FIT1045
Sample Exam 1

Note: this sample exam is designed for an exam of duration 3 hours, to get a realistic representation of the final exam, you could attempt two thirds of the questions within 2 hours or try sample exam 2.
We have left this sample exam as is rather than shrinking it as you could find all of the questions included valuable for your own learning

For Questions 1-6 circle only one letter for each question corresponding to the correct response.

Question 1 [1 mark]

What is printed by the following code?

```
def anon(n):
    if n % 2 == 1:
        return 0
    else:
        return 1 + anon(n//2)

print(anon(36))
```

(a) 5
(b) 4
(c) 2
(d) 1

Question 2 [1 mark]

Suppose you have the following function:

```
def secret(aList):
    val = 0
    for i in range(0, len(aList)//2):
        val += aList[i]*aList[len(aList)-i-1]
    return val
```

What is printed by the following code?

```
myList = [4, 3, 2, 1, 5]
print(secret(myList))
```

(a) 7
(b) 23
(c) 25
(d) 27

2

Blank Page for Working

Question 3 [1 mark]

The base case for Merge sort is a list of size?

(a) 0
(b) 1
(c) Both 0 and 1 <- note that while 0 could be a valid base case, it is not a useful one
(d) None of the above.

Question 4 [1 mark]

A function g(n) is said to be O(f(n)) if there exists constants k and L such that.

(a) g(n) > k*f(n) for all n < L
(b) g(n) < k*f(n) for all n > L
(c) g(n) < k*f(n) for all n < L
(d) g(n) > k*f(n) for all n > L

Question 5 [1 mark]

How many solutions are there for the 3 Queens problem?

(a) 0
(b) 1
(c) 2
(d) None of these.

Question 6 [1 mark]

An O(nlog(n)) algorithm always runs faster than an $O(n^2)$ algorithm. True or False? Why?

(a) False. For small n, constant factors may dominate the running time.
(b) True. O(nlog(n)) complexity is better than $O(n^2)$.
(c) True, but only if the input given to both algorithms is the same.
(d) False. $O(n^2)$ complexity is better than O(nlog(n)).

4

Blank Page for Working

Question 7 [2 + 2 + 2 = 6 marks]

(a) Give a definition of a process.

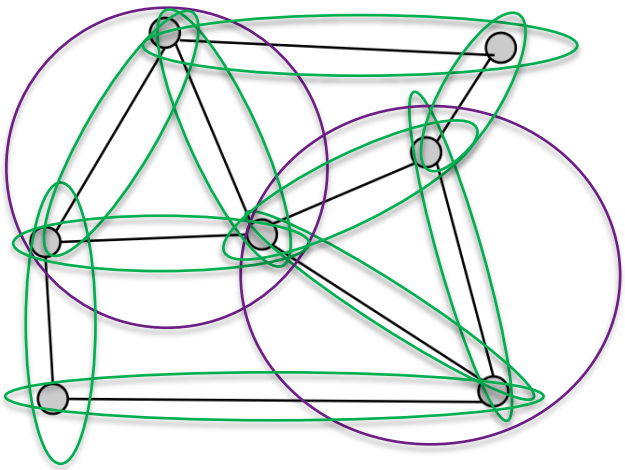
Your answer to this should cover the definition of a process (see lectures in week 12)

(b) Give a definition of a Hamiltonian cycle in a connected graph G.

You might be uncertain between Eulerian and Hamiltonian, Eulerian is where every edge is visited once, so what's Hamiltonian?

You should also include in your answer how a cycle/circuit differs from a path/trail

(c) How many cliques of each size less than 5 are there in the following graph?



There are two of size 3, 10 of size 2 which gives 12 useful cliques; cliques of size 1 are basically useless as they tell you nothing about the graph itself

6

Blank Page for Working

Question 8 [1 + 1 + 1 + 1 = 4 marks]

This question is about time complexity. For each of the given Python functions, state the time complexity in big O notation and provide a brief explanation.

(a)

```
def total_func(n):
    total = 0
    for k in range(n):
        for j in range(n-k, 0, -1):
            total + k*j
    return total
```

the inner list here is dependant on the variable of the outer loop. The easiest way to think about this is to consider what happens for the smallest value of k and the largest value of k

(b)

```
def fraction_func(n):
    fraction = 1
    for k in range(100):
        for j in range(k):
            fraction = k + j + 1/fraction
    return fraction
```

Note how the outer loop is based on a set value. Will the time taken to run this loop change as n increases?

(c)

```
def another_total_func(n):
    total = 0
    for k in range(n):
        total += k
    for k in range(10*n):
        total += num
    return total
```

there are two loops here but they are sequential so one happens after the other; how does this affect the complexity?

(d)

```
def mid_func(n):
    low = 0
    high = n
    while low <= high:
        mid = (low + high)//2
        low = mid + 1
    return mid
```

This may be familiar to you from binary search, note how every iteration low becomes one more than the half way point. How often must this happen before low exceeds high?

4

Blank Page for Working

Question 9 [5 marks]

Write a Python function, **unique**, which takes as input a sorted list of strings, and returns **True** if the all the items in the list are unique. Otherwise the function should return **False**.

To perform well in this task, you will need to write a function which accepts a list as an input and considers each item in that list. How you determine whether any given item is a duplicate depends on whether you are taking advantage of the knowledge that the list is sorted. Consider the following example:

[1,2,5,7,7,20,200,200,200,1000,1205,5325]

How would you know whether 7 was a duplicate? What would the algorithm for this be?

You should be able to terminate early if you realise any given item is a duplicate

A good implementation of this should run in linear time in the worst situation (and constant in the best)

5

Blank Page for Working

Question 10 [1 + 3 + 2 = 6 marks]

The number comparisons for a version of Quick sort, **C(n)**, is defined by the following relations.

C(0) = 1 and C(n) = C(n//2) + n + 1,

(a) Give the value of **C(3)**.

C(3) = C(3//2) + 3 + 1
C(3) = C(1) + 3 + 1
C(1) = C(0) + 1 + 1
Hence what is C(3) ?

(b) Write a recursive Python function which has as input a non-negative integer, **n**, and returns **C(n)**.

Your base case here should be where n is 0
You would be performing addition of n and 1 to the result of the recursive call

(c) State and explain the time complexity for your function.

Note how the value of n reduces by a factor of 2 with each new recursive call.
Hence how many recursive calls are there (in terms of n?) this will tell you the complexity

6

Blank Page for Working

Question 11 [5 + 5 = 10 marks]

Consider the 4 Queen problem. Suppose we use the representation of a list of numbers where the kth number represents the row which the kth Queen, $\mathbf{Q_k}$, is in, e.g., [2, 1, 3, 0] would represent the following board.

Row 0				Q_3
Row 1		Q_1		
Row 2	Q_0			
Row 3			Q_2	

(a) Write a Python function, **lastQueenOk**, which has as input a list of numbers representing the positions of the Queens on 4x4 board. This function should return **True** if no Queen is attacking the Queen represented by the last number in the list, otherwise it should return **False**.

You should have a look at your code for workshops 6 and 10 as you likely solved this then. Things to keep in mind, you can terminate early if you discover that ANY queen can attack the last queen as the answer will always be **False**

A good strategy here would be to consider each queens vs the last queen and consider whether they are in the same row, column or diagonal. The most involved of these is the diagonal checking for which you might like to consider that a diagonal attack can occur when the two queens are the same horizontal distance as the vertical distance

(b) Using the function, **lastQueenOk**, write a Python function**, isSolution**, which has as input a list of numbers representing the positions of the Queens on 4x4 board. This function should return **True** if no Queen is attacking any other Queen, otherwise it should return **False**.

This should simply be the case of running your previous function for every queen in order

10

Blank Page for Working

Question 12 [5 marks]

Suppose you have a collection of **n** items. All the items have the same weight, **w**, and you can choose at most **one** of each item. Write a Python function which is given as input, the capacity of the knapsack, **capacity**, a list (sorted in ascending order) of values, **values**, of each item, and the weight, **w**, and returns the maximum value that the knapsack can hold.

Note that due to each item having identical weight, the only difference between items is their value. This means that unlike the standard version of the knapsack problem, a greedy approach could be chosen that consistently works. How would greed be applied in this case?

5

Blank Page for Working

Question 13 [5 marks]

Write a Python function, **duplicate,** which takes two lists sorted in ascending order) as input and returns a list of items that appear in both lists.

Given that you know that BOTH lists are sorted, this suggests you can do something similar to (but not identical to) Question 9. The difference here would be that you would need two separate position variables which may change at different times.

5

Blank Page for Working

Question 14 [6 marks]

Insert the following numbers, in the order they appear, into a Heap. You are allowed to choose whether the Heap is a min-Heap or a max-Heap.

10 6 11 -6 13 2

Show the Heap after each number has been inserted. The answer should consist of **6 Heaps.**

Max-heap = no children of any given node can exceed it

Min-heap = no children of any given node can be less than the parent

Invalid heaps:
Min: [20,1,90] as 1 is smaller than 20 (its parent)
Max: [20,9,1,3,5,0,7] as 7 is bigger than its parent (1)

Review your answer with this in mind, each diagram should include one additional node and the node should be added to the end and risen appropriately

Question 15 [2 marks]

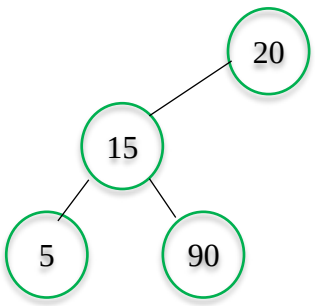
Insert the following numbers, in the order they appear, into Binary Search Tree.

10 6 11 -6 13 2

Show the Binary Search Tree after all the numbers have been inserted.

If you answer is correct you should be able to check every single node and confirm that the contents of every single item in its left subtree is smaller and right subtree is larger.

For example the tree below is NOT a valid BST since 90 is in the left subtree of 20 but 90 > 20



8

Blank Page for Working

Question 16 [2 + 6 = 8 marks]

Consider the problem of finding all the permutations of N different items.

a) Describe how a partial solution can be represented as list of numbers.

So long as the partial solution can represent the numbers that are currently included and their order your answer is correct

b) Show how you could use backtracking to solve this problem, when N = 3.

In particular, show a search tree and indicate on your diagram for each position the corresponding partial solution (represented as a list of numbers).

This really depends on your representation, but as long as you have branching points for each remaining option it should be fine. For example where 1 has been chosen already only 2 and 3 remain (giving two paths from that). You should also include the empty case where nothing has been chosen yet.

8

Blank Page for Working

Question 17 [4 + 3 + 3 = 10 marks]

(a) Explain why if a NP-complete problem could be solved in polynomial time, then P = NP.

As part of your answer you should cover the idea of polynomial reduction and the relation between NP-Complete problems and other NP problems

(b) Give **3** examples of NP problems given in lectures and state their certificates.

Any three with their certificates are valid (such as vertex cover)

(c) Describe the Halting problem.

You should describe what this problem actually entails and what we have learned from this problem about the nature of algorithms.

10

Blank Page for Working

Question 18 [1 + 1 = 2 Marks]

(a) Give an example of a sorting method that has linear time complexity is the best case.

There are many examples of this, as a hint think about sorting algorithms which perform well when the list is already sorted.

(b) Give an example of a sorting method that uses divide and conquer and has quadratic time complexity in the worst case.

This is really looking for one particular answer… you'll need to be fast to figure it out

Question 19 [5 Marks]

Write a Python function, **isVertexCover,** that checks whether a list of vertices, **vertexList,** is a vertex cover in a given graph.

The graph is represented as an adjacency matrix, **graphTable,** where **graphTable[j][k] = 1** if vertex **j** is adjacent to vertex **k**. The function, **isVertexCover**, takes **vertexList** and **graphTable** as input and returns **True** if the vertices in **vertexList** form a vertex cover of the graph represented by **graphTable**, and returns **False** otherwise.

Remember that vertex covers are a set where every single edge in the entire graph has one end (at least) in the vertex cover.
You'll need to go through every edge in the graph and check whether at least one of the vertices that edge connects are in the list of vertices that you've been given.
There are neat ways to do this and messy ways, a cute (but inefficient) way to do this is to get the sum of the adjacency matrix columns for each vertex in the list given and compare with the sum of the whole thing. Other ways might involve only going through the edges of vertices not in the set and checking where the other end goes.

7

END OF EXAM