**Lab3**

**Jimmy Palma**

**610756**

1. **Goofy has thought of a new way to sort an array arr of n distinct integers:**

**Step 1: Check if arr is sorted. If so, return.**

**Step 2: Randomly arrange the elements of arr (Hint: this can be done in** **O(n))**

**Step 3: Repeat Steps 1 and 2 until there is a return.**

**Answer the following:**

**Will Goofy’s sorting procedure work at all? Explain**

No always, like the work is randomly, in the worst case never get the result

**What is a best case for GoofySort?**

When the array is already sorted

**What is the running time in the best case?**

To verify if the array is sorted, we have to loop the array at least once, therefore the running time will be θ(n)

**What is the worst-case running time?**

When never get the sorted array because the algorithm works randomly

**Is the algorithm inversion-bound?**

It is not inversion-bound in all the cases, for example in the case the array is not sorted would do one comparison, but in the next step the array get sorted so the algorithm will finish. In this scenario the algorithm is 2n, and the inversion is n(n-1)/4. # comparison < # inversion

1. **Interview Question. An array A holds n integers, and all integers in A belong to the set {0, 1, 2}. Describe an O(n) sorting algorithm for putting A in sorted order. Your algorithm may not make use of auxiliary storage such as arrays or hashtables (more precisely, the only additional space used, beyond the given array, is O(1)). Give an argument to explain why your algorithm runs in O(n) time.**

Algorithm sort

Input <- array

Output <- sorted array

numZero, numOnes, numTwos <- new Varianles

for (int i = 0; i < array.size; i++)

If array.get(i)== 0 then

Increment NumZero

Else if array.get(i) == 1 then

Increment numOnes

Else if array.get(i) == 2 then

Increment numTwos

for (int i = 0; i < numZeros; i++)

Array.add(0)

for (int i = 0; i < numOnes; i++)

Array.add(1)

for (int i = 0; i < numTwos; i++)

Array.add(2)

/////////// code /////////

public static Integer[] sortZeroOneTwo(int[] arr){

if (arr == null || arr.length == 0)

return null;

List<Integer> finalList = new ArrayList<Integer>();

int zeros = 0, ones = 0, twos = 0;

for (int i = 0 ; i < arr.length; i ++){

if (arr[i] == 0)

zeros++;

else if (arr[i] == 1)

ones ++;

else

twos ++;

}

for (int i = 0; i < zeros; i++){

finalList.add(0);

}

for (int i = 0; i < ones; i++){

finalList.add(1);

}

for (int i = 0; i < twos; i++){

finalList.add(2);

}

return finalList.stream().toArray(Integer[]::new);

}

1. **BubbleSort**
2. **Improve the BubbleSort implementation so that when the input array becomes sorted after some runs of outer for loop, the algorithm will stop. Call your new Java file BubbleSort1.java.**
3. **Recall that in BubbleSort, at the end of the first pass through the outer loop, the largest element of the array is in its final sorted position. After the next pass, the next largest element is in its final sorted position. After the ith pass (i=0,1,2,...), the largest, second largest,..., i+1st largest elements are in their final sorted position. Use this observation to cut the running time of BubbleSort in half. Implement your solution in code, and prove that you have improved the running time in this way. Call your new Java file, which contains the improvements from this problem and the previous problem, BubbleSort2.java.**
4. **In this lab folder, I have given you an environment for testing sorting routines. Insert into this environment the original BubbleSort file along with your new BubbleSort1 and BubbleSort2 classes, and run the SortTester class. What are the results? Are the results what you expected? Explain why the running times turned out the way they did.**

362 ms -> BubbleSort1

411 ms -> BubbleSort2

487 ms -> BubbleSort

The results means BubbleSort1 is more efficient, this is because the algorithm sorted the array and after one loop more of comprobation it finished.

The BubbleSort2 does not break the algorithm anytime, only reduce the last elemtent, therefore is not efficient as BubbleSort1

1. **Interview Question. You are given a length-n array A consisting of 0s and 1s, arranged in sorted order. Give an o(n) algorithm that counts the total number of 0s and 1s in the array. Your algorithm may not make use of auxiliary storage such as arrays or hashtables (more precisely, the only additional space used, beyond the given array, is O(1)). You must give an argument to show that your algorithm runs in o(n) time.**

public static Integer[] countZeroOne(int[] A) {

List<Integer> values = new ArrayList<Integer>();

if(A.length == 0) {

values.add(0);

values.add(0);

return values.stream().toArray(Integer[]::new);

}

else if(A[0] == 1) { //must be all 1s since sorted

values.add(0);

values.add(A.length);

return values.stream().toArray(Integer[]::new);

}

else if(A[A.length-1] == 0) { //must all be 0s

values.add(A.length);

values.add(0);

return values.stream().toArray(Integer[]::new);

}

return *countZeroOneRecursive*(A, 0, A.length-1);

}

public static Integer[] countZeroOneRecursive(int arr[], int first, int last){

List<Integer> values = new ArrayList<Integer>();

int mid = (first + last)/2;

if(arr[mid] == 0 && arr[mid+1] == 1) {

values.add(mid +1);

values.add(arr.length - (mid+1));

return values.stream().toArray(Integer[]::new);

}

if(arr[mid] == 0 && arr[mid+1] == 0) {

return *countZeroOneRecursive*(arr, mid+1, last);

}

if (arr[mid] == 1) {

return *countZeroOneRecursive*(arr, first, mid -1);

}

return null;

}