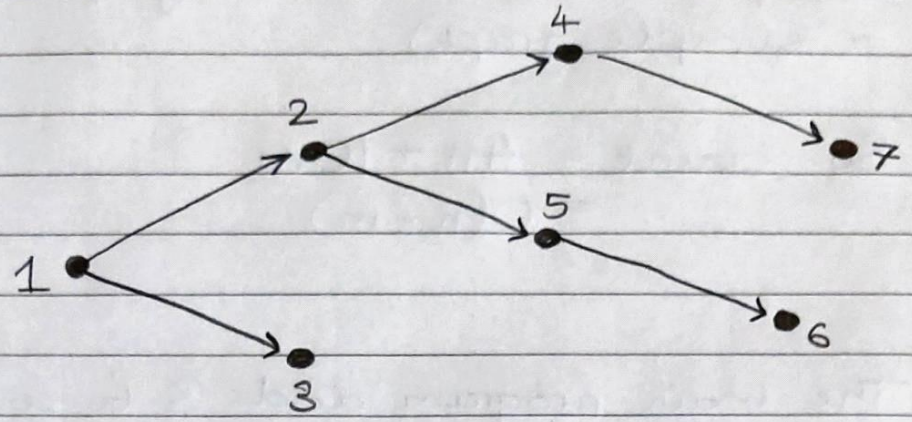
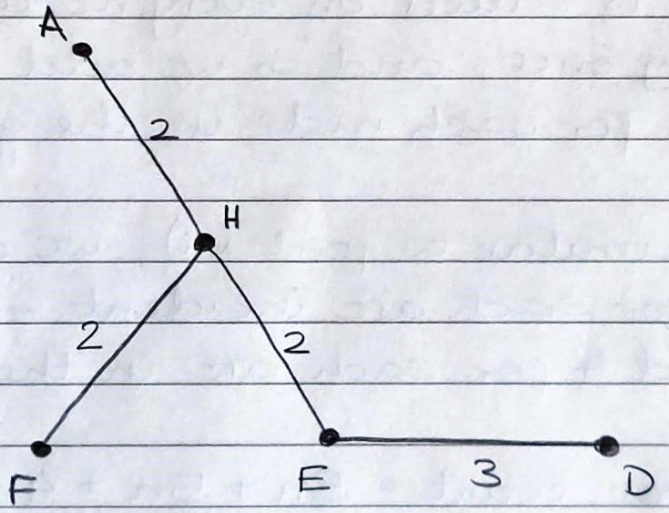
Graphs and Algorithms

1 a) Topological sort order: 1, 3, 2, 5, 6, 4, 7
Order entered: 1, 2, 4, 7, 5, 6, 3
Order exited: 7, 4, 6, 5, 2, 3, 1



Topological sort order is reverse of exit order

b) Order added: A, H, E, F, D



Shortest path is AHED, with total cost 7

c) i) For each node we add 4 to count. For each node we also add 5 for each arc incident on that node. Since each arc is incident on two nodes, we add 10 for each arc in the graph (assuming a simple graph).

So count = $4n + 10m$
$= O(n+m)$

ii) The main program adds 3 to count for each node in the graph. Then iterate is always called on unvisited nodes (because it is always under the condition "if not visited[x]" or similar) and the main program attempts to call it for every node in the graph. Thus iterate is called on every node in the graph exactly once, and so we add 3+2=5 to count (again) for each node in the graph.

And, similar to part (ii), we add 2 to count for each arc incident on each node, so add 4 for each arc in the graph.

This gives count = $3n + 5n + 4m$
$= 8n + 4m$
$= O(n+m)$

d) M[0] = 0
found[0] = true

```
for i = 1 to n:
    M[i] = c[i]
    found[i] = (c[i] ≠ 0)

    for j = 1 to i:
        if found[i-j] and found[j]:
            if not found[i]:
                M[i] = M[i-j] + M[j]
            else:
                M[i] = min(M[i], M[i-j] + M[j])
            found[i] = true

if not found[n]:
    return "not found"

return M[n]
```