Version POK/3

EGT3
ENGINEERING TRIPOS PART IIB

Wednesday 24 April 2024 9.30 to 11.10

Module 4M26

ALGORITHMS AND DATA STRUCTURES

*Answer not more than **three** questions.*

All questions carry the same number of marks.

*The **approximate** percentage of marks allocated to each part of a question is indicated in the right margin.*

*Write your candidate number **not** your name on the cover sheet.*

STATIONERY REQUIREMENTS
Single-sided script paper

SPECIAL REQUIREMENTS TO BE SUPPLIED FOR THIS EXAM
CUED approved calculator allowed
Engineering Data Book

10 minutes reading time is allowed for this paper at the start of the exam.

You may not start to read the questions printed on the subsequent pages of this question paper until instructed to do so.

You may not remove any stationery from the Examination Room.

1 (a) The code below is designed to implement deletion in a Binary Search Tree but contains a number of mistakes. Identify each mistake and explain how it can be fixed. [35%]

```
class Node:

    def __init__(self, key, parent, left, right):
        self.key = key
        self.parent = parent
        self.left, self.right = right, left

class BinarySearchTree:

    def __init__(self, root):
        self.root = root

    def shift_nodes(self, old, src):
        if not old.parent:
            self.root = src
        elif old = old.parent.left:
            old.parent.left = src
        else:
            old.parent.right = src
        if src:
            src.parent = old.parent

    def delete(self, u):
        if not u.left:
            self.shift_nodes(u, u.right)
        elif not u.right:
            self.shift_nodes(u, u.right)
        else:
            succ = self.maximum(u.left)
            if succ != u.right:
                self.shift_nodes(succ, succ.left)
                succ.right = u.right
                succ.right.parent = succ
            self.shift_nodes(u, succ)
            succ.left = u.left
            succ.left.parent = succ
```

(b) Does the deletion operation in a Binary Search Tree have the following property? *For any pair of nodes* `u` *and* `v` *belonging to a Binary Search Tree* `T`*, the result of calling* `T.delete(u)` *then* `T.delete(v)` *is identical to the result of calling* `T.delete(v)` *then* `T.delete(u)`. Explain your reasoning. [15%]

(c) In at most three steps, describe a strategy to solve the following problem. Given a Binary Search Tree T and an integer k, find the kth smallest key in the tree. [15%]

(d) In at most three steps, describe a strategy to solve the following problem. A Binary Search Tree T has exactly two nodes' keys swapped such that the Binary Search Tree property no longer holds. Transform T into a valid Binary Search Tree without modifying its structure. [15%]

(e) A sequence of keys $k_1, k_2, k_3, \ldots$ is to be inserted into a Binary Search Tree T.

(i) Give the *worst case* asymptotic complexity of this operation as a function of the number of keys, n, inserted into T so far. Define a deterministic sequence of unique keys that induces this behaviour. [10%]

(ii) Give the *best case* asymptotic complexity of this operation as a function of the number of keys, n, inserted into T so far. Define a deterministic sequence of unique keys that induces this behaviour. [10%]

2 (a) The code below aims to implement the quicksort algorithm. Unfortunately, this implementation contains a number of mistakes. Identify each mistake and explain how it can be fixed. [35%]

```
def partition(A: list, low: int, high: int) -> int:
    pivot = high
    pivot_val = A[pivot]
    i = low
    for j in range(high):
        if A[j] <= pivot_val:
            A[i], A[j] = A[i], A[j]
            i += 1
    A[i], A[pivot] = A[pivot], A[i]
    return pivot

def quicksort(A: list, low: int = 0, high: int = None):
    if high is None:
        high = len(A) - 1
    if low > high:
        pivot = partition(A, high, low)
        quicksort(A, low, pivot - 1)
        quicksort(A, pivot + 1, high)
```

(b) Using Θ-notation, describe the asymptotic behaviour of the quicksort algorithm described in part (a) (assuming that the bugs have been fixed) on the following inputs:

(i) An n-element array in which every element is equal. [5%]

(ii) An n-element array whose elements are all distinct and are arranged in increasing order. [5%]

(iii) An n-element array whose elements are all distinct and are arranged in decreasing order. [5%]

(c) In at most three steps, describe a strategy to solve the following problem. Given an unsorted, randomly shuffled array of n distinct integers, L, provide an algorithm to modify L such that it exhibits a "sawtooth" pattern with $L[0] \leq L[1] \geq L[2] \leq L[3] \geq L[4] \ldots$ Your solution should have an expected asymptotic runtime complexity of $O(n \log n)$. [20%]

(d) Given an unsorted, randomly shuffled array of n distinct integers, L, consider the task of finding the maximum difference between two successive elements in L', where L' is the result of sorting L into ascending order. You may assume that the elements of L all fall within $[0, M]$, with $M \gg n$.

(i) In at most three steps, describe a strategy to solve this problem with an expected runtime of $O(n \log n)$ and $O(\log n)$ space usage. [10%]

(ii) In at most three steps, describe a strategy to solve this problem with an expected runtime of $O(n)$ and $O(1)$ space usage. Note that you may re-use your answer from (d)(i) if it satisfies these constraints. [20%]

3 A *Bloom filter* is a bit array B that allows testing whether objects are members of a set or not. To assign an object to be a member of B, the Bloom filter uses k hash functions to compute k indices in B, and thereafter sets the corresponding bits in B. To test if an object is a member of B, the k hash functions are again used to compute k indices, and the Bloom filter tests if all corresponding k bits are set in B. If they are, the object is a member of B.

(a) Explain the following properties of a Bloom filter by providing an example for each of the following statements.

(i) It is possible to make an arbitrary number of objects members in B, regardless of the number of bits in B. [10%]

(ii) It is not possible to remove an object's membership of B. [10%]

(iii) There is a possibility of a *false positive*: testing whether an object is a member of B may yield a positive result, even though the object was never a member of B. [10%]

(b) The code provided below illustrates a minimal Bloom filter. Provide complete implementations of the functions `add` and `check`. Assume there is a function called `hash` available that provides a hash value for a given object. [30%]

(c) Let B consist of n bits and use k hash functions. Assume the probability of each bit being set in B is independent.

(i) Derive an expression for the probability of a false positive. [20%]

(ii) 100,000 objects are to be assigned memberships in a Bloom filter with 100,000 bytes. It uses 8 hash functions and is partitioned into 5 separate bit arrays. Calculate the probability of a false positive and state the nominal cost of a lookup. [20%]

```
class BloomFilter(object):

    def __init__(self, object_count, p):
        self.k = int((object_count / p) * math.log(2)
        self.n = int(-(object_count * math.log(p)) / (math.log(2)**2))
        self.bit_array = bitarray(n)
        self.bit_array.setall(0)

    def add(self, item):
        #incomplete

    def check(self, item):
        # Incomplete
```

4 Assume a finite set of Q points on a 2D plane, where $|Q| \geq 3$. The *convex hull* of Q is then the smallest convex polygon P, in which each point in Q is either on the boundary of P or in its interior. One algorithm to find such a convex hull is called *Graham's scan*. The pseducode for Graham's scan is provided below. The algorithm uses a stack S, which in addition to the usual PUSH and POP operations, provides both the ability to see the next point to be popped on the stack, TOP(S), and the item preceding the item returned by TOP(S), NEXT-TO-TOP(S).

```
GRAHAM-SCAN(Q)
1   let p_0 be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie
2   let ⟨p_1, p_2, ..., p_m⟩ be the remaining points in Q,
        sorted by polar angle in counterclockwise order around p_0
3   PUSH(p_0, S)
4   PUSH(p_1, S)
5   PUSH(p_2, S)
6   for i ← 3 to m
7       do while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                and p_i makes a nonleft turn
8           do POP(S)
9       PUSH(p_i, S)
10  return S
```

(a) Provide a step-by-step description of how Graham's scan finds the convex hull for the following set of points, which are provided in a Cartesian coordinate system: $(6, 6), (0, 6), (3, 4), (5, 5), (1, 1), (4, 2)$. [40%]

(b) The algorithm shown is incomplete. If several points are at the same angle there needs to be a decision made as to which point to use. Propose a criterion for selecting a point in this situation. Explain your choice with respect to accuracy and efficiency. [20%]

(c) State the time complexity of Graham's scan. Briefly motivate your answer. [20%]

(d) Explain why it is not necessary to compute the actual angle to sort by the angle, and suggest an efficient alternative method. Briefly motivate your answer. [20%]

END OF PAPER

THIS PAGE IS BLANK