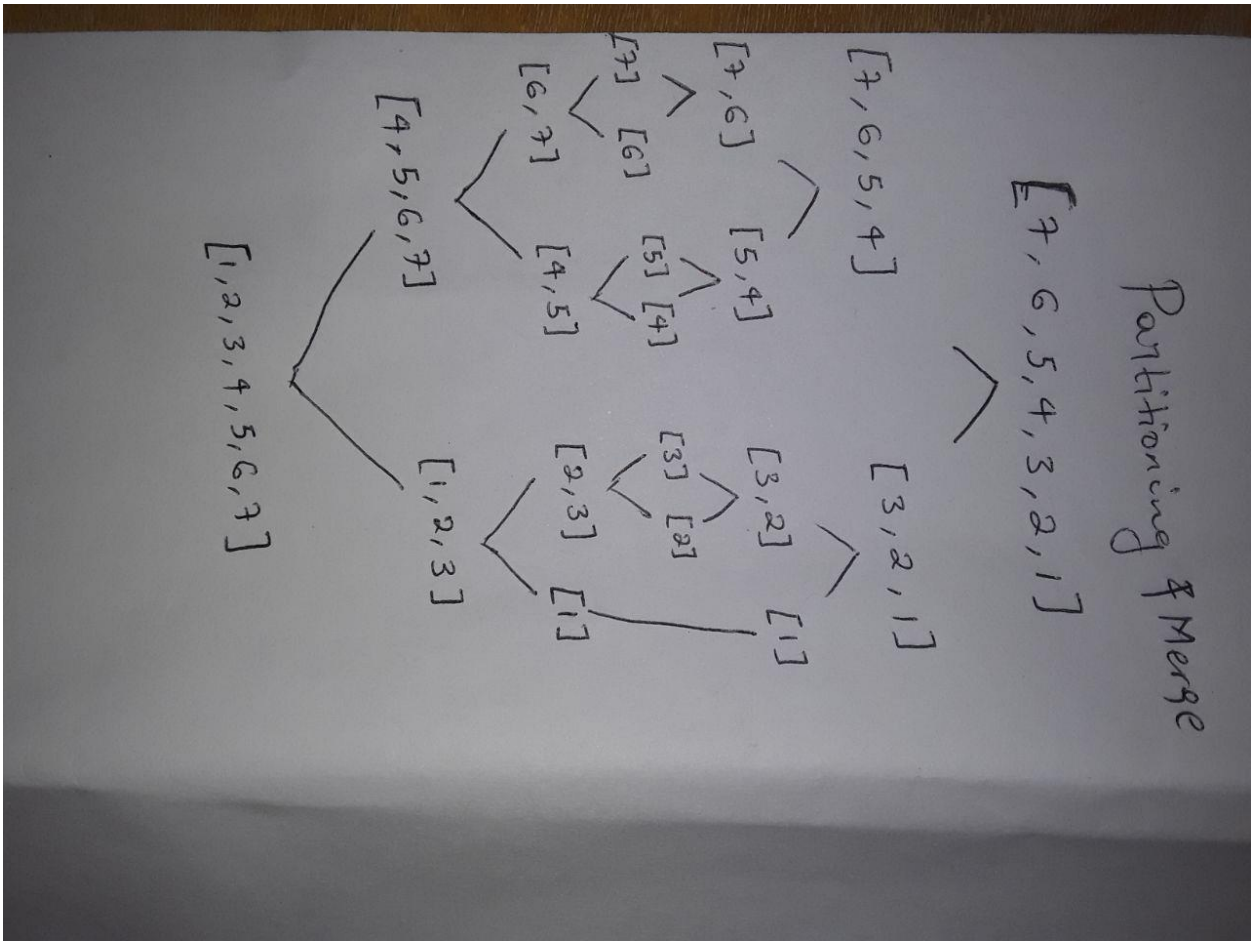
Lab assignment solutions

Problem 1:

Selection sort is not a stable algorithm, because given it swaps elements. For example, if the array to be sorted is {2,2,1}, after passing through Selection sorting algorithm it will become {1,2,2}.

The other two sorting algorithms, bubble sort and insertion sort, do not swap elements unless they are adjacent and unequal. Therefore, bubble sort and insertion sort are stable algorithms.

Problem 2:



Problem 3:

a.

```
Algorithm mergeSort(S)
    Input: sequence S with n integers
    Output: sequence S sorted
    if S.size() ≤ 1 then return S
    if S.size() ≤ 20 then
        insertionSort(S);
    else
        (S1, S2) ← partition(S, n/2);
        mergeSort(S1)
        mergeSort(S2)
    S ← merge(S1, S2)
    return S;
```

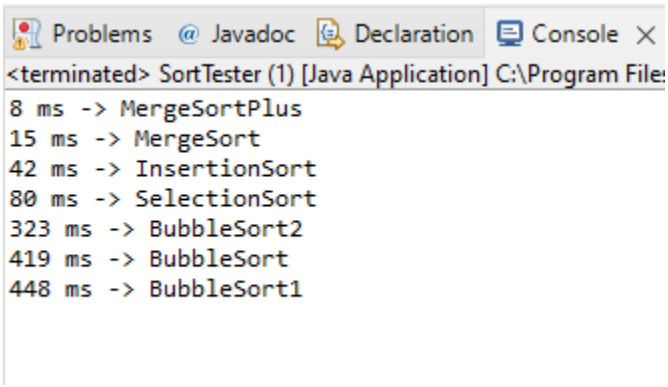
b. MergeSortPlus.java

```java
package sortroutines;

import runtime.*;

public class MergeSortPlus extends Sorter {
    final int ARRAY_SIZE = 33;
    final int MAX_VAL = 1000;
    int[] theArray;

    // public sorter
    public int[] sort(int[] input) {
        int n = input.length;
        int[] tempStorage = new int[n];
        theArray = input;
        mergeSort(tempStorage, 0, n - 1);
        return theArray;
    }

    void mergeSort(int[] tempStorage, int lower, int upper) {
        if (lower == upper) {
            return;
        }
        if (upper - lower < 20)
            insertionSort(theArray, lower, upper);

        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
        }
    }

    // insertion sort for a portion of an array , from anArray[from] to
anArray[to]
    public static void insertionSort(int[] anArray, int from, int to) {
        int len = to - from + 1;
        if (anArray == null || len <= 1) {
            return;
        }
        int temp = 0;
        int j = 0;
```

```
        for (int i = 1; i < len; ++i) {
            temp = anArray[i];
            j = i;
            while (j > 0 && temp < anArray[j - 1]) {
                anArray[j] = anArray[j - 1];
                j--;
            }
            anArray[j] = temp;
        }
    }

    /** Merges the ranges [lower, mid] and [midPlusOne,upper] in place */
    private void merge(int[] tempStorage, int lower, int midPlusOne, int upper) {
        int pos = 0; // tempStorage index
        int i = lower;
        int j = midPlusOne;
        int n = upper - lower + 1; // total number of elements to rearrange

        // view the range [lower,upper] as two arrays
        // [lower, mid], [midPlusOne,upper] to be merged

        while (i < midPlusOne && j <= upper) {
            if (theArray[i] <= theArray[j])
                tempStorage[pos++] = theArray[i++];
            else
                tempStorage[pos++] = theArray[j++];

        }
        while (i < midPlusOne) {
            tempStorage[pos++] = theArray[i++];
        }
        while (j <= upper) {
            tempStorage[pos++] = theArray[j++];
        }
        // replace the range [lower,upper] in theArray with
        // the range [0,n-1] just created in tempStorage
        for (j = 0; j < n; ++j) {
            theArray[lower + j] = tempStorage[j];
        }
    }

    // set up routines
    public static void main(String[] args) {
        MergeSortPlus ms = new MergeSortPlus();
        // ms.testMerge();
```

```
        int[] arr = { 1, 4, 2, 5, 6, 1, 7, 9, 0 };
        int[] returnArr = ms.sort(arr);
        for (int i : returnArr) {
            System.out.print(i + " ");
        }
    }
}
```

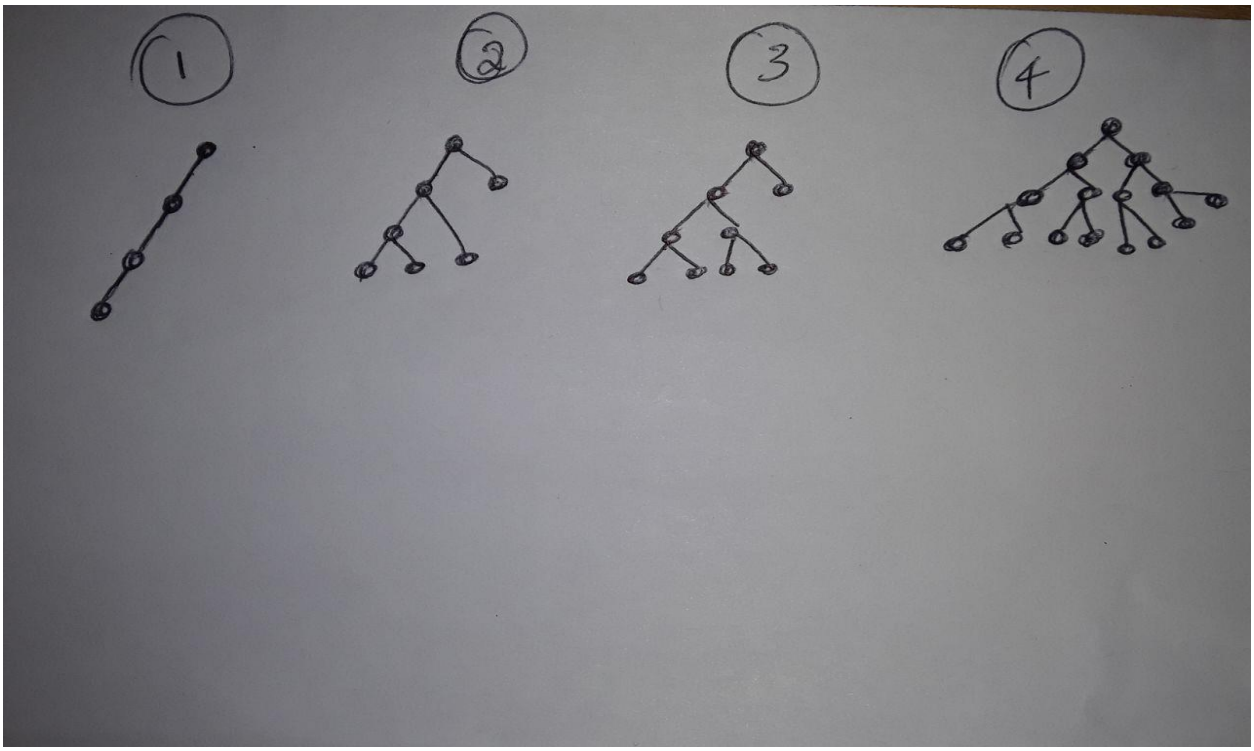
c. I tested the runtime using the sortTest file provided in the previous example.



The MergeSortPlus runs faster than the MergeSort when the the minimum limit for doing the insertion sort is 20, but for a limit of 100, MergeSort will be faster.

Problem 4:

a.



b.

Yes, its true.

c. A binary tree of height 1 has at most 2 nodes. A binary tree of height 2 has almost height 4 because the maximum number of leaves is obtained when all every node has two child nodes until the height is 2. Similarly, a binary tree of height n, will have at most twice of a tree of height (n-1). Since a tree of height 1 has 2^1 leaves, we can use Mathematical induction to show that a tree of height n has at most 2^n leaves.