1	2	3	4	5	6	Total

The University of Texas at Austin
Department of Electrical and Computer Engineering

EE w360C — Algorithms — Summer 2013

Exam I — Friday 21 June 2013

EID: _____

Instructions.

- You have 60 minutes to complete the exam. The maximum possible score is 70.
- The exam consists of 6 questions and 5 printed pages.
- It is a closed book / closed notes exam. No calculators, laptops, or other devices are allowed.
- Write your answers legibly on the test pages. Use back of test pages for scratch work. Show intermediate answers and process of solving questions.
- If there is any confusion, write down your assumptions and proceed to answer the question.
- In questions where you have to write an algorithm, you can describe it with reference to the algorithms we discussed in class. For example, you can say BFS with some additional operation at one step. Or use the output of topological sort directly etc.

1. What stable matching is found by the Gale-Shapley algorithm (in which men propose) for the preference lists given below:

	10 pts

Women	1	2	3	4	5
A	V	W	X	Y	Z
B	X	W	Z	Y	V
C	Y	Z	V	X	W
D	Z	X	Y	V	W
E	V	Y	Z	X	W

Men	1	2	3	4	5
V	A	B	E	C	D
W	A	C	D	B	E
X	D	B	E	A	C
Y	E	D	C	B	A
Z	C	D	A	B	E

Solution:

- V proposes to A, (V, A) formed
- W proposes to A, nothing changes
- W proposes to C, (W, C) formed
- X proposes to D, (X, D) formed
- Y proposes to E, (Y, E) formed
- Z proposes to C, (Z, C) formed, W is now free
- W proposes to D, W stay free
- W proposes to B, (W, B) formed

20 pts

2. Consider the following problem. You're given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$—that is, the sum $A[i] + A[i+1] + \ldots + A[j]$. (The value of array entry $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.) Here's a simple algorithm to solve this problem.

```
For i = 1, 2, ..., n
  For j = i+1, i+2, ..., n
    Add up array entries A[i] through A[j]
    Store the result in B[i, j]
  Endfor
Endfor
```

(a) (5 pts) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

Solution: The outer for loop runs at most n items, the inner loop runs at most n times, and the summation is over at most n elements. All the other operations take constant time. So $f = n^3$ and algorithm is $O(n^3)$

(b) (5 pts) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

Solution: For lower bound, consider the first $n/3$ iterations of outer loop. For each of these iterations, the inner loop runs at least $2n/3$ times. Consider the last $n/3$ iterations of this inner loop. Number of elements to be added for these inner loop iterations is at least $n/3$. Thus there are at least $(n/3)^3 = 1/27n^3$ addition operations. Hence the algorithm is $\Omega(n^3)$.

(c) (10 pts) Give a different algorithm to solve this problem, with an asymptotically better running time.

Solution:

```
currentSum = 0;
For i = 1, 2, ..., n
  currentSum = A[i];
  For j = i+1, i+2, ..., n
    currentSum = currentSum + A[j]
    Store the current sum in B[i, j]
  Endfor
Endfor
```

This algorithm runs in $\Theta(n^2)$

3. A binary tree is a rooted tree in which each node has at most two children. Show that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.
[Hint: Induction]

10 pts

Solution: Lets represent count of leaf nodes by L and count of nodes with two children by T. We want to prove $L_i = T_i + 1$ for all i.
Base case: For $n = 1$, $L_1 = 1$ and $T_1 = 0$. Hence $L_1 = T_1 + 1$.
Inductive case: Given a tree with $n + 1$ nodes, remove an arbitrary leaf l from this tree. From inductive hypothesis we know that $L_n = T_n + 1$ for the remaining n node tree. Now lets add back l. Consider two cases:
Case I: l had a sibling Adding l increases L_n by 1 as l is a new leaf and T_n by 1 as the parent of l has two children now. So $L_{n+1} = L_n + 1$ and $T_{n+1} = T_n + 1$ which implies that $L_{n+1} = T_{n+1} + 1$.
Case II: l had no sibling Adding l does not change L_n as the parent is no longer a leaf but l is a new leaf. Also T_n does not change as the parent did not have two nodes before or after adding l. Thus $L_{n+1} = L_n$ and $T_{n+1} = T_n$ and therefore $L_{n+1} = T_{n+1} + 1$.

4. Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$. Show that there must exist some node v, not equal to either s or t, such that deleting v from G destroys all s-t paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.)
[Hint: BFS Tree]

10 pts

Solution: Considers layers formed by the BFS algorithm starting from s. L_0 contains s alone by definition. Lets say t was reached in L_d. Then d is strictly greater than $n/2$. But there are at most $n - 2$ nodes other than s and t to fit in at least $n/2$ layers $L_1, \ldots, L_{d-1}$. Thus some layer must have one node alone. Lets call that node v. Because of BFS tree property that only adjacent layers have edges, no node in a layer reached earlier than v can have an edge to a layer reached after v i.e. all edges go through v. Hence deleting v disconnects the graph and destroys all s-t paths.

5. A Hamiltonian path is a path that visits every vertex of an input graph exactly once. Give an algorithm to efficiently (i.e., in polynomial time) determine whether or not a given DAG contains a Hamiltonian path.
[Hint: Topological Order]

10 pts

Solution: If a hamiltonian path exists in a DAG, then there is only one possible topological ordering which puts nodes in the order of this path. Any two nodes not put in the order of hamiltonian path will form a back-edge and hence it wouldn't be a topological ordering.
Algorithm works similar to topological sort except that we try to find a node with in-degree 0 only from neighbors of last node visited.

```
Set S=V, where V is the set of all vertices.
While V ≠ φ
  Find a node v ∈ S with in−degree 0.
  If no such node
    Declare hamiltonian path does not exist and Terminate
  EndIf
  Set S = {w|(v, w) ∈ E} where E is the set of all edges
  Remove v from DAG
EndWhile
Declare hamiltonian path exists
```

6. Let G be a graph on n nodes, where n is an even number. Prove that if every node of G has degree at least $n/2$, then G is connected.
[Hint: Contradiction]

10 pts

Solution: Assume G is not connected, then there are nodes x and y such that there is no path between them but both of them are connected to at least $n/2$ other nodes. Since there are only $n-2$ remaining nodes, there must be a node z adjacent to both x and y. But then x-z-y connects x and y. Hence a contradiction.