Algorithms and Data Structures

Final Exam

August 8, 2022

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

$$n\,(n + \log n), n\, \log n, \log n, (\log n)^2, 1\,000\,000\,000, n^{3/2}$$

(b) Give a sharp Big-Oh upper bound and a sharp Big-Omega lower bound for the running time of the following Python function which takes as input a Python list `S` of length n.

```
def mystery_function (S):
    for j in range(len(S)):
        for k in range(j+1, len(S)):
            if S[j] == S[k]:
                return False
    return True
```

(c) What is `mystery_function` good for?

(d) Can you suggest a different implementation of `mystery_function` which has a running time of $O(n \log n)$?

Note: No need to write Python code – a high-level description in words suffices. Be sure to justify that your suggestion achieves the claimed running time.

(5+5+5+5)

2. True or false? Explain your answer in 1–2 sentences.

(a) A pre-order traversal of a tree with n nodes runs in $O(n)$ time.

(b) Accessing the next element in a pre-order traversal of a tree can always be done in $O(1)$ time.

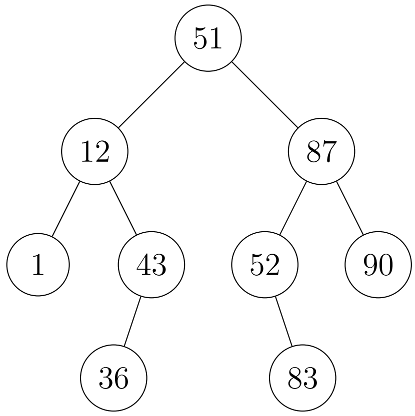
(c) Accessing the next element in a breadth-first traversal of a binary tree can always be done in $O(1)$ time.

(d) Inserting an element into a hash table can always be done in $O(1)$-time.

(e) There are binary search trees where a search can take $\Omega(n)$ time.

(2+2+2+2+2)

3. (a) Draw the resulting binary search tree when you remove the root node 51 from the following tree.



(b) Draw an example of an AVL-tree where the removal of an element triggers more than one rotation.

(c) Suppose you have an implementation of an AVL-tree. Argue that you can use it to sort a list in $O(n \log n)$ time.

(5+5+5)

4. Let `G` be an undirected graph without weights. The graph distance between vertices `u` and `v` is then defined as the length of the shortest path between `u` and `v`. The following function computes the graph distance, where `LinkedQueue` is an implementation of the standard queue data type, and `neighboring_vertices(w)` is an iterator over all vertices connected to `w` by an edge.

```
def distance(G, u, v):
    Q = LinkedQueue()
    Q.enqueue((u,0))
    while not Q.is_empty():
        w, d = Q.dequeue()
        if w == v:
            return d
        for s in G.neighboring_vertices(w):
            Q.enqueue((s, d+1))
```

(a) How does this algorithm work? (You may argue by analogy with one of the tree traversal algorithms.)

(b) In certain cases, this algorithm may fail spectacularly. Why and when?

(c) What is the running time, in Big-Oh notation, of this algorithm if every vertex has an edge to 5 others? And if the graph is complete, i.e., every vertex has an edge to every other vertex?

(d) Give a modification of the algorithm that reduces the running time to $O(n+m)$, where n is the number of vertices and m is the number of edges in the graph. (You may state the modification in precise language – no need to write Python code.) Give a brief reasoning why your modification respects the required running time bound.

(e) Does your modification avoid the problem from part (b)? Why or why not?

(5+5+5+5+5)

5. Write an algorithm `search`, in pseudo-code or in Python, which returns the position of an element with key `k` in a binary search tree `T` if the element is found, and returns `None` otherwise.

(10)