Miscellaneous

1. Find all non-isomorphic connected simple graphs with 4 or fewer vertices. How many did you find?

---

2. Determine (up to isomorphism) how many rooted binary trees there are with 5 or fewer vertices.

Shortest Paths

3. Dijkstra's algorithm.
   (a) Explain what the algorithm finds.
   (b) Give any restrictions on the algorithm.
   (c) Describe how the algorithm runs.
   (d) Run the algorithm on the weighted directed graph G with initial vertex a where

$$G = \{a : \{b : 2,\ c : 3\},\quad b : \{c : 2,\ d : 4\},\quad c : \{b : 1,\ d : 1,\ e : 4\},$$
$$d : \{e : 2,\ f : 3\},\quad e : \{d : 1,\ f : 2\},\quad f : \{\}\}$$

Heaps

4. What properties must a binary tree have in order to be a *heap*?

---

5. Recall the function HEAPIFY(A, k) where A is a binary tree and k is an index.
   (a) What conditions on A and k are assumed by this function?
   (b) What does this function accomplish?

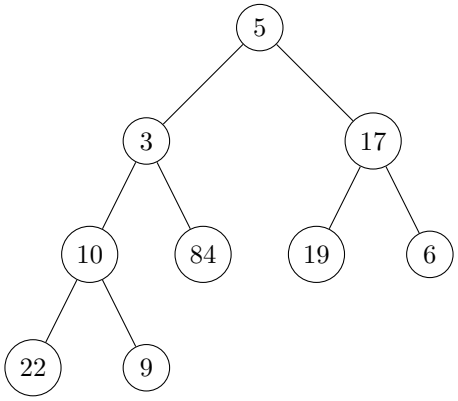
---

6. Recall the function

   BUILD-HEAP(A)

   ```
   1  heap-size(A) ← length(A)
   2  for k ← ⌊length(A)/2⌋ downto 1
   3      do HEAPIFY(A, k)
   ```

   Run BUILD-HEAP(A) on the following binary tree showing the step by step transformation of the tree.



---

7. Run the function HEAP-EXTRACT-MAX on your heap from Problem 6 showing the step by step transformation of the tree.

Binary Search Trees

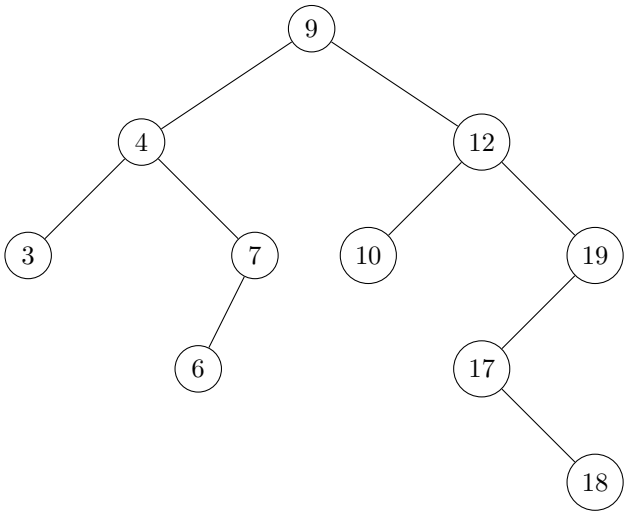
8. What properties must a binary tree have in order to be a binary *search* tree?

---

9. Compute the expected height of a randomly generated binary search tree having 3 vertices with distinct keys.

---

10. The numbers displayed in the binary search tree T below are the keys of the nodes. Run the function TREE-DELETE(T, z) where z has key 12.



---

11. Run the function TREE-INSERT(T, w) where T is the tree you obtained in problem 10 and w is a node with key 12.

Sorting

12. What is the official name of the python `sort` function given below?

```
def sort(A):
    if len(A) <= 1:
        return A
    a,L,R = A[0],[],[]
    for k in A[1:]:
        if k < a:
            L.append(k)
        else:
            R.append(k)
    return sort(L) + [a] + sort(R)
```

---

13. How many total comparisons would be required if the above `sort` function was run on the list $L = [1, 2, 3, 4, 5, 6, 7, 8]$?