CS570
Analysis of Algorithms
Spring 2016
Exam I

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	20	
Problem 3	15	
Problem 4	15	
Problem 5	15	
Problem 6	15	
Total	100	

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/]
All stable matchings are perfect matchings.

[TRUE/]
In class, we showed that choosing requests with earliest finish time will lead to an optimal solution to the interval scheduling problem. An optimal solution can also be achieved by choosing requests with latest start time.

[TRUE/]
The total cost of relaxation steps in Dijkstra's shortest path algorithm will be lower if we used a Fibonacci heap as opposed to a binary heap.

[/FALSE]
BFS can be used to find the shortest path between two nodes in any graph as long as the edge costs are all positive.

[/FALSE]
If function $f=O(n^3)$ and $g=O(n^2)$, then $f/g=O(n)$.

[/FALSE]
A graph has a unique MST if and only if all its edges have different weights.

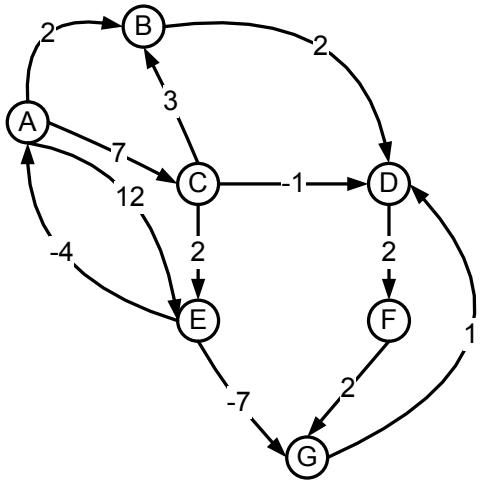
[TRUE/]
A binary heap A has each key randomly increased or decreased by 1. The random choices are independent. We can restore the heap property on A in linear time.

[/FALSE]
If item A is an ancestor of item B in a heap (used as a Priority Queue) then it must be the case that the Insert operation for item A occurred before the Insert operation for item B.

[TRUE/]
Prim's algorithm works correctly even when there are negative cost edges in the graph.

[/FALSE]
Every directed acyclic graph has exactly one topological ordering.

2) 20 pts
Consider the following directed, weighted graph:



a. Even though the graph has negative weight edges, step through Dijkstra's algorithm to calculate *supposedly* shortest paths from *A* to every other vertex. Show your steps in the table below. Cross out old values and write in new ones, from left to right within each cell, as the algorithm proceeds. Also, list the vertices in the order that Dijkstra's finds their shortest path.

Vertex	Found at Step	Distance	Path
A	0		
B	1	2	A
C	4	7	A
D	2	4	B
E	6	12,9	C
F	3	6	D
G	5	8	F

b. Dijkstra's algorithm found the wrong path to some of the vertices. For **just the vertices where the wrong path was computed**, indicate *both* the path that was computed and the correct path.

Computed path to G is A,B,D,F,G but shortest path is A,C,E,G.
Computed path to D is A,B,D but shortest path is A,C,E,G,D.
Computed path to F is A,B,D,F but shortest path is A,C,E,G,D,F.

c. List the minimum number of edges that could be removed from the graph so that Dijkstra's algorithm would happen to compute correct answers for all vertices in the remaining graph. (2 pts)

Remove the edge from E to G.

3) 15 pts

You are given a set of n intervals on the x-axis (a line): $[s_1, f_1]$; $[s_2, f_2]$; ……; $[s_n, f_n]$, where the i^{th} interval has left endpoint at $x = s_i$ and its right endpoint at $x = f_i$. Your goal is to select the minimum number of intervals whose union is the same as the union of all intervals. Give an efficient algorithm for this problem, analyze its running time, and prove that it is correct.

Algorithm**:**

1. Sort {si} in non-decreasing order
2. Of all the intervals with minimum si, select the one that has the latest fi.
3. Then take the last finishing time of all that overlap this, and so on until either all intervals have been considered, or until no overlapping ones remain.
4. If the latter case, go back to finding the minimum si of those uncovered. This can be implemented in O(n log n).

Proof:

Suppose the set of intervals we choose according to the greedy solution G is $[s\_g_1, f\_g_1]$, ……, $[s\_g_m, f\_g_m]$. An optimal solution OPT for covering the union is $[s\_o_1, f\_o_1]$, ……, $[s\_o_n, f\_o_n]$, and we have m<= n.

Consider the following solution OPT1: $[s\_g_1, f\_g_1]$, $[s\_o_2, f\_o_2]$ ……, $[s\_o_n, f\_o_n]$. Since s_g_1 is the earliest starting point, while f_g1 is the latest finishing point of the intervals with the earliest starting point, we have $[s\_g_1, f\_g_1]$ covers $[s\_o_1, f\_o_1]$. Therefore, OPT1 is also an optimal solution.

Suppose OPT_k: $[s\_g_1, f\_g_1]$, ……., $[s\_g_k, f\_g_k]$, $[s\_o_{k+1}, f\_o_{k+1}]$……, $[s\_o_n, f\_o_n]$ is optimal solution. Then consider OPT_{k+1}: $[s\_g_1, f\_g_1]$, ……., $[s\_g_{k+1}, f\_g_{k+1}]$, $[s\_o_{k+2}, f\_o_{k+2}]$……, $[s\_o_n, f\_o_n]$. Since f_g_{k+1} is the latest finish time of either the overlapping interval with $[s\_g_1, f\_g_1]$ U …….U $[s\_g_k, f\_g_k]$, or the non-overlapping interval with the earliest starting time s_g_{k+1}, then $[s_g_1, f_g_1]$U …….U $[s\_g_{k+1}, f\_g_{k+1}]$ covers $[s\_g_1, f\_g_1]$ U …….U $[s_g_k, f_g_k]$U$[s_o_{k+1}, f_o_{k+1}]$, . Therefore, OPT_{k+1} is also an optimal solution.

Based on the above induction, it follows that $[s\_g_1, f\_g_1]$, ……, $[s\_g_m, f\_g_m]$ is an optimal solution, and m = n. (Note that after selecting $[s\_g_m, f\_g_m]$ according to the greedy strategy, there should be no interval left)

4) 15 pts

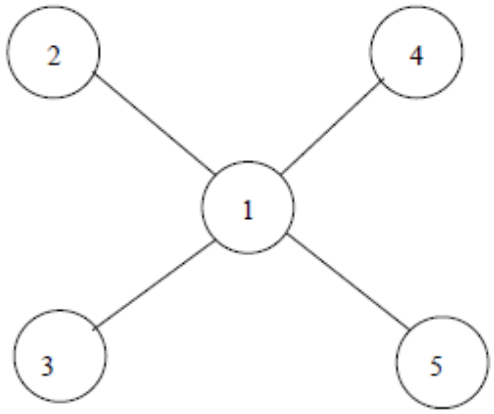
We have two routines for graph traversal - DFS(G, s) and BFS(G, s) - where G is a graph and s is the starting node in G. These two procedures will create a DFS tree and a BFS tree rooted at s respectively.

CLAIM: If G = (V, E) is a connected, undirected graph then the height of DFS(G, t) tree (rooted at an arbitrary node t) is always larger than or equal to the height of any of the BFS trees created by BFS(G, x).

Either prove this claim is true or provide a counterexample.

Note: we are comparing the height of one DFS tree rooted at t to the heights of |V| BFS trees each rooted at a different node in G.

Solution:



Counter example:

The DFS can start from any arbitrary node. So we pick the center node. For BFS we pick any of the boundary nodes. Then the height of DFS tree is 1, whereas the height of BFS tree is 2. Thus the claim is wrong.

5) 15 pts

Consider an array *A* containing *n* **distinct** integers. We define a local minimum of *A* to be an *x* such that *x=A[i]*, for some $0 \leq i < n$, with $A[i-1] > A[i]$ and $A[i] < A[i+1]$. In other words, a local minimum *x* is less than its neighbors in *A* (for boundary elements, there is only one neighbor to consider). As an example, suppose A = [10, 6, 4, 3, 12, 19, 18]. Then A has two local minima: 3 and 18.

a. Prove that **any** array *A* will have **at least** one local minimum

Proof. Assume there is no local minimum. Since A[0] is not a local minimum, and because all integers are distinct, we have A[1] < A[0]. Since A[1] is not a local minimum either, it must be the case that A[2] < A[1]…. Inductively, A[i] < A[i-1] for all 0 < i < n. Then A[n-1] is a local minimum. Contradiction.

b. Describe an algorithm using the divide and conquer technique to find **a** local minimum. Note that *A* might have multiple local minima, but you only have to locate and return one.

```
LocalMin(A[1…n])
if n = 1: return A[1]
else:
        m := floor(n/2)
        if A[m] is a local minimum: return A[m]
        else:
                if A[m-1]<A[m]<A[m+1]: return LocalMin(A[1…m-1])
                else: return LocalMin(A[m+1…n])
```

c. Express a recurrence equation for the running time of your algorithm and solve the recurrence.

T(n) = T(n/2) + θ(1)

Solution: T(n) = θ(log n)

6) 15 pts

You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, F)$ with respect to these weights. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly update the minimum spanning tree T to reflect this change, without re-computing the entire tree from scratch. There are four cases. In each case give a linear-time algorithm for updating the tree (if the tree needs to be updated).

(a) $e \in E - F$ and $\hat{w}(e) > w(e)$.
(b) $e \in E - F$ and $\hat{w}(e) < w(e)$.
(c) $e \in F$ and $\hat{w}(e) < w(e)$.
(d) $e \in F$ and $\hat{w}(e) > w(e)$.

Solution:

(a) Since e is not in T, increasing the weight of e cannot change the cost of T (an MST). So T is still an MST.
(b) Add e to T and call this new graph H. Use DFS to find a cycle in H. Find the highest weight edge on this cycle and delete it from H. The resulting graph is a tree and it follows from the cycle property that it is an MST. This all takes $\Theta(|V|)$ time.
(c) Since e is in T, decreasing its weight decreases the cost of an MST and T has this new weight. So T is still an MST.
(d) Delete e from T. This results in two trees A and B. Search through E to find the lightest edge that connects A and B (this lightest edge can still be e). This all takes $\Theta(|V| + |E|)$ time.

(a) 3.5 points.
- Indicating that MST needs no updates gets you 3.5 points.

(b) 4 points.
- Giving a linear time algorithm correctly updating the MST gets you 4 points.
- Giving a not working algorithm or some working non-linear algorithm such as Kruskal's gets you 2 points.
- Giving algorithms finding a cycle containing e in G (should be in T instead) gets you 3 points.

(c) 3.5 points.
- Indicating that MST needs no updates or needs only update on the edge cost gets you 3.5 points

(d) 4 points.
- Giving a linear time algorithm correctly updating the MST gets you 4 points.
- Giving a not working algorithm or some working non-linear algorithm such as Kruskal's gets you 2 points