NAME: ____________________

CSE 421
Introduction to Algorithms
Sample Midterm Exam Fall 2014

DIRECTIONS:

- Answer the problems on the exam paper.
- You are allowed one cheat sheet.
- Justify all answers with proofs, unless the facts you need have been proved in class or in the book.
- If you need extra space use the back of a page
- You have 50 minutes to complete the exam.
- Please do not turn the exam over until you are instructed to do so.
- Good Luck!

1	/25
2	/25
3	/25
4	/25
Total	/100

1. (25 points, 5 each) For each of the following problems answer **True** or **False** and BRIEFLY JUSTIFY you answer.

   (a) $n^{2.1} = O(n^2 \log n)$.
   False. $n^{0.1}$ grows faster than $\log n$, as we discussed in class.

   (b) There is a polynomial time algorithm for deciding whether a graph is bipartite or not.
   True. We can use breadth first search to check whether a graph is bipartite or not.

   (c) If an undirected connected graph G has a unique heaviest weight edge e, then e cannot be part of any minimum spanning tree.
   False. If the edge is the only edge that connects a particular vertex, it must be included in every spanning tree.

   (d) If all edges in a graph have weight 1, then there is an $O(m + n)$ time algorithm to find the minimum spanning tree, where m is the number of edges and n is the number of vertices.
   True. In this case all spanning trees have the same weight. So we can use breadth first search to find a spanning tree.

(e) If $T(n) \leq 10T(n/3) + n^3$, $T(1) = 1$, then $T(n) = O(n^3)$. True. By the master theorem, since $3^3 > 10$, $T(n) = O(n^3)$.

2. (25 points) A perfect matching of an undirected graph on $2n$ vertices is a matching of size n, namely n edges such that each vertex is part of exactly one edge. Give a polynomial time algorithm that takes a tree on $2n$ vertices as input and finds a perfect matching in the tree, if such a matching exists. HINT: Give a greedy algorithm that tries to match a leaf in each step. *Solution*: To find the perfect matching, proceed as follows:

```
Input: A tree T.
Result: A perfect matching in the tree, if one exists.
Set M to be an empty set;
while T has vertices in it do
    if T has a vertex ℓ with deg(ℓ) = 1 then
        Let p be the neighbor of ℓ ;
        Add {p, ℓ} to M;
        Delete the vertices p, ℓ from T;
    else
        Output "no matching";
    end
end
Output M;
```

Algorithm 1: Perfect Matching Algorithm for Trees

Analysis: First we show, if the above algorithm outputs M, M is a matching of size n between the vertices such that each vertex is part of exactly one edge. This is because whenever we match two vertices p, ℓ we immediately delete them. Furthermore, the algorithm successfully outputs M when T has no more vertices. Since T has originally $2n$ vertices, the latter means $|M| = n$.

Coversely, suppose the above algorithm outputs "no matching" when there exists a matching M^* of size n. But observe that every vertex of degree 1 throughout the algorithm must be matched to its unique neighbor. Therefore, we haven't made any incorrect decisions. Furthermore, we know that every tree has a leaf, so the above algorithm will find a leaf p and match it in the only way possible. If this causes another neighbor of p to lose all of its edges, then there can be no perfect matching.

Runtime: All steps are polynomial time, so the runtime is polynomial time.

3. (25 points) A contiguous subsequence of a list S is a subsequence made up of consecutive elements of S. For instance, if S is

$$5, 15, -30, 10, -5, 40, 10,$$

then $15, -30, 10$ is a contiguous subsequence but $5, 15, 40$ is not. Give a polynomial time algorithm that takes n numbers as input, and outputs the contiguous sequence of maximum sum. HINT: Let $OPT(i)$ be maximum sum of all contiguous sequences that end at i, and show how to compute $OPT(i)$ for every value of i.

Solution: We solved this problem in class; so you can just say it is solved in class. Also, note that the problem only asks for a polynomial time algorithm. So, one can in principal return the simplest solution: For all interval $[x_i, \ldots, x_j]$ sum up all the numbers in the interval and take the maximum over all possible intervals.

Since, there are at most n^2 many intervals and we can compute the sum of numbers in each interval in time $O(n)$ the above algorithm runs in time $O(n^3)$ which is a polynomial in n.

4. (25 points) Given *sorted* array of n distinct integers, arranged in increasing order $A[1,n]$, you want to find out whether there is an index i for which $A[i] = i$. Give an algorithm that runs in time $O(\log n)$ for this problem. HINT: Consider the algorithm that compares $A[\lceil n/2 \rceil]$ and $\lceil n/2 \rceil$, and uses that comparison to recurse on either the first half or the second half of the array. Prove that if $A[\lceil n/2 \rceil] > \lceil n/2 \rceil$, such an i cannot be in last $n - \lceil n/2 \rceil$ coordinates, and if $A[\lceil n/2 \rceil] < \lceil n/2 \rceil$, then such an i cannot be in the first $\lceil n/2 \rceil$ coordinates. *Solution*:

```
Input: A sorted array A
Result: i such that A[i] = i, if such an i exists
Let k = 1, j = n;
while j − k > 1 do
  | Set ℓ = ⌊(j+k)/2⌋;
  | if A[ℓ] = ℓ then
  |   | Output ℓ.
  | else if A[ℓ] > ℓ then Set j = ℓ;
  | ;
  | else Set k = ℓ;
  | ;
end
if A[k] = k then
  | Output k;
else if A[j] = j then Output j ;
;
else Output "No such index";
;
```

Algorithm 2: Binary Search

Analysis: If $A[\ell] > \ell$, then it must be the case that any index i with $A[i] = i$ is in the interval $[k, \ell]$. This is because for all $j \geq \ell$,

$$A[j] \geq j - \ell + A[\ell] > j - \ell + \ell = j.$$

In the first inequality we have used that A is sorted array of distinct integers and in the second one we used that $A[\ell] > \ell$.

Similarly, if $A[\ell] < \ell$, it must be the case that the index we want is in the interval $[\ell, j]$. Thus the above algorithm correctly halves the size of the interval we are looking for, in each run of the while loop.

Runtime: Because each time we halve the size of the interval we are looking for, the runtime satisfies: $T(n) \leq T(n/2) + O(1)$. Thus $T(n) \leq O(\log n)$.