Q1 (i) (a) O(n^2) [or O(n2) if you want to use formatting, or O(n squared) will do]

(b) O(n log n)

(c) O(n^2)

(ii)

***Note - quickest way to do these is to solve by hand on paper, then type the solutions in.***

(a) (Insertion Sort)

*8 15 3 23 7 19 25 4 12 18 (start)*

*8 | 15 3 23 7 19 25 4 12 18 (0 comparisons)*

*8 15 | 3 23 7 19 25 4 12 18 (1)*

*3 8 15 | 23 7 19 25 4 12 18 (2)*

*3 8 15 23 | 7 19 25 4 12 18 (1)*

*3 7 8 15 23 | 19 25 4 12 18 (4)*

*3 7 8 15 19 23 | 25 4 12 18 (2)*

*3 7 8 15 19 23 25 | 4 12 18 (1)*

*3 4 7 8 15 19 23 25 | 12 18 (7)*

*3 4 7 8 12 15 19 23 25 | 18 (5)*

*3 4 7 8 12 15 18 19 23 25 (4)*

*27 comparisons*

*(b) heapsort*

*8 15 3 23 7 19 25 4 12 18 (start)*

*8 | 15 3 23 7 19 25 4 12 18 (0)*

*15 8 | 3 23 7 19 25 4 12 18 (1) (building a max-heap)*

*15 8 3 | 23 7 19 25 4 12 18 (1)*

*23 15 3 8 | 7 19 25 4 12 18 (2)*

*23 15 3 8 7 | 19 25 4 12 18 (1)*

*23 15 19 8 7 3 | 25 4 12 18 (2)*

*25 15 23 8 7 3 19 | 4 12 18 (2)*

*25 15 23 8 7 3 19 4 | 12 18 (1)*

*25 15 23 12 7 3 19 4 8 | 18 (2)*

*25 18 23 12 15 3 19 4 8 7 (3)*

*23 18 19 12 15 3 7 4 8 |* ***25*** *(2) (and now destroying the heap)*

*19 18 8 12 15 3 7 4|* ***23******25*** *(2)*

*18 15 8 12 4 3 7 |****19******23******25*** *(2)*

*15 12 8 7 4 3 |* ***18 19******23******25*** *(2)*

*12 7 8 3 4 |* ***15******18 19******23******25*** *(2)*

*8 7 4 3 |* ***12******15******18 19******23******25*** *(1)*

*7 3 4 |* ***8******12******15******18 19******23******25*** *(1)*

*4 3 |* ***7******8******12******15******18 19******23******25*** *(1)*

*3 |* ***4 7******8******12******15******18 19******23******25*** *(0)*

***3*** ***4 7******8******12******15******18 19******23******25*** *(0)*

28 comparisons

***NOTE: heapsort is lengthy to write out line by line. If the exam asks to do heapsort, show that you can do the first couple of steps, show the max-heap that results in the middle, then show the first couple of steps to turn that heap into a sorted list.***

(c) quicksort (NOTE- error corrected in the comparison count - explanation at end of paper)

*8* 15 3 23 7 19 25 4 12 18 (start)

*7* 4 3 **8** *23* 19 25 15 12 18 (11)

*3* 4 **7** **8** *12* 19 18 15 **23** *25* (10)

**3** *4* **7** **8** **12** *19* 18 15 **23** **25** (6)

**3** **4** **7** **8** **12** *15* 18 **19** **23** **25** (3)

**3** **4** **7** **8** **12** **15** *18* **19** **23** **25** (2)

**3** **4** **7** **8** **12** **15** **18** **19** **23** **25** (0)

32 comparisons

(iii) Sort the input using an efficient sort algorithm - say mergesort to get the best O(n log n) worst case complexity. Then do

mindiff = array[1] - array[0]

minplace = 0

for i from 1 to n-2

if array[i+1] -array[i] < mindiff

mindiff = array[i+1] - array[i]

minplace = i

return array[minplace], array[minplace+1]

Complexity of above algorithm is O(n), so total worst case complexity is from sorting, which is O(n log n)

Q2

Top level is a list of (vertex,adjacency list) pairs. Using [x,y,z,...] to show a list, e(x,y,w) to show an edge with weight w between vertex with label x and vertex with label y.

[(A,[e(A,B,4), e(A,C,5)]),

(B,[e(A,B,4), e(B,C,7), e(B,E,6)]),

(C,[e(A,C,5), e(B,C,7), e(C,D,5)]),

(D,[e(C,D,5), e(D,E,3)]),

(E,[e(B,E,6), e(D,E,3)])]

(ii) (a) do a linear search of the list for an entry (x,*list*) and return the pair.

(b) run the method for (a)above then extract the list from the pair

(iii)

Prim:

Free: { a:{}, b:{}, …, m:{} }

Pq : [ cost, (vertex,edge) ] Pq: [ infinite, (v, None) for all v ]

Locs: { v: index }

Mst: {}

Pq: remove a

Locs: remove a

Free: remove a

Mst: {}

Pq add {5, (b, a-b]}, [7, (…

Remove {5, (b, a-b)]

Free: remove b

Locs: remove b

Mst: { e(a,b) }

Pq: add

Djikstra:

Open: apq {0:{a}} {key:(vertex)}

Locs: { a: 0} { vertex: index in pq }

Closed: empty {]

Preds: { a: None }

Open: remove a

Locs: remove a

Closed: {a: (0, None)}

Preds: {a: None}

Open: add {5:b}, {10:f}, {e:7}

Locs: add b, f, e

Closed: {a: (0, None)}

Preds: {a:None, b:a, f:a, e:a}

Open: remove min {5:b}

Locs: remove b

Closed: {a:(0,None), b:(5,a)}

Preds: {a:None, b:a, f:a, e:a}

###################################################

PQ: [Each edge as (weight,edge) pair, in a APQ with the minimum cost edge at the front]

MST: {}

Dict: [a:{a}, b:{b}, c:{c}, d:{d}, e:{e}, f:{f}, g:{g}, h:{h}, i:{i}, j:{j}, k:{k}, l:{l}, m:{m}]

PQ: remove min cost edge e(F,J,3)

MST: {e(F,J,3)}

Dict: [a:{a}, b:{b}, c:{c}, d:{d}, e:{e}, f:{f,j}, g:{g}, h:{h}, i:{i}, j:**F**, k:{k}, l:{l}, m:{m}]

(using capital letters to indicate a tree found at another vertex key)

PQ: remove min cost edge e(F,G,4)

MST: {e(F,J,3), e(F,G,4)}

Dict: [a:{a}, b:{b}, c:{c}, d:{d}, e:{e}, f:{f,g,j}, g:**F**, h:{h}, i:{i}, j:**F**, k:{k}, l:{l}, m:{m}]

PQ: remove min cost edge e(B,D,4)

MST: {e(F,J,3), e(F,G,4), e(B,D,4)}

Dict: [a:{a}, b:{b,d}, c:{c}, d:**B**, e:{e}, f:{f,g,j}, g:**F**, h:{h}, i:{i}, j:**F**, k:{k}, l:{l}, m:{m}]

PQ: remove min cost edge e(A,B,5)

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5)}

Dict: [a:**B**, b:{a,b,d}, c:{c}, d:**B**, e:{e}, f:{f,g,j}, g:**F**, h:{h}, i:{i}, j:**F**, k:{k}, l:{l}, m:{m}]

PQ: remove min cost edge e(E,I,5)

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5),e(E,I,5)}

Dict: [a:**B**, b:{a,b,d}, c:{c}, d:**B**, e:{e,i}, f:{f,g,j}, g:**F**, h:{h}, i:**E**, j:**F**, k:{k}, l:{l}, m:{m}]

PQ: remove min cost edge e(G,K,5)

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5),e(E,I,5),e(G,K,5)}

Dict: [a:**B**, b:{a,b,d}, c:{c}, d:**B**, e:{e,i}, f:{f,g,j,k}, g:**F**, h:{h}, i:**E**, j:**F**, k:**F**, l:{l}, m:{m}]

PQ: remove min cost edge e(H,L,5)

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5),e(E,I,5),e(G,K,5),e(H,L,5)}

Dict: [a:**B**, b:{a,b,d}, c:{c}, d:**B**, e:{e,i}, f:{f,g,j,k}, g:**F**, h:{h,l}, i:**E**, j:**F**, k:**F**, l:**H**, m:{m}]

PQ: remove min cost edge e(G,H,6)

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5),e(E,I,5),e(G,K,5),e(H,L,5),e(G,H,6)}

Dict: [a:**B**, b:{a,b,d}, c:{c}, d:**B**, e:{e,i}, f:{f,g,h,j,k,l}, g:**F**, h:**F**, i:**E**, j:**F**, k:**F**, l:**F**, m:{m}]

PQ: remove min cost edge e(K,L,6)

*no change - same tree*

PQ: remove min cost edge e(J,M,6)

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5),e(E,I,5),e(G,K,5),e(H,L,5),e(G,H,6),e(J,M,6)}

Dict: [a:**B**, b:{a,b,d}, c:{c}, d:**B**, e:{e,i}, f:{f,g,h,j,k,l,m}, g:**F**, h:**F**, i:**E**, j:**F**, k:**F**, l:**F**, m:**F**]

PQ: remove min cost edge e(A,E,7)

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5),e(E,I,5),e(G,K,5),e(H,L,5),e(G,H,6),e(J,M,6),e(A,E,7)}

Dict: [a:**B**, b:{a,b,d,e,i}, c:{c}, d:**B**, e:**B**, f:{f,g,h,j,k,l,m}, g:**F**, h:**F**, i:**B**, j:**F**, k:**F**, l:**F**, m:**F**]

(and not ... i:**E** ... as was in the previous version -change continues below)

PQ: remove min cost edge e(C,D,7)

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5),e(E,I,5),e(G,K,5),e(H,L,5),e(G,H,6),e(J,M,6),e(A,E,7),e(C,D,7)}

Dict: [a:**B**, b:{a,b,c,d,e,i}, c:**B**, d:**B**, e:**B**, f:{f,g,h,j,k,l,m}, g:**F**, h:**F**, i:**B**, j:**F**, k:**F**, l:**F**, m:**F**]

PQ: remove min cost edge e(J,K,7), e(K,M,7)

*no change - same tree*

There was meant to be an edge between D and G with weight 8. 8 is there in the sketch, but the edge itself is missing. Selecting that edge joins the two trees together. The MST is then of size 12, and so the algorithm stops. If you didn't consider that edge, then B-F, E-F or E-J would be the next edge, and that would also end the algorithm.

MST: {e(F,J,3), e(F,G,4), e(B,D,4), e(A,B,5),e(E,I,5),e(G,K,5),e(H,L,5),e(G,H,6),e(J,M,6),e(A,E,7),e(C,D,7),e(D,G,8)}

Dict: [a:**F**, b:**F**, c:**F**, d:**F**, e:**F**, f:{a,b,c,d,e,f,g,h,i,j,k,l,m}, g:**F**, h:**F**, i:**F**, j:**F**, k:**F**, l:**F**, m:**F**]

(Note: all entries should now point to F's tree - in previous version some were still pointing to B)

***NOTE - this would take a long time to type out in full during an exam. If the exam was to ask for Kruskal's algorithm, show that you know what data structures to use and what they would look like, show a couple of steps to show that you know what the algorithm is doing and how the data structures change, and then show the final output, listing the edges in the MST in the order you would have added them***

(iv)

for each vertex in the graph

remove the vertex and its associated edges and store them

pick one vertex in the graph

run a depth-first search to get a list of L of vertices reached

if L is not of size n-1 (i.e. at least one vertex in addition to removed one is not reached)

return TRUE

restore the removed vertex and its edges

Single iteration:

O(deg of that vertex) to remove vertex

O(*n+m*) for the DFS

O(deg of that vertex) to replace vertex and edges

So over all iterations, every edge gets removed twice and restored twice, so O(4\*m) = O(m) for edge removals. So complexity comes from the n repeats of DFS.

So complexity is O(n\*(n+m))

END OF PAPER

**The Comparison Count in Quicksort**

The pseudocode in the lecture for Quicksort says that each call to quicksort specifies the pivot and the end of the sublist that is being sorted. We search to the right, beginning with the first item after the pivot, looking for an item bigger than the pivot - count 1 comparison for each item we examine, but once we go beyond the last item in the sublist, do not count a comparison against whatever we find there (since we know we have gone off the end). We search to the left from the last item, looking for smaller items, and count one comparison for each item we examine. Once we reach the pivot, do not count a comparison against the pivot, since we know we have gone too far.

So, when sketching out the progress, if we start with the line

*7* 4 3 **8** *23* 19 25 15 12 18

there are two sublists to search -- first from 7 to 3, and then from 23 to 18

Searching the first sublist, where 7 is the pivot

(searching to the right, starting with 4)

4 against 7 (& keep going)

3 against 7 (& keep going)

off the end of this sublist

(searching to the left, starting at 3)

3 against 7 (wait)

searches crossed, so swap 3 with 7

Now search the 2nd sublist, which starts with pivot 23

(search to the right, starting with 19)

19 against 23 (& keep going)

25 against 23 ( wait)

(search to the left, starting at 18)

18 against 23 (wait)

swap 25 and 18

(continue searching to the right, starting again at 15)

15 against 23 (& keep going)

12 against 23 (& keep going)

25 against 23 (wait)

(continue searching to the left, starting again at 12)

12 against 23 (wait)

searches crossed, so swap 12 with 23

giving

3 4 **7** **8** 12 19 18 15 **23** 25

which took 10 comparisons (count the lines above with 'against' in them)