1. We showed in class that the expected (or average) height of a randomly built binary search tree is $\Theta(\log n)$. This is different from the average depth of a node in a fixed binary search tree. Describe a binary search tree on $n$ nodes such that the average depth of a node in the tree is $\Theta(\log n)$, but the height of the tree is $\omega(\log n)$.

**Solution:** Consider the following binary search tree. There is a complete binary tree on $n - \sqrt{n \log n}$ nodes and then there is a path of length $\sqrt{n \log n}$ hanging off from one of the leaves. This tree has height $\Theta(\sqrt{n \log n} = \omega(\log n)$, yet the average depth is $\Theta(\log n)$. (15 points)

2. In an experiment you pick at random a bit string of length 5. Consider the following events: $E_1$: the bit string chosen begins with 1, $E_2$: the bit string chosen ends with 1, $E_3$: the bit string chosen has exactly three 1s.

   (a) Find $p(E_1|E_3)$.
   (b) Find $p(E_3|E_2)$.
   (c) Find $p(E_2|E_3)$.
   (d) Find $p(E_3|E_1 \cap E_2)$.
   (e) Determine whether $E_1$ and $E_2$ are independent.
   (f) Determine whether $E_2$ and $E_3$ are independent.

**Solution:** We will need the following probabilities:

\[
p(E_1) = p(E_2) = \frac{1}{2}, \quad p(E_3) = \frac{\binom{3}{2}}{2^5} = \frac{5}{16},
\]
\[
p(E_1 \cap E_2) = \frac{1}{4}, \quad p(E_1 \cap E_3) = p(E_2 \cap E_3) = \frac{\binom{4}{2}}{2^5} = \frac{3}{16}.
\]
Then we have

a.) \( p(E_1|E_3) = \frac{p(E_1 \cap E_3)}{p(E_3)} = \frac{3}{16} = \frac{3}{5}. \)

b.) \( p(E_3|E_2) = \frac{p(E_2 \cap E_3)}{p(E_2)} = \frac{3}{16} = \frac{3}{8}. \)

c.) \( p(E_2|E_3) = \frac{p(E_2 \cap E_3)}{p(E_3)} = \frac{3}{16} = \frac{3}{5}. \)

d.) \( p(E_3|E_1 \cap E_2) = \frac{p(E_1 \cap E_2 \cap E_3)}{p(E_1 \cap E_2)} = \frac{3}{32} = \frac{3}{8}. \)

e.) They are independent, since \( p(E_1 \cap E_2) = \frac{1}{4} = \frac{1}{2} \cdot \frac{1}{2} = p(E_1) \cdot p(E_2). \)

f.) They are not independent, since \( p(E_2 \cap E_3) = \frac{3}{16} \neq \frac{1}{2} \cdot \frac{5}{16} = p(E_2) \cdot p(E_3). \)

(20 points)

3. Prove the combinatorial identity

\[
\sum_{k=1}^{n} k \binom{n}{k}^2 = n \binom{2n-1}{n-1}.
\]

**Solution:** Combinatorial proof: On both sides we count the number of ways we can select a committee of \( n \) members with a chairperson from a group of \( n \) mathematics professors and \( n \) computer science professors, such that the chairperson of the committee is a mathematics professor. (15 points)

4. Give an \( O(n) \)-time dynamic programming algorithm for the maximum subarray problem. That is, given an array of real numbers \( a_1, a_2, \ldots, a_n \), your algorithm must compute the maximum sum \( \sum_{i=j}^{k} a_i \), where \( 1 \leq j \leq k \leq n \). Draw the subproblem graph. How many vertices and edges are in the graph?
Solution: Here is the algorithm:
Maximum-subarray(n)
let $M[1..n]$ be a new array
$M[1] = a_1$
for $i = 2$ to $n$
    $M[i] = max(M[i - 1] + a_i, a_i)$
we find the maximum $M$ of $M[i], 1 \leq i \leq n$
return $M$

Each number in the sequence is computed from the previous number in the sequence. The running time is clearly $O(n)$. The subproblem graph consists of $n$ vertices, $v_1, \ldots, v_n$. For $i = 2, \ldots, n$, vertex $v_i$ has one leaving edge: to vertex $v_{i-1}$. No edge leaves vertex $v_1$. Thus, the subproblem graph has $(n - 1)$ edges. (15 points)

5. Not every greedy strategy works. Show that if in the activity-selection problem we always select the activity of least duration (another greedy strategy) from among those that are compatible with the previously selected activities we might not get an optimal solution.

Solution: Assume we have the following three activities: $a_1$ with $s_1 = 0, f_1 = 3, a_2$ with $s_2 = 2$ and $f_2 = 4$ and $a_3$ with $s_3 = 3$ and $f_3 = 6$. The suggested greedy approach selects just $\{a_2\}$, but the optimal solution selects $\{a_1, a_3\}$. (15 points)

6. Consider the following problem: we have to buy licenses for $n$ pieces of software, but we can buy at most one license per month. Currently each license is selling for $\$100$. But $t$ months from now license $j$ will cost $100 \cdot r_j^t$, where $r_j > 1$ is a growth rate factor and $r_i \neq r_j$ if $i \neq j$. Design a fast greedy algorithm that determines the order in which the licenses should be bought so that the total amount of money spent is minimized. Show that your algorithm gives an optimal answer. What is the running time?

Solution: Let us sort the $r_i$’s in decreasing order and we buy the licenses in this order. The running time is $O(n \log n)$. We claim that this gives an optimal answer. Otherwise assume indirectly that there is an optimal solution that differs from this. Then this solution contains
an “inversion”, i.e. there must exist two neighboring months $t$ and $t+1$ such that the price increase rate of the license bought in month $t$ (say $r_1$) is less than that bought in month $t+1$ (say $r_2$), so $r_1 < r_2$. Let us swap these two purchases, while the rest of the purchases are the same. We get a better solution (which is a contradiction) if

$$100(r_2^t + r_2^{t+1}) < 100(r_1^t + r_2^{t+1}).$$

This is true if

$$r_1^{t+1} - r_1^t < r_2^{t+1} - r_2^t,$$

or

$$r_1^t(r_1 - 1) < r_2^t(r_2 - 1).$$

But this is true since $r_i > 1$ for all $i$ and $r_1 < r_2$. (20 points)