Decision Mathematics 1

Pearson Edexcel · International A-Level · Mathematics

Decision Mathematics 1: algorithms, networks and linear programming

Decision Mathematics 1 (D1, paper WDM01/01, 1 hour 30 minutes, 75 marks, first assessment June 2019) is the discrete option of the IAL: P1–P2 are assumed, and every method is an ALGORITHM – a finite list of exact instructions. No formula booklet entries beyond the basics: the marks are for executing procedures precisely and stating results in the required form. This reference covers sorting and packing algorithms, minimum spanning trees, shortest paths, the Chinese postman and travelling salesman problems, critical path analysis and linear programming.

House rules for algorithm answers:

Task	Required form
sorting	list the list after EACH pass (or pivot choices for quick sort)
bin packing	name the algorithm (first-fit, first-fit decreasing, full-bin) and show each placement
graph algorithms	state the order edges/nodes are chosen; show final values (labels, total weight)
route inspection	list the odd nodes, the pairings and their weights, before totalling
linear programming	define variables, objective and EVERY constraint with non-negativity

Sorting and packing algorithms

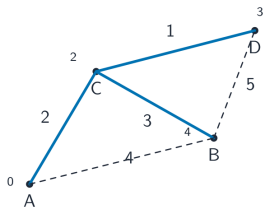
The **bubble sort** compares adjacent items, swapping out-of-order pairs, one pass at a time; a pass with no swaps ends it. The **quick sort** picks the MIDDLE item of the list as pivot, partitions the remaining items into "less than pivot" and "greater than pivot" sublists (keeping their order) and recurses – always quote the pivot and the list after each partition. The spec pins the middle-item choice to the method defined in its glossary, so exercise lists here keep an odd number of items and the middle item is the one at position $\frac{n+1}{2}$. **Bin packing** : first-fit places each item in the first bin with room; first-fit decreasing sorts items largest-first then applies first-fit; full-bin packing looks for combinations filling a bin exactly.

Worked example – quick sort (original). Sort 9, 2, 7, 5, 3, 4, 6. The middle item (position 4 of 7) is 5: less = 2, 3, 4; greater = 9, 7, 6, giving 2, 3, 4, **5**, 9, 7, 6. Sublist 2, 3, 4 (pivot 3): 2, **3**, 4 – sorted. Sublist 9, 7, 6 (pivot 7): 6, **7**, 9 – sorted. Final list:

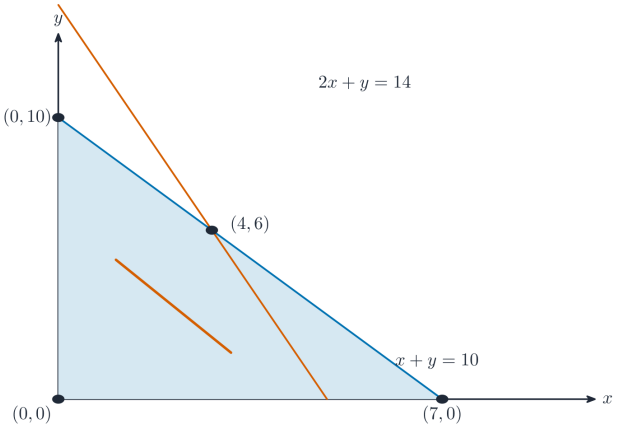
2, 3, 4, 5, 6, 7, 9.

Worked example – bin packing (original). Bins of capacity 10; items 7, 5, 4, 3, 6, 2 (total 27, so at least 3 bins). First-fit: 7→b1, 5→b2 (7+5 > 10), 4→b2 (9), 3→b1 (10), 6→b3, 2→b3 (8): bins (7,3), (5,4), (6,2). First-fit decreasing: 7, 6, 5, 4, 3, 2: 7→b1, 6→b2, 5→b3 (6+5 > 10), 4→b2 (10), 3→b1 (10), 2→b3 (7): bins (7,3), (6,4), (5,2) – also 3 bins.

Minimum spanning trees



MST = AC + CD + CB = 6; Dijkstra: A 0, C 2, B 4, D 3 via A–C–D



$P = 3x + 2y$: vertices give 0, 21, 24, 20 — max at $(4, 6)$

A **minimum spanning tree (MST)** connects every vertex at least total weight. **Kruskal's algorithm**: sort edges by weight, add the smallest that does NOT form a cycle (reject and move on when it would). **Prim's algorithm**: grow a tree from any start vertex, repeatedly adding the cheapest edge from the tree to a new vertex. The matrix form of Prim: cross out the start row, ring the cheapest remaining entry in the column of the chosen vertex, and continue. Kruskal is edge-driven (good on a drawn network); Prim is vertex-driven (good on a matrix). Both give the same total; edges may differ only when weights tie.

Worked example (original). Edges AB 4, AC 2, BC 3, BD 5, CD 1: Kruskal takes CD 1, AC 2, BC 3 (B joins – no cycle), then rejects AB 4 and BD 5 (cycles): MST total 6. Prim from A: AC 2, CD 1 (cheapest edge from {A,C} to a new vertex), CB 3: the same tree, and the total 6 ✓. Always list the edges in the order chosen and give the total.

Shortest paths

Dijkstra's algorithm labels vertices with the smallest total distance found so far: start vertex label 0; repeatedly pick the unlabelled vertex of smallest temporary label, make it permanent, and update its neighbours (new = permanent

+ weight, only if smaller). Record the order of permanence and the final labels; the route is recovered by tracing back through vertices whose label equals the final label minus the connecting weight.

Worked example (original). Edges AB 4, AC 2, BC 1, BD 5, CD 8. Labels: A = 0 permanent; C = 2 permanent; B = min(4, 2+1) = 3 permanent; D = min(3+5, 2+8) = 8 permanent. The shortest route from A to D has length 8: A–C–B–D (trace back: 8 = 3 + 5 at B, 3 = 2 + 1 at C).

The Chinese postman problem

The **route inspection (Chinese postman) problem** walks every EDGE at least once and returns to the start. Total length = total of all edges + the shortest way to duplicate edges so every vertex becomes even. Find the odd vertices (at most four), list their pairings, choose the pairing of least total weight, and duplicate those edges. State the pairing before the total.

Worked example (original). Edges AB 4, AC 2, BC 3, BD 5, CD 1 (total 15). Degrees: A 2, B 3, C 3, D 2 – the odd nodes are B and C, so duplicate BC (weight 3): the shortest closed route walking every edge at least once has length 15 + 3 = 18.

The travelling salesman problem

The **classical travelling salesman problem** tours every VERTEX exactly once and returns, on a complete graph with the triangle inequality. Bounds: a lower bound is the MST of the network doubled is an UPPER bound route (walk each MST edge twice); a better upper bound applies shortcuts. Lower bounds come from MST-based arguments (e.g. the MST itself, or deleting a vertex and adding its two cheapest edges). The **nearest neighbour algorithm** builds a tour: from the start vertex repeatedly go to the nearest unvisited vertex, then return. Convert to a complete network of shortest distances first when the graph is not complete (via Dijkstra or inspection).

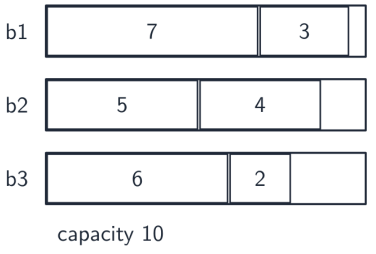
Worked example (original). Complete network: AB 4, AC 2, AD 7, BC 3, BD 5, CD 6 (the triangle inequality holds throughout). Nearest neighbour from A: A–C (2), C–B (3), B–D (5), D–A (7): tour length 17 – an upper bound. The MST is AC + BC + BD = 10, so doubling it gives the crude upper bound 20, improved to 17 by shortcuts; a lower bound from deleting A (MST of B, C, D is 3 + 5 = 8, plus A's two cheapest edges AC + AB = 2 + 4 = 6) is 14. The optimal tour lies between 14 and 17.

Critical path analysis

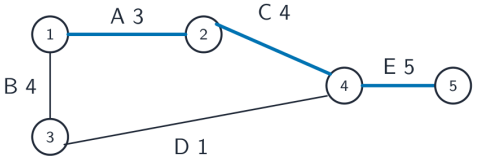
Model a project as an **activity-on-arc network** from a precedence table; **dummies** (zero-weight arcs) resolve dependencies that shared endpoints cannot express. The **critical path** is the longest chain through the network:

the project duration. Compute **earliest event times** (forward pass: the max of arrivals) and **latest event times** (backward pass: the min of departures). An activity's **total float** = latest finish − earliest start − duration; critical activities have zero float. **Gantt (cascade) charts** bar each activity from its earliest start with its float as a trailing space; scheduling allocates workers day by day.

Worked example (original). Activities A (3 days), B (4), C (4 after A), D (1 after B), E (5 after C and D). Chain A–C–E = 12 days; chain B–D–E = 10 days: the critical path is A–C–E and the project takes 12 days; D's total float = 12 − (4 + 1 + 5) = 2 days.



first-fit: (7,3), (5,4), (6,2) — three bins, none over capacity



event times 0; 3 / 4; 7; 12

critical path A–C–E = 12 days; D has float 2

Linear programming

Formulate: define the decision variables, write the objective (maximise/minimise) and every constraint as an inequality, including $x, y \geq 0$. Solve two-variable problems graphically: shade the feasible region, then use the **ruler method** (slide a ruler along the objective's gradient) or the **vertex method** (evaluate the objective at each vertex of the feasible region). Integer solutions: test integer points near the optimal vertex inside the region.

Checklist before you call this unit done

- Execute bubble and quick sorts showing every pass and pivot; bin pack by first-fit, first-fit decreasing and full-bin with placements shown.
- Find MSTs by Kruskal and by Prim (node and matrix forms); draw a network from a matrix and back.
- Run Dijkstra with labels in order of permanence; recover the route, not just the length.
- Solve route inspection: odd nodes, all pairings with weights, least duplication, total walk length.

- Bound the classical TSP: doubled MST and shortcuts for upper bounds, MST-deletion for lower bounds; run nearest neighbour on the complete shortest-distance network.
- Build activity networks with dummies; forward and backward passes for event times; total floats; identify the critical path; draw the Gantt chart and schedule.

Worked example (original). Activities A (3 days), B (4), C (4 after A), D (1 after B), E (5 after C and D). Chain A–C–E = 12 days; chain B–D–E = 10 days: the critical path is A–C–E and the project takes 12 days; D's total float $= 12 - (4 + 1 + 5) = 2$ days.

Worked example – linear programming (original). Maximise $P = 3x + 2y$ subject to $x + y \leq 10$, $2x + y \leq 14$, $x, y \geq 0$. The feasible region's vertices are $(0, 0)$, $(7, 0)$, $(4, 6)$, $(0, 10)$; evaluating P: 0, 21, 24, 20 – the maximum is $P = 24$ at $(4, 6)$, where both constraints are tight.

- Formulate linear programs fully; solve graphically by ruler or vertex methods; find best integer solutions.