ALGORITHMS - a finite sequence of step by step instructions.

- Flow Charts — START/END (oval), Instruction (rectangle), Decision (diamond). Use tables to record your steps

- Bubble Sort
  - Use to put into ascending/descending order
  - Work through the list by comparing pairs of adjacent items in the list
  - If the items are correct, leave them
  - If the items are incorrect, switch them
  - Continue until you finish the first pass
  - Repeat until no swaps.

- Quick Sort
  - Use to put into ascending/descending order
  - Choose the pivot using $\left(\frac{n+1}{2}\right)$
  - Write all the items less than the pivot (in the same order) in a sublist
  - Write the pivot
  - Write all the items remaining (in the same order) after the pivot
  - Repeat until all items have been chosen as pivots.

- Bin Packing
  - Always remember to find the lower bound.
    - Add all items together
    - Divide by height of the bin.
  - If your result is the same as the lower bound after applying the algorithm, then this is an optimal solution.

First-Fit
- → Take items in the order given
- → Place each item in the first available bin that can take it. Start at bin 1 every time.
- ⊕ Quick to apply ⊖ Not likely to be good solution

First-Fit Decreasing
- → Sort items (using another algorithm) so they are decreasing
- → Apply first fit algorithm to new list.
- ⊕ Usually get a fairly good solution & its easy ⊖ May not get optimal solution

Full-Bin
- → Use observation to find combinations that will fill a bin. Pack these first.
- → Any remaining items to be packed using first fit
- ⊕ Usually get a good solution ⊖ More difficult with more items/complex.

Binary Search
- Ensure list is ordered (may have to sort)
- $\frac{n+1}{2}$, use as pivot (round up if not an integer)
- Compare midpoint with what we are trying to find.
- Discard the part where it must not be
- Repeat until found / proven its not in the list.

GRAPHS & NETWORKS

- A graph consists of vertices/nodes which are connected by arcs/edges.
- If a graph has a number associated with each edge, this is called a weighted graph.
- The degree/valency of a vertex is the number of edges that meet at that vertex
- A trail which traverses (goes down) every arc and starts and ends at the same vertex is called an Eulerian cycle.
- A walk is a route through the graph
- A path is a walk which no vertex is visited more than once.
- A trail is a walk in which no edge is visited more than once.
- A cycle is a walk in which the end and start vertex are the same and no vertex is visited more than once.
- Hamiltonian cycle is a cycle which includes every vertex.
- A spanning tree is a subgraph which includes all vertices and is also a tree.
- An adjacency matrix describes the number of arcs joining the corresponding vertices.
- A distance matrix has entries that represent the weight of each arc.

Kruskal's Algorithm

- Use to find the minimum spanning tree.
  1. Sort arcs into ascending order of weight
  2. Consider the next arc of least weight (Remember, we do not want to form a cycle) If there is a choice of equal weight, consider each in turn.
  3. Repeat until all vertices are connected.

Prim's Algorithm

- Use to find the minimum spanning tree.
  1. Choose any vertex to start the tree.
  2. Select an arc of least weight that joins a vertex already in the tree to one not yet in the tree. If there is a choice of equal weight, chose any
  3. Repeat until all connected.
  4. List the order of arcs that were added.

Dijkstra's Algorithm

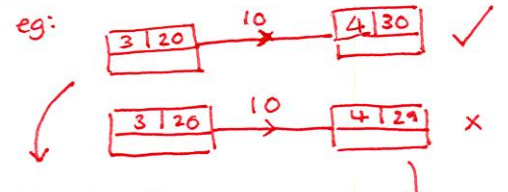
- Use to find the shortest path

Steps:
1. Label start vertex with 0
2. Look at all the vertices that can be reached directly from the start. Temporary label with weights
3. Select the vertex with the smallest temporary label and make it permanent
4. Now look at all routes from the new vertex that connect directly temporary label with the sum of the weight.
5. Choose the least weight including the original vertices to find the second node
6. Repeat until all vertices have a permanent label.

THEN...

Identify the shortest route
1. Work backwards from the end node
2. You can only go down paths that have the same weight as the difference in the nodes

eg:



30 - 20 = 10
arc weight is 10 ✓

29 - 20 = 9
arc weight is 10
x

State the length once you get back to start

Prim's can also be used from a table.
1. Delete the row of your chosen vertex
2. Number the column for your chosen vertex
3. Circle the lowest undeleted entry
4. The circled entry becomes the next arc
5. Repeat 1-4 until all rows are deleted
6. List arcs in the order they were added

Route Inspection

- Find the total of all the weights in the network.
- Identify the odd vertices
- Pair the odd vertices in all possible ways and for each pairing add the weights between each pair of nodes.
- Identify the pairing for which the total weight is shortest. Add this distance to the total of the weights for the network. This is the total weight of the shortest route.

*BE CAREFUL, WHEN PAIRING, THE LENGTHS MAY BE MISLEADING AND MAY WEIGH MORE, SO EXPLORE ALL ROUTES POSSIBLE *

EULERIAN - if the graph has all even nodes, it is eulerian. The shortest route is the same as the total weight.

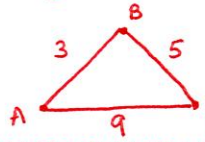
There can be a number of routes available and the question may ask for you to write your route.
*It is important that you make a note of the edges that must be repeated * (This will be the edges of the original odd vertices)

SEMI-EULERIAN - starts and finish at different nodes these nodes will both be odd. To make the network Eulerian you would need to add arcs and these would need to be traversed twice

The travelling Salesman

Triangle Inequality :

longest side of any triangle ≤ sum of the two shorter sides



AC ≤ AB + BC
9 ≤ 3 + 5 ✓✓

Classical Problem : each vertex must be visited exactly once before returning to the start

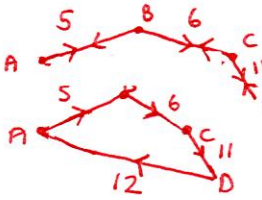
Practical Problem : each vertex must be visited at least once before returning to the start.

*If you use a table of least distances then the classical and practical method are equivalent *

BOUNDS :

Minimum Spanning Tree : UPPER

① Find MST using Prim's / Kruskal's
② Double this total (this is your INITIAL UPPER BOUND)
③ Try to find shortcuts using arcs not in the MST to decrease the total weight.



5+6+11 = 22
22×2 = 44
or
5+6+11+12 = 34

Minimum Spanning Tree : LOWER

① Remove each vertex in turn, together with its arcs. (usually stated which)
② Find the residual minimum spanning tree (this is just the MST after removal of one vertex)
③ Add to the RMST 'the cost' of reconnecting with the deleted vertex using the two shortest distinct arcs and note the total.
④ The greatest of these totals is used for the lower bound
⑤ Make the lower bound as high as possible to reduce size of interval where optimal solution lies.
⑥ Only if you find a Hamiltonian cycle or if upper = lower then the solution is optimal.

Nearest Neighbour : UPPER

① Select each vertex in turn as a starting point (This will usually be told for which one to start)
② Nearest Neighbour
- Choose 1 vertex
- Select the arc with the least weight
- This is node 2, you can only look down this column to find the next.
- Repeat until all vertices are added.

①	A	B	② C	D
A	–	5	4	7
B	5	–	8	3
C	④ 4	8	–	2
D	7	3	⑥	–

even though 5 is smaller than 6, this is not connected to C ∴ cannot be used.

③ Once all vertices have been used as the starting vertex, select the tour with the smallest length as the upper bound.

Critical Path Analysis

When tasks can not take place before something else has happened means it is dependent.
- An activity network is drawn to represent the events.
- Node 0 is the SOURCE
- Final node is the SINK.

Dummys - a dummy has NO time or cost, it simply represents that an activity cannot start before another ends.

eg: 3 cannot happen until both 1 & 2 are finished.

Scheduling Activities

- Always make sure to find the early and late times.

early / late — Always start with early then work backwards for late.

Early - earliest time of arrival at the event allowing time for completion of all preceeding events.
Late - the latest time the event can be left without extending time needed for the project
Float = latest finish time − earliest start time − duration
Critical Activities → float of 0 & A path from source-sink = critical path

GANT CHARTS

- Always start with your critical path
- Number scale shows time passed.
- Include late times.
- Float is seen

Lower bound for total number of workers.

$$\frac{\text{sum of all activity}}{\text{critical time}}$$

Always round up

Linear Programming

① Define the decision variables (x, y, z)
② State the objective (maximise or minimise with objective function.)
③ Write the constraints as inequalities.

To graph - label feasable region
For max point, look for last point covered by objective line leaving feasable region
For min point look for first point covered by objective line entering feasable region