Question 1. [8 MARKS]

Implement classes TrackRunner and RunnerRoster in subparts (a) and (b). Include class and method docstrings, but no examples are required.

Part (a) [4 MARKS]

First implement class TrackRunner to record an individual runner's information. a TrackRunner must have:

- an __init__ method
- an identity string
- an age (integer)
- a record of all this runner's race results, including the distance (in metres) and time (in seconds) for each race
- a method to add a new race result
- a method to report the best (shortest) time for each distance this runner has raced

```
class TrackRunners:
    def __init__(self, identity, age):
        self._id = identity
        self._age = age
        self.record = {}

    def add_result(self, dis, t):
        if dis in self._id:
            self.record[dis].append(t)
        else:
            self.record[dis] = [t]

    def report_best(self):
        res = {}
        for dis, t in self.record.items():
            res[dis] = min(t)
```

Part (b) [4 MARKS]

Implement class RunnerRoster to record information for an entire cohort of TrackRunners. A RunnerRoster must have:

- an init method
- a data structure to record multiple TrackRunners
- a method to add new TrackRunners
- a method to report the fastest TrackRunner, and their time, for each distance

```
class RunnerRoster:
    def __init__(self):
        self.manager = []

    def add_trackrunner(self, runner):
        self.append(runner)

    def report_fastest(self):
        res = {}
        for runner in self.manager:
            best = runner.report_best()
            for d, t in best:
                if d in res:
                    if t <= res[d][1]:
                        res[d] = [runner, t]
                else:
                    res[d] = [runner, t]

        return res
```

This page has been left intentionally (mostly) blank, in case you need space.

Question 2. [6 MARKS]

Below is a configuration of an unsolvable grid peg solitaire puzzle, where the "*" character represents pegs, and the "." character represents spaces. Allowable moves are a horizontal or vertical jump of one peg over an adjacent peg into a space, and the peg jumped over is removed.

```
* * .
* * *
* * *
```

Part (a) [3 MARKS]

Show the extensions of this configuration. Underneath each extension, show its extensions.

```
. . *        * * *
* * *        * * .
* * *        * * .

* . *        * * *
. * *        . . *
. * *        * * .

. . *        * * *
* . *        * * .
* * *        . . *
```

Part (b) [3 MARKS]

From the original configuration plus the extensions in part (a), indicate which might be the first 3 configurations considered by breadth first_solve, and which might be the first 3 configurations considered by depth-first solve. Briefly explain.

Question 3. [8 MARKS]

Read the API for classes **LinkedlistNode** and **LinkedList**, as well as the docstring for function merge. Then implement function **merge**. Note that you may only use **LinkedList** and **LinkedListNode** methods that are in the API, and that **listl** and **list2** must have all attributes correctly set when you are done.

```
def merge(listl, list2):
    """
    Merge listl and list2 by placing list2's nodes into the
    correct position in listl to preserve ordering. When
    complete listl will contain all the values from both lists,
    in order, and list2 will be empty.

    Assume listl and list2 contain comparable values in non-
    decreasing order

    @pararo LinkedList listl: ordered linked list
    @param LinkedList list2: ordered linked list
    @rtype: None

    >>> listl - LinkedList()
    >>> listl.append(l)
    >>> listl.append(3)
    >>> listl.append(5)
    >>> list2 = LinkedList()
    >>> list2.append(2)
    >>> list2.append(6)
    >>> merge(listl, list2)
    >>> print(listl.front)
    1 -> 2 -> 3 -> 5 -> 6 ->
    """
```

This page has been left intentionally (mostly) blank, in case you need space.

Question 4. [8 MARKS]

Read the docstring and example for concatenate_flat below, and then implement it.

```
def concatenate-flat(list-):
    """
    Return the concatenation, from left to right, of strings contained
    in flat (depth 1) sublists contained in list-, but no other strings.

    Assume all non-list elements of list- or its nested sub-lists are
    strings

    Oparam list list-: possibly nested sub-list to concatenate from
    @rtype: str

    >>> concatenate-flat(["five", [["four", "by"], "three"], ["two"]])
    if len(list_) == 0:
        return ""
    elif all(isinstance(sub, str) for sub in list_):
        return "".join(s for s in list_)
    else:
        return "".join(concatenate_flat(sub) for sub in list_ if isinstance(sub,
    list))
```

Question 5. [8 MARKS]

Read the declaration of class **Tree** in the API, and the docstring and examples of **pathlength_sets** below. Then implement **pathlength_sets**

```
def pathlength_sets(t):
    """
    Replace the value of each node in Tree t by a set containing all
    path lengths from that node to any leaf.  A path's length is the
    number of edges it contains.

    @param Tree t: tree to record path lengths in
    @rtype: None

    >>> t = Tree(5)
    >>> pathlength_sets(t)
    >>> print(t)
    {0}
    >>> t.children.append(Tree(17))
    >>> t.children.append(Tree(13, [Tree(11)]))
    >>> pathlength_sets(t)
    >>> print(t)
    {1, 2}
      {0}
      {1}
        {0}
    """
```

Question 6. [8 MARKS]

Read the API for class BinaryTree as well as the docstring and example for swap_even below. Then implement swapeven.

```
def swap-even(t, depth=0):
    H ii ir
    Swap left and right children of nodes at even depth.

    Recall that the root has depth 0, its children have depth 1,
    grandchildren have depth 2, and so on.

    @param BinaryTree t: tree to carry out swapping on.
    @parara int depth: distance from the root
    @rtype: None

    >>> bl = BinaryTree(1, BinaryTree(2, BinaryTree(3)))
    >>> b4 = BinaryTree(4, BinaryTree(5), bl)
    >>> print(b4)
        1



        5
    <BLANKLINE>
    >>> swap-even(b4)
    >>> print(b4)
        5
    4
        1
                3
            2
    <BLANKLINE>
    n N it
```

Question 7. [5 MARKS]

From the list, circle the big-oh expression that gives the best upper bound for each code fragment, and briefly explain your choice.

Part (a) [2 MARKS]

```
sum, i = 0, 0
while i**2 < n:
    sum = sum + i
    i += 2
```

$O(1)$ $O(\log_2 n)$ $O(\sqrt{n})$ $O(n)$ $O(n \log_2 n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$

Part (b) [2 MARKS]

```
i, j, sum = 1, 1, 0
while i < n**3:
    while j < n:
        sum = sum + i
        j += 1
    i = i + n
```

$O(1)$ $O(\log_2 n)$ $O(\sqrt{n})$ $O(n)$ $O(n \log_2 n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$

Part (c) [2 MARKS]

```
i, sum = 0, 0
while  i < 3 * n:
    if i % 2 == 0:
        j = 1
        while j < n:
            sum = sum + j
            j = j * 2
    else:
        j = 1
        while j < n:
            sum ~ sum * j
            j += (n // 5)
    i = i + 5
```

$O(1)$ $O(\log_2 n)$ $O(\sqrt{n})$ $O(n)$ $O(n \log_2 n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$

Part (d) [2 MARKS]

```
i, sum = 0, 0
while i * 2 < n:
    sum = sum + i
    i = i + 1
```

$O(1)$ $O(\log_2 n)$ $O(\sqrt{n})$ $O(n)$ $O(n \log_2 n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$