Name: _____

Entry number: _____

There are 4 questions for a total of 15 points.

---

1. Answer the following questions:

   (a) (½ point) State true or false: In any undirected graph, the sum of degrees of odd-degree vertices is always an even. (*Degree of a vertex is the number of neighbouring vertices it has.*)

   (a) **True**

   (b) (½ point) State true or false: In any strongly connected undirected graph with at least two vertices, there are always two vertices that have the same degree.

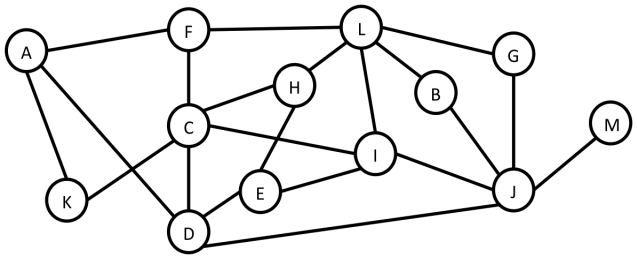
   (b) **True**

   (c) (½ point) State true or false: For any directed graph G with $n \geq 2$ vertices and any two vertices s, t in G, the number of distinct paths from s to t is $O(2^n)$

   (c) **False**

   (d) (½ point) State true or false The graph given below is a bipartite graph.

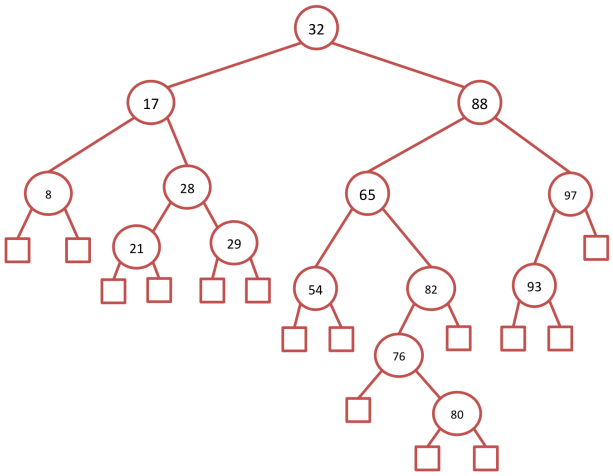
   (d) **True**



   (e) (½ point) State true or false Any strongly connected directed graph with at least two vertices contains a cycle.

   (e) **True**

(f) (½ point) Consider the binary search tree below. Draw the binary search tree obtained after performing `remove(88)`.



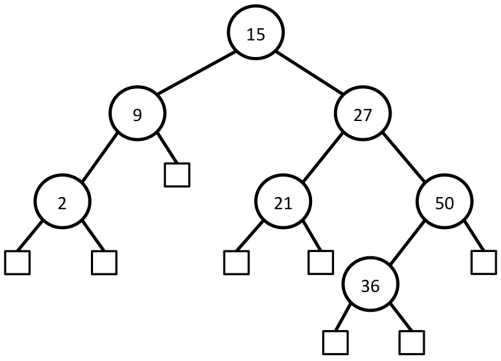
(g) (1 point) Consider the array A representing a min-heap below (only keys are shown) and answer the question that follow:

Array index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Key	3	10	5	13	17	6	11	15	16	21	18	9	8	23

Give the array after performing one `removeMin()` operation.

Array index	1	2	3	4	5	6	7	8	9	10	11	12	13
Key	5	10	6	13	17	8	11	15	16	21	18	9	23

(h) (1 point) Show the AVL tree that results after each of the integer keys $9, 27, 50, 15, 2, 21$, and 36 are inserted, in that order, into an initially empty AVL tree. (*You may want to show steps for partial grading*)



2. (4 points) You are given an array $A[1...n]$ representing a min-heap and an integer k. You have to design an algorithm to output all the keys in the array that are less k.
(*For example if* $k = 6$ *and the array* A *is as shown below, then your algorithm should output the keys* 3 *and* 5.)

Array index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Key	3	10	5	13	17	6	11	15	16	21	18	9	8	23

Give pseudocode for your algorithm and discuss running time. Let m be the number of keys in A that are smaller than k. The running time of your algorithm should be $O(m)$.

Solution: Here is the pseudocode for the algorithm. The function call should be `SmallerThan(`$A, n, k, 1$`)`.

```
SmallerThan(A, n, k, i)
  - if(A[i] ≥ k OR i > n)return
  - Print(A[i])
  - SmallerThan(A, n, k, 2i)
  - SmallerThan(A, n, k, 2i + 1)
```

Running time: Let the number of elements smaller than k in the array A be m. The above algorithm is a recursive algorithm. There are constant number of operations performed in every recursive call before further recursive calls are made. So, the total running time of the algorithm is proportional to the total number of recursive calls that are made during the execution of the algorithm. For each element that are smaller than k, at most two recursive calls will be made. Note that no recursive calls are made with respect to elements that are at least k. So, the total number of recursive calls will be bounded by $2m$.

3. (3 points) Consider the following basic java implementation of Binary Search Tree that we have used in the lecture. Complete the body of the method `PrintKeys` below that is supposed to print all the keys in the tree in a sorted order (increasing order). Your algorithm should run in $O(n)$ time where n is the number of keys in the binary search tree. (*The methods for insert, search, delete are not shown below*)

```
class Node{
   public int key;//This denotes the key stored at the node
   public String value;//This denotes the value stored in the node.
   public Node leftChild;//This is a reference to the left child
   public Node rightChild;//This is a reference to the right child
   public Node parent;//This is a reference to the parent of the node
   public Node(){parent = leftChild = rightChild = null;}
}
public class BinarySearchTree{
   public Node root;
   public int size;
   public BinarySearchTree(){size=0;root=null;}

   //This method checks if the given node is a leaf
   public boolean isLeaf(Node N){
       if(N.leftChild == null && N.rightChild == null)return(true);
       else return(false);
   }
   ... other methods

   //This method should print the keys in sorted order
   public void PrintKeys(){
       InOrderTraverse(root);
  }

   public void InOrderTraverse(Node N){
       if(isLeaf(N))return;
       InOrderTraverse(N.leftChild);
       System.out.println(N.key);
       InOrderTraverse(N.rightChild);
   }
}
```

4. (3 points) You are given an unsorted array $A = A[1...n]$ containing n distinct integers. Design an algorithm that outputs the smallest k elements in the array A. The running time of your algorithm should be $O(n + k \log n)$. Give pseudocode and discuss running time.

Solution: Here is the pseudocode for the algorithm.

```
SmallElements(A, k, n)
  - BottomUpHeapConstruction(A, n)
  - For i = 1 to k
      - Print(A[1])
      - A[1] ← A[n]
      - n ← n − 1
      - DownHeapBubble(A, n, 1)


DownHeapBubble(A, n, i)
  - c ← i
  - if(2i > n and 2i + 1 > n)return;
  - elseif(2i ≤ n and 2i + 1 > n){if(A[c] > A[2i]) c ← 2i}
  - else{
      if(A[c] > A[2i])c ← 2i
      if(A[c] > A[2i + 1])c ← 2i + 1
  }
  - if(c = i)return;
  - else{swap(A, i, c); DownHeapBubble(A, n, c)}


BottomUpHeapConstruction(A, n)
  - h ← ⌊log n⌋
  - For i = h − 1 to 0
      - For j = 2^i to 2^(i+1) − 1
          - DownHeapBubble(A, n, j)
```

We use the array based bottom-up heap construction discussed in the class to convert array A to one representing a min-heap. After this we perform the remove minimum operation on this heap to obtain the k smallest elements.

Running time: Constructing the heap will take time $O(n)$ and performing k remove minimum operations costs $O(k \log n)$ time. So, the total running time is $O(n + k \log n)$.