Test 2 Design and Analysis of Algorithms

Question 1. (30 points)

```c
void optsearchtree(int n, const float p[], float &minavg, index R[][]) {
    index i, j, k, diagonal;
    float A[i..n+1][0..n];

    for (i = 1; i <= n; i++) {
        A[i][i-1] = 0;
        A[i][i] = p[i];
        R[i][i] = i;
        R[i][i-1] = 0;
    } // end for

    A[n+1][n] = 0;
    R[n+1][n] = 0;
    for (diagonal = 1; diagonal <= n-1; diagonal++) {
        for (i = 1; i <= n-diagonal; i++) {
            j = i + diagonal;
            A[i][j] = min(min(A[i][k-1] + A[k+1][j]) + \sum_{m=1}^{j} p[m]
                           R[i][j] = a value of k above that gave the minimum;
        } // end for
    } // end for
    minavg = A[1][n];
} // end optsearchtree
```

(a) For the following keys and probabilities of access, use the above algorithm to fill the A and R arrays, which can be used to determine the optimal binary search tree.

<table>
<thead>
<tr>
<th>Keys:</th>
<th>Key_1 Abe</th>
<th>Key_2 Bob</th>
<th>Key_3 Cal</th>
<th>Key_4 Doug</th>
<th>Key_5 Eli</th>
</tr>
</thead>
</table>

\[
\begin{array}{cccccc}
A: & 0 & 1 & 2 & 3 & 4 & 5 \\
1 & 0 & 0.25 & 0.55 & \text{0.55+1.4} = 1.05 & \text{0.65+0.7+1.35 = 2.85} & \text{1.05+1.25 = 2.05} \\
2 & 0 & 0.15 & 0.5 & 0.25+0.25 = 0.7 & 0.65+0.75 = 1.4 & 0.35+0.25 = 0.55 \\
3 & 0 & 0 & 0.2 & 0.4 & 0.4+0.6 = 1.0 & 0.3 \\
4 & 0 & 0 & 0 & 0.1 & 0.55 & 0.3 \\
5 & 0 & 0 & 0 & 0 & 0.5 & 0.3 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

R:

\[
\begin{array}{cccccc}
R: & 0 & 1 & 2 & 3 & 4 & 5 \\
1 & 0 & 1 & 1 & 2 & 2 & 3 \\
2 & 0 & 2 & 3 & 3 & 3 & 3 \\
3 & 0 & 3 & 3 & 3 & 5 & 5 \\
4 & 0 & 4 & 4 & 4 & 5 & 5 \\
5 & 0 & 5 & 5 & 5 & 5 & 5 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

(b) Draw optimal binary search tree here:

```
(b) DRAW the optimal binary search tree in the box at the top of the page.
```
Question 2. (35 points) Consider a dynamic-programming algorithm to compute $a^n$ based on the formulas:

\[
\begin{align*}
    a^0 &= 1, & \text{for } n = 0 \\
    a^1 &= a, & \text{for } n = 1 \\
    a^n &= a^{n-2}a^{n-2}, & \text{for even } n > 1 \\
    a^n &= a^{(n-1)/2}a^{(n-1)/2}, & \text{for odd } n > 1
\end{align*}
\]

(a) Complete the below powerOf algorithm which fills the array answers from index 0 (holds value of $a^0$) to index n (holds value of $a^n$). For example, a call to powerOf(2, 9) would fill the array answers with:

\[
\begin{array}{cccccccccc}
    0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
    \hline
    1 & 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 \\
\end{array}
\]

```java
number powerOf(number a, integer n) {
    number answers[0..n];
    answers[0] = 1
    answers[1] = a
    for (i = 2; i <= n; i++) {
        if (i mod 2 == 0) { // even
            answers[i] = answers[i/2] * answers[i/2];
        } else { // odd
            answers[i] = answers[(i-1)/2] * answers[(i-1)/2] * a;
        }
    }
    return answers[n];
} // end powerOf
```

(b) Suggest a way to speed up the above dynamic-programming algorithm.

A couple methods:

1. Use memoization to recursively solve only smaller problems need for $a^n$

2. Only loop through $n/2$ and calculate $a^n$ from either answers[n/2] or answers[(n-1)/2] in return statement. (Saves about half the work.)

3. Recursively calculate smaller problem, but square its result.
Question 3. (35 points) A college professor has covered all the "required" material for a course and has time to cover 3 more topics. She has a list of 10 possible topics ("A" to "J" below) that she could cover. To help decide how to fill the remaining class time, she had each of her 20 students to rank their first ("1"), second ("2"), and third ("3") picks from the list of 10 possible topics. Sample, summary data from the student surveys would look like:

<table>
<thead>
<tr>
<th>Students</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Student 2</td>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Student 3</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

a) Describe in **English** a greedy algorithm to select the "most popular" three topics to be covered using the summary data from the student surveys.

For each topic column, count number of times it was chosen and the sum of the choices. Select three topics with highest choice counts, but use smallest choice sum as tie-breaker.

b) Write your greedy algorithm using high-level pseudo-code. (English steps are sufficient)

Assume n students and m topics in summary table like above. Assume Os for non-chosen topics.

```plaintext
for topic = 1 to m do
    choiceCounts[topic] = 0
    choiceSum[topic] = 0
    for student = 1 to n do
        if summaryTable[student][topic] != 0 then
            choiceCounts[topic] += 1
            choiceSum[topic] += summaryTable[student][topic]
        end if
    end for
    find best topic
    find second best topic
    find third best topic
end for
```

c) What would be the theta notation, Θ( ), of your algorithm for n students and m topics?

Θ(n x m)