FIT1008 – Intro to Computer Science
Tutorial 1

Solution
Semester 1, 2018

Exercise 1

Solution

Part i & iv

This algorithm has two nested loops, i.e., one of the loops (referred to as the inner loop) is place inside the other (referred to as the outer loop), and, thus, it is executed for every iteration of the outer loop. For an list of length 7 the algorithm will print "Outside" 7 times, since it prints it once for each element in the list. Given this reasoning, it is clear we can easily generalise this result: for an list of length N, the algorithm will print "Outside" N times.
It will also print " Inside" 49 times, since the inner loop is executed 7 times (once per element in the list) for each iteration of the outer loop. Since there are 7 iterations of the outer loop (one per element in the list), we have 7 * 7 = 49. If you run the algorithm you will see how "Inside" is printed 7 times for each print of "Outside". Again, we can easily generalise this result: for an list of length N, the algorithm will print "Inside" N * N = N^2 times.

Part ii & iv

The value returned in count by the algorithm is 56, since it is incremented by 1 every time we print "Outside" and every time we print " Inside", which gives 49+7=56. Generalising as before, we could say that for an list of length N, the algorithm will return the value $N^2 + N$.

Part iii & iv

For the list 1, 2, 3, 4, 5, 6, 7 the value printed for sum is 224, since both inner and outer loop add all the elements in the list once (1+2+3+4+5+6+7=28) but the inner loop is called 7 times. This results in 28+28*7 = 224. For list 1, 2, 3, . . . , N, the sum of its elements is $(N^2 + N)/2$, thus, it would be $(N^2 + N)/2 + N * (N^2 + N)/2 = (N^2 +$

$N + N^3 + N^2)/2 = (N^3 + 2N^2 + N)/2$. You can check that for $N = 7$ that results in 224.
There is a clear relationship between the value of count (N^2+ N) and BigO time complexity, since this value counts the number of times each loop is performed. The algorithm is then $O(N^2)$.

Exercise 2

Solution

```
module isPalindrome(letters) {
        i <- 0
        j <- length(letters) - 1

        while (i < length(letters) / 2) {
                if (letters[i] != letters[j]) {
                        return False
                }

                i <- i + 1
                j <- j - 1
        }

        return True
}
```

Exercise 3

Solution

```
module printCommonElements(list1, list2) {
        for each element i in list1 {
                for each element j in list2 {
                        if list1[i] == list2[j] {
                                print(list1[i])
                        }
                }
        }
}
```

Can be added if tutorial 1 is too short

Exercise 4

Solution

- Heaps vs BSTs
  - heaps have the properties:
    * that all levels are full (filled left to right) except (possibly) the bottom level
    * that a parent is no smaller than either child (in a max heap or no larger in a min heap)
  - binary search trees have the properties:
    * all elements in the right subtree of a node have keys larger than that node
    * all elements in the left subtree of a node have keys smaller than that node
- stacks vs queues
  - for stacks, access is only given to the top of the stack, additions and removals both occur there leading to the elements entering first will be those first to leave
  - for queues, we remove from the front and add to the read, both ends are accessible but for different actions; hence the first to enter is the first to leave the queue.
- iteration vs recursion
  - iteration involves using a loop to repeat code; the block of code inside the loop will be repeated until the condition of the loop is no longer met
  - recursion involves using function calls to repeat code; in this case an if statement is used to represent the termination of the equivalent loop and successive iterations of the equivalent loop are handled by function calls to the same function with a change in argument.