1 Question 1

There are following cases for the potential value of result[i][j] during evaluation:

\[
result[i][j] = \max ( \\
1. \text{result}[i-1][j] + \text{indel} \\
2. \text{result}[i][j-1] + \text{indel} \\
3. \text{if } A[i] \neq B[j] \\
\quad \text{result}[i-1][j-1] + \text{mismatch} \\
\quad \text{result}[i-1][j-1] + \text{ext_match} \\
\quad \text{else if } A[i] == B[j] \\
\quad \text{result}[i-1][j-1] + \text{match} \\
) \\
\]

\text{match}(A, B)\{
 m = A.length \\
 n = B.length \\
 let c[0...m, 0...n] be new tables \\
 c[0,0] = 0; \\
 for i = 1 to m \\
 \quad c[i,0] = c[i-1,0] + \text{indel} \\
 for i = 1 to n \\
 \quad c[0,i] = c[0,i-1] + \text{indel} //initialize
for i = 1 to m
    for j = 1 to n
        if A[i] == B[j]
            { 
                    c[i,j] = c[i-1,j-1] + ext_match  //previous match is the same letter
                else
                    c[i,j] = c[i-1,j-1] + match
            }
        else
            { 
                c[i,j] = max(c[i-1,j] + indel, c[i, j-1]+indel, c[i-1, j-1] + mismatch)
            }
    }

Time complexity O(mn)

2 Question 2:

Potential function: \( \Phi(i) = 2 \times \text{number of items in queue}_1 \)

Enqueue: amortized cost \( c' = 1 + 2 = 3 \);
Dequeue: amortized cost \( c' = 1 + 0 = 1 \);
//Move all dequeue n item from queue 1, and enqueue n
//item to queue 2, so actually cost is 2n
Move all: amortized cost \( c' = 2n - 2n = 0 \);

The amortized cost is O(n). Other reasonable approach will be accepted

3 Question 3:

for each element in sets, add a field named ‘root’, when it is union with other
sets, update all the root field to new root, so that find can be done in O(1)
(by examine the ‘root’ value directly). Updating root field cause at most
O(c). So, total time complexity for every union operate takes O(c). So for
n operations, the total time complexity would be O(n).

4 Question 4:

using stack
DFS(G,v)  ( v is the vertex where the search starts )
Stack S := {};  ( start with an empty stack )
for each vertex u, set visited[u] := false;
push S, v;
while (S is not empty) do
  u := pop S;
  if (not visited[u]) then
    visited[u] := true;
    for each unvisited neighbour w of u
      push S, w;
  end if
end while
END DFS()

time Complicity O(V+E)

5  Question 5:

Prove by contradiction: assume there exist more than one single linear path.
There must exist a branch in the graph, to cover other edges in the branch, you must transverse back to the node, which means there is a cycle in the graph and conflict with the question.