1.

2^(sqrt(2log(n))
(sqrt(2))^log(n)
2 ^ log(n)
nlog(n)
n (log(n)^3)
2^n^2
2^2^n

2.

Run BFS -> Output adjacent list: List<List< Integer, Integer>>

If no cycle, the BFS should output the same number of edges as the original graph.

Else, there is a cycle. Trace back from the node in the missing edge to the common ancestor in the BFS tree to find the cycle.

```
output = BFS(s)
if output == graph (every edge appears in the output):
    return there is no cycle
else:
    return there is a cycle
def find_cycle( node that is on the missing edge of BFS output):
    visited[node] = True
    stack = [node]
    while stack:
        current = stack.pop()
        for adj_v in current:
            if visited[adj_v]:
                continue
            if adj_v != paretnt:
                visited[adj_v] = True
                cycle.append(adj.v)

    return cycle
```

The time complexity of this algorithm is O(V+E) because each vertex is visited only once.

If there is a cycle in graph G, the BFS will have different output compared with the graph G. Since there is a cross edge that cannot be detected by BFS.

If the algorithm detects cycle, there must be a vertex that has two different parents, so the graph must have a cycle in it.

3.

Binary tree with one node has 1 leaf and 0 node with two children.
Assume there is a binary tree has N node with 2 children. Add one node will result in N-1 node with 2 children and N leaves.
Assume there is a binary tree has N – 1 node with 2 children. Add one node will result in N node with N+1 leaves.
By induction, binary tree that has N node with 2 children will have N+1 leaves

4.

According to the book, a planar graph satisfy E <= 3*V – 6.
A complete graph K 5 has 10 edges, but planar graph with 5 vertices must have 9 edges or less, so K5 is not a planar graph.

5.

Operation	1	2	3	4	5	6	7	8	…	n
Cost	1	2^1	1	2^2	5	6	7	2^3	…	Log2(n)

Total_cost = $2 * \frac{(1 - 2^{log2(n)})}{1-2} + (n - log2(n))$
AC = total_cost / n = O(3)

6.

Insert	Old Size	New Size	Copy
1	1	3	0
2	3	5	1
3	5	7	2
4	7	9	3

Assume (2^n) + 1 inserts, there will be 2^n copies
AC = $(2^n + 1 + 2^n)/(2^n + 1)$ = O(2)

7. we can use a modified Dijstra’s Algorithm. Time complexity O(E* log(V))

```
function max_min_weight(graph, s, t):
    weight = [-inf] * len(v)
    prev = [None] * len(v)
    weight[s] = inf
    Q = all vertices in the graph
    While Q:
        U = vertex with highest weight in weight (priorityqueue)
        U = Q.pop()
        If weight[u] = -inf:
            Break
        For v in adj[u]:
            Temp = max(weight[v], min(weight[u], difference  between u, v)
            If temp > weight[v]:
                Weight[v] = temp
                Prev[v] = u
                Q.update(weight[v])
    Return weight[t]

Function find_max_min_path(graph, s, t, prev):
    Path = []
    Path.append(t)
    Current = t
    While prev[current] != s:
        Path.append(prev[current])
        Current = prev[current]
    Path.append(s)
    Return reversed(path)
```