Final Exam

Name: Guangfeng Lin

Student #: 30131213

1. A dual is a powerful mathematical concept, in optimization problems, the identification of a dual problem is almost always coupled with the discory of a polynomial-time algorithm. Duality often helps provide a proof of optimality. The original problem is a primal problem when referring to a dual problem.

2. The adjacency list is a set of edge sets adjacent to each vertex in the graph, where the set refers to the unorderset. When we attach an attribute to an adjacency list. This representation can used to represent a weighted graph. The weights of edges can be describe as lists of pairs.

3. Get result from Johnson's algorithm
return D(u,v)

4. There are two properties
① optimal substructure
② overlapping sub-problems

- Optimal substructure giving an optimization problem that can be constructed from optimal solution of its subproblem. Optimal substructure is usually use to determine the usefulness of dynamic programming of and greedy algorithm.
- overlapping sub-problems is occuring when the problem can be broken down into subproblems that are reused several times.

5. R is in P, yes R in P, and R in NP as well.

6. a. Algorithm: using modify merger sort and merge.

```
Merge_sort (int arr[], int left, int right)
    if (left < right)
        int m = left + (right - 1)/2 ;
        Merge-sort (arr, left, m);
        Merge-sort (arr, m+1, right);
        Merge (arr, left, m, right);

Merge (int arr[], int left, int m, int right)
    int i;
    int j;
    int count, int k = left.
    for i=0 to m-left+1
        left[i] = arr[left+i];
    for j=0 to right-m
        right[j] = arr[m+1+j];
    while (i< m-left+1 && j< right-m)
        if (left[i] > right[j])
            count++;
            arr[k] = right[j];
            j++;
        else
            arr[k] = left[i];
            i++;
        k++;
    while (i< m+1+j)
        arr[i-] = left[i];
        i++;
        k++;
    while (j< right-m)
        arr[k] = R[j];
        j++;
        k++;
return count;
```

since this algorithm is modifying from merge sort. the time complexity is the same as merge sort that is $O(n \log n)$.

b. Every acyclic graph has n-1 edges. Thus the sum of the degrees of all the vertices of any tree have to 2(n-1). However if there are $n \geq 2$ vertices has at least two verti with degree 1, then the sum of the degrees of all the vertices must be at lease 2(n-1)+1, this is lead to a contradiction. //

C. Algorithm:

```
A[1...n];
int i;
int j;
int count = 0;
for i=0 to n-1
    int value = value + A[i];
    if value < 0
        count ++;
        value = A[i];
return count;
```

Using greedy algorithm as above

e. Algorithm:

exchange vertex constraint to an edge

```
modified_ford-Fulkerson(G, s, t)
    for each vertex v in G.Vertex
        v.flow = 0
    while there is a path p from s to t in residual network G_f
        C_f(P) = min{C_f(v) : v is in p}
        for each vertex V in p
            if V is in G vertex
                V.flow = V.flow - C_f(P);
            t.flow = t.flow + C_f(P);
            s.flow = s.flow - C_f(P);
    return t.flow.
```

We find a path from residual network, return the maximum flow as t.flow, the running time is $O((V+E)|f^*|)$.