General Instructions

1. You may use the class notes and homework assignments, or any handwritten material. No electronic devices are allowed.

2. The work is to be your own and you are expected to adhere to the UMass Boston honor system.

3. Write your answers in the available spaces, using the back of the page if needed. Write clearly and concisely and try to avoid cursive.

4. Please explain your answers if needed but do it briefly.

5. You may use any proof technique we showed in class or any other technique, as long as it constitutes a mathematical proof. Remember that a proof by example is generally good only to show that something is NOT true.

6. If you base your answer on a homework question state exactly which question it was.

Good Luck!

Name: __________________________
1. (30%) Medians and order statistics: If we had an algorithm that finds the median of a sequence in linear time (worst case) – `findMedian(A,p,r)` which returns the index of the median of the sequence `A[p..r]`, describe a worst-case linear time algorithm that finds any order statistics. Provide a brief runtime analysis.

**Answer:** You can use the `findMedian` to modify the randomizedSelect procedure for order statistics. It basically guarantees that we always split the array in two, therefore guarantees $O(n \log n)$ performance. Specifically:

- If the array has one element - return it.
- `findMedian`
- If the median is what we need to find, return it.
- Otherwise, use median index as a pivot to Randomized Partition.
- If the order statistics is smaller than the median, repeat with the smaller half, otherwise with the bigger half.

Since we always split the array in half, a $O(n \log n)$ performance is guaranteed.

2. Binary Search Trees: (40%):

   (a) (20%) Show how to sort a set of $n$ numbers in $O(n \log n)$ time using only the following binary search tree operations: Minimum, insert and successor. You may assume each operation takes $O(\log n)$ time. Your initial input is just the array of numbers in random order.

   **Answer:** Do the following:

   - Insert all $n$ elements.
   - Find the minimum
   - Call successor $n-1$ times.

   If each operations takes $O(\log n)$, the whole procedure is $O(n \log n)$.

   (b) (20%) Let $b_n$ be the number of all possible binary search trees with $n$ nodes. Trivially, $b_0$ is 1. Show that $b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k}$.

   **(Hint:** Don’t look for hidden catches. Think combinatorially. I’m looking for quite a simple explanation).**

   **Answer:** Each one of the $n$ nodes can be the root. For each root, if it is the $k^{th}$ smallest element, we have $k-1$ nodes on the left subtree and $n-k$ nodes on the right subtree. Each one of the subtree can be arranged $b_{k-1}$ and $b_{n-k-1}$ ways, resp. The summation is over the selection of roots. Notice that the order statistics are indexed from 1 but the summation index is from 0, don’t get confused. I accepted any answer that got the principle right.

3. Dynamic Programming:

   (a) (8%) See question Question 2b above. Draw all the possible trees for 2 and 3 nodes (assume the keys are \{1, 2\} and \{1, 2, 3\}).

   **Answer:**

   See here:
(b) (9%) The question can be solved using Dynamic Programming. Based on 3a, show how many binary search trees can be constructed for 4 nodes by filling the table below (you can and should use the formula in 2b which can be used as recursion and DP):

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>b_n</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>14</td>
</tr>
</tbody>
</table>

(c) (8%) What is the run time of the DP algorithm as a function of n? Explain briefly.

**Answer:** The run time is \(O(n^2)\) since for each \(i\) we have to go over all the previous \(i-1\) substrees.