CSCI 3104: Algorithms **Oscar Delgado**
Problem Set 6 (70 points) **Summer 2019, CU-Boulder**

1. *(15 points) Give an $O(VE)$-time algorithm for computing the transitive closure of a directed graph $G = (V, E)$. Compute its asymptotic running time.*

   You can use the floyd-warshal alg algorithms to to compute the transitive closure. you can start by creating an nxn matrix A filled with 0's. WE can determine the vertices reachable from a particular vertex in O(E), since we can copute it for eeach v in V. We have to assign each edge weight to 1. then we have $\delta(v, u) \leq |E|$ for all u in V. if j is reachable from i then it would 1 otherwise it would be 0. It's asymtotic running time would be $O(V^3)$

2. *(15 points) Grog –master of pictures– needs your help to compute the in- and out-degrees of all vertices in a directed multigraph G. However, he is not sure how to represent the graph so that the calculation is most efficient. For each of the three possible representations, express your answers in asymptotic notation (the only notation Grog understands), in terms of V and E, and justify your claim.*

   (a) *An* adjacency matrix *representation. Assume the size of the matrix is known.*

   For edge list, we have a list of all the edges so E. Everytime we add a degree we at it to the Vertic and it's vertices and we need to make sure it isn't a duplicate so we need to check all the edges before so it would be $O(V * E)$

   (b) *An* edge list *representation. Assume vertices have arbitrary labels.*

   For adjacency list, its a array of sort with a list of adjecent vertices. So when going through array using i, you add the degrees on the i vertex and also to the vertices in the list and when the vertices is less than i, you check A[vertices] list if i exists in the list and if does, don't add it because its a duplicate. So, because you don't need to check the whole array again its $O(V * E)$

   (c) *An* adjacency list *representation. Assume the vector's length is known.*

   For adjacency matric, its much simplier to check for duplicates. So you go through each list of vertex, and lets say i is for vertix and j for the vertices, if 1 is present it means there is an edge so you add degree to the vertix and vertices, you go through it VxV and if j ¡ i check if Matrix[j][i] is 1 and if it is don't add the degree because its a duplicate. For this it would be $O(V^2)$

3. *(40 points) Consider a valleyed array $A[1, 2, \ldots, n]$ with the property that the subarray $A[1 \ldots i]$ has the property that $A[j] > A[j+1]$ for $1 \leq j < i$, and the subarray $A[i \ldots n]$ has the property that $A[j] < A[j+1]$ for $i \leq j < n$. For example, $A = [16, 15, 10, 9, 7, 3, 6, 8, 17, 23]$ is a valleyed array.*

   (a) *Write a recursive algorithm that takes asymptotically sub-linear time to find the minimum element of A.*

```
1. def minA(A,s,l):
2.    x=s , y=l , found = false , min=s
3.    while found != true and x<(x+y)/2
4.        if A[x+1] > A[x]
5.            found=true
6.            min=x
7.            break
8.        if A[y-1]>A[y]
9,            found=true
10.          min=y
11.          break
12.       x++ , y--
13.  return min
```

   (b) *Prove that your algorithm is correct. (Hint: prove that your algorithm's correctness follows from the correctness of another correct algorithm we already know.)*

   Loop invariant: Before every iteration the subarray A[s....i-1] and A[j+1...l] does not contain the smallest element because it is not found if array A has the property that $A[j] > A[j+1]$ for $1 \leq j < i$, and the subarray $A[i \ldots n]$ has the property that $A[j] < A[j+1]$ for $i \leq j < n$.

   Init: (found is not true, i=s , j=l): Both the subarrays are empty so they can't contain smallest element so trivially holds.

   maint (i=x, j=y): Assume our LI hold for i-1 and j+1 meaning A[s...i-1] and A[j+1....l] doesn't have smallest element and are decreasing. Then checking i and j we have 3 cases. 1. $A[i+1] > A[i]$ (line 4), meaning we found the valley on the first half of array then would assign min to i and exit loop so A[s...i] and A[j...l] don't contain smallest element withholding the LI. otherwise 2.$A[j-i] > A[j]$ (line 7), meaning we found the valley on the second half of array and assigns min to j and exits loop so A[s...i] and A[j...l] don't contain smallest element withholding the LI otherwise 3. the valley was not found and it continues to new iteration meaning A[s...i] and A[j...l] don't contain smallest element withholding the LI.

Term (found is true,i=j): following the property of A, a valley exist in A so the loop will go until found checking all the way until reaching the middle which would be the last place the valley could be so i=j so A[l...i-1] and A[j+1...l] doesn't contain the smallest element so LI holds.

(c) *Now consider the multi-valleyed generalization, in which the array contains k valleys, i.e., it contains k subarrays, each of which is itself a valleyed array. Let $k = 2$ and prove that your algorithm can fail on such an input.* Well because there are 2 valleys there are 2 possible min. My algorithm right now would find the first valley's min and return that but that other valley min could possibly be smaller and thus not returning the smallest value. ex A = [5, 4 , 6 , 2 , 6 , 7], here the alg. would hit the first valley A[1]=4 and return that but the other valley A[3]=2 has a smaller value thus failing.

(d) *Suppose that $k = 2$ and we can guarantee that neither valley is closer than $n = 4$ positions to the middle of the array, and that the "joining point" of the two singly valleyed subarrays lays in the middle half of the array. Now write an algorithm that returns the minimum element of A in sublinear time. Prove that your algorithm is correct, give a recurrence relation for its running time, and solve for its asymptotic behavior.*

```
1. def minA_updated(A,n):
2.   mid = n/2
3.   m1 = minA(A , 0 , mid-1)
4.   m2 = minA(A, mid , n-1)
5.   return min(m1,m2)
```

Direct proof. We are given that the valley are n=4 position from the middle of the array so the mid is n/2. so line 3 does minA(A,0,mid-1) which is a subarray of A and is A'=A(0....mid-1) that containes 1 of the valleys and line 4 does the other half min(A,mid,n-1) which a subarray of A and is A"=A(mid...n-1) conatins the other valley. Both min of both valleys are found with alg minA which is proven in B. last thing would be checking which of the 2 mins are smaller and this is done in line 5 whiling returning the smallest of the 2 and we also know that $A' \cup A'' = A$

making the min of those 2 values the min of A.

The recurrence relation is T(n)=T(n/2)+T(n/2)+c which when solving is O(log(n))