Dynamic Programming, Divide and Conquer, Data Structures

Please complete this problem set by October 7, 2021 at 11:59PM. You **can only use 1 make up day for this assignment** because I will be releasing the solutions on Saturday.

Problem 0 – Dynamic Programming (25%)

Zoe starts at the top left corner of an $n \times n$ square grid of numbers and must end at the bottom right corner. The numbers can be positive or negative. At each step, Zoe can go left, right, or down (but not up), and she can never revisit a square that she has already visited. Find an $O(n^2)$ algorithm to find the maximum sum of numbers Zoe can visit.

Problem 1 – Divide and Conquer (25%)

A mad scientist accidentally left her cloning machine on and open while vacationing in Storrs, Connecticut. While she was away, a murder of crows managed to get inside her office and use the cloning machine. The mad scientist returned from her vacation surprised to have her office full of crows, but excited to test a new hypothesis. As is well known, crows are very intelligent birds, and the scientist suspects that one or more crows may have taken advantage of the cloning machine to gain a selective genetic advantage.

We say that two crows are *identical* if we genetically sequence the crows and their DNA is 99.99% identical. The mad scientist has a machine that will determine whether or not two crows are genetically identical, but it is costly to operate so she wants to minimize its use.

The only operation you are allowed to perform is to take two crows and determine if they are genetically identical. Develop an $O(n \log n)$ algorithm that, given n crows, determines if there is a set of more than $n/2$ crows that are genetically identical.

Problem 2 – SillySort (25%)

Prove that SillySort(L) terminates and correctly sorts its input list of numbers L.

Algorithm 1: A sorting algorithm only used by silly people.

```
def sillysort(L):
    while true:
        if L is sorted:
            return L
        else:
            compute an arbitrary index i such that L[i] > L[i+1]
            swap L[i] and L[i+1]
```

Problem 3 – Data structures (25%)

1. We saw in class the algorithm for building a max-heap

   Algorithm 2: Building a max heap.

   ```
   import math
   def buildmaxheap(A):
       A.heapsize = A.length
       for i in range(int(math.floor(A.length/2)),0,-1):
           maxheapify(A,i)
   ```

   Consider the following alternative way to build a max-heap

   Algorithm 3: Building a max heap.

   ```
   def buildmaxheap2(A):
       A.heapsize = 1
       for i in range(2,A.length+1):
           maxheapinsert(A,A[i])
   ```

   Prove that buildmaxheap and buildmaxheap2 do not always create the same heap by providing a counterexample.

2. What is the worst case time complexity of buildmaxheap2? Explain your reasoning.