Assignment 3: Counting, algorithms, complexity and graphs

Name: Trac Duc Anh Luong
Student ID: 103488117

Counting

1.
a)
i)
- All 5 tasks are done: 5! = 120
- 4 out of 5 tasks are done: P(5, 4) = 120
- 3 out of 5 tasks are done: P(5, 3) = 60
- 2 out of 5 tasks are done: P(5, 2) = 20
- 1 out of 5 tasks is done: P(5, 1) = 5
- 0 out of 5 tasks is done: P(5, 0) = 1
  => Number of patterns = 120 + 120 + 60 + 20 + 5 + 1 = **326**

ii) Patterns containing only 3 types: P(5, 3) = **60**

iii) At least 4 types => 4 or 5 types
Start with playing game => 3 choices left
- Pattern with 4 types, starting with game: 1 * P(4, 3) = 24
- Pattern with 5 types, starting with game: 4! = 24
  => Number of patterns = 24 + 24 = **48**

b)
i)
1st case:
- Every ingredient can be used or not => 2 options for each ingredient = 2^6 ways
- You cannot use 0 ingredient => - 1 way

Ways to combine = 2^6 - 1 = **63**
2nd case: If every ingredient has to be used up, the way to combine is C(6, 6) = **1**

ii) 3 ingredients only = C(6, 3) = **20**

c)
i)

- 12 figures left, 3 figures go in the 1st location: C(12, 3) = 220
- 9 figures left, 3 figures go in the 2nd location: C(9, 3) = 84
- 6 figures left, 3 figures go in the 3rd location: C(6, 3) = 20
- 3 figures left, 3 figures go in the 4th location: C(3, 3) = 1
  => Ways to arrange figures = 220 * 84 * 20 * 1 = **369600**

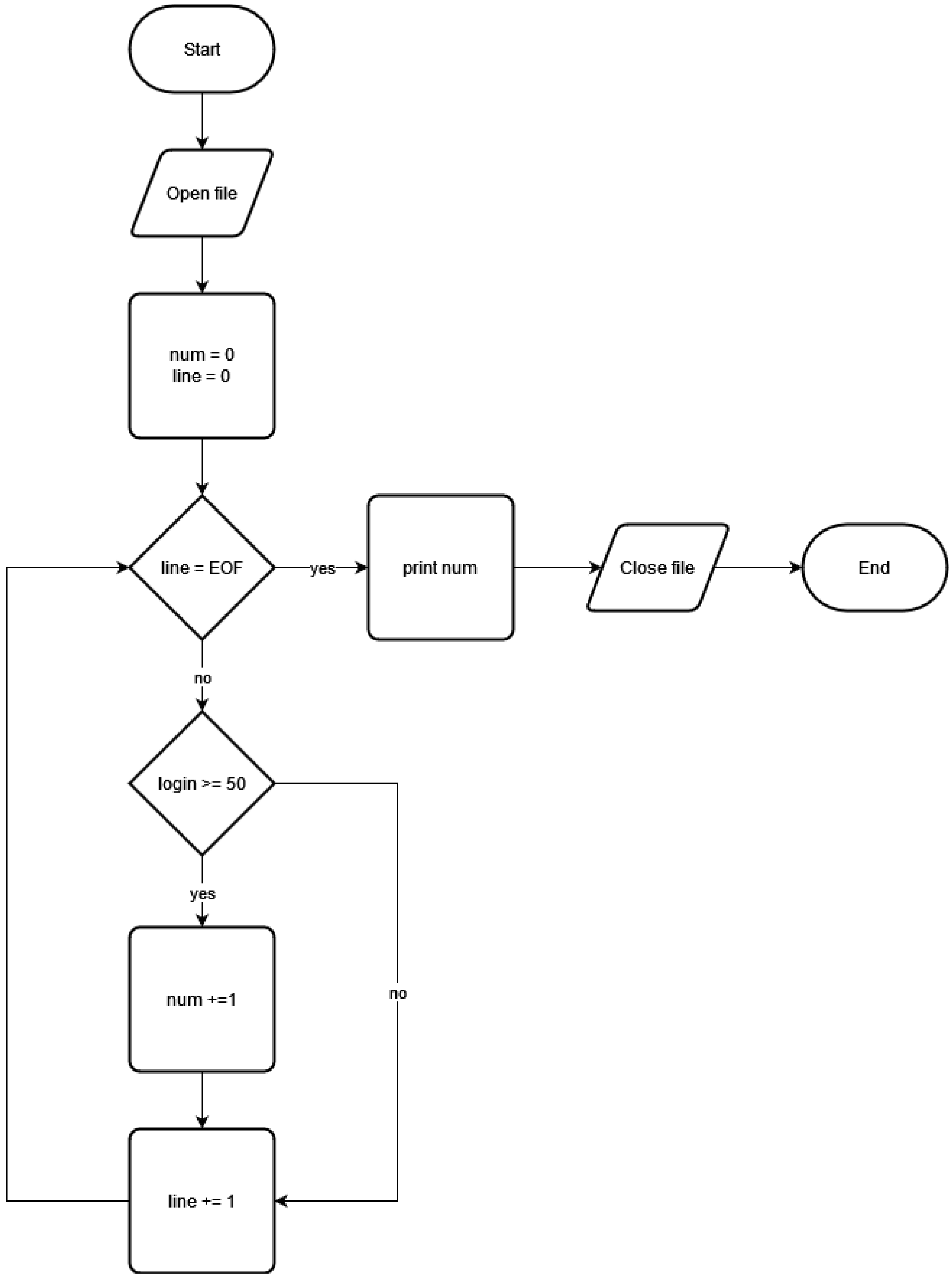
ii)

- 12 figures: 12!
- 3 figures at each of the 4 locations: 3! * 3! * 3! *3!
  => Ways to arrange figures = 12! / (3! * 3! * 3! *3!) = **369600**

d)
For this problem, we need to use the Pigeonhole principle. The principle states that if n items are put into m containers, with n > m, then at least one container must contain more than 1 item. In this case, the available days are the containers. Each container has 2 consecutive days as we need 1 day to study and 1 day to break. Therefore, we have 14 / 2 = 7 pigeon holes, or m = 7. 10 study sessions will be the items that we put into the containers, therefore, n = 10. Using the formula, we have:
n = km + 1
⇔ 10 = 7k + 1
⇔ k = 9/7
⇔ At least 1 of the sets will contain k + 1 = 16/7 ≈ 2.29 objects
In conclusion, if there are 10 study sessions planned, we will study on consecutive days at least once in the next fortnight.

Algorithms

2.
a. (next page)



b. The complexity of my program is O(n) as runtime will depend on the integer value "line". As "line" increases, the runtime will also increase. This is a linear runtime.

3.

a. I will recommend algorithm X, which has a worst-case runtime complexity of O(n^2) as it is faster and has less running time than algorithm Y, which has a worst-case runtime of O(n^3), regardless of your n inputs.

b. My advice will be based on the input size of n.
For example, we have 2 algorithms of f1(n) = n^2 and f2(n) = n + 1000.
We can see that:

- the complexity of f1(n) is Θ(n^2)
- the complexity of f2(n) is Θ(n)

When we input a small value of n = 5 into the 2 algorithms, f1(5) = 25 and f2(5) = 1005. In this case, Θ(n^2) is faster than Θ(n) (as 25 < 1005). From this example, we can see that for each n < 32.1267292, Θ(n^2) is faster than Θ(n). However, as n → ∞, Θ(n) will be more efficient as it is faster than Θ(n^2).
To conclude, if the value of n is small, I will advise them to choose algorithm X, which has an average-case time complexity of Θ(n^2). If the value of n → ∞, I will advise them to choose algorithm Y, which has an average-case time complexity of Θ(n).

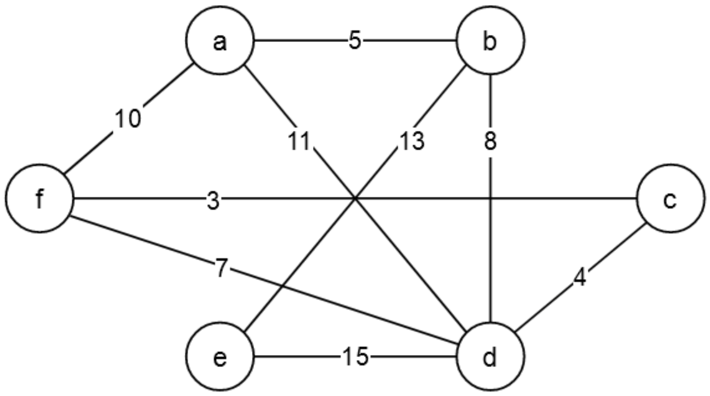
4.

```
function sum_odd(n)
    if n <= 0 then
        return 0
    else if binary_ones(n) % 2 = 1 then
        return n + sum_odd(n - 1)
    else then
        return sum_odd(n -1)
    end if
end
```

Graphs and trees

5.

a.



b.

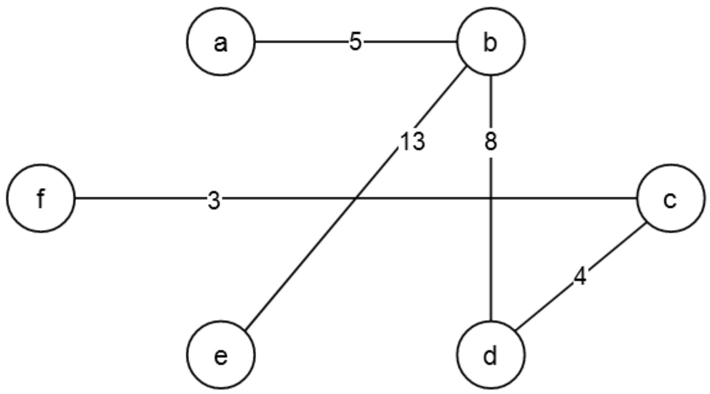
i.

After reordering the edges by weight:

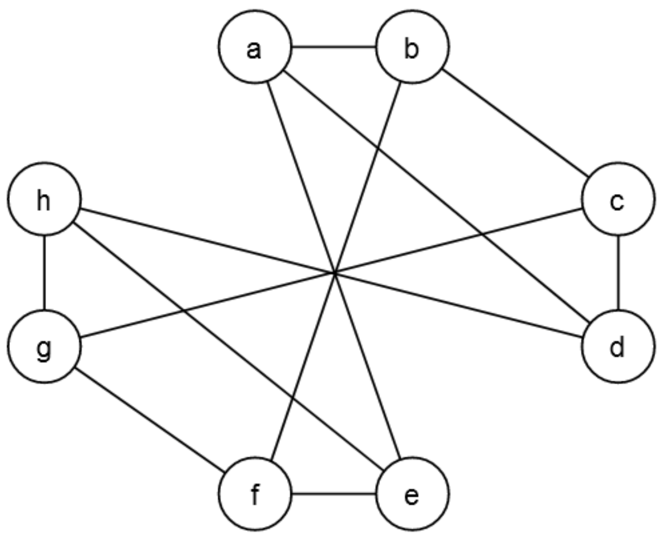
E = [{c, f}, {c, d}, {a, b}, {d, f}, {b, d}, {a, f}, {a, d}, {b, e}, {d, e}]

E	keep/discard
{c, f}	keep
{c, d}	keep
{a, b}	keep
{d, f}	discard
{b, d}	keep
{a, f}	discard
{a, d}	discard
{b, e}	keep
{d, e}	discard

ii



6.



This is not an Eulerian cycle because every vertex has 3 degrees (odd).
This is a Hamiltonian cycle because each vertex is visited exactly once, apart from the starting and ending vertex which is the same.
Example:
Cycle order of the graph, starting from “a”: a → b → c → d → h → g → f → e → a