1. (a) The algorithm written in C:

```c
int T(int n) {
    int sum = 0;
    if (n==0 || n==1)
        return 2;
    for(int j=1;j<=n-1;j++)
        sum+=T(j)*T(j-1);
    return sum;
}
```

Let $N(n)$ denote the number of operations required to calculate $T(n)$. Note that $N(2) = 2$. So if $n$ is even, from the algorithm above we have:

$$
N(n) = \sum_{i=1}^{n-1} 2 \cdot N(i) - N(n-1) + N(0) + 2 \cdot (n-1) \\
\geq N(n-1) + N(n-2) \geq 2 \cdot N(n-2) \\
\geq 2 \cdot 2 \cdot N(n-4) \geq \cdots \geq 2^{\frac{n}{2}} .
$$

When $n$ is odd, it’s the same idea. Hence it’s exponential.

(b) An algorithm in C:

```c
T[0]=T[1]=2;
for(j=2;j<=n;j++)
{
    T[j] = 0;
    for(k=1;k<j;k++)
        T[j]+=T[k]*T[k-1];
}
return T[n];
```

(c) An algorithm in C:

```c
T[0]=T[1]=2;
T[2]=T[0]*T[1];
for(j=3;j<=n;j++)
return T[n];
```

2. The solution to the problem of the shortest common subsequence for three strings is essentially identical to the solution with 2 strings. By treating the $T(i,j,k)$’s as array entries and updating the table in the appropriate way we can get the following $O(n^3)$ time algorithm. Let $a$ be the length of $A$, $b$ be the length of $B$, and $c$ be the length of $C$. 
for j = 0 to b
  for k = 0 to c
    T(0,j,k)=0

for k = 0 to c
  for i = 0 to a
    T(i,0,k)=0

for i = 0 to a
  for j = 0 to b
    T(i,j,0)=0

for i = 1 to a do
  for j = 1 to b do
    for k = 1 to c do
      if a(i) = b(j) = c(k) then
        T(i,j,k)=T(i-1,j-1,k-1) + 1
      else
        T(i,j,k)= MAX(T(i, j-1, k), T(i-1, j, k), T(i, j, k-1))

3. [This solution was adapted from Brian Wongchaowart’s homework writeup.]

(a) The path taken by the algorithm through the table to reconstruct
    the shortest common super-sequence is highlighted in bold.
    Let M be the name of our table, let A = zxyyzz and B = zzyxzy.

          M     B
            j=    0 1 2 3 4 5 6
                z z y x z y
    i=0        0 1 2 3 4 5 6
    1 z        1 1 2 3 4 5 6
    2 x        2 2 3 4 4 5 6
    A 3 y      3 3 4 4 5 6 6
    4 y        4 4 5 5 6 7 7
    5 z        5 5 5 6 7 7 8
    6 z        6 6 6 7 8 8 9

Following this path from M[6,6] to M[0,0], the shortest common
super-sequence of A and B is zzyxzyyzz. (For details see part c.)

(b) If the strings A and B have length m and n, respectively, the table entry
    M[m, n] gives the length of the shortest common super-sequence
    of A and B. In this example, the bottom-right entry in the table is 9,
    which is the length of the shortest common super-sequence of zxyyzz
    and zzyxzy.
(c) The letters of string A label the rows of the table, and the letters of string B label the columns. The shortest common super-sequence is constructed in reverse order starting from the bottom-right entry. A letter is added to the beginning of the partial super-sequence constructed so far as one moves past its row or column, so moving left adds the letter at the top of the current column, and moving up adds the letter at the head of the current row. A diagonal move is only allowed when the current row and column are marked with the same letter; that letter is added once to the common super-sequence but is accounted for (moved over) in both words.

The idea is to “reverse” the code that built the table, eventually reaching the upper-left corner (the entry containing the 0) in as few moves as possible, at which point every letter of both words will have been added to the super-sequence. Note that in the first row and column there is obviously only one possible direction in which to move.

Some pseudocode for the trace back:

\[
\begin{align*}
  i &= m, \ j = n, \ S = \varepsilon \\
  \text{while } M[i, j] > 0 & /\text{do until we’ve reached upper left corner of } M \\
    & \quad \text{if } i == 0 /\text{we’re in the first row} \\
    & \quad \quad S = b_j + S; \ /\text{add the letter from } B \\
    & \quad \quad j - -; \ /\text{move left one column} \\
    & \quad \text{else if } j == 0 /\text{we’re in the first column} \\
    & \quad \quad S = a_i + S; \ /\text{add letter from } A \\
    & \quad \quad i - -; \ /\text{move up one row} \\
    & \quad \text{else if } a_i == b_j \text{ then} \\
    & \quad \quad S = a_i + S; \ /\text{add this letter (it’s in both } A \text{ and } B) \\
    & \quad \quad i - -, \ j - -; /\text{move diagonally up and left} \\
    & \quad \text{else if } M(i - 1, j) \leq M(i, j - 1) \text{ then} \\
    & \quad \quad S = a_i + S; \ /\text{add letter from } A \\
    & \quad \quad i - -; /\text{move up one row} \\
    & \quad \text{else} \\
    & \quad \quad S = b_j + S; /\text{add letter from } B \\
    & \quad \quad j - -; /\text{move left one column} \\
  \end{align*}
\]

end if

end while

return S

4. We present an algorithm to compute the minimum edit distance of two strings. Note that:

(a) If it were possible to convert \( a_1, \ldots, a_{m - 1} \) into \( b_1, \ldots, b_n \), one could complete the transformation of \( A \) into \( B \) by deleting \( a_m \).

(b) If it were possible to convert \( a_1, \ldots, a_m \) into \( b_1, \ldots, b_{n - 1} \), one could
complete the transformation by adding $b_n$ to $A$.

(c) If it were possible to convert $a_1, \ldots, a_{m-1}$ into $b_1, \ldots, b_{n-1}$, one could complete the transformation by replacing $a_m$ with $b_n$.

(d) If string $A$ is empty and $B$ is not empty then the conversion can only be done by inserting all remaining characters of $B$ into $A$.

(e) If string $B$ is empty and $A$ is not empty then the conversion can only be done by deleting all remaining characters in $A$ (assuming we can’t replace a character with the empty character).

Now let $A[i, j]$ be the minimum cost of transforming $a_1, \ldots, a_i$ into $b_1, \ldots, b_j$. The algorithm is:

```
MinimumEditDistance(A, B)
A[0, 0] = 0
for i = 1 to m
    A[i, 0] = i * 3
for j = 1 to n
    A[0, j] = j * 4
for i = 1 to m
    for j = 1 to n
        if $a_i = b_j$ then
```

Starting from $A[m, n]$, we can trace backwards through the table to determine which operations were performed at each step.

5. [This solution is courtesy Matthias Grabmair (in collaboration with Ian Wong).]

For $K_1, K_2, K_3, K_4, K_5$, our algorithm produces the following table of expected access times:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.5</td>
<td>.6</td>
<td>.85</td>
<td>1.4</td>
<td>2.15</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>.05</td>
<td>.2</td>
<td>.55</td>
<td>1.05</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>.1</td>
<td>.4</td>
<td>.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>.2</td>
<td>.65</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Given this table, we can produce the table of roots that correspond to the above access times:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hence, the tree looks as follows:

```
  1
 / \
 4   / \
 /   3 5
/     /
2
```

The expected access time of the tree is found in the upper right corner of the table, 2.15. The tree can be constructed from the table above as follows. Start with the upper right corner (entry [2,5]) in the table of roots to see that node 1 is at the root of the tree. Hence the remaining nodes 2 through 4 are all in one subtree that is the right child of the root. The subtree has minimal weight when the root is 4, given in position [2,5] of the table. Since node 4 is the new root node, 5 inevitably becomes 4’s right child node. For the subtree that is the left child of node 4, we need to figure out which of 2 or 3 is the root. According to table cell [2,3] it is node 3.