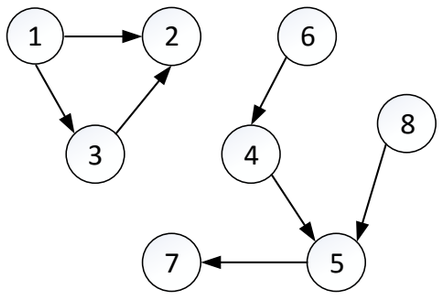
PART 1 - MULTIPLE CHOICE

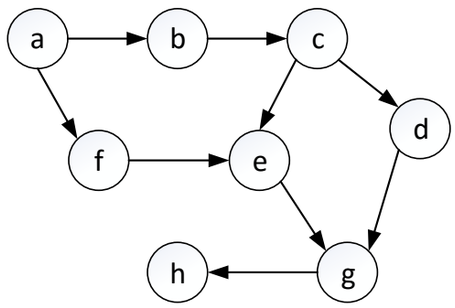
Questions

1. (2 marks) Is this graph:



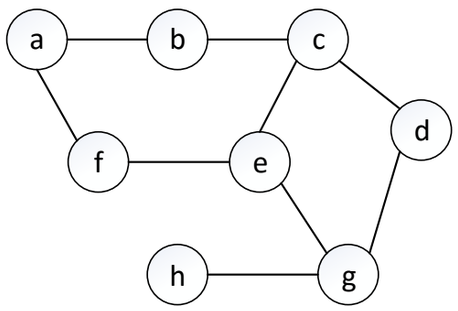
A. Weighted and connected
B. Weighted and not connected
C. Unweighted and connected
D. Unweighted and not connected

2. (2 marks) Is this graph:



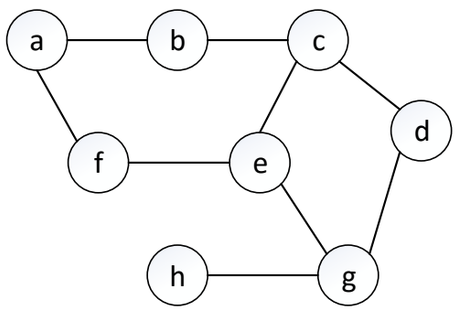
A. Directed and cyclical
B. Directed and acyclical
C. Undirected and cyclical
D. Undirected and acyclical

3. (3 marks) Which one of these sequences of vertices **was not** generated from a DFS (Depth First Search) traversal of the graph on the left? Assume ties are broken by the alphabetical order of the matrices.



A. b, a, f, e, c, d, g, h
B. a, b, c, d, g, e, f, h
C. d, c, b, a, f, e, g, h
D. e, c, d, g, h, f, a, b

4. (3 marks) Which one of these sequences of vertices **was** generated from a BFS (Breadth First Search) traversal of the graph on the left? Assume ties are broken by the alphabetical order of the matrices.



A. c, b, e, d, a, f, g, h
B. a, b, f, c, d, e, g, h
C. e, c, f, g, b, d, a, h
D. h, g, d, e, c, b, f, a

5. (2 marks) Which is the best (i.e. fastest) complexity for an algorithm:

   A. $O(n \log_2 n)$

   B. $O(n)$

   C. $O(n^2)$

   D. $O(2^n)$

6. (3 marks) What is the **exact cost** of this algorithm as a function of n:

```
myloops (n)
        for i=0 to n-1 do
                for j=i downto 0 do
                        // the basic operation is here
```

   A. $n(n-1)/2$

   B. n^2

   C. $n^2/2$

   D. $n(n+1)/2$

7. (4 marks) Assuming that the basic operation in this algorithm is the number of function calls of mysum, what is the **asymptotic cost** of this algorithm as a function of n:

```
mysum (n)
        if n=0 or n=1       return n
        else    return mysum(n div 2) + mysum(n div 2)
```

   A. $O(\log_2 n)$

   B. $O(n)$

   C. $O(n^2)$

   D. $O(2^n)$

8. (2 marks) Do these two algorithms cost the same asymptotically:

```
sum1(n)
        result = 0
        for i=1 to n do
                result = result + i
        return result
```

```
sum2(n)
        if n=0          return 0
        else    return sum2(n-1) + n
```

   A. Yes

   B. No

PART 2 - SHORT ANSWERS

9. (2 marks) You have been asked to improve an algorithm which manipulates a high volume of data and which is running much too slowly. You analysed this algorithm mathematically and proved that its complexity was $O(n^3)$. You then made some radical changes to it, analyzed it again and concluded that its new complexity was $O(n^2)$. Is this good news? explain your answer.

▶ Yes it is good news: when n is large enough, which is probably the case here since we are dealing with a high volume of data, $O(n^2)$ is significantly faster than $O(n^3)$.

10. (3 marks) You have continued your mathematical analysis of the algorithm you revised in the previous question and have now concluded that its complexity is $\Theta(n^2)$. Is this good news? explain your answer.

▶ No this is not good news: if the algorithm is $\Theta(n^2)$ this means that in addition to having a quadratic upper bound, it has a quadratic lower bound. So the asymptotic performance of this particular algorithm cannot be improved in a significant way.

11. (3 marks) Can you design an $O(n^2)$ algorithm to generate all the possible bit strings of length n? Explain your answer

▶ No: there are 2^n bit strings of length n, so the size of the solution to this problem is $O(2^n)$. It is not possible to have an $O(n^2)$ algorithm which generates 2^n solutions.

12. Here is an algorithm to find the position of a substring in a string.

```
// This function returns the positon of "sub" in "string"
// or the size of string if sub is not a substring of string
substring (sub, string)
        n=length(string)
        m=length(sub)
        for i=0 to n-m-1 do
                matchedchars=0
                for j=0 to m-1 do
                        if sub[j]=string[i+j]       matchedchars = matchedchars+1
                if matchedchars=m       return i
        return n
```

a) (1 mark) What is the size of this algorithm? (i.e. what variables with the cost be a function of?)

▶ n and m

b) (1 mark) Circle the basic operation of this algorithm which will be used to compute its complexity

▶ See highlight

c) (2 marks) What is the best case cost of this algorithm and when does it happen?

▶ The best case scenario is when string starts with sub. This will be detected at i=0, and right after the inner loop has executed once. The best case cost will be 1×m = m

d) (2 marks) What is the worst case cost of this algorithm and when does it happen?

▶ The worst case scenario is when sub is not in the string but the entire sub must be checked at each iteration of the outer loop to discover this. For example:

- string = "0…01" (with n-1 0's)
- sub = "0…02" (with m-1 0's)

Each inner loop will be fully executed m times
The outer loop will also be fully executed n-m time
So the cost is (n-m)×m

Another worst case scenario with the same cost would be

- string = "0…01" (with n-1 0's)
- sub = "0…01" (with m-1 0's)

e) (2 marks) What is the average case cost of this algorithm?

▶ (n-m)×m/2 if the sub is found in the middle of string

f) (5 marks) Rewrite this algorithm to improve its average case complexity

```
substring (sub, string)
        n=length(string)
        m=length(sub)
        for i=0 to n-m-1 do
                matchedchars=0
// Replacement for for loop:
                while (matchedchars <m and
                                sub[matchedchars]=string[i+ matchedchars]) do
                        matchedchars = matchedchars+1
                if matchedchars=m       return i
        return n
```

g) (3 marks) Explain why your new algorithm is faster

▶ At each starting point i in string, this algorithm stops comparing sub with string as soon as they stop matching, instead of comparing the entire sub with the entire string[i..i+m-10 size m