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 encourage you only to look at these after you have considered the guidance. You will learn very little from memorising the solutions to these and the exam questions will be different. You will also learn far more by discussing your answers with another person than you will from reading our solutions. These solutions are not the best or only solution, please contact your lecturer if you are uncertain (as your solution may still be correct even if it does not match the given solution)

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

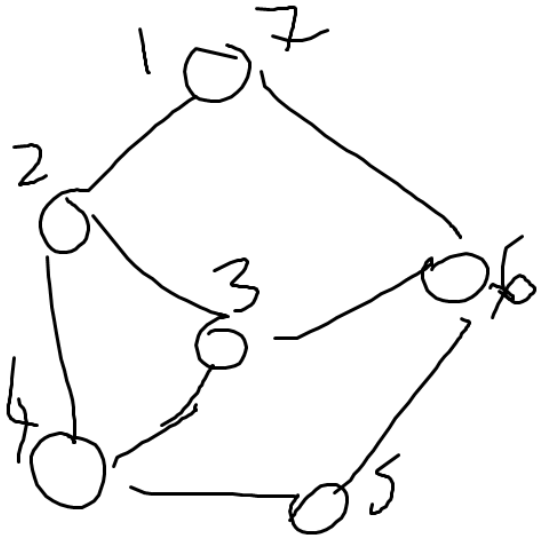
Question 7 [2 + 2 + 2 = 6 marks]

(a) Give a definition of a process.

Your answer to this should cover the definition of a process (see lectures in week 12) Sequence of steps, effectiveness, input, output, definiteness

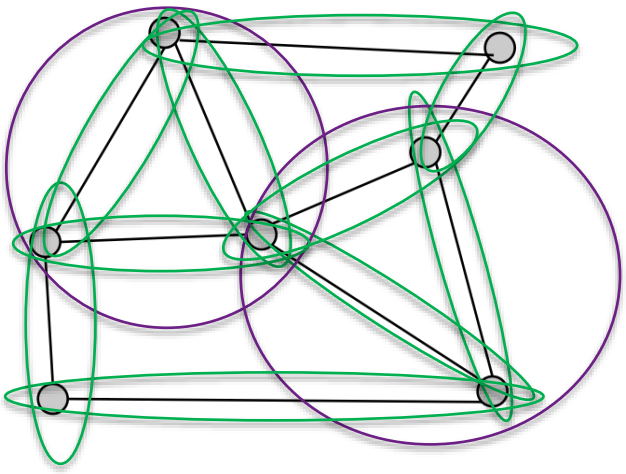
(b) Give a definition of a Hamiltonian cycle in a connected graph G.

A Hamiltonian cycle is a walk that visits every single vertex in a graph precisely once and returns back to the start, for example consider this graph



vertices here are numbered based on the order they may be visited in a hamiltonian cycle

(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 outer loop is n dependent, the inner loop is dependent on the variable of the outer loop making in n*k or O(n*n)

(b)

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

the outer loop is constant (not dependent on n) and the inner loop is dependent on the outer loop's variable, this makes it 100*k and given k is at most 100 this is at most 100*100 which is still O(1)

(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
```

the first loop is n dependent as is the second loop. They collectively give n + 10n = 11n which in big O notation is O(n) as the constants are irrelevent

(d)

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

each iteration reduces the number of remaining items by half, a constant amount of work is done each iteration hence it is simply how many times can you halve n which is logn making the complexity O(logn)

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**.

- Correct function definition (1)
- Iterates over each item in the list (1)
- Either iterates over every other item in the list (or only considers adjacent ones) (1)
- For any pair of strings checks equality (1)
- returns true where unique (0.5)
- returns false where not unique (0.5)

one **possible** answer

```
def unique(aList):
        pos = 0
        while pos < len(aList):
                if (pos+1) < len(aList):
                        if aList[pos] == aList[pos+1]:
                                return false
        return true
```

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
C(0) = 1
C(3) = ((1) + 1 + 1) + 3 + 1 = 7

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

Rubric:

- Correct base case (0.5)
- Correctly returns results (0.5)
- Correct recursive call (1)
- Correctly adds to determine result for particular value of n (0.5)
- Correct function definition (0.5)

Most likely implementation:

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

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

Log(n)
Each call to C(n) results in a call with half the input. The input can be halved as many times as what power of 2 n is which is logn. The amount of work being done with each call is constant (addition)

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**.

Possible implementation

```
def lastQueenOk(partial):
        Queen = partial[-1] #the last queen in the solution so far
        for colNum in range(len(partial)-1): #-1 because want to not compare with self
                if sameRow(colNum,-1,partial):
                        return false
                if sameDiag(colNum,-1,partial)
                        return false
        return true

def sameRow(col1,col2,partial):
        return partial[col1] == partial[col2]

def sameDiag(col1,col2,partial):
        return abs(col1-col2)==abs(partial[col1]-partial[col2])
```

- Considers each prior queen (1)
- Compares against the very last queen (0.5)
- Considers whether they share a row (1)
- Considers whether they share a column (1)
- Correct function definition (0.5)
- Correctly returns results true where none can attack last queen (0.5)
- Correctly returns false where any can attack last queen (0.5)

10

(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**.

Possible implementation:

```
def isSolution(board):
   for length in range(len(board)):
      if not lastQueenOk(board[:length-1]):
         return False
   return True
```

- Correct function definition (0.5)
- Correctly iterates over all subsets of the list of numbers (2)
- Correctly runs their lastQueenOk function (1)
- If at any point this gives false isSolution also gives false (1)
- If all combintations give true then isSolution gives true (0.5)

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.

Possible implementation:

```
def knapsack(values,capacity,w):
    remain = capacity
    current = len(values)-1
    totalValue = 0
    while remain > 0 and current >= 0:
        if (remain - w) >= 0:
            totalValue+=values[current]
            remain -= w
            current-=1
    return totalValue
```

- Correct function definition (0.5)
- Starts with the largest value (0.5)
- Ensures choosing this does not exceed capacity (1)
- Somehow updates capacity remaining or used up so far (0.5)
- Updates cost (0.5)
- Restricts to only one of each (0.5)
- Handles the case of having used up all the items (0.5)
- Moves along to choose the next biggest value (0.5)
- Returns the result (0.5)

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.

Possible implementation:

```
def duplicate(listA,listB):
    aPos = 0
    bPos = 0
    dupes = []
    while aPos < len(listA) and bPos < len(listB):
        aVal = listA[aPos]
        bVal = listB[bPos]
        if aVal == bVal:
            dupes.append(aVal)
            aPos+=1
            bPos+=1
        elif aVal < bVal:
            aPos +=1
        else:
            bPos +=1
    return dupes
```

- Correct function definition (0.5)
- Considers each element of list 1 (1)
- Considers each element of list 2 (1)
- Compares elements in list one against elements in list 2 (1)
- Where items match these are added to a list for later returning (1)
- Returns results (0.5)

5

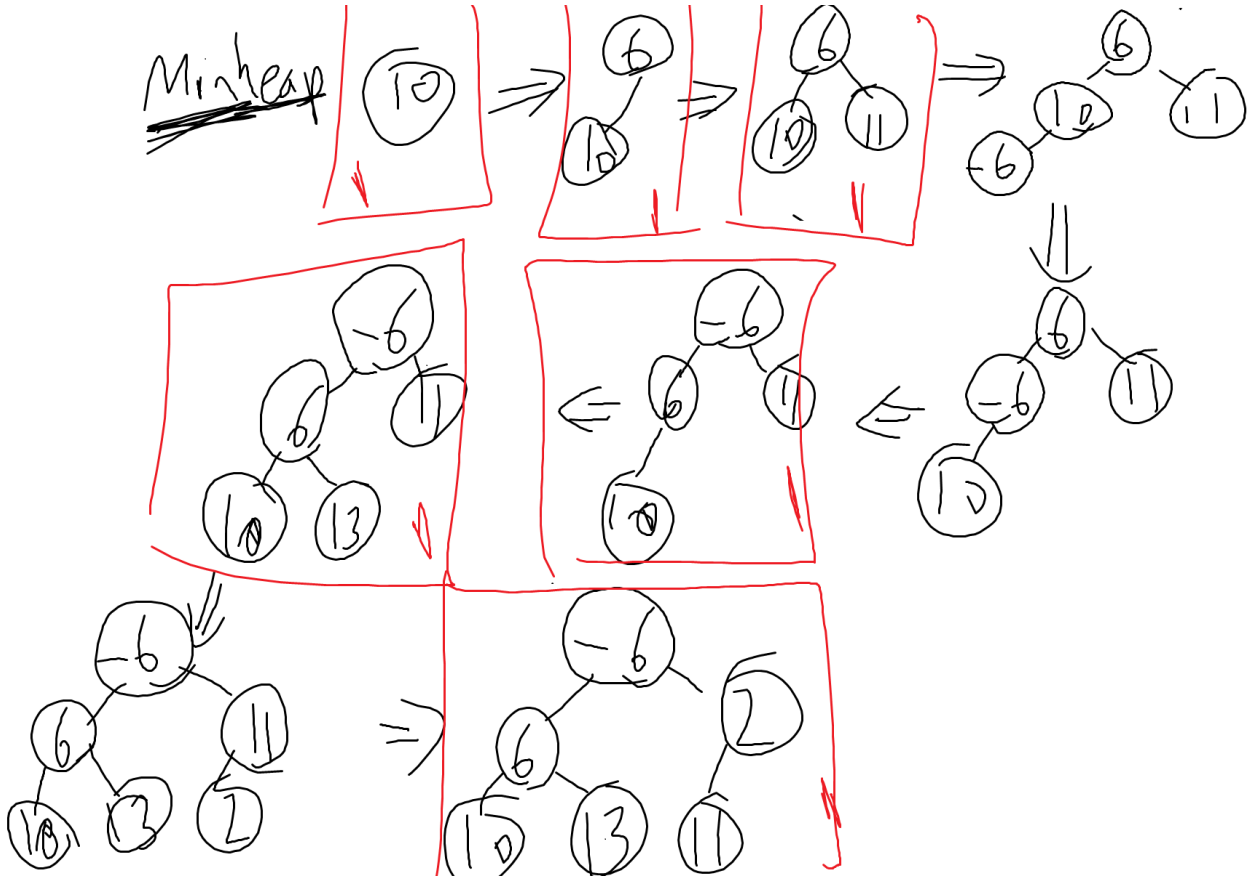
Blank Page for Working

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.**



In red squares are what would be assigned marks, consequential marks awarded based on whether correct choices are made afterwards

Question 15 [2 marks]

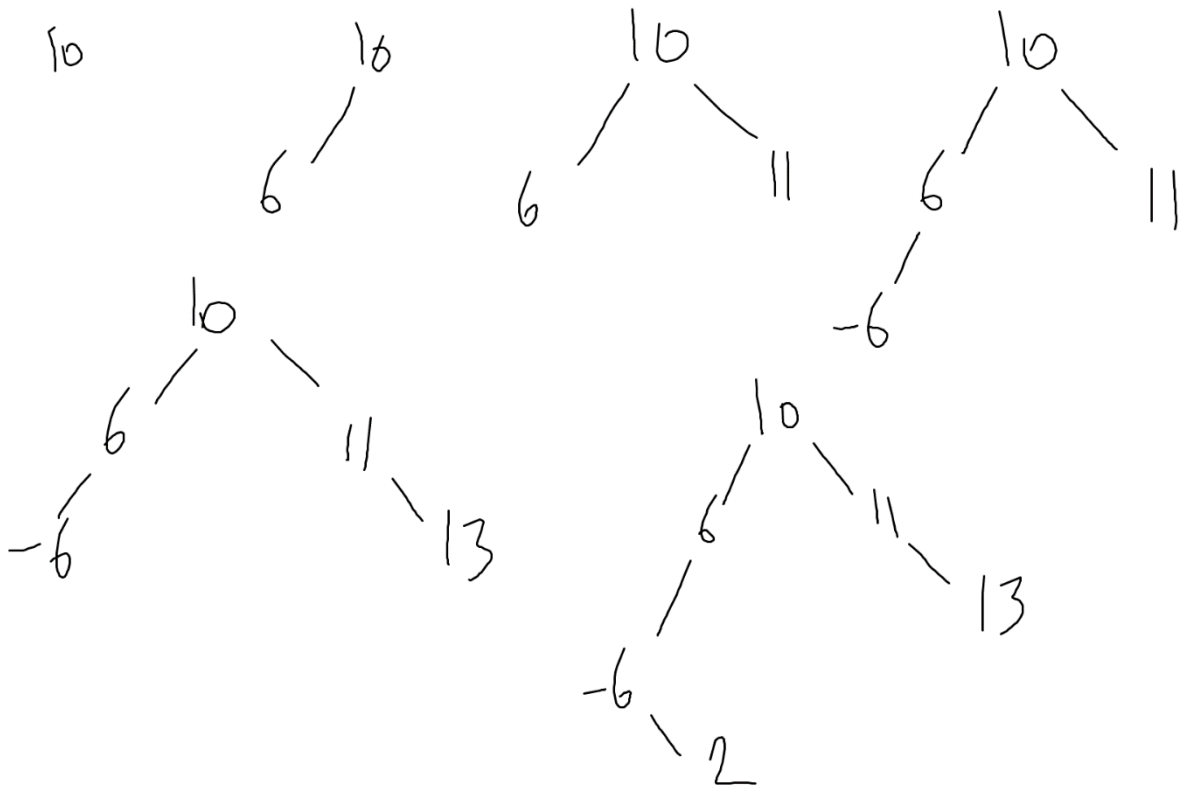
Insert the following numbers, in the order they appear, into Binary Search Tree.

10 6 11 -6 13 2

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

See next page:

8



Blank Page for Working

Question 16 [2 + 6 = 8 marks]

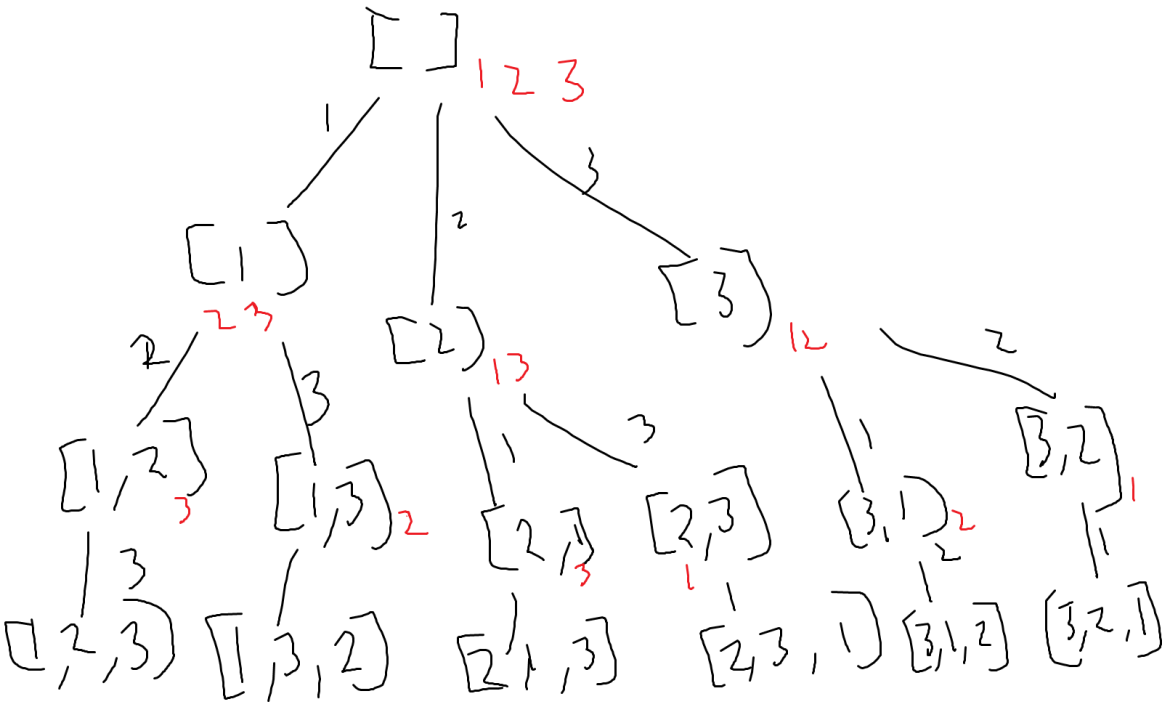
Consider the problem of finding all the permutations of N different items.

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

An example would be [1,5] for N of 5, here we currently have selected 1 first and then 5 and the number represents which item it is

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



8

Blank Page for Working

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

(a) Explain the difference between the P and NP classes of problems; What would be the consequences of discovering for certain that these are different?

NP Problems are those which can have certificates (possible solutions) **checked for correctness** in polynomial time; P problems are those who are known to have algorithms to **solve** them in Polynomial time. If we knew for certain that these were different classes, then we would know that there are some problems which we will never be able to find efficient ways to solve (and we would know to stop trying to find P algorithms for them) – we would also have more certainty about the security of certain kinds of encryption etc. P=/=NP is what a majority of computer scientists believe to be the case.

(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)

Vertex cover – set of vertices
Traveling salesperson – ordered set of vertices
3 colouring problem – a set of colours for each vertex
+ various others

(c) Suggest a problem for which no algorithm exists, explaining why this is the case.

Consider the process in lectures for getting the decimal representation of a real number. The issue can arise that some do not have finite representations (consider 1/3); this remains true regardless of the number system (eg. 0.1 in base 2 has a non-finite representation). No algorithm can be produced to yield an infinite sequence as by definite it could never terminate and so must not be an algorithm

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.

Insertion sort is linear for a sorted list, counting sort / bin sort is linear for bounded data sets (ex. Data in the range of 1 to 100).

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

We only showed you merge and quicksort. Merge sort is always nlogn, whereas quicksort can be quadratic with a poor choice of pivot

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** and **len(graphTable)** returns the number of vertices in the graph. 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.

Possible implementation:

```
def isVertexCover(graph,cover):
    for source in range(len(graph)):
        for dest in range(len(graph[source])):
                #graph[source][dest] is an edge if is 1
                if graph[source][dest]==1:
                        found = false
                        for vertex in cover:
                                if source==vertex or dest==vertex:
                                        found = true
                        if found==false:
                                return false
    return true
```

- considers every vertex in the graph (0.5)
- considers every other vertex in the graph (0.5)
- using this it finds each edge in the graph (0.5)
- considers every vertex in the vertex set (1)
- these vertices are compared against the chosen edge to determine if at least one end of the edge is in the vertex set (1)
- if neither end is it returns false (1)
- if ALL edges have at least one end in the vertex cover it returns true (0.5)

7

END OF EXAM