

**Answer Sheet for On-Line Examinations**

**Summer 2020**

**Student Number: 118364841**

**Module Code: *CS2516***

**Programme and Year of study: Data Science and Analytics II**

(e.g., Arts III, Food Science II)

**Date and Time of Examination 30/04/2020 14:00**

**Place X in box if registered with DSS:**

**Place X in box if availing of a spelling and grammar waiver**

(for marking guidelines, see <https://www.ucc.ie/en/dss/publications/>)

**ALL ANSWERS SUBMITTED WILL BE CHECKED BY TURNITIN**

**Student Declaration**

This is to certify that the work I am submitting is my own and has been done by me solely and not in consultation with anyone else. Neither I nor anyone else have submitted this work for assessment, either at University College Cork or elsewhere. I have read and understood the regulations of University College Cork concerning plagiarism. Where breaches of the declaration are detected, these will be reviewed under UCC student conduct and plagiarism policies. Any breach of the examination rules is a serious issue and can incur penalties.

**Please start your answers on the next page**

Q1.

1. (a) O(n^2)

(b) O(nlogn)

(c) O(n^2)

(ii)

*17 23 15 31 49 30 52 46 (start)*

***15*** *23 17 31 49 30 52 46 (7)*

***15 17*** *23 31 49 30 52 46 (6)*

***15 17 23*** *31 49 30 52 46 (5)*

***15 17 23******30*** *49 31 52 46 (4)*

***15 17 23******30 31*** *49 52 46 (3)*

***15 17 23******30 31 46*** *52 49 (2)*

***15 17 23 30 31 46 49*** *52**(1)*

***15 17 23 30 31 46 49 52*** *(0)*

*Total: 28*

*(iii)*

*17 23 15 31 49 30 52 46 (start)*

*17 23 15 31 30 49 46 52 (4)*

*15 17 23 31 30 46 49 52 (6)*

*15 17 23 31 30 46 49 52 (6)*

***15 17 23 30 31 46 49 52*** *(5)*

*Total: 21*

*(iv)*

*17* 23 15 31 49 30 52 46 (start)

*15* **17** *23* 31 49 30 52 46 (9)

**15 17** **23** *31* 49 30 52 46 (6)

**15 17** **23** *30* **31** *49* 52 46 (6)

**15 17** **23** **30** **31** *46* **49** *52* (4)

**15 17** **23** **30** **31** **46** **49** **52** (0)

Total: 25

(v)

Fastest: mergesort

Slowest: selectionsort

(vi)

Quicksort would be most efficient.

Quicksort worst case time complexity is O(n^2) while mergesort is O(nlogn).

However the expected time complexity of quicksort is O(nlogn) and is usually faster than mergesort.

Quicksort is most efficient in a completely randomised list. It is best before sorting a list with quicksort, to randomise it, because sorting an already sorted list is quicksorts worst case.

Already sorted is the worst case because the list isn’t being partitioned in half on each step, so its reduced slower.

Q2.

(i)

Top level is a list of (vertex, adjacency map) 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, [key1:value1, key2:value2, …] to represent dictionaries.

[(A,[B:e(A,B,7), C:e(A,C,6), D:e(A,D,12), E:e(A,E,5)]),

(B,[A:e(A,B,7), C:e(B,C,8), D:e(B,D,4)]),

(C,[A:e(A,C,6), B:e(B,C,8)]),

(D,[A:e(A,D,12), B:e(B,D,4), E:e(D,E,6)]),

(E,[A:e(A,E,5), D:e(D,E,6)])]

(ii)

(a) Do a linear search for vertex x and find the edge using its dictionary and the vertex y as the key.

(b) do a linear search for vertex x and linearly check each of its edges weights. Return the smallest weighted edge found.

(iii)

Open is the apq.[cost to vertex, vertex]

Output shows the resulting path

Preds: shows the predecessor of vertices

Open: [a]

Closed: []

Preds: [a:None]

Open: [i, h, f, b]

Closed: [a:(0,None)]

Preds: [a:None, f:a, h:a, b:a, i:a]

(remove minimum from open 5:i)

(add adjacent vertices of I not in closed)

Open: [h, f, b,j,l,k]

Closed: [a:(0,None), i:(5,a)]

Preds: [a:None, f:a, h:a, b:a, j:I, l:i, k:I]

(remove minimum from open and preds 6:h)

(add adjacent vertices of h not in closed or update if cost is better than before)

Open: Open: [f, b,j,l,k,]

Closed: : [a:(0,None), i:(5,a), h:(6,a)]

Preds: [a:None, f:a, h:a, b:a, j:I, l:i, k:h]

And so on…

To the end…

Open: []

Closed: […., g:(12,e),…]

Preds: [a:None, …g:e, …]

You can trace out the path by checking the predecessor of each vertex in the path fom g to a.

e.g. g predecessor is e, e predecessor is b, b predecessor is a.

(iv)

I would use a Binary heap.

It allows quick addition and removals from the apq because the elements are sorted in a binary heap with the minimum element always at the top.

Time complexity of remove\_min and adding element is O(logn) because of its binary tree structure. This is efficient.

(v)

A regular person looking at the map would have a general idea of which direction to go by just looking at it, seeing which roads go in the general path of their destination. Therefore they check the roads going the right way only.

However djikstras algorithm checks every possible path from each vertex in each step and does this until we reach the answer of the shortest path.

We could improve the efficiency by computing path weights between all pairs of adjacent vertices at the start and storing in a matrix or table. The matrix stores the cost between all pairs but at the start we just find the adjacent pairs. We can continue the djikstra algorithm using the information we find from the path pairs and update the paths in the table as we continue through the algorithm.

We would have to ensure that the matrix is regularly updated as we go through each step of the matrix.