Problem Set 2, CS 5800 Spring 2017

Due: Monday, 1/23, 11AM

Problem 1 (30 pts)**.** Give an algorithm running in time $O(n)$ and space $O(1)$ which sorts an array of n elements in the range $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

Problem 2 (40 pts)**.** Let $A[1..n]$ be an array of n integers between 1 and n^2. Give an $O(n)$-time $O(n)$-space algorithm to determine if the elements of A are distinct.

Let $A[1..n]$ and $B[1..n]$ be two arrays of n integers between 1 and n^2. Give an $O(n)$-time $O(n)$-space algorithm to determine if A and B have a common element.

Problem 3 (30 pts)**.** Prove that radixsort sorts correctly an array of $n = 2$ elements with $d = 2$ digits.

```
Problem 2 & 1 See next page

Problem 3:

      Proof:

            We represent the given array [a, b] as:
                  [ ( 10 * a1 + a0 ) , ( 10 * b1 + b0) ]
                  using the given assumption that d = 2.
                  We then have:
                        abs(a0-b0) < 10 (Fact1)

            If a1 > b1:
                  We then have: a - b >= 10 (Assumption1)
                  Assumption1 & Fact1 -> a < b (Assumption2)
                  The second Counting Sort inside the Radix Sort will place
                  a & b based on a1 and b1 alone, end result being [a,b] which
                  matches Assumption2. -> Case consistency achieved.

            else if a1 < b1:
                  We then have: b - a >= 10 (Assumption3)
                  Assumption3 & Fact1 -> a > b (Assumption4)
                  The second Counting Sort inside the Radix Sort will place
                  a & b based on a1 and b1 alone, end result being [b,a] which
                  matches Assumption4. -> Case consistency achieved.

            else: # being a1 == b1:
                  We then have b - a == 0 (Assumption5)
                  Then the second Counting Sort will not change the position of
                  a & b, therefore the final result will only base on the result
                  of first Counting Sort. (Assumption6)
                  if a0 > b0:
                        We have end result [a,b] based on Assumption6
                        -> Case consistency achieved.
                  if a0 < b0:
                        We have end result [b,a] based on Assumption6
                        -> Case consistency achieved.
                  if a0 == b0:
                        We have end result [a,b] based on Assumption6
                        -> Case consistency achieved.

            Proof: all cases exhausted & all cases consistency achieved
```

```
def myCountingSort(lst,n,key=lambda x:x):
     # nothing fancy three-loop Counting Sort
     count  = [0    for i in xrange(n)]
     result = [None for i in xrange(len(lst))]
     for i in lst:
          count[key(i)-1]+=1
     for i in xrange(n):
          if i==0: continue
          count[i]+=count[i-1]
     for i in reversed(lst):
          result[count[key(i)-1]-1] = i
          count[key(i)-1] -= 1
     return result
######################################################################

def myRadixSort(lst):
     # invariant: all element in lst < len(lst)^2
     n = len(lst)
     def trans(num):
          # return given number converted into N-based number
          # invariant: given number is no bigger than N^2
          return (num/n,num%n)
     def revert((digit1,digit0)):
          # revert N-based 2-digit number back to decimal number
          return digit1*n+digit0

     lst = [trans(i) for i in lst]

     for i in range(2):
          lst = myCountingSort(lst,n,key=lambda x: x[i])

     return [revert(i) for i in lst]

######################################################################

# PS2.1
def PS2_1(lst):
     return myCountingSort(lst,10)

# PS2.1
def PS2_2_1(lst):
     prev = None
     for i in myRadixSort(lst):
          if prev==i:
               return True
          prev = i
     return False

# PS2.2
def PS2_2_2(lst1,lst2):
     # no need to use two pointer hahaha
     return PS2_2_1(lst1+lst2)
```