Prof. Emdad Khan

September 2019
Lab#4

Group 1

Group members:
Asad Ali Kanwal
Aser Ahmad Ibrahim Ahmad
Jean Wilbert Volcy
Zayed Hassan

1. Problem 1

Insertion sort: it is stable since the loop which compares each two elements only swaps them if the right-side element is larger than the left-side element. This means that if two equal elements are found, they are not swapped.

Bubble sort: although the it compares all pairs of elements; it swaps only the pairs where the left element is larger than the right element. So, if a pair with two equal elements is found, it remains as it is and not swapped. So, it is a stable sort.

Selection sort: it depends on a subroutine to find the position of the minimum element in the array ($minpos$). If the ($minpos$) subroutine finds a minimum element, it records its position. If another equal element is found, it is not recorded because the comparison condition evaluates to true only if a smaller element than the current element is found. In other words: smallest elements are fetched by the $minpos$ subroutine in their same order in the original array. So, the selection sort is stable.

2. Problem 2

Solution is in the file (Problem 2.pdf).

3. Problem 3

A. Pseudo code of the MergeSortPlus:

```
Algorithm MergeSortPlus (A)
Input: unsorted array A
Output: sorted array A
if (A.length ≤ 20) then
    go to InsertionSort (A)
    return A
(A1, A2) = partision (A, A.length / 2)
MergeSortPlus (A1)
MergeSortPlus (A2)
A = merge (A1, A2)
return A

Algorithm merge (A1, A2)
Input: two arrays A1, A2
Output: M: merged array of A1 and A2
initialize empty array M of size !A1.length + !A2.length
while (!A1.empty & !A2.empty) do
    if (!A1.first ≤ !A2.first) then
        M.addLast(A1.removeFirst)
    else
        M.addLast(A2.removeFirst)
while (!A1.empty) do
    M.addLast(A2.removeFirst)
while (!A2.empty) do
    M.addLast(A1.removeFirst)
return M

Algorithm InsertionSort (A)
Input: unsorted array A
Output: sorted array A
temp = 0
j = 0
for (i = 1; i < A.length; i++) do
    temp = A[i]
    j = i
    while (j > 0 & temp < A[j – 1]) do
        A[j] = A[j – 1]
        j--
    A[j] = temp
```

B. The java code is in the path `Problem3\src\MergeSortPlus.java`.

C. The test is in the path `Problem3\src\testMergeSort.java`. The test procedure is as follows:

1. A number of arrays with sizes of 10, 100, 1000, 10000, 1000000, 10000000 is generated with random element values.
2. Each array is copied twice. One copy is sorted using the merge sort. The other is merged using merge sort plus. Time elapsed is recorded for both.

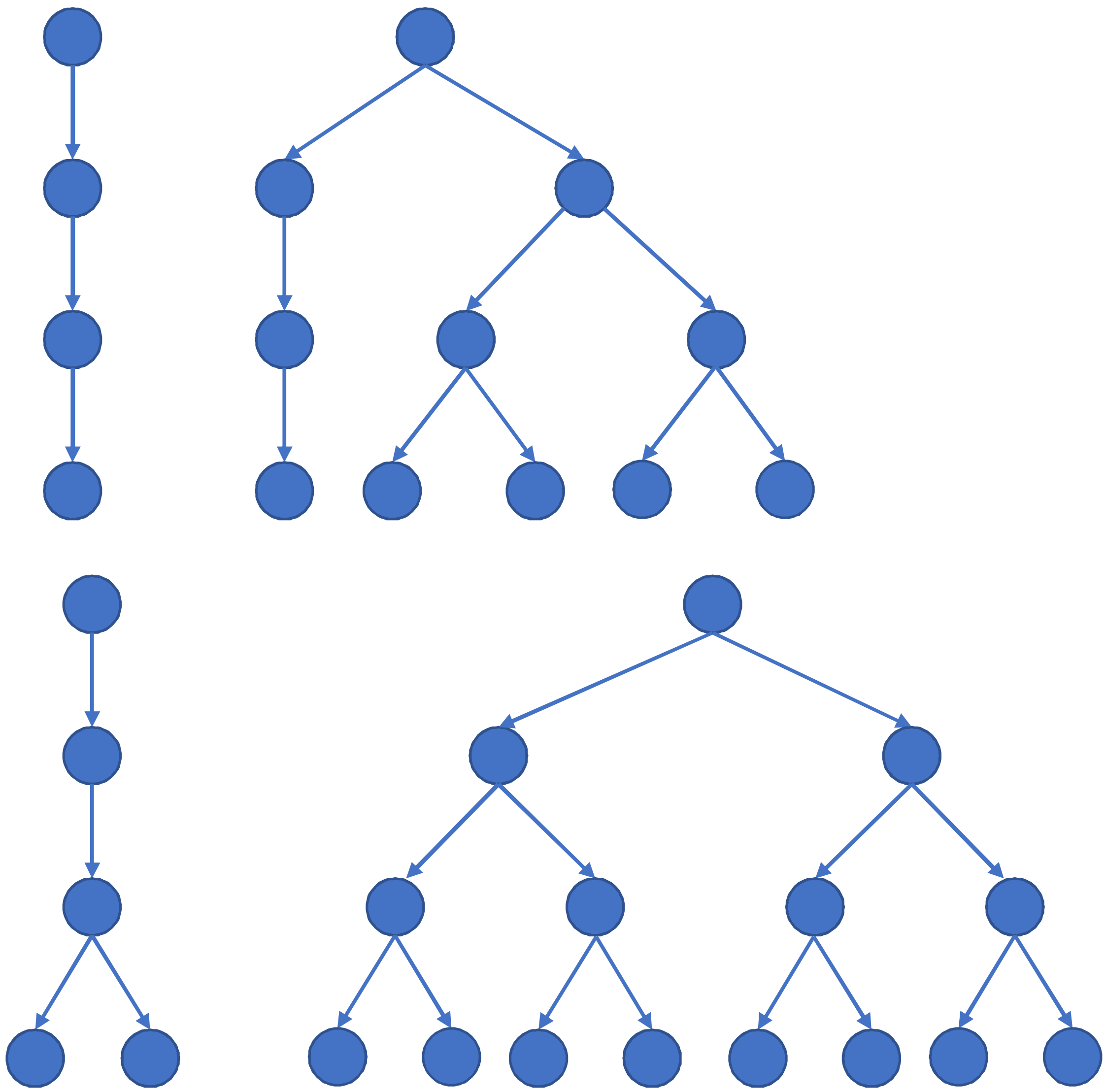
Test results: time (in mS) was typically the same for array sizes les than 1000 elements. For larger sizes, the merge sort plus was superior by an increasing rate, with a rough average 30% improvement in the running time.

The results are believed to be conclusive because:

1. They cover a large range of array sizes ($10^1 - 10^7\ elements$).
2. Identical arrays are tested each time for both algorithms. So, the algorithms have equal chances.
3. The test is run using the same IDE and performed using the same java class. Also, they were performed several times and they gave similar results.

4. Problem 4

a. Drawings of binary trees:



b. According to the 4 trees above, the number of leaves is:
The first: $1 < 8$, the second: $5 < 8$, the third: $2 < 8$, the fourth: $8 \leq 8$.
That is, all trees generated satisfy the mentioned condition.

c. For a binary tree, the maximum number of leaves, n, can be calculated according to the formula:

$$n = 2^n$$