DECISION 1 Revision Notes

1. Sorting (assuming sorting into ascending order)

a) BUBBLE SORT

Make sure you show comparisons clearly and label each pass

First Pass

8 4 3 6 1
4 8 3 6 1
4 3 8 6 1
4 3 6 8 1
4 3 6 1 8

Second Pass

4 3 6 1 | 8
3 4 6 1 | 8
3 4 6 1 | 8
3 4 1 6 | 8

Third Pass

3 4 1 | 6 8
3 4 1 | 6 8
3 1 4 | 6 8

Fouth Pass

3 1 | 4 6 8
1 3 | 4 6 8

Step 1
Compare first two numbers. If the smaller number is on the right, swap the two numbers – write the remainder of the list.

Step 2
Move one step forwards in the list and compare the two numbers. If the smaller is on the right swap the two numbers – write the remainder of the list

Step 3
Repeat Step 2 until the two numbers on the extreme right have been compared, this completes the FIRST PASS.

Step 4
SECOND PASS repeat Step 1 – 3 but no need to compare last 2 in the list (final place already sorted)

Step 5
THIRD PASS repeat Step 1 – 3 but no need to compare last 3 places as last 2 already sorted……..etc
STOP when a complete pass produces no swaps – show this!

b) SHUTTLE SORT

Make sure you show comparisons clearly and label each pass

First Pass

7 3 5 8 1
3 7 5 8 1

Second Pass

3 7 5 8 1
3 5 7 8 1

Third Pass

3 5 7 8 1
(no swap)

Fourth Pass

3 5 7 8 1
3 5 7 1 8
3 5 1 7 8
3 1 5 7 8
1 3 5 7 8

SHUTTLE-SORT

First Pass
Compare first two numbers – swap if necessary – rewrite the remainder of the list.

Second Pass
Compare second and third number – swap if necessary – rewrite the remainder of the list. Now compare the first two numbers –swap if necessary and re-write the list….IF not swap needed start the next pass

…….

Final Pass
The pass which starts by comparing the second to last and last numbers. **Make sure you complete the pass indicating comparisons made.**

BUBBLE AND SHUTTLE

For a list of n numbers the list will definitely be sorted after the $(n-1)^{th}$ pass

Maximum number of comparisons/swaps = $\frac{n(n-1)}{2}$

Shuttle can be more efficient than bubble as 'pass' stops when no swap needed.

c) SHELL SORT

This is the one which involving splitting the lists into groups – sorting the groups – merging the groups and splitting the lists again into smaller groups – the final step is to shuttle sort the last grouping which involves all elements in the list.

Step 1

For a list of n values work out INT(n/2) = m divide the list into m subgroups.

Step 2

Shuttle sort each subgroup – you may need to keep a record of comparisons/swaps but jut rewrite each subgroup in order. Merge the groups to form 1 list again

Step 3

Work out INT(m/2) = divide the list into this many subgroup – shuttle sort groups and merge

Continue until groups size = 1 and shuttle sort the complete list. **To gain full marks it is not necessary to show the individual passes of the Shuttle – see opposite**

37 36 15 16 33 28 18 25 *INT(8/2) = 4*

37 33 } 4 groups
36 28
15 18
16 25

33 28 15 16 37 36 18 25 INT(4/2) = 2

33 15 37 18 } 2 groups
28 16 36 25

15 16 18 25 33 28 37 36 INT(2/2) = 1

15 16 18 25 28 33 36 37

Take care with odd numbers INT(9/2) = 4

d) QUICK SORT

Take care with notation
-underline PIVOT value in list
-box in pivot value in resorted list –
- once boxed the value stays boxed until the end

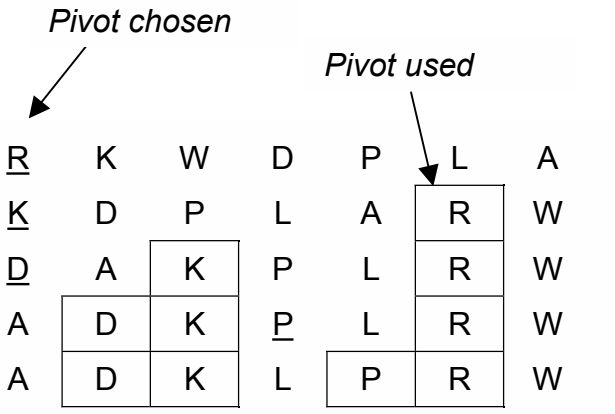
Step 1

Underline the first value in the list (PIVOT) – rewrite the list so that any values smaller than this are to the left - keep the original order of the values (see opposite)

Step 2

Draw a box around the pivot – your list is now split into two groups – repeat step 1 choosing the fist value in the LHS as the pivot and then repeat for the RHS.

Continue until none of the sublists have more than 2 numbers (eg. No more than 1 unused value between pivots)



2. Algorithms

Tracing an algorithm means showing how the variables change value as the computer would run through the program

- List the variables used (in the order they are introduced) as column headings in a table
- If the **PRINT** command is used – write clearly PRINT : and list the words/values the computer would print
- Watch out for **INEQUALITLES -** 'less than' **or** 'less than or equal to'
- When you think you have finished tracing – start again and check the values you have listed – particularly the last ones to make sure you didn't need to repeat again.

If you are asked to state the purpose of the algorithm – try to use correct mathematical terms

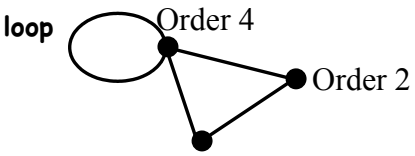
Division – quotient and remainder Square Numbers/Square roots

Factors – Multiples Integer values /Roots

If you are asked to state values of the variables for which the algorithm would fail – you will usually need to state **inequalities**

If you are asked to change/add additional commands to the algorithm – make sure you give the line an appropriate number.

3. Graphs – definitions



Node / Vertex – usually represent places –

- Degree / Order number of edges at the node

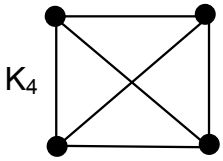
Arc / Edges – represent roads / pipes / cables

- Digraph if one or more of the edges has direction – one way

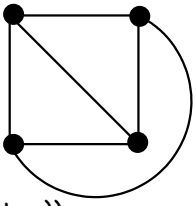
Connected Graph – it is possible to get from any node to another – not necessarily a direct route

Complete Graph (K_n) - there is a direct route between any 2 nodes/vertices

Number of edges in K_n = $\frac{n(n-1)}{2}$



K_4 drawn as a **PLANAR** graph (no edges crossing)



Hamilton Cycle –visits every vertex once (and only once apart from the first/last vertex))

–starts and finished at the same vertex

–does not use any edges more than once (does not need to include every edge)

n nodes	**(n -1)!** Hamilton cycles

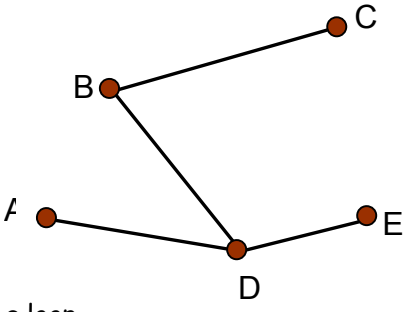
Eulerian Graph – traversable – possible to draw without lifting pen

– the degree/order of all of the nodes are even

The complete graph K_n is Eulerian if n is odd

4. MINIMUM SPANNING TREES – KRUSKALS or PRIMS

n nodes – the minimum span tree will have n –1 edges



- always a good idea to draw out your chosen minimum spanning tree
- clearly state it's minimum weight

e.g. Minimum weight = 28 m

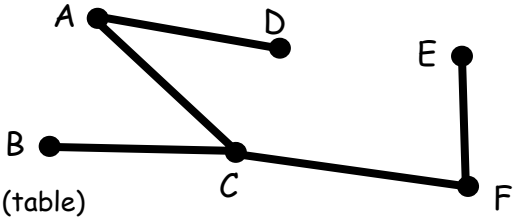
KRUSKALS

1) List the edges in ascending order of size/weight
2) Choose the smallest edge –
3) Choose the next smallest edge –
4) Keep selecting edges in order of size but ignore any that would make a loop.

WORKINGS

List the edges as you choose them with their weights + draw the tree as you work

AD: 4
CF: 7 } either order
CB: 7
FE: 8
AC: 11 **TOTAL = 37**



Disadvantage : difficult to do from a matrix (table)

GREEDY ALGORITHM – at each stage you make the most obviously advantageous choice without 'thinking' ahead'

PRIMS

From a graph

1) Choose the smallest edge (or the starting point given)
2) Find the smallest edge that is joined to your first choice edge
3) You can only choose edges that connect to the tree already drawn – always choose the smallest available – UNLESS it forms a loop
4) Continue until all the nodes are connected

WORKINGS

List the edges as you choose them with their weights + draw the tree as you work

From a Table

1) Choose a starting vertex and delete all elements in that vertex's row and highlight its column
2) Neglecting all deleted terms, scan **all** highlighted columns for the lowest available element and circle that element
3) Delete the circled element's row and highlight its column
4) Repeat steps 2 and 3 until all rows deleted
5) The spanning tree is formed by the circled arcs

	A	B	C	D	E
A	~~-~~	~~9~~	~~3~~	~~7~~	~~4~~
B	9	-	5	7	2
C	(3)	~~5~~	~~-~~	~~8~~	~~9~~
D	7	7	8	-	3
E	4	2	9	3	-

	A	B	C	D	E
A	~~-~~	~~9~~	~~3~~	~~7~~	~~4~~
B	9	-	5	7	2
C	(3)	~~5~~	~~-~~	~~8~~	~~9~~
D	7	7	8	-	3
E	(4)	~~2~~	~~9~~	~~3~~	

	A	B	C	D	E
A	~~-~~	~~9~~	~~3~~	~~7~~	~~4~~
B	9	-	5	7	(2)
C	(3)	~~5~~	~~-~~	~~8~~	~~9~~
D	7	7	8	-	3
E	(4)	~~2~~	~~9~~	~~3~~	

	A	B	C	D	E
A	~~-~~	~~9~~	~~3~~	~~7~~	~~4~~
B	~~9~~	~~-~~	~~5~~	~~7~~	(2)
C	(3)	~~5~~	~~-~~	~~8~~	~~9~~
D	7	7	8	-	3
E	(4)	~~2~~	~~9~~	~~3~~	

	A	B	C	D	E
A	~~-~~	~~9~~	~~3~~	~~7~~	~~4~~
B	~~9~~	~~-~~	~~5~~	~~7~~	(2)
C	(3)	~~5~~	~~-~~	~~8~~	~~9~~
D	7	7	8	-	(3)
E	(4)	~~2~~	~~9~~	~~3~~	

Workings

-Full and clear workings shown in a table
-List of the edges chosen, in the order chosen, with their weights
-diagram of minimum tree
- the minimum total weight

5. SHORTEST PATH BETWEEN 2 POINTS

DIJSKTRAS (labelling each node with the distance from the start)

- Label your starting point 0 and draw a box round it 0
- Look at the vertices connected to your boxed node and label **all** of them with the distance from start
- Choose the smallest label and 'box it in' then look at nodes connected to this point
- **Continue until all nodes are boxed** –
  - **do not stop just because you may have reached the destination**

WORKINGS

- make sure the starting node has a clearly boxed 0
- show 'crossing out' clearly when you need to re-label a node ~~34~~ ~~32~~ 29
- check that **all totals are boxed** before looking for the solution
- to find the solution –**work backwards** - looking at differences between boxed values

State clearly

- the route that should be taken for the shortest path between the 2 vertices given
- the length /weight of the chosen route.

If after finding the shortest path the question then changes a variable or inserts an additional route :
- look at the marks for the next part to give you an idea of the complexity of the new problem- you may need to consider more than one alternative route.

LIMITATION OF DIJSKTRAS – doesn't work if any of the weights are **NEGATIVE-**(may be that a lorry running along a particular edge may in someway produce a profit when you are actually trying to minimise cost).

6. CHINESE POSTMAN (Route inspection problem)

Travels along every **edge** at least once before returning to the start in the minimum distance.

Check if the graph is **Eulerian** /Traversable = LOOK if all of the **nodes are even**
If the question asks why is it not possible to Start and Finish at ……
ANSWER: nodes are odd therefore the graph is non-traversable

WORKINGS

- list all of the odd nodes in the network (A, C, D, E)
- List the different ways in which the odd nodes could be paired AC and DE = 6 + 14 =20
- find the shortest distance for each of the pairs AD and CE = 11 + 8 = 19
  AE and CD = 12 + 8 = 20
- choose the pairing which gives the smallest total (AD and CE = 19)
- add together all of the weights in the network plus additional edges found (AD and CE 19)

STATE your total weight clearly
STATE clearly a route around the network which includes the extra edges you have added

The number of possible pairings
2 nodes – 1 pairing
4 nodes – 3 pairings n nodes $(n-1)\times(n-3)\times(n-5)\times\ldots\times 3\times 1$ pairings
6 nodes – 15 pairings

6. Travelling Salesman (Upper and Lower Bounds)

Hamilton Cycle – a tour/route which contains every vertex **exactly** once – for small numbers of nodes may be possible to list the hamilton cycles but no algorithm **(n-1)!** different cycles for n nodes – instead we use Upper and Lower bounds

Lower bound –
- delete one of the vertices (often told which one)
-find a minimum spanning tree (Kruskals or Prims) for the remaining vertices
-add to the spanning tree to weights of the two smallest edges from the deleted vertex
You may need to repeat this for all vertices-the biggest answer is the best LOWER BOUND
Likely to be single nodes in the Lower Bound solution therefore unlikely to be the optimal solution as a complete tour would not be possible using the edges used.

Upper bound – (Nearest Neighbour)
(You may need to complete a matrix to show the shortest distance between pairs of nodes)
- choose a starting node (may be given in the question)
-join this node to the nearest node
- continue - join your new node to the nearest node (as long as it hasn't already been used) – when you have included all nodes – join the last node you picked back to the start

WORKINGS
If working from a table – circle the edges chosen and delete the rest of the row and column – do not confuse with PRIMS - you can only choose from the current column not previous ones used
LIST the edges in the order they were chosen and their weights
The TOTAL of the weights is the UPPER BOUND

Upper bound is often a tour which may be improved on – Nearest Neighbour is a HEURISTIC Algorithm – will give a satisfactory solutions but not necessarily the optimal
You may need to repeat this for all vertices-the smallest answer is the best UPPER BOUND

When stating the possible length of the optimal solution make sure you use **≤** not just **<**

You have found the **optimum solution** if **Upper bound** = **Lower Bound**

7. Matchings

Bipartite graph – has 2 distinct sets of vertices (no edges within the group of vertices)
MATCHING – one or more edge connecting a vertex in one group to a vertex in the other – **a vertex can only be used once**
COMPLETE MATCHING – matches every vertex in one group to one in the second group
MAXIMAL MATCHING – is one where that uses the greatest possible number of edges (complete is always maximal)

To improve a matching (find a maximal matching)
- Always start with the initial matching (usually given in the question)
-Start with an unmatched node in the LHS
- Move to RHS along and edge NOT in INITIAL MATCHING
-Move back to LHS along and edge IN the INITIAL MATCHING
- Continue until you reach an unmatched vertex in the RHS

WORKINGS

Initial Matching : A-G, B-M, C-H, D-S **Solution**

Include edges in alternating but not in initial

Alternating Path: F – A + G – C + H - E AF ,CG , EH

Include edges in initial but not in alternating

BM, DS

You may need to repeat the process more than once – if you haven't found a maximal matching – investigate different starting points if more than one unmatched vertex in the LHS

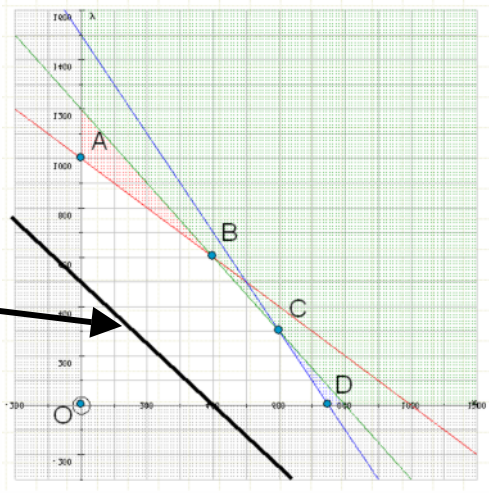
9. LINEAR PROGRAMMING

The inequalities

- identify the variables to use in your inequalities – question may state 'x' is….and 'y' is ….
- it may help you to make a table with x and y as column headings and the factors needed as the row headings with their totals shown at the end of the rows)
- check for the word **integer** in the question – often for the inequalities
- state clearly the **objective function** – what you are trying to maximise or minimise
- don't forget the inequalities for the lower limits of x and y e.g $x \geq 0$, $y \geq 3$
- make sure you use the same units throughout (e.g £ , kg…)

The graph

- label you axes clearly
- label the lines and you plot them
- shade out the area you cannot use
- don't forget to shade below the axes if necessary.
- plot the **objective function** – label this



Finding the optimal solution

- if maximising look for point further away from origin and objective line moves out of feasible region - if minimising look for point closest to origin
- use simultaneous equations to calculate the (x,y) at the vertex being investigated
- calculate the value of the objective function using this x and y
- if **integer** solutions needed (use your common sense) investigate points in the feasible region near the initial vertex chosen

TAKE CARE with questions involving percentages – if an inequality involves 40% of the total produced – your inequality should involve 0.4(x + y)

If you have to find the lines defining the Feasible Region use the equation $y = mx + c$ to work out the equations of the lines plotted (m is the gradient and c is the y intercept)