ict1ai.

A spanning tree T of an undirected connected graph G is such that T, a tree, is a subgraph of G and nodes(T) = nodes(G)

1aii.

Order: 1,2,7,3,8,5,6,4

1b.

Assume that there isn’t a sink node nor a source node. Then every node must have at least 1 outgoing arc, and at least 1 ingoing arc. Apply DFS.

For each node traversed, we use one of its outgoing arcs to continue DFS and add it to a list of traversed nodes, say traversed.

If we reach a node and it points to one of the nodes we have already traversed, this is a cycle, so this is done.

If not, we continue until we reach a node x such that { } = nodes(G) \ traversed. But as we have assumed that x must have an outgoing arc, x must point to some other node y in nodes(G). But as nodes(G) \ traversed = { }, then we must have already visited y. Then this is a cycle.

As our graph is acyclic, then it can’t be the case that we have no sink nodes and no source nodes. Hence, there must exist at least one sink node and at least one source node.

1c.

Count = 2n + 6m. This is of order O(n + m).

Since a simple graph can have a maximum of n\*(n-1)/2 arcs, we can write m in terms of n and the result would be of order O(n^2).

2

1di.

A minimum spanning tree of a connected weighted graph (G, W) is a spanning tree such that for all spanning trees T in G, weight(T) >= weight(MST).

[weight(G’) function is a function defined as the total sum of weights of the parcs in G’, a subgraph of G, using W]

1dii.

Find max(W), and apply the function to all weightings w in W, max(W) - w, such that the order of arcs are now inverted (i.e largest weighted arc should now be 0 and the smallest), to get the weight matrix W’.

Apply Kruskal’s/Prim’s on (G, W’). This gives a MST in (G,W’) but a MaxST in (G, W).

Correctness:

Assume our algorithm does not return the MaxST. Then whilst our algorithm returns a spanning tree B, there exists a spanning tree A that is the MaxST, such that weight(A) > weight(B).

But by our algorithm, we can then find the relation

(|nodes(G)| - 1) \* max(W) - weight(A) = weight(A’) < (|nodes(G)| - 1) \* max(W) - weight(B) = weight(B’), where A’ and B’ are MSTs returned by Kruskal’s/Prim’s.

[|nodes(G)| - 1 is derived from the fact that spanning trees, which are trees, will have n - 1 arcs.]

But this is a contradiction as then B’ wouldn’t be a MST if weight(A’) < weight(B’), breaking the correctness of Kruskal’s/Prim’s.

Hence, this is a contradiction.

2ai.

Insertion sort on [1,4,3,2,5] is 7 comparisons needed.

2aii.

For a pair-swapped list of size n even >= 2, e.g [2,1,4,3,6,5,8,7], we need 1 + 3(n - 2) / 2 comparisons. To get to there, we notice that the example list requires [0,1,1,2,1,2,1,2] comparisons for each element. We can then generalise this with larger lists.

To sort the first 2 elements, we require 0 and 1 comparisons respectively. We observe that the relation holds: Each value in a pair is greater than the ones before it.

Hence, for each pair except the *1st (in*dex 0 and 1), we require 1 (simple check and appending) to insert the first of the pair, and then 2 comparisons for the second of the pair.

Hence, we get the equation 1 + 3(n - 2) / 2.

2bi. Given E = [3,5,16,7,2,10,9,13]. Draw heap where we fill along.

\*cba to draw\*

2bii.

E’ = [16, 13, 10, 7, 2, 3, 9, 5]

\*cba to draw\*

2ci.

An undirected graph G has a Hamiltonian path if there exists a path that visits every node exactly once in

G.

2cii.

HPN can be defined to be the problem ∃P. E(G, x, y, P) such that P is a path in G, and E is the decision problem that verifies if a path P has start and endpoints x and y respectively, is a hamiltonian path in G.

As P is a path, the number of nodes in P must be bounded by |nodes(G)| as we can’t have repeated nodes. Hence, |P| is polynomially bounded, and thus HPN is in NP.

2ciii.

We know that HAMPATH is NP-Complete so HAMPATH is in NP and all decision problems NP reduce to HAMP ATH.

We know that HPN is in NP, so to show that HPN is NP-Complete, we just need to also show that all decision problems in NP reduce to HPN. We can do by transitivity when we show HAMPATH reduces to HPN.

Take arb. Graph G, we construct G’ by adding 2 extra nodes x and y to G, and make both x and y connect to every single node in G. This construction is in p-time since we add p-many nodes (2) and arcs (2\*|nodes(G)|).

To prove HAMPATH(G) -> HPN(G’, x, y)

Assume HAMPATH(G), then there exists a Hampath P in G; name its starting node ‘a’ and finishing node ‘b’. Then we have a path P’ from x to y, by adding edge (x, a) and (y, b) to P.

Clearly P’ visits every node in G exactly once, also P’ visits x and y exactly once. Hence P’ visits every node in G’ exactly once. Therefore P’ is a Hampath of G’, we get HPN(G’, x, y).

To prove HPN(G’, x, y) -> HAMPATH(G)

Assume HPN(G’, x, y), then there exists a Hampath P’ in G’ with starting node x and finishing node y. Let a be the node connecting x in P’ and b be the node connecting y in P’.

By removing x and y from P’, we get a path P from a to b.

As P’ visits all nodes in G’ exactly once, it must visit all nodes in G exactly once. Therefore P visits all nodes in G exactly once. Hence P is a hampath of G.

Overall, HAMPATH(G) <-> HPN(G’, x, y), HAMPATH <= HPN

As HAMPATH reduces to HPN, then as for all decision problems D in NP, D <= HAMPATH, then D also reduces to HPN by transitivity.

Hence, as HPN is in NP and is NP-Hard (all decision problems in NP reduces to HPN), then HPN is NP-Complete.