# Algorithms and Data Structures

Mock Exam

July 9, 2024

1. (a) Simplify the following Big-Oh resp. Big-Omega expressions as much as possible:

   (i) $O(\log n) + O(n \log n) + O((\log n)^4)$

   (ii) $\Omega(\log n) + \Omega(\log \log n) + \Omega(\log \log \log n)$

   (iii) $O(1 + 2n + 3n^2 + 4n^3)$

   (iv) $O\left(\sum_{i=1}^{n} i^2\right)$

   (v) $O(n) + \Omega(n)$

   (b) What is the running time of the following code as a function of $n$? Give a Big-Oh upper bound and a Big-Omega lower bound.

```
1 def idle(n):
2     r = 0
3     i = n
4     while i>0:
5         r += i
6         i = i//2
7     return r
```

(10+5)

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

   (a) Python lists (i.e., dynamic arrays) can be used to efficiently implement a stack, using `L.append(e)` and `L.pop()` for push and pop, respectively.

   (b) Python lists (i.e., dynamic arrays) can be used to efficiently implement a queue, using `L.append(e)` and `L.pop(0)` for enqueue and dequeue, respectively.

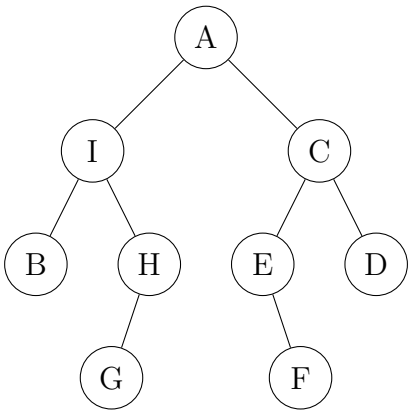
   (c) A heap can be searched more efficiently than an unsorted array.

   (d) A heap can be searched more efficiently than a sorted array.

   (e) Breadth-first traversal of a binary tree takes worst-case $O(1)$ time for each node.

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

5. Insert the keys 4, 9, 3, 7, 5, 6 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. A function takes as input a list of integers `L` of length $n$ and an integer value `total`. It returns a tuple of elements from the list with sum `total`, if possible, and `None` otherwise. Describe an algorithm that can do this in $O(n \log n)$ time. (5)

7. In the following, $G$ is a graph in a "map of maps" representation. An example is the following:

```
G = {'A' : {'B', 'E', 'D'},
     'B' : {},
     'C' : {'B', 'E', 'F'},
     'D' : {'B'},
     'E' : {'D'},
     'F' : {'A', 'D', 'E'}}
```

   (a) Draw a representation of this graph in the plane.

   (b) What does the following function do if it is executed on a graph `G` above?

```
def mystery_function(p):
    for n in G[p]:
        if n not in v:
            v[n] = p
            mystery_function(n)

v = {}
v['A'] = None
mystery_function('A')
```

(c) What data structure is encoded in `v` if this code is run on a general graph `G`?

(d) If `n=len(G)`, what is the running time of this algorithm? Give a Big-Oh upper bound and a Big-Omega lower bound. Justify your answer using your knowledge about the running time of the native Python dictionary operations.

(e) Sketch, using Python or Python-like pseudocode, a "breadth-first search" (BFS) traversal of a graph of this type.

(2+3+2+3+5)