Lab 4

Houssam Eddine ATIF

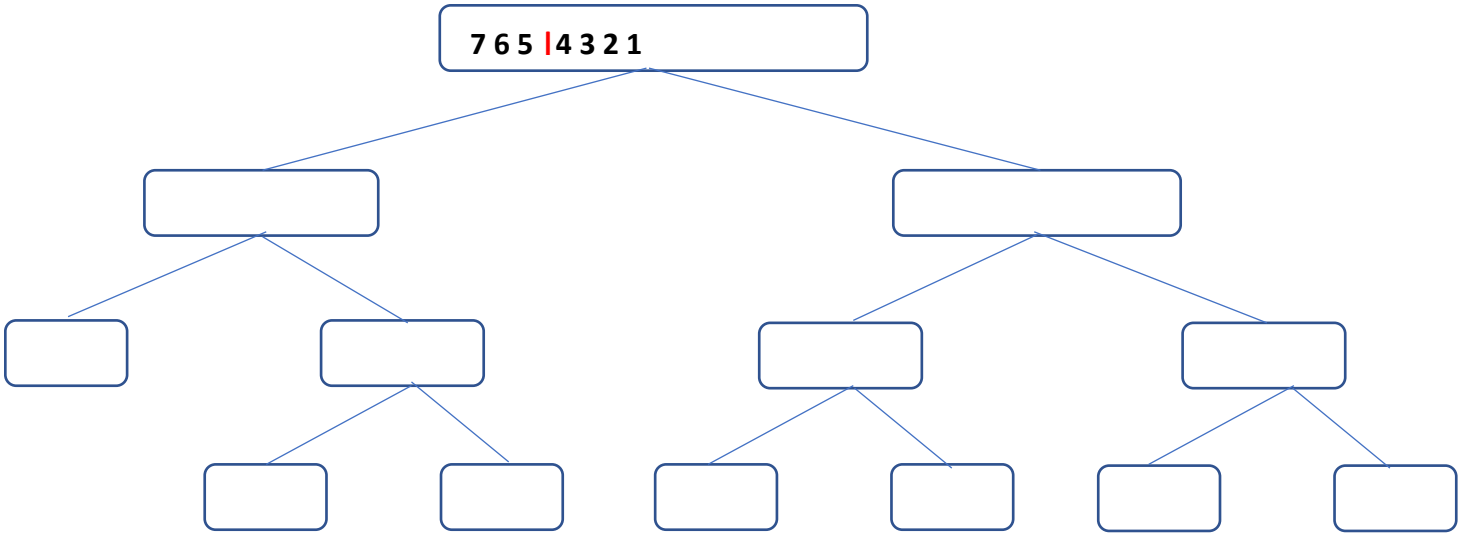
N# 610165

1/

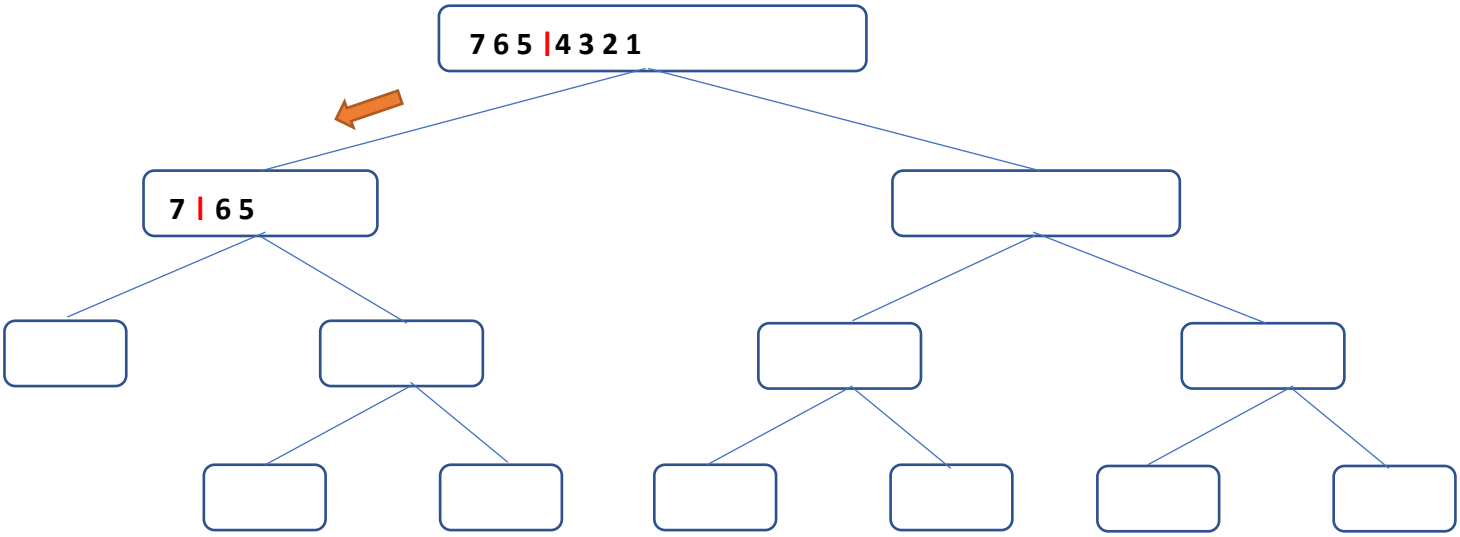
1- BubbleSort is a stable sorting algorithm because two equal elements will never be swapped.
2- InsertionSort is a stable sorting algorithm because two equal elements will never be swapped.
3- SelectionSort is not a stable sorting algorithm as it changes the relative order of elements with the same value in sorting procedure.

2/

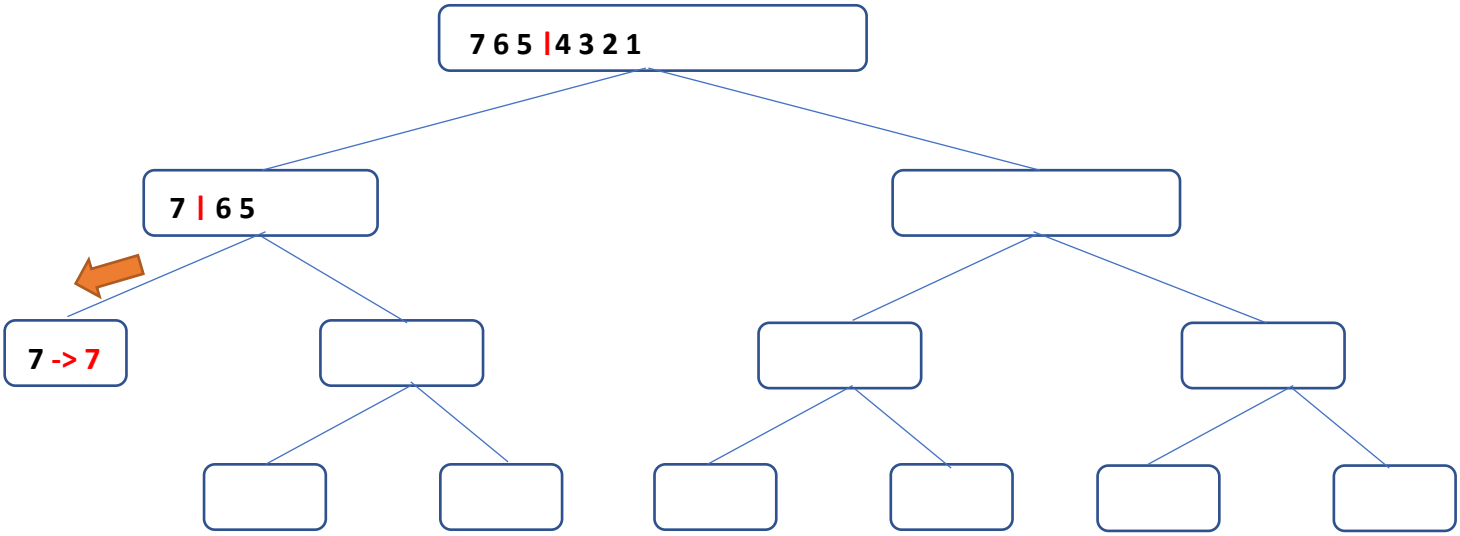
Partition



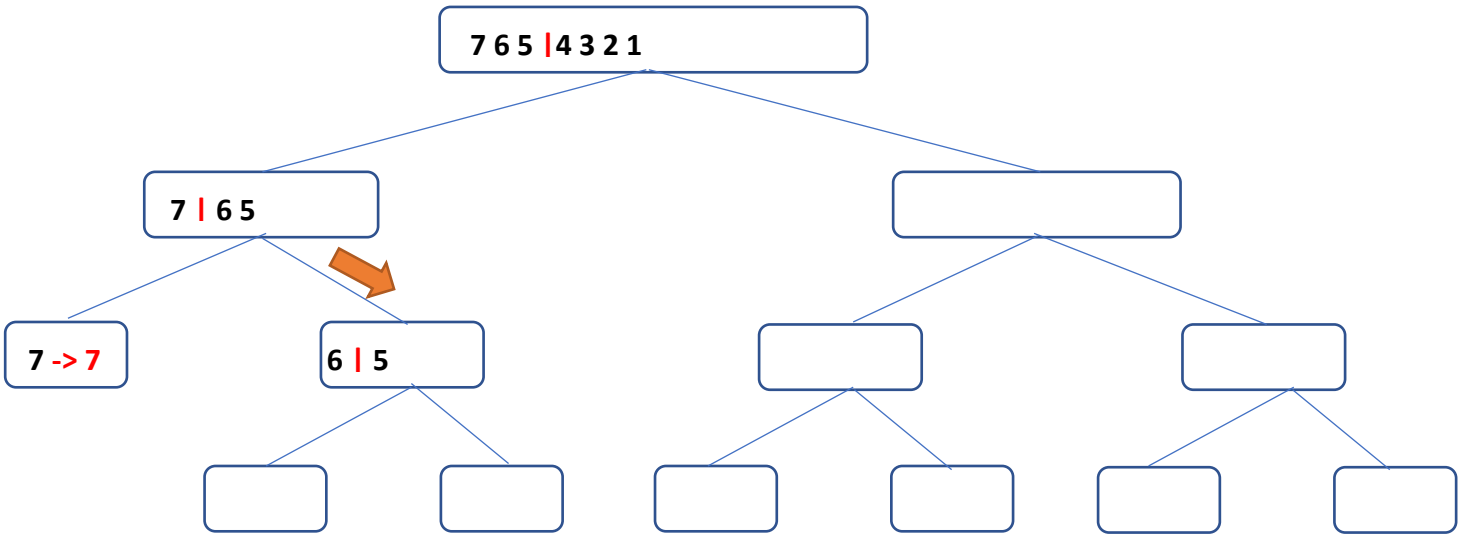
Recursive call, partition



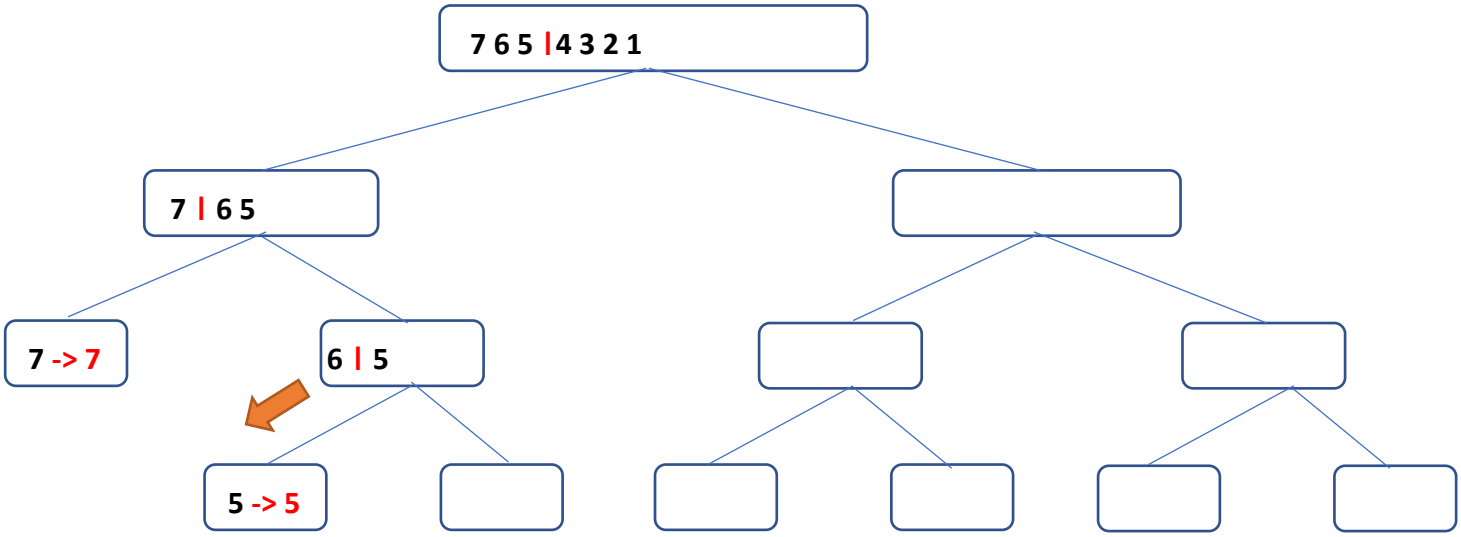
Recursive call, base case



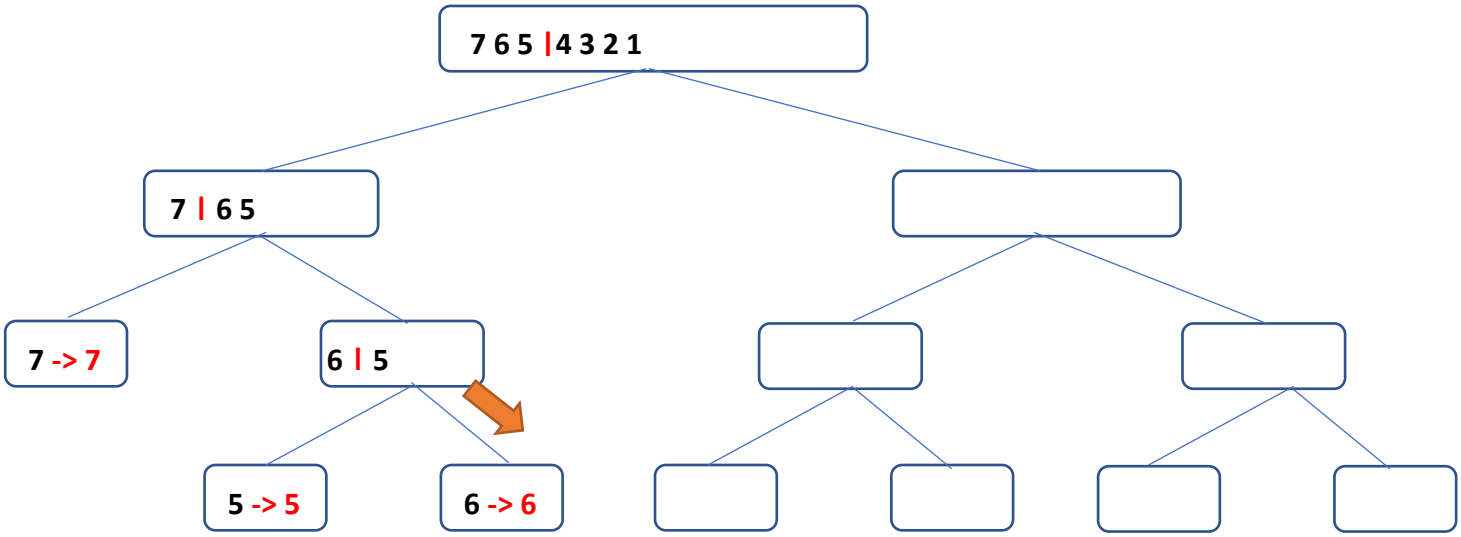
Recursive call, base partition



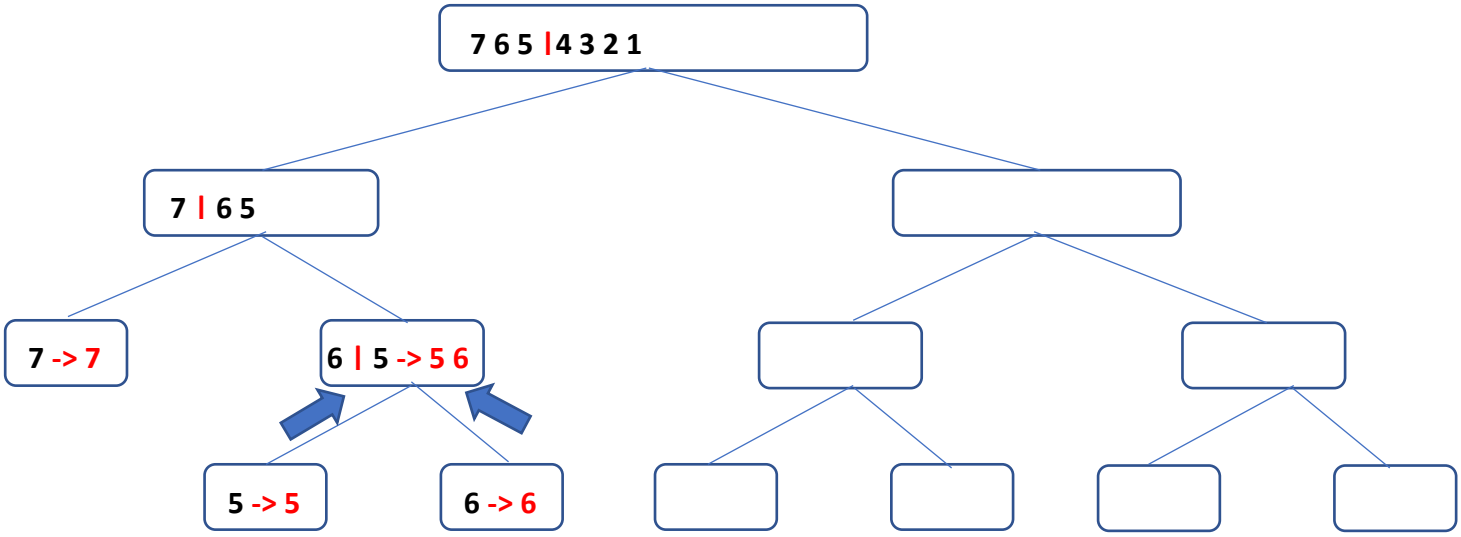
Recursive call, base case



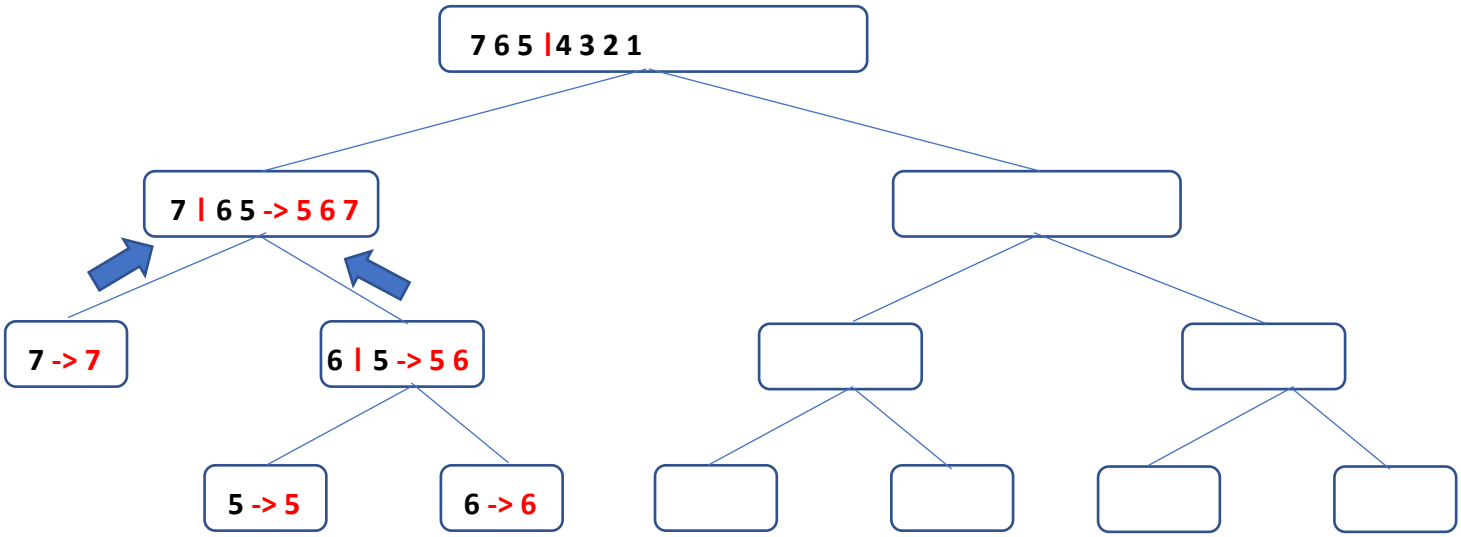
Recursive call, base case



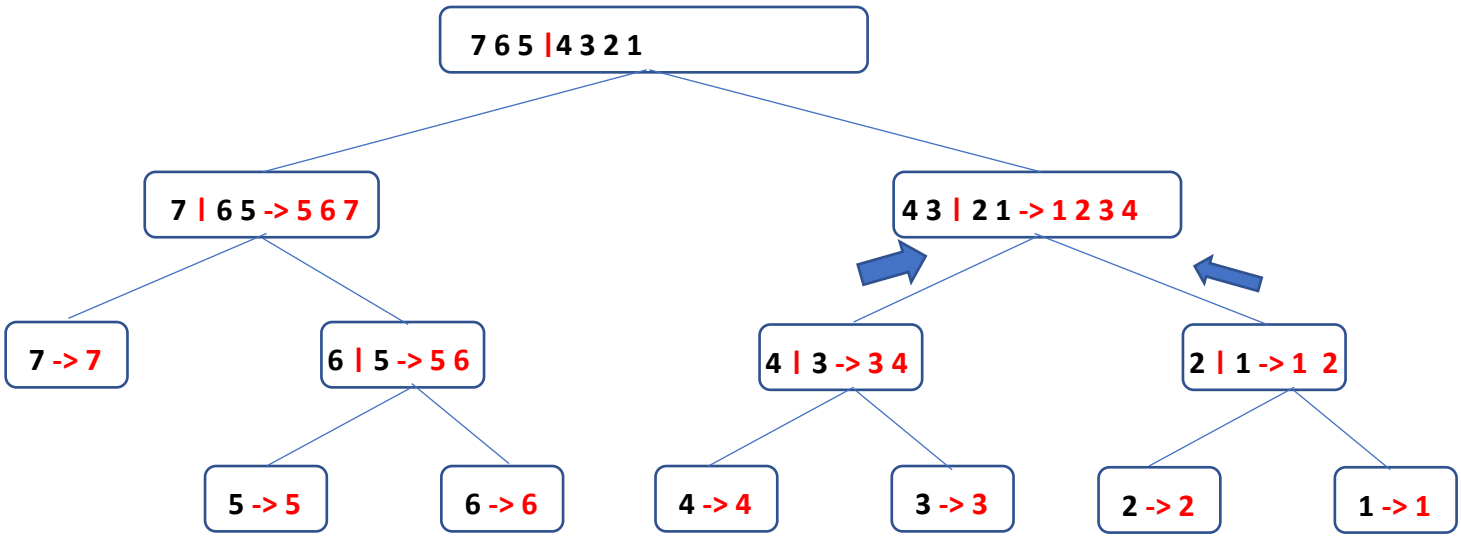
Merge



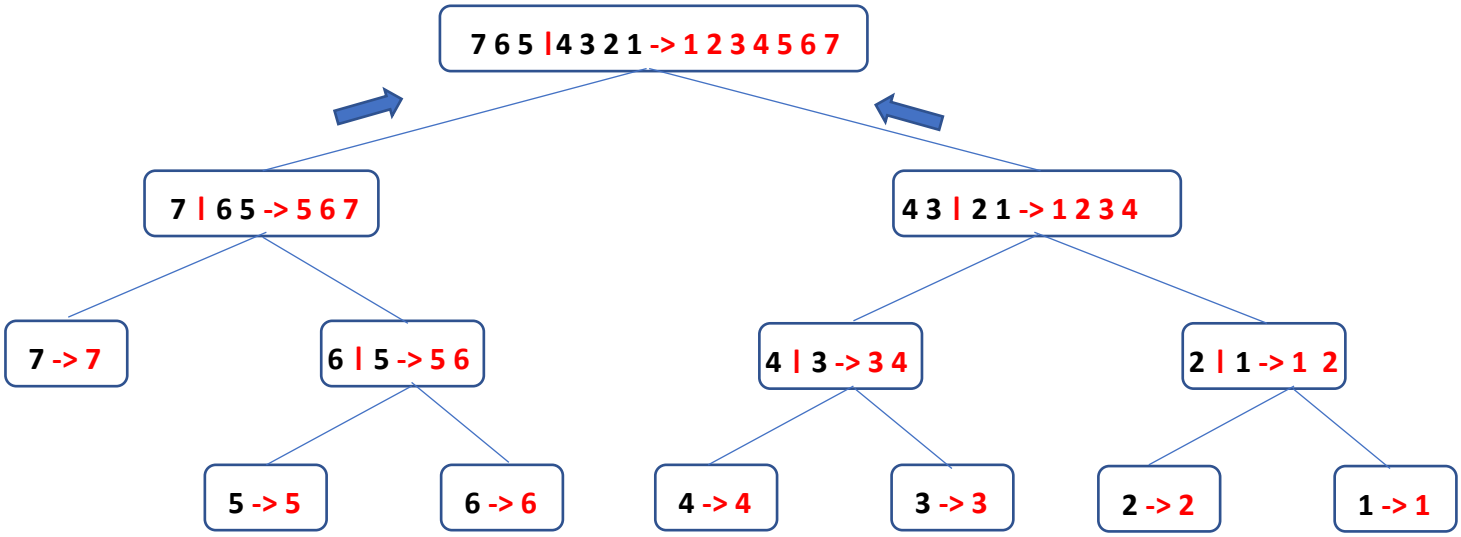
Merge



Recursive call, …, merge, merge



Merge



3/

A/

Algorithm mergeSortPlus(S)

Input sequence S with n integers

Output sequence S sorted

if S.size() <= 20 then

  InsertionSort(S)

if S.size() > 1 then

  (S1 , S2) ← partition(S, n/2)

  mergeSort(S1)

  mergeSort(S2)

  S ← merge(S1 , S2)

return S

B/

```
void mergeSort(int[] tempStorage, int lower, int upper) {
              if(upper==lower) {
                    return;
              }

              if(upper-lower<=20){

                    if(tempStorage == null || tempStorage.length <= 1) {
                          return;
                    }
                    int temp = 0;
                    int j = 0;
                    for(int i = lower+1; i < upper+1; ++i) {
                      temp = theArray[i];
                      j=i;
                      while(j>lower && temp < theArray[j-1]){
                         theArray[j] = theArray[j-1];
                          j--;
                      }
                      theArray[j]=temp;
                    }

                      return;
              }

              else {
                    int mid = (lower+upper)/2;
```

```
                mergeSort(tempStorage,lower,mid);  //sort left half
                mergeSort(tempStorage,mid+1, upper); //sort right half
                merge(tempStorage,lower,mid+1,upper); //merge them
            }
        }
```

B/

```
234 ms -> MergeSortPlus
290 ms -> MergeSort
7526 ms -> InsertionSort
```

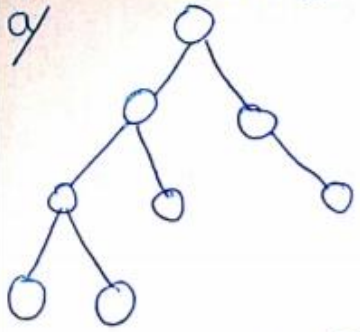
The MergeSortPlus is the fastest now. When we have small length of array like in this example 20, the insertion sort algorithm performs better and it has faster running time compared with a merge sort algorithm and this is related of the number of the constant operations. In insertion sort algorithm the number of constant operations is less than the merge sort algorithm.

4/

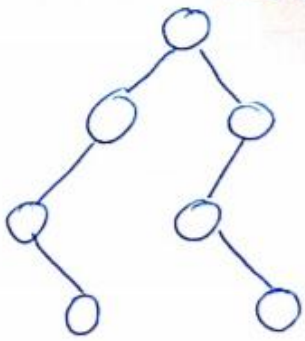
4/

a/

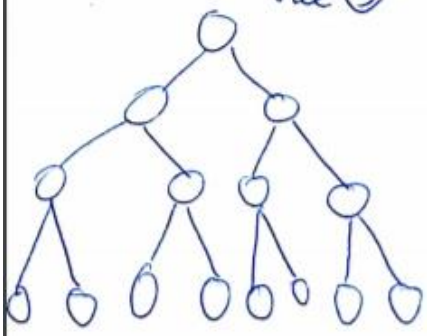
tree ①



tree ②



tree ③



b/ → tree ① has 8 nodes and 4 leaves $< 2^3$ ⇒ true

→ tree ② has 7 nodes and 2 leaves $< 2^3$ ⇒ true

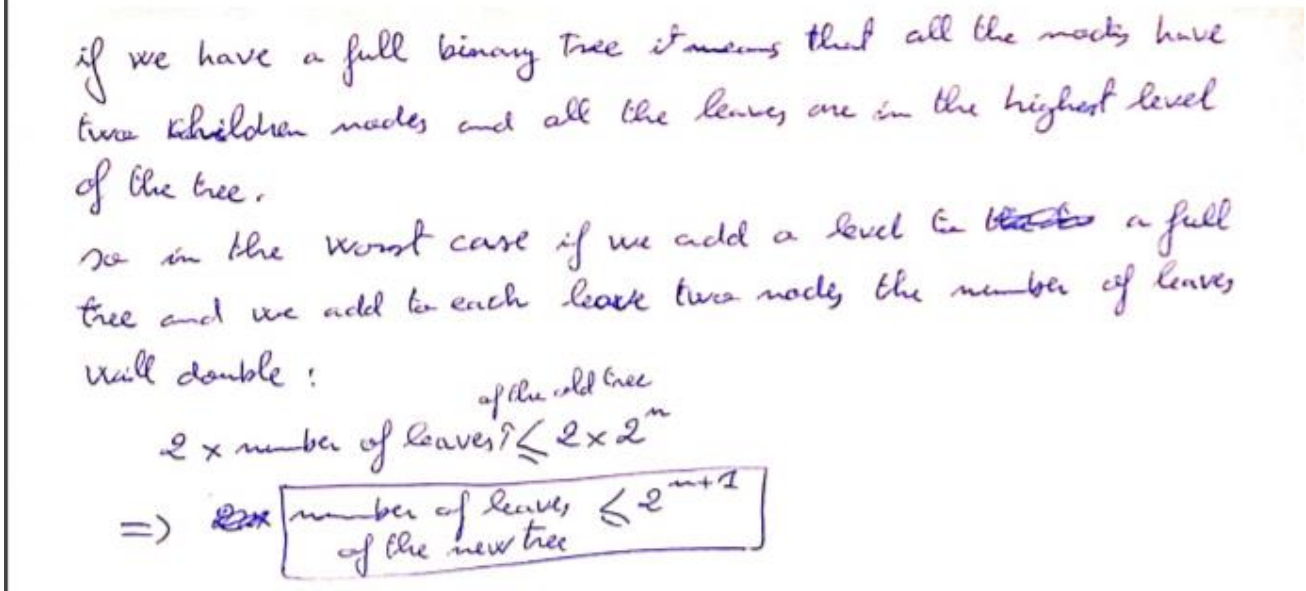
→ tree ③ has 15 nodes and 8 leaves $= 2^3$ ⇒ true

c) → we have $2^0 = 1$ ~~is a level~~ a tree that has just level 0 it contains just 1 node : the root which is true.

→ we suppose that $2^n \leq$ the number of leaves of a tree (n is the number of levels or the high) is true and prove with induction that $2^{n+1} \leq$ the number of leaves of a tree

→ ~~prove~~

~~add leave~~ to add on other level we have to add at least 1 node to the leave in the highest level.



5/

```
public static void reverse(int[] a, int i, int j) {

            if(i>j) return;
            int temp=a[i];
                a[i]=a[j];
                a[j]=temp;
                  reverse(a,i+1,j-1);
      }
```

This algorithm traverses the half of the array, in each step it switches the value with the value of the opposite index of this array. So this algorithm call it self n/2, which mean the running time is O(n).