

CS435DE - Lab 4

Saurab Ghimire

417555

Problem 1: Solution

For any sorting algorithm to be stable, it needs to make sure the elements with equal values have the same order and the order is maintained.

Insertions sort is stable because it picks elements one by one and inserts them into their correct position in a sorted place. So, if two elements are equal, the algorithm does not make any swap and ensures the original order.

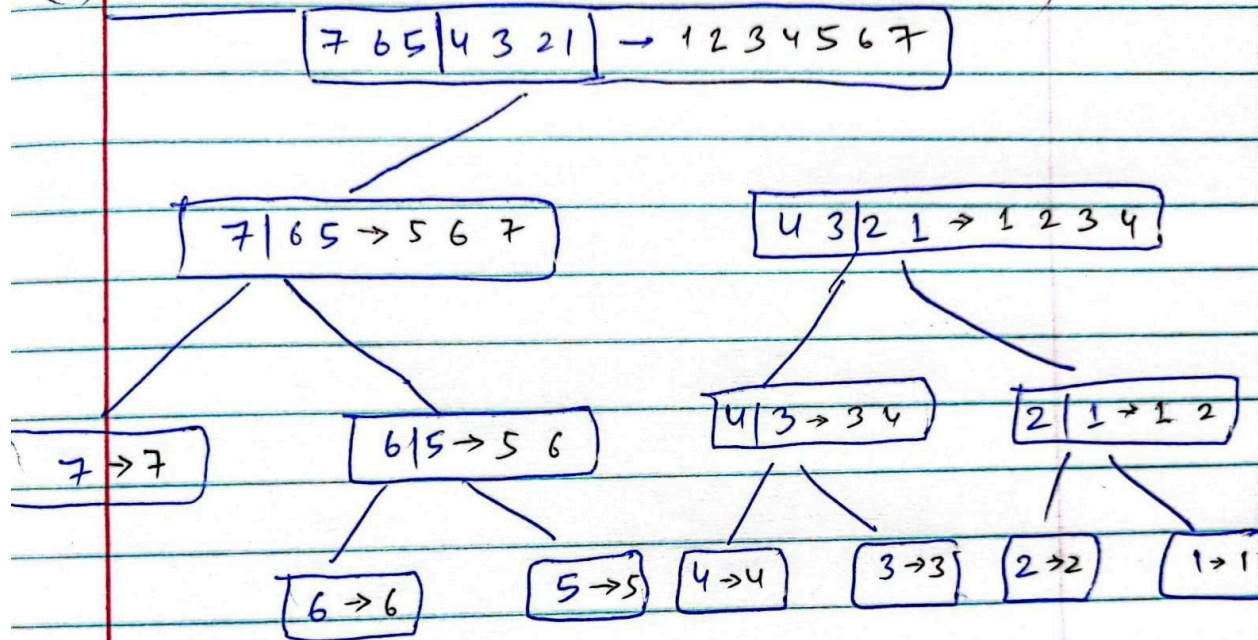
Bubble sort is stable because it swaps adjacent elements only if they are not in the appropriate order. So, if two elements are equal, the algorithm does not make any swap.

Selection sort selects the smallest element and swaps it with the first unsorted element. So if two elements are equal, an element in later stage of the array might be swapped before the identical element that appeared originally earlier. Hence the original order is not maintained. So, selection sort not a stable sorting algorithm.

Problem 2: Solution


Saurab Ghimire (617555)

(2) Solution:



Problem 3: Solution

Pseudo code for mergeSortPlus

```
1  Algorithm MergeSortPlus(A)
2  Input: Array A of n integers
3  Output: Sorted array A
4
5  if  $n \leq 20$  then
6      InsertionSort(A)
7      return
8
9  mid  $\leftarrow n / 2$ 
10 left  $\leftarrow A[0 : \text{mid}-1]$ 
11 right  $\leftarrow A[\text{mid} : n-1]$ 
12
13 MergeSortPlus(left)
14 MergeSortPlus(right)
15
16 A  $\leftarrow \text{merge}(\text{left}, \text{right})$ 
17
18 Algorithm InsertionSort(A)
19 Input: Array A of size n
20 Output: Sorted array A
21
22 for i  $\leftarrow 1$  to n-1 do
23     key  $\leftarrow A[i]$ 
24     j  $\leftarrow i - 1$ 
25     while j  $\geq 0$  and  $A[j] > \text{key}$  do
26         A[j + 1]  $\leftarrow A[j]$ 
27         j  $\leftarrow j - 1$ 
28      A[j + 1]  $\leftarrow \text{key}$ 
29
```

Using the subroutines in the given code, we can implement the code for mergesort plus where if $\text{right} - \text{left} + 1 \leq 20$, we use insertion sort.

```

1 package lab4;
2 import sortroutines.InsertionSort;
3
4 public class MergeSortPlus {
5     InsertionSort insertionSort = new InsertionSort();
6     public void mergeSortPlus(int[] arr, int left, int right) {
7         if (right - left + 1 <= 20) {
8             insertionSort.sort(arr);
9             return;
10        }
11
12        int mid = left + (right - left) / 2;
13
14        mergeSortPlus(arr, left, mid);
15        mergeSortPlus(arr, left: mid + 1, right);
16
17        mergeSortPlus(arr, left, right);
18    }
19 }

```

To verify the execution time between mergeSort and mergeSortPlus, I generated array of large elements and recorded execution times. With mergeSortPlus having insertionSort for more than 20, it was faster for smaller arrays, but for large inputs, performance was similar. The results are not conclusive.

Problem 4: Solution