Lab4

# Problem 1

1. BubbleSort is stable sorting algorithms because this block of code

if(arr[j]> arr[j+1]){  
 swap(j,j+1);  
}

show us that BubbleSort only swap current element with next element if current element is greater than next element.

1. SelectionSort is not stable because this block of code

int nextMinPos = minpos(i,len-1);  
swap(i,nextMinPos);

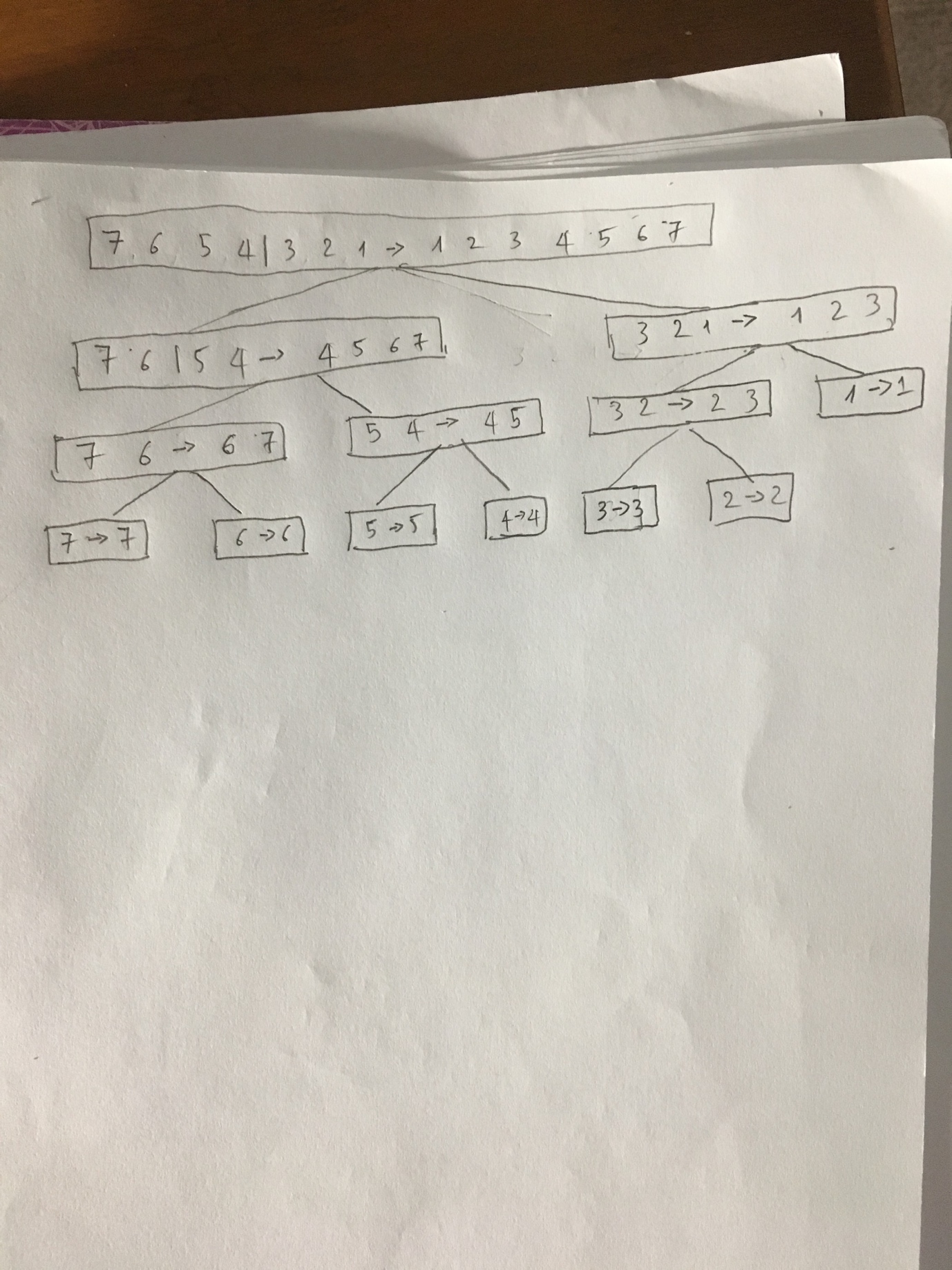
show us that SelectionSort will swap current element with the next minimum element even next minimum element is equal to current element.

1. InsertionSort is stable sorting algorithms because this block of code

while(j>0 && temp < anArray[j-1]){  
 anArray[j] = anArray[j-1];  
 j--;  
}

show us that only if the *temp* will be stuck if it meets an element less than it and won’t do anything.

# Problem 2



# Problem 3

Algorithm mergeSort(S)  
 Input sequence S with n integers  
 Output sequence S sorted  
 if S.size() > 1 then  
 if S.size() <= 20 then  
 insertionSort(S)  
 else  
 (S1, S2) <- partition(S, n/2)  
 mergeSort(S1)  
 mergeSort(S2)  
 S <- merge(S1, S2)  
 return S

# Problem 4

A

public void bubbleShort(int[] arr) {  
 for (int i = 0; i < arr.length; i++) {  
 boolean isSwap = false;  
 for (int j = 0; j < arr.length - 1; j++) {  
 if (arr[j] > arr[j + 1]) {  
 swap(j, j + 1);  
 isSwap = true;  
 }  
 }  
 if (!isSwap) {  
 break;  
 }  
 }  
}

I use a flag variable isSwap to check the input Array is sorted or not. With the sorted array isSwap never turn to be true then break the outer for loop.

The run time now is O(n) because: The outer for loop will run n time, the inner for loop run n – 1 time and the because the array is already sorted so isSwap never turn to true then the outer loop is broken. So the run time = n + n – 1 = 2n – 1 (O(n))

B

private void bubbleSort(){  
 int len = arr.length;  
 for(int i = 0; i < len; ++i) {  
 for(int j = 0; j < len - i - 1; ++j) {  
 if(arr[j]> arr[j+1]){  
 swap(j,j+1);  
 }  
 }  
 }  
}

After i time the outer loop run, the element from n – i – 1 to n – 1 will be in final sorted order so the inner loop only need run from 0 to n – i – 1.

The reduce time = 1 + 2 + 3 + … + n – 1 = n2 / 2 => the running time is reduced a haft but still O(n2)

C.

The result is

477 ms -> BubbleSort2

498 ms -> BubbleSort1

532 ms -> BubbleSort

This is not exactly what I expected.

I expect the running time of BubbleSort2 is a haft of BubbleSort.

The running time is show like above (BubbleSort1 & BubbleSort2 < BubbleSort) because the BubbleSort1 & BubbleSort2 are improving version of BubbleSort, we already reduce some redundancy calculating.