Question 1

```
public static void DFS(Vertex a) {

    /* Complete the code */
    System.out.print(a.getID() + " ");

    a.setMarked();//visit the first vertex and mark it as visited
    TreeMap<Vertex, Integer> adj = a.getNeighbours();//create a treemap of all the vertices
    Iterator<Vertex> i = adj.keySet().iterator();//iterate through the vertices
    while(i.hasNext()){//while there are vertices remaining
        Vertex next = i.next();
        if(!next.isMarked()){//if the next vertex hasn't been visited
            DFS(next);
        }
    }
```

Question 2

i. Preferred data type is a string where we assume that the first 3 characters are the shortened month values, followed by 2 characters that denote the day of the month. For example, the 12th of January is represented as "Jan12", and the 2nd of February is "Feb02". This is because the substring() method can be used to split the string into the month component and day component as it is faster to check if there is a birthday with the same month, and then check whether it's the same day.
Problem: check that 2 people have the same birthday; decision problem
Input: string array of *n* birthdays
Output: true if there are two people with a birthday on the same day, false otherwise.

ii. Pseudocode:
Boolean SameBirthday(string[] b)
   a. Int x = number of elements in b
   b. If(x<=365 || LeapYear && b<=366){ //if there are more elements than days in the year
      i. Return true;
   c. }
   d. For loop iterates through array b{
      i. If i.substring(0,2) == j.substring //where i is the first element, j is the element that gets incremented
         1. If i.substring(3,5) == j.substring(3,5)
            a. Return true
   e. Return false

iii. The worst case occurs if for the *n* values in the array of birthdays, various items in the list have the same month, but different days. This results in the algorithm stopping at each set of values with the same month and iterating through the different days, then moving on to the next month to repeat the process. Therefore, the big-O notation of this algorithm is O(n).