Algorithms and Data Structures

Final Exam

July 30, 2024

1. (a) Order the following functions by their asymptotic growth rate:

$$n, \exp n, \log n, \exp(n^2), \log(\log n), (\log n)^2$$

   (b) An algorithm takes as input an n-element sequence of integers. It iterates through all of its elements, executing an $O(\log n)$-time computation if the value of the current element is even, resp. an $O(n)$-time computation if the value of the current element is odd. What are the best-case and worst-case running times of the algorithm?

   (c) In the following, `T` is an instance of a standard binary tree class. It contains n nodes. What does the following code do? What is its running time as a function of n? Give a Big-Oh upper bound and a Big-Omega lower bound.

```
def do_something(T, p):
    if p is not None:
        do_something(T, T.left(p))
        print(T.element(p))
        do_someting(T, T.right(p))

do_something(T, T.root())
```

(5+5+5)

2. Are the following statements true or false? Explain your answer in 1–2 sentences.

   (a) A singly linked list can be used to implement a stack such that “push” and “pop” execute in $O(1)$-time.

   (b) A singly linked list can be used to implement a queue such that “enqueue” and “dequeue” execute in $O(1)$-time.

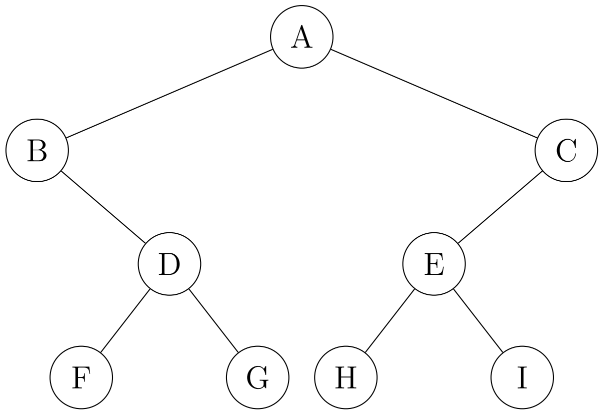
   (c) An AVL tree can be used to sort a list $O(n \log n)$-time.

   (d) A heap can be used to sort a list in $O(n \log n)$-time.

   (e) A skip list can be used to sort a list in $O(n \log n)$-time.

(2+2+2+2+2)

3. Give the pre-order, in-order, and post-order traversals of the following binary tree:



(5)

4. Insert the keys 6, 9, 4, 3, 5, 7 into an initially empty heap. Show the heap at each step of insertion. (5)

5. Insert the keys 6, 9, 4, 3, 5, 7 into an initially empty splay tree. Show the splay tree at each step of insertion.

   (You may consult the attached splay tree "cheat sheet".) (5)

6. Describe an algorithm that takes an undirected, connected graph $G = (V, E)$ as input and returns `True` if the graph is a tree and `False` otherwise. (5)

7. Consider the following greedy strategy for finding a shortest path from vertex *start* to vertex *goal* in a given connected graph.

   (a) Initialize path to *start*.

   (b) Initialize set *visited* to $\{start\}$.

   (c) If *start*=*goal*, return path and exit. Otherwise, continue.

   (d) Find the edge (*start*, v) of minimum weight such that v is adjacent to *start* and v is not in *visited*.

   (e) Add v to path.

   (f) Add v to *visited*.

   (g) Set *start* equal to v and go to step (c).

   Does this greedy strategy always find a shortest path from *start* to *goal*? Either explain intuitively why it works, or give a counterexample. (5)