Lab 4

① i) Insertion Sort is stable because insert elements from the beginning depending of the comparison result.

example: 6 5 3_{II} 1 3_{I}

5 6

3 5 6

1 3_{II} 5 6

1 3_{I} 3_{II} 5 6

From the previous example you can see each element from left is compared with each element of the right side and insert the element when find a smaller one. there is no equal comparison.

ii) Bubblesort is not stable because this algorithm compare two elements and swap them.

example: 3_{I} 2 1 3_{II}

2 3_{I}

2 1 3_{I}

1 2 3_{I} 3_{II}

iii) Selection sort is not stable because this algorithm swap elements depending on who is the minimum, if the minimum is already there it won't swap it.

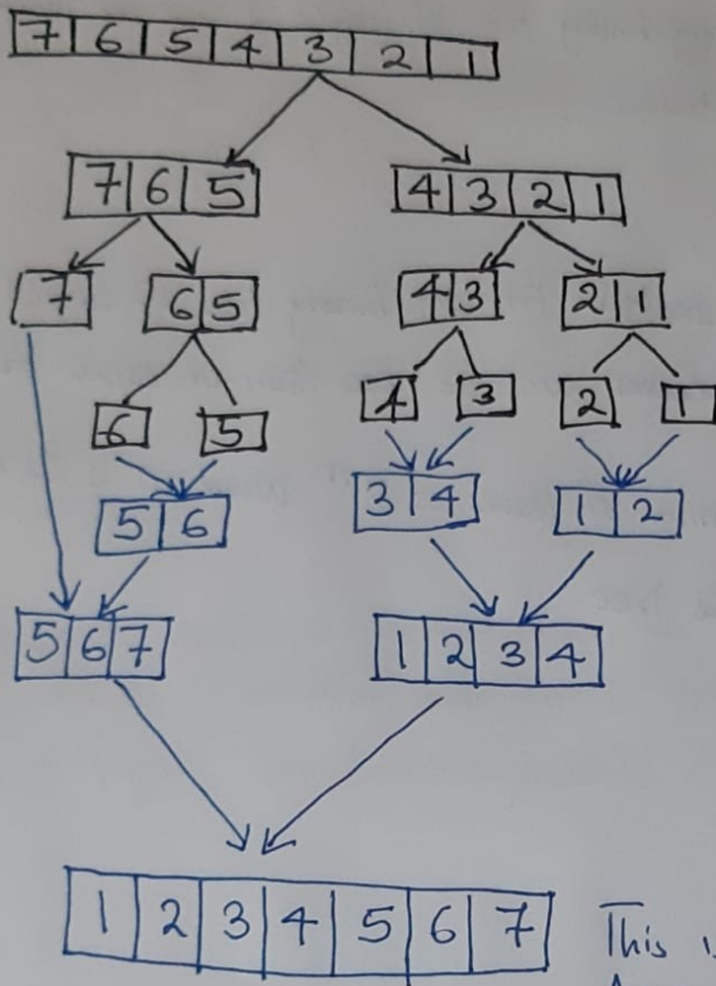
example: 3_{II} 2 1 3_{I}

1 2 3_{II} 3_{I}

→ it won't swap 3_{II} to 3_{I}, thus it is not stable.

2 Performing merge sort algorithm on the array [7,6,5,4,3,2,1]

* Start by partition the array in halfs until it's of length 1 for each subset.



the next step is to start merging the subset starting with smallest numbers

This is the final sorted Array using the divide an conquer algorithm known as merge sort.

3. A. The Algorithm

A. Pseudo-code.

The quick sort & insertion sort Algorithm is the same as the one given in class. The only difference is the merge one.

```
Algorithm   doMerge (lowerIndex, higherIndex)

Input :  Array A & the lower & higherIndex of the array to
         be merged

Output : Merged array starting from lower to higher in
         sorted order

if lowerIndex < higherIndex   then
   middle ← (lowerIndex + higherIndex)/2 ;
   if middle > 20    then
      doMerge (lowerIndex, middle)
      doMerge ( middle+1 , higherIndex)
   insertion Sort ( lowerIndex, higherIndex)
```

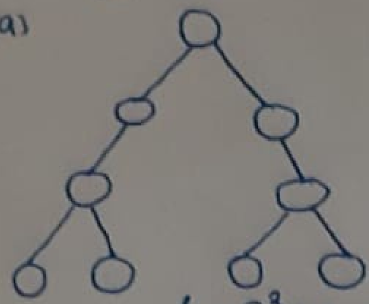
B. The Java Code

```java
package mergesort;
public class MergeSortPlus {
    private int[] array;
    private int length;
    public static void main(String a[]) {
        int[] inputArr=new int[100];
        for(int i=0;i<100;i++) {
            inputArr[i]=(int)(Math.random()*100);
        }
        MergeSortPlus m = new MergeSortPlus();
        m.sort(inputArr);
        for (int i : inputArr) {
            System.out.print(i);
            System.out.print(" ");
        }
    }
    public void sort(int inputArr[]) {
        this.array = inputArr;
        this.length = inputArr.length;
        doMergeSort(0, length - 1);
    }
    private void doMergeSort(int lowerIndex, int higherIndex) {
        if (lowerIndex < higherIndex) {
            int middle = (lowerIndex + higherIndex) / 2;
            // Below step sorts the left side of the array
            if(middle>20) {
                doMergeSort(lowerIndex, middle);
                // Below step sorts the right side of the array
                doMergeSort(middle + 1, higherIndex);
            }
            // Now merge both sides
            insertionSort(lowerIndex, higherIndex);
        }
    }
    private void insertionSort(int lowerIndex, int higherIndex) {
        int temp = 0;
        int j = 0;
        for (int i = lowerIndex; i <= higherIndex; i++) {
            temp = array[i];
            j = i;
            while (j > 0 && temp < array[j - 1]) {
                array[j] = array[j - 1];
                j--;
            }
            array[j] = temp;
        }
    }
}
```

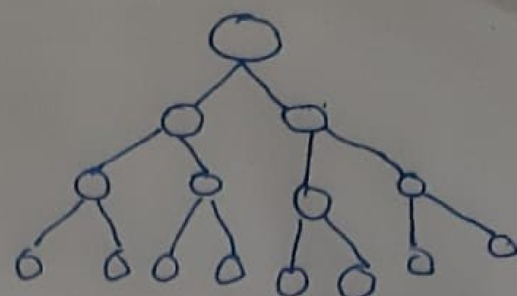
C. For us we tested and compared the MergeSort with the MergeSortPlus algorithm by using the below java code and every time the MergeSort algorithm performs better .

```
MergeSortPlus m = new MergeSortPlus();
long t1=System.nanoTime();
m.sort(inputArr);
long t2=System.nanoTime();
System.out.println(t2-t1);
MergeSort m1 = new MergeSort();
long t3=System.nanoTime();
m1.sort(inputArr);
long t4=System.nanoTime();
System.out.println(t4-t3);
```

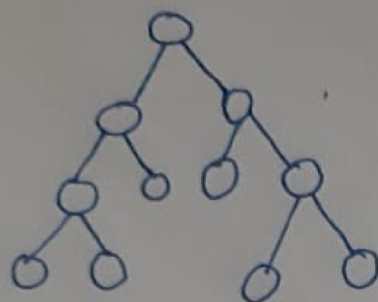
4(a) hen l = number of leaves $h = 3 =$ height



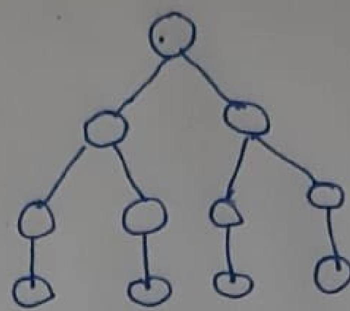
$l = 2^2 = 4 \leq 2^3 = 8$



$l = 2^3 = 8 \leq 2^3 = 8$



$l = 2^3 - 2 = 6 < 2^3 = 8$



$l = 2^2 < 2^3$

4(b) Based on the tree diagrams above, each tree has less than or e
to 2^3 leaves. This proves the statement that

Every binary tree of height 3 has at most $2^3 = 8$

is true.

4(c) Based on this analysis, For any binary tree of height n
the number of leaves will be less than or equal to 2^n
i.e

number of leaves $= 2^n$ where n is the height

of the tree.