CSC-421 Applied Algorithms and Structures Fall 2018-19

Student: **Mark Davis**
Instructor: Iyad Kanj
Office: CDM 832
Phone: (312) 362-5558
Email: ikanj@cs.depaul.edu
Office Hours: Monday & Wednesday 4:00-5:30
Course Website: https://d2l.depaul.edu/

Assignment #1

(Due September 26)

Remarks

- For the questions on this assignment, if needed, you may assume that sorting n numbers can be done in time $O(n \lg n)$ (e.g., using Heap Sort). If you need to sort, you can directly apply such a sorting algorithm (without writing the pseudocode), and claim that it runs in $O(n \lg n)$ time, where n is the number of elments/numbers being sorted.
- When asked to give an algorithm that meets a certain time bound, you need to give the algorithm (pseudocode/description) and analyze its running time to show that it meets the required bound; giving only the algorithm is not enough to receive full credit.
- Please upload your submission as a single PDF file on D2L. If your submission consists of more than one file, convert all your files into a single PDF file and upload it.

1. Given a collection of n nuts and a collection of n bolts, arranged in an increasing order of size, give an $O(n)$ time algorithm to check if there is a nut and a bolt that have the same size. The sizes of the nuts and bolts are stored in the sorted arrays $NUTS[1..n]$ and $BOLTS[1..n]$, respectively. Your algorithm can stop as soon as it finds a single match (i.e, you do not need to report all matches).

Actual working code:

```
def elementMatch(x,y):
  i=0
  j=0
  n=len(x)

 while(i<n and j<n): # O(n)
    if x[i]==y[j]:
      return True
    if x[j]<y[i]:
      j+=1
    else:
      i+=1
  return False
```

Total time = O(n)

2. Let $A[1..n]$ be an array of distinct positive integers, and let t be a positive integer.

(a) Assuming that A is sorted, show that in $O(n)$ time it can be decided if A contains two distinct elements x and y such that $x + y = t$.

Actual working code:

```
def equalssum(x, total):
  count = 0
  i=0
  n=len(x)
  j=n-1
  count=0


  while(i<j): # O(N)
    diff = total-x[i]
    if x[j]==diff and x[i]!=x[j]:
      return True
    elif x[j]>diff:
      j-=1
    else:
      i+=1
  return False

nums = [5,10,20,30,35,60,110,200]

for test in nums:
  print(equalssum(nums,test))
```

Total time = O(n)

(b) Use part (a) to show that the following problem, referred to as the 3-Sum problem, can be solved in $O(n^2)$ time:

3-Sum
Given an array $A[1..n]$ of distinct positive integers that is not (necessarily) sorted, and a positive integer t, de- termine whether or not there are three distinct elements x, y, z in A such that $x + y + z = t$.

Actual working code:

```
def equalssum3(x, total):
  count = 0
  i=0
  n=len(x)

# FOR LOOP = O(N)
  for num in x:
    diff = total - num
    i=0
    j=n-1

    # WHILE LOOP = O(N)
    while(i<j):
      sum=x[i]+x[j]
      if sum==diff and x[i]!=x[j] and x[j]!=num and x[i]!=num:
        return True
      elif sum>diff:
        j-=1
      else:
        i+=1

  return False
```

Total time = O(n) * O(n) = O(n²)

3. Let $A[1..n]$ be an array of positive integers (A is not sorted). Pinocchio claims that there exists an $O(n)$-time algorithm that decides if there are two integers in A whose sum is 1000. Is Pinocchio right, or will his nose grow? If you say Pinocchio is right, explain how it can be done in $O(n)$ time; otherwise, argue why it is impossible.

Impossible to do in O(n) time without sorting (sorting would be greater than O(n)). Without sorting, no additional information is gained after each iteration regarding the remaining positive integers, and no significant algorithmic optimization can be done. Worst case, no pair of elements will add up to 1000. This implies either computing the sum of every combination of two numbers **(O(n^2))**, or pre-sorting then searching the array O(nlg(n))+ O(n) = **O(nlgn).**

4. Let $A[1..n]$ be an array of points in the plane, where $A[i]$ contains the coordinates (x_i, y_i) of a point p_i, for $i = 1, \ldots, n$. Give an $O(n \lg n)$ time algorithm that determines whether any two points in A are identical (that is, have the same x and y coordinates).

Step 1: ***O(nlgn)***
Sort N points by y-coordinate

Step 2: ***O(n)***
Iterate all (n-1) pairs of points to compare if $x_i == x_{i+1}$ *and* $y_i == y_{i+1}$
return (ifTrue)

Total time = O(nlgn) + (n) = O(nlgn)

5. Textbook, page 1020, exercise 33.1-4.

Step 1:
*for p in points****: # O(n)***
*for q in points: **# O(n)***
if (not (q == p)) then compute the angle between q and p
*sort all angles ascending****: #O(nlgn)***

*Step 3****:***
sequential search for two identical angles having identical x and y values, ***O(n)***
if angle1 == angle2 and p1==q1
then return True

Total time = O(n² lgn) + (n) = O(n² lgn)