Question 1. [9 MARKS]

Consider the following Python code which simulates rolling a pair of dice and counting the number of rolls until we reach n pairs. We are interested in the number of time the `random.randint` method is called, which corresponds to the number or rolls. *Do not express the complexity in $\mathcal{O}$ notation but give exact expressions.*

```
from random import randint

# Attempt to roll n pairs. Use a maximum of 10n rolls.
def countRolls(n):
    count = 0
    tries = 0
    while count < n and tries < 10*n:
        die1 = randint(1,6)      # roll the first die
        die2 = randint(1,6)      # roll the second die
        if die1 == die2:
            count += 1
        tries += 2      # count these 2 rolls even if we don't get a pair
    return count, tries
```

Part (a) [1 MARK]

Perform a best-case analysis of `countRolls`.

Part (b) [3 MARKS]

Perform a worst-case analysis of `countRolls`.

Part (c) [5 MARKS]

Perform an average-case analysis of `countRolls`. You do not need to simplify your expressions.

Question 2.

Let $a_1, a_2, \ldots, a_n$ be a sequence of real numbers, for some $n \geqslant 1$. A SUM-BOX is an ADT that stores the sequence and supports the following operations (S is a given SUM-BOX):

- PARTIAL-SUM(S, m): return $\sum_{i=1}^{m} a_i$, the partial sum from a_1 to a_m ($1 \leqslant m \leqslant n$).
- CHANGE(S, i, y): change the value of a_i to a real number y.

Design a data structure that implements SUM-BOX, using an **augmented AVL tree**. The worst-case runtime of both PARTIAL-SUM and CHANGE must be in $\mathcal{O}(\log n)$. Describe your design by answering the following questions.

(a) What is the key of each node in the AVL tree? What other attributes are stored in each node?

(b) Write the pseudo-code of your PARTIAL-SUM operation, and explain why your code works correctly and why its worst-case running time is $\mathcal{O}(\log n)$. Let $S.root$ denote the root node of the AVL tree.

(c) Describe in clear and concise English how your CHANGE operation works, and explain why it runs in $\mathcal{O}(\log n)$ time while maintaining the attributes stored in the nodes of the AVL tree.