1. Fibonacci numbers are a sequence of integers defined recursively as follows: \( F_0 = F_1 = 1 \) and \( F_n = F_{n-1} + F_{n-2}, \quad n \geq 2. \)

(a) The following recursive function computes \( F_n \), but is extremely inefficient. Explain very briefly why it is inefficient.

```c
int F(int n){
    if (n <= 1) return 1;
    else return F(n-1) + F(n-2); }
```

(b) Let \( T(n) \) be the total time for computing \( F(n) \) by the above method. Write a recurrence for \( T(n) \).

(c) Prove by induction that \( T(n) \geq A \cdot 2^\lfloor n/2 \rfloor \) for some constant \( A \). Be sure to show the arithmetic completely.
2. Suppose we want every node in a **Binary-Search-Tree (BST)** to contain the total number of descendants of that node (which is the total number of nodes in the subtree rooted at that node). Thus, each node will have four fields: **KEY** (the key value stores at that node), **DES** (number of descendants), **LC** (left-child pointer), **RC** (right-child pointer).

(a) Write the code for inserting a new element into BST, while updating the descendant information for all the nodes that are effected. (Note that BST is not rebalanced.)

(b) What is the time complexity of an insert operation in terms of the number of nodes in the tree, \( n \), and the height of the tree, \( h \).
3. (a) Insert element 72 in the following AVL tree. (First complete the picture by drawing lines from each node to its children.) Show the result after the initial insert, and then after each rotate operation needed to rebalance.

```
50
  30    70
 20  55   80
  52  60  75  90
```

(b) Delete element 52 from the following AVL tree. Show the result after the initial delete, and after each rotate operation needed to rebalance.

```
50
  30    70
 20  35  55   80
10  52  75   90
   72  78  85  99
```

(c) Delete element 52 from the following AVL tree. Show the result after the initial delete, and after each rotate operation needed to rebalance.

```
50
  30    70
 20  35  55   80
10  52  75   90
   72  78
```
4. Given an array $A[0..n-1]$ and a small constant $k$. Suppose that array $A$ at the beginning is “nearly sorted” such that for every element $A[i]$, at most $k$ elements to the left of $A[i]$ are greater than $A[i]$. We define such a sequence to be $k$-near-sorted. For example, the following sequence is 2-near-sorted:

$$[8, 1, 2, 7, 3, 4, 5, 6].$$

(a) Write an efficient algorithm (write the code) for completely sorting an array $A$ which is $k$-near-sorted.

(b) What is the time complexity of this algorithm in terms of $n$ and $k$?

(c) Illustrate the algorithm for the sequence below, where $k = 2$.

$$[8, 1, 2, 7, 3, 4, 5, 6].$$
5. Given the following directed graph, with $n = 9$ vertices, defined by its adjacent lists.

(a) Show the breadth-first spanning tree for the graph, starting with the source vertex 1.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$L_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3, 6, 7</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>3, 8</td>
</tr>
<tr>
<td>7</td>
<td>8, 9</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) Show the depth-first spanning tree, starting with the source vertex 1.