T of size N=N1+N2. The resulting array T should contain all the values sorted in descending order. **3pts**

Example: **T1**: 1,4,5,9. **T2**: 2,3,7,8 → T: 9,8,7,5,4,3,2, 1.

```
void mergeArraysDescending(int T1[], int N1, int T2[], int N2, int T[]) {

  int i = N1-1, j = N2-1, k = -1;

  while (i >= 0 && j >= 0) {

      k++;

      if (T1[i] < T2[j]) {

          T[k] = T2[j];

          j--;

     } else {

          T[k] = T1[i];

          i--;

     }

      }

  while (i >= 0) {

      k++;

      T[k] = T1[i];

      i--;

  }

  while (j >= 0) {

      k++;

     T[k] = T2[j];

     j--;

  }

}
```