Problem 1 (10 pts)

Arrange the following in increasing order of the asymptotic growth rate. Explain your answer. All logs are in base 2.

$$
\begin{aligned}
&(a) \quad f_1(n) = 2^{2\sqrt{\log n}} \\
&(b) \quad f_2(n) = 2^{\log(n^2)} \\
&(c) \quad f_3(n) = \frac{n(\log\log n)^{99}}{(\log n)^{99}} \\
&(d) \quad f_4(n) = (n!)^2 \\
&(e) \quad f_5(n) = 4^{(2^{\log n})} \\
&(f) \quad f_6(n) = n^{n\log n} \\
&(g) \quad f_7(n) = \log(n!) \\
&(h) \quad f_8(n) = 2^{\frac{\log n}{\log\log n}} \\
&(i) \quad f_9(n) = 2^{\log n - \log\log n} \\
&(j) \quad f_{10}(n) = (4^2)^{\log n}
\end{aligned}
$$

Solution:

We begin by simplifying some of the functions and then comparing their growth rates.

Step 1. Simplify the Functions:

- For $f_2(n) = 2^{\log(n^2)}$:
$$2^{\log(n^2)} = n^2.$$
- For $f_{10}(n) = (4^2)^{\log n}$:
$$(4^2)^{\log n} = 16^{\log n} = n^{\log 16} = n^4 \quad (\text{since } \log_2 16 = 4).$$
- For $f_5(n) = 4^{(2^{\log n})}$:
$$2^{\log n} = n \quad \Longrightarrow \quad f_5(n) = 4^n = 2^{2n}.$$
- For $f_9(n) = 2^{\log n - \log\log n}$:
$$2^{\log n - \log\log n} = 2^{\log\left(\frac{n}{\log n}\right)} = \frac{n}{\log n}.$$
- For $f_7(n) = \log(n!)$: Using Stirling's approximation, $\log(n!) \sim n\log n$.

- For $f_4(n) = (n!)^2$: Since $n! \sim \left(\frac{n}{e}\right)^n$, we have $(n!)^2$ growing super-exponentially.
- For $f_6(n) = n^{n \log n}$: Taking logarithms yields

$$\log f_6(n) = n \log n \cdot \log n = n(\log n)^2,$$

which grows even faster than $(n!)^2$.

Step 2. Compare the Growth Rates:

1. $\boldsymbol{f_1(n) = 2^{2\sqrt{\log n}}}$: The exponent $2\sqrt{\log n}$ increases very slowly as $n \to \infty$.
2. $\boldsymbol{f_8(n) = 2^{\frac{\log n}{\log \log n}}}$: Its exponent $\frac{\log n}{\log \log n}$ grows faster than $2\sqrt{\log n}$ for large n.
3. $\boldsymbol{f_3(n) = \frac{n\,(\log \log n)^{99}}{(\log n)^{99}}}$: This function is nearly linear (proportional to n) but is reduced by a factor of $(\log n)^{99}$, making it grow slower than a linear function.
4. $\boldsymbol{f_9(n) = \frac{n}{\log n}}$: This is almost linear but only divided by a single $\log n$ factor; hence, it grows faster than $f_3(n)$.
5. $\boldsymbol{f_7(n) = \log(n!)}$: With Stirling's approximation, we have $\log(n!) \sim n \log n$, which grows faster than $n/\log n$.
6. $\boldsymbol{f_2(n) = n^2}$: A polynomial of degree 2; thus, it grows faster than $n \log n$.
7. $\boldsymbol{f_{10}(n) = n^4}$: A polynomial of degree 4, so it grows faster than n^2.
8. $\boldsymbol{f_5(n) = 4^n}$: An exponential function; exponentials eventually outgrow any polynomial.
9. $\boldsymbol{f_4(n) = (n!)^2}$: The factorial grows super-exponentially, so $(n!)^2$ grows even faster than 4^n.
10. $\boldsymbol{f_6(n) = n^{n \log n}}$: Rewriting $f_6(n)$ as $\exp\big(n(\log n)^2\big)$ shows that it has an enormous growth rate, exceeding even that of $(n!)^2$.

Step 3. Final Ordering:

Arranging the functions in increasing order of their asymptotic growth rates gives:

$$\boxed{f_1(n) < f_8(n) < f_3(n) < f_9(n) < f_7(n) < f_2(n) < f_{10}(n) < f_5(n) < f_4(n) < f_6(n).}$$

Problem 2 (10 pts)

You have a collection of books and need to arrange them on shelves by color. Each shelf can hold only books of the same color, but you don't know the colors directly. Instead, you are given pairs of books known to be the same color. This relation follows an equivalent relation (reflexive, symmetric, and transitive). Your task is to write a code to determine the minimum number of shelves needed, ensuring that no two books of different colors share a shelf.

Submission: Please submit your source code file to Canvas, along with a screenshot attached in this document showing the output from running the three examples. The hard-coded program will not be accepted.

Example 1:
Input:

```
books = ['u', 'v', 'w', 'x']
pairs = [('u', 'v'), ('v', 'w')]
```

Output:

```
print(min_shelves(books, pairs))  # Output: 2
```

Explanation: Books u, v, and w are all connected (same color). Book x is not connected to any other book and thus needs its own shelf. Result: 2 shelves.

Example 2:
Input:

```
books = ['a', 'b', 'c', 'd', 'e', 'f']
pairs = [('a', 'b'), ('b', 'c'), ('d', 'e')]
```

Output:

```
print(min_shelves(books, pairs))  # Output: 3
```

Explanation: Books a, b, and c are of the same color. Books d and e are of the same color. Book f is alone and needs its own shelf. Result: 3 shelves.

Example 3:
Input:

```
books = ['x', 'y', 'z']
pairs = [('x', 'y')]
```

Output:

```
print(min_shelves(books, pairs))  # Output: 2
```

Explanation: Books x and y are connected (same color), but book z is not connected to any other book. Hence, we need two shelves. Result: 2 shelves.

Hint: This problem can be abstracted as finding the connected components in a graph. Each book is a node, and each pair of books that are the same color represents an edge between two nodes. Your goal is to identify how many disconnected groups (connected components) of books there are. Each connected component will correspond to a shelf. You can use Depth-First Search (DFS) or Breadth-First Search (BFS) to find these connected components efficiently.

Problem 3 (30 pts)

You have a 5-gallon jug and a 3-gallon jug, both initially empty. Your goal is to have exactly **4 gallons** of water in the 5-gallon jug, and **0 gallons** in the 3-gallon jug.

You are allowed the following operations:

1. **Fill** any of the jugs completely.
2. **Pour** water from one jug into the other until the first jug is empty or the second jug is full.
3. **Empty** the contents of a jug.

Objective:

1. (5 pts) Describe a method to reach the goal state (4 gallons in the 5-gallon jug and 0 gallons in the 3-gallon jug). You can solve this puzzle step by step in any way that works, without needing to apply a general algorithm. Assume (m, n) denote m gallons in the 5-gallon jug and n gallons in the 3-gallon jug.

   Example

   (a) Start with empty jugs: (0, 0)
   (b) Fill jug 3: (0, 3)
   (c) Pour from jug 3 to jug 5: (3, 0)
   (d) Fill jug 3: (3, 3)
   (e) ...
   (f) Pour from jug 3 to jug 5: (4, 0) **(Goal reached)**

2. (25 pts)Now we generalize the problem to two jugs of any size, and we hope to find steps to reach a target volume in one jug, e.g. two jugs of 7 and 5 gallons for a target of 6 gallons. Write a program to automate the solution to this problem. The program should find the minimum number of steps and print each step taken to reach the goal state.

   Function definition:

```
reach_target_volumn(size_A, size_B, target_vol)
```

Output:

```
Steps for size_A=7, size_B=5, target_vol=6:
(0, 0)
(7, 0)
(2, 5)
...
(6, 5)
```

If there is no solution, return: "unreachable"

Submission: Please submit your source code file to Canvas and attach a screenshot in this document showing the output when running your code with these two test cases.

```
size_A=6, size_B=4, taget_vol=1;
size_A=11, size_B=5, taget_vol=8;
```

Hint: Treat each state of the jugs as a vertex (m, n), where m and n are the amounts of water in the jugs. Operations (filling, pouring, emptying) are edges between states. Use Breadth-First Search (BFS) to find the shortest path from the initial state $(0, 0)$ to the target state (x, y) or return "unreachable".