CSE 2320

Final Exam Review

Instructor : Donna French

Final Exam

The Final Exam will be broken down into 5 quizzes that will need to be taken between midnight, August 16th and midnight August 18th.

This is a total of 48 hours during which you will need to schedule time to take 5 FEQs (**F**inal **E**xam **Q**uizzes).

Each quiz is considered to be 30 minutes worth of the Final Exam which would be 2.5 hours in person.

Final Exam

I break it up into pieces in order to have fewer issues with quiet time and internet connectivity.

You will have to run through the LockDown Browser and Monitor process for each quiz, but please remember that set up time does not count in your quiz time. You will have 30 minutes for each quiz once the quiz actually starts.

If you experience issues and miss a quiz, then your **lowest** score from the other 4 quizzes will be used to replace that missed quiz.

This only applies to one missed quiz – any other missed quizzes will be a 0.

Final Exam

The five quiz scores will be taken together to make a Final Exam grade.

No quiz will be dropped

asking for that is like asking that a certain section of an in person Final Exam be dropped.

The first FEQ – FEQ1 – will be over this week's material

Heaps

Radix Sort

FEQ1 – Radix Sort

Question 1 1 pts

Given this list of integers, show the steps/stages of Radix Sort. Answer **MUST BE** entered using any leading zeros. 003 is correct. 3 is not.

3	73	42	10	99	85	4	956	199

FEQ1 – Radix Sort

Question 1 1 pts

Given this list of integers, show the steps/stages of Radix Sort. Answer **MUST BE** entered using any leading zeros. 003 is correct. 3 is not.

3	73	42	10	99	85	4	956	199
003	073	042	010	099	085	004	956	199

FEQ1 – Radix Sort

Question 1 1 pts

Given this list of integers, show the steps/stages of Radix Sort. Answer MUST BE entered using any leading zeros. 003 is correct. 3 is not.

3	73	42	10	99	85	4	956	199
003	073	042	010	099	085	004	956	199
010	042	003	073	004	085	956	099	199

FEQ1 – Radix Sort

Question 1 1 pts

Given this list of integers, show the steps/stages of Radix Sort. Answer MUST BE entered using any leading zeros. 003 is correct. 3 is not.

3	73	42	10	99	85	4	956	199
003	073	042	010	099	085	004	956	199
010	042	003	073	004	085	956	099	199
003	004	010	042	956	073	085	099	199

FEQ1 – Radix Sort

Question 1 1 pts

Given this list of integers, show the steps/stages of Radix Sort. Answer MUST BE entered using any leading zeros. 003 is correct. 3 is not.

3	73	42	10	99	85	4	956	199
003	073	042	010	099	085	004	956	199
010	042	003	073	004	085	956	099	199
003	004	010	042	956	073	085	099	199
003	004	010	042	073	085	099	199	956

FEQ1 - Heap

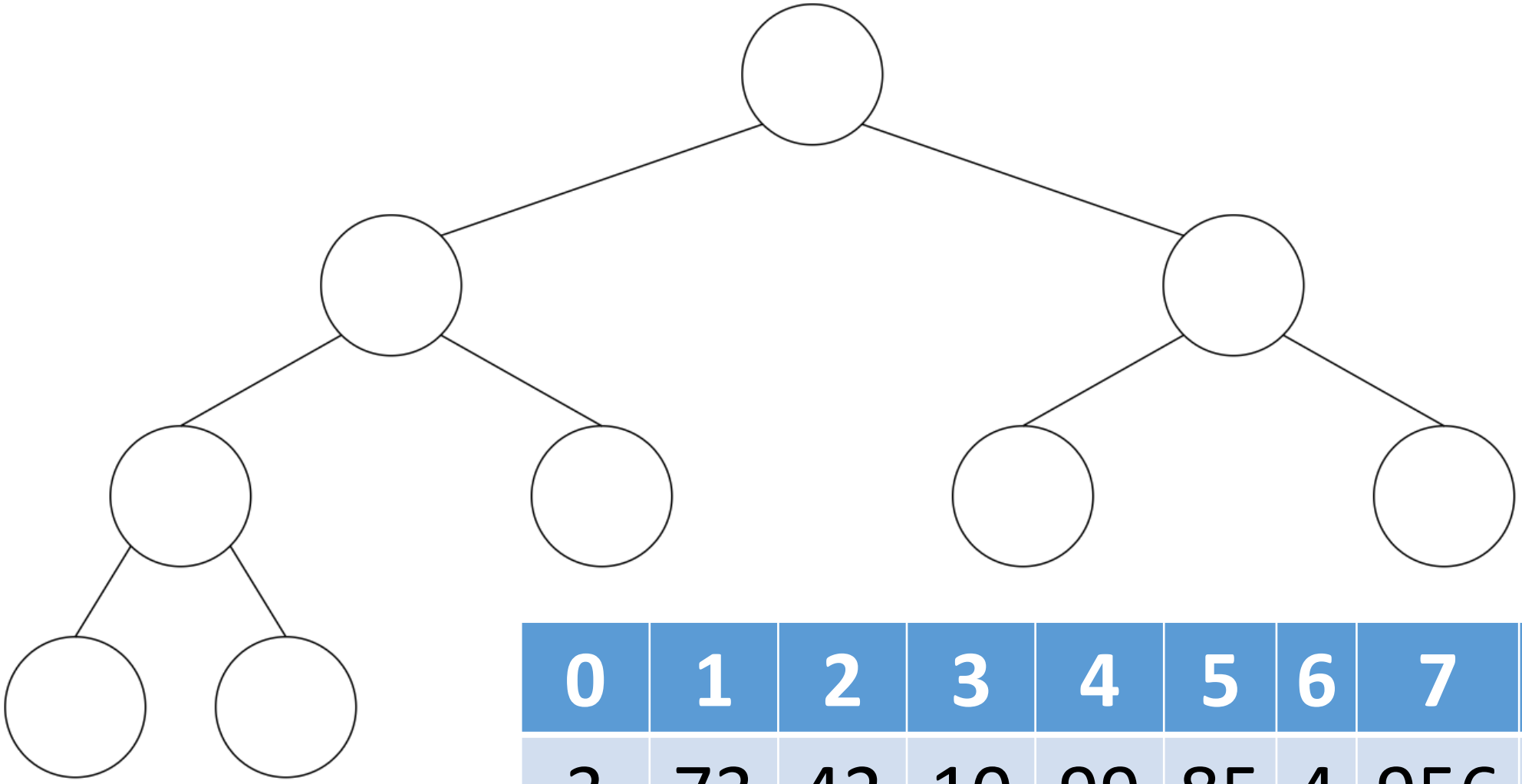
Question 2 1 pts

Given this list of integers, create a max heap. Fill in the array to show how the max heap would be stored.

3,73,42,10,99,85,4,956,199

0	1	2	3	4	5	6	7	8

FEQ1 - Heap



0	1	2	3	4	5	6	7	8
3	73	42	10	99	85	4	956	199

FEQ1 - Heap

Question 2 1 / 1 pts

Given this list of integers, create a max heap. Fill in the array to show how the max heap would be stored.

3,73,42,10,99,85,4,956,199

0	1	2	3	4	5	6	7	8
956	199	85	73	99	42	4	10	3

FEQ2 – Binary Search

Given an array, show the steps taken (the portions of the array that will be searched) during a binary search.

Enter search value 123

Search array -> {14,17,111,114,117,123,128,139,141,152,166,173,714}

{14,17,111,114,117,123}

{114,117,123}

{123}

FEQ2 – Binary Search

Given an array, show the steps taken (the portions of the array that will be searched) during a binary search.

Enter search value 3

Search array -> {2,3,4,10,11,15,40,42,47,49,50}

{2,3,4,10,11}

{2,3}

{3}

FEQ2 – Binary Search

Given an array, show the steps taken (the portions of the array that will be searched) during a binary search.

Enter search value 1835

Search array -> {156,287,323,489,587,654,719,823,954,1834,3472,76387,98981}

{823,954,1834,3472,76387,98981}

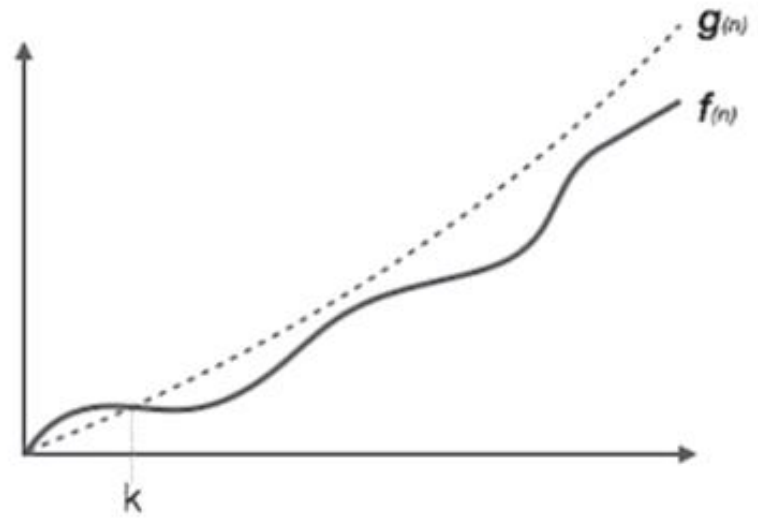
{3472,76387,98981}

{3472}

FEQ2 – Asymptotic Notation

Big Oh Notation, O

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.



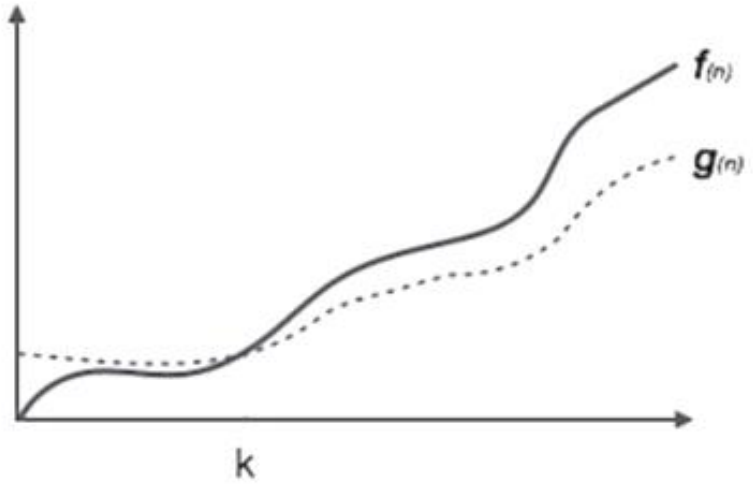
For example, for a function ***f*(n)**

$O(f(n)) = \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } f(n) \leq c.g(n) \text{ for all } n > n_0. \}$

FEQ2 – Asymptotic Notation

Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.



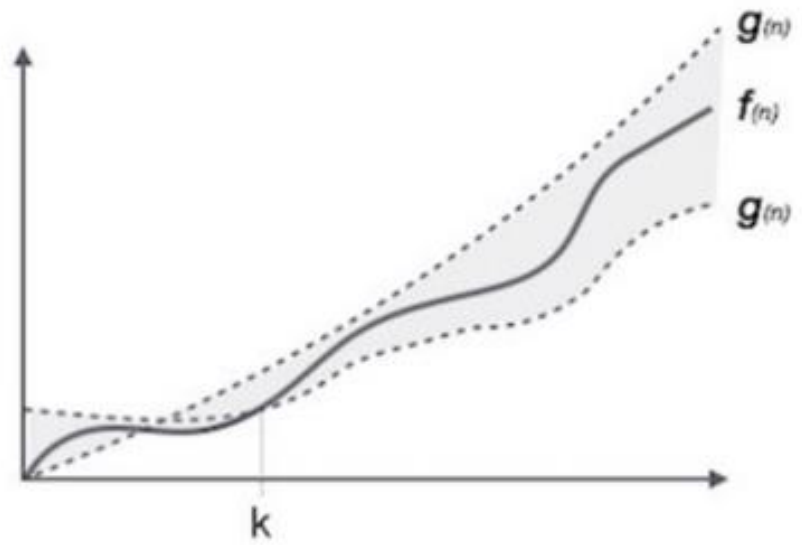
For example, for a function ***f*(n)**

$\Omega(f(n)) \geq \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } g(n) \leq c.f(n) \text{ for all } n > n_0. \}$

FEQ2 – Asymptotic Notation

Theta Notation, θ

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows −



$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0$. $\}$

FEQ2 – Asymptotic Notation

Common Asymptotic Notations

Following is a list of some common asymptotic notations −

constant	–	O(1)
logarithmic	–	O(log n)
linear	–	O(n)
n log n	–	O(n log n)
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$

FEQ2 - Recursion

```c
#include <stdio.h>

#define MIN 24
#define MAX 27

void FunctionR(int a, int b)
{
    if(b > a)
    {
        printf("%d&", a);

        if ((a % 5) == 0)
            printf("\n");

        FunctionR(a+1, b);
    }
}

int main(void)
{
    FunctionR(MIN, MAX);

    return 0;
}
```

24&25&
26&

Pass 1

FunctionR(24,27)

27 > 24, print "24&"

Pass 2

FunctionR(25,27)

27 > 25, print "25&" and newline

Pass 3

FunctionR(26,27)

27 > 26, print "26&"

Pass 4

FunctionR(27,27)

27 ≯ 26

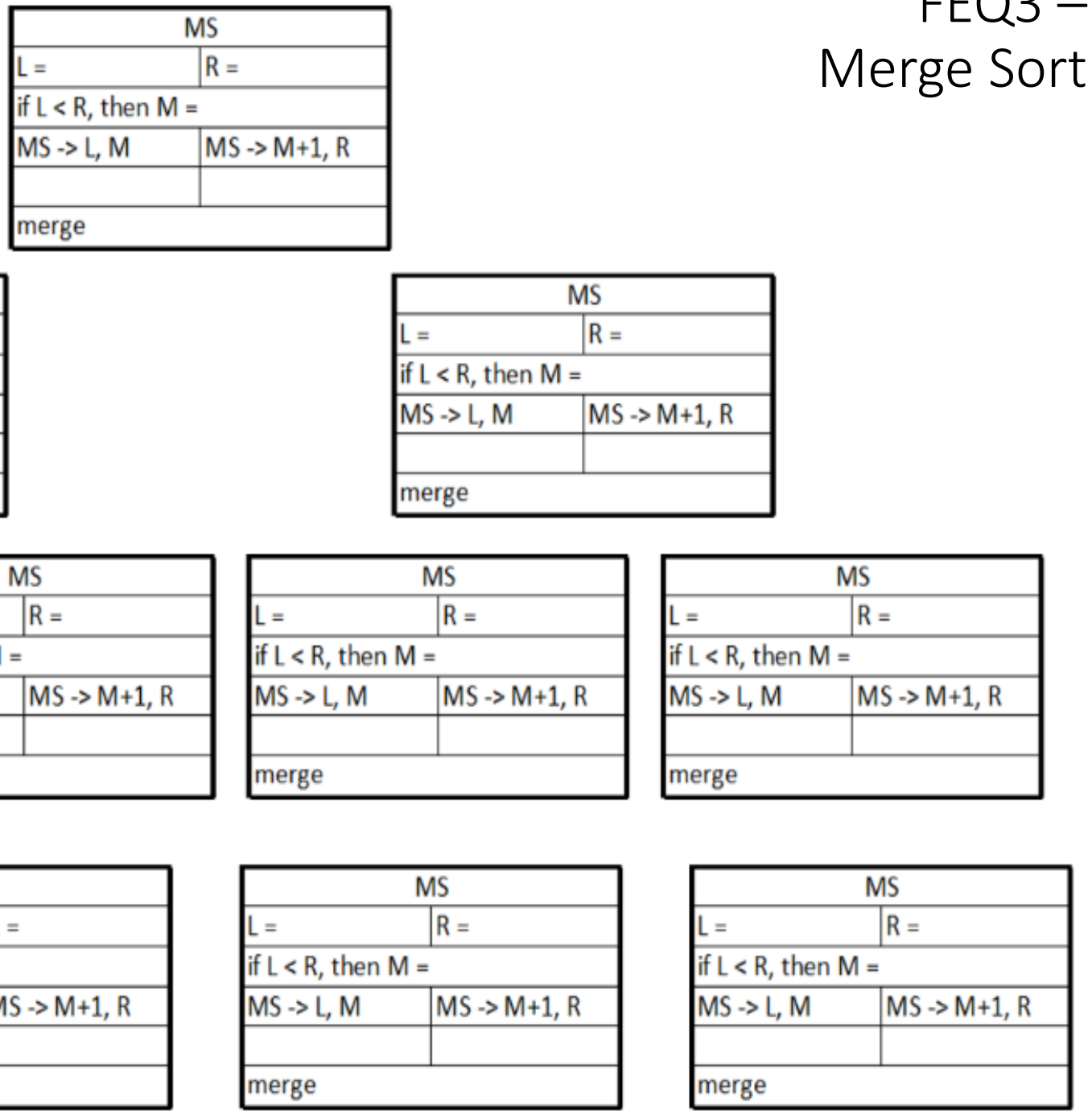
FEQ3 – Merge Sort

The array is {12,11,5,13,7,6}

Given the merge sort code, determine the order of the calls to merge() and label them starting with A. For each call to merge(), fill in the left, middle, right values passed to merge(). Fill in the contents of the L(left) array and R(right) array that merge() creates. Fill in what the array will look like after each call of merge() completes.

FEQ3 – Merge Sort

The array is {12,11,5,13,7,6}



FEQ3 – Merge Sort

The array is {12,11,5,13,7,6}

Given the merge sort code, determine the order of the calls to merge() and label them starting with A. For each call to merge(), fill in the left, middle, right values passed to merge(). Fill in the contents of the L(left) array and R(right) array that merge() creates. Fill in what the array will look like after each call of merge() completes.

Step	left	middle	right	L	R	array
A	0	0	1	12	11	11,12,5,13,7,6
B	0	1	2	11,12	5	5,11,12,13,7,6
C	3	3	4	13	7	5,11,12,7,13,6
D	3	4	5	7,13	6	5,11,12,6,7,13
E	0	2	5	5,11,12	6,7,13	5,6,7,11,12,13

FEQ3 – Quick Sort

Given this array, {9,7,5,12,11}, what will print when the Quick Sort with print function is run?

You MUST include all steps even if they are the same as each other. You are showing what would print.

You are showing that you understand that some of the swaps are swapping numbers with themselves.

Do not leave out any lines - you are showing what the program will print.

Be sure to following the formatting of the print as shown in the code.

FEQ3 – Quick Sort

{9,7,5,12,11}

Swap 9 with 9	**{9,7,5,12,11}**
Swap 7 with 7	**{9,7,5,12,11}**
Swap 5 with 5	**{9,7,5,12,11}**
Final Swap 12 with 11	**{9,7,5,11,12}**
Final Swap 9 with 5	**{5,7,9,11,12}**
Swap 7 with 7	**{5,7,9,11,12}**
Final Swap 9 with 9	**{5,7,9,11,12}**

```
int partition (int A[], int low, int high)
{
      int i, j = 0;
      int pivot = A[high];

      i = (low - 1);

      for (j = low; j < high; j++)
      {
            if (A[j] < pivot)
            {
                  i++;
                  swap(&A[i], &A[j]);
                  printArray(A);
            }
      }
      swap(&A[i + 1], &A[high]);
      printArray(A);

      return (i + 1);
}
```

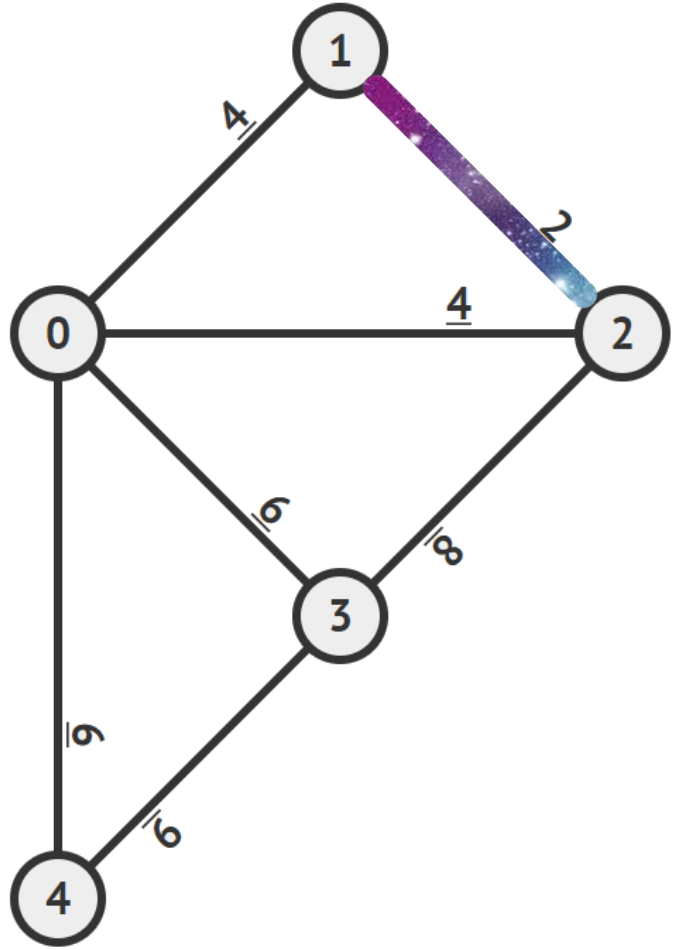
```
void QuickSort(int A[], int low, int high)
{
      if (low < high)
      {

            int ndx = partition(A, low, high);

            QuickSort(A, low, ndx - 1);
            QuickSort(A, ndx + 1, high);
      }
}
```

FEQ4 – Prim's Algorithm

Use Prim's Algorithm to find the MST starting at Vertex 2.



From V2, we can go to

(V2,V1,2)
(V2,V0,4)
(V2,V3,8)

Prim's would order them by weight/cost.

Prim's would pick (V2,V1,2) and remove it from the list

FEQ4 – Prim's Algorithm

From V1, we can go to

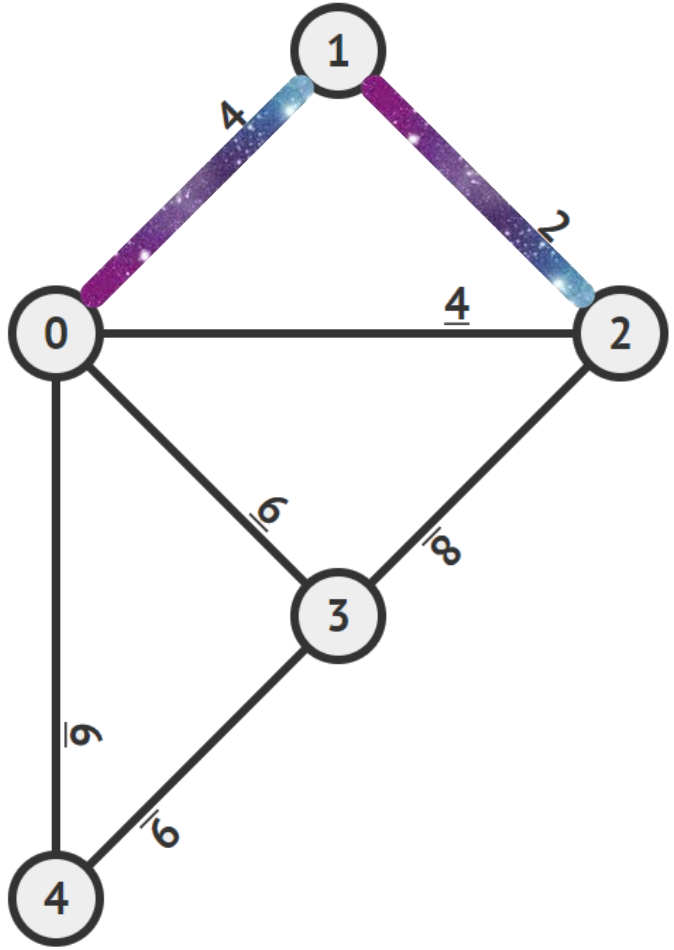
(V1,V0,4)

so it would be added to the list in order.

(V1,V0,4)
(V2,V0,4)
(V2,V3,8)

Prim's would pick (V1,V0,4) and remove it from the list.

(V2,V0,4) causes a cycle and is removed.



FEQ4 – Prim's Algorithm

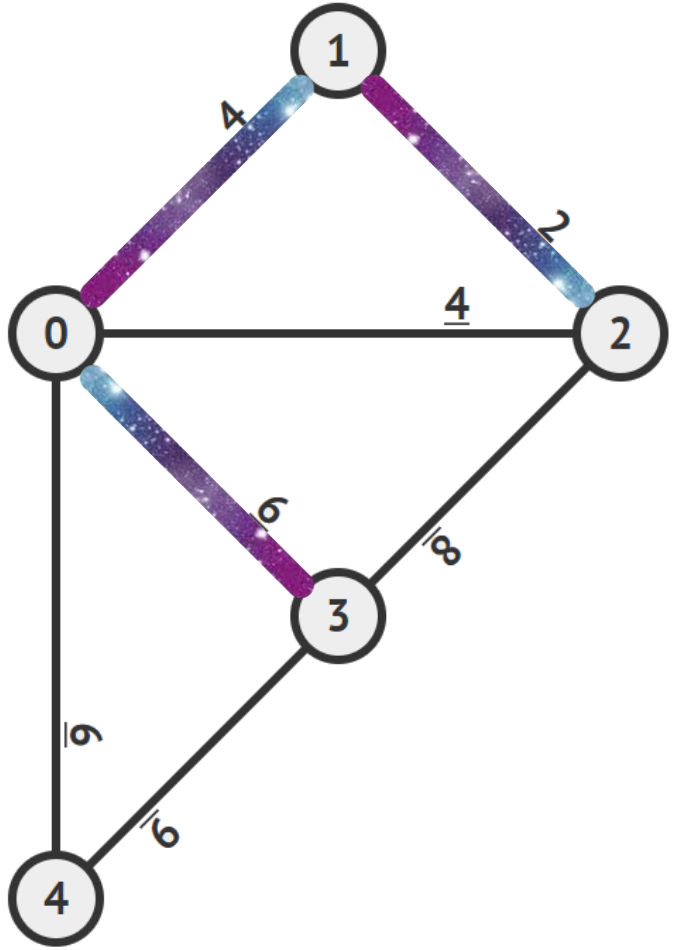
From V0, we can go to

(V0,V2,4)
(V0,V3,6)
(V0,V4,6)

(V0,V2,4) would cause a cycle and would not be added.

(V0,V3,6)
(V0,V4,6)
(V2,V3,8)

Prim's would pick (V0,V3,6) and remove it from the list.



FEQ4 – Prim's Algorithm

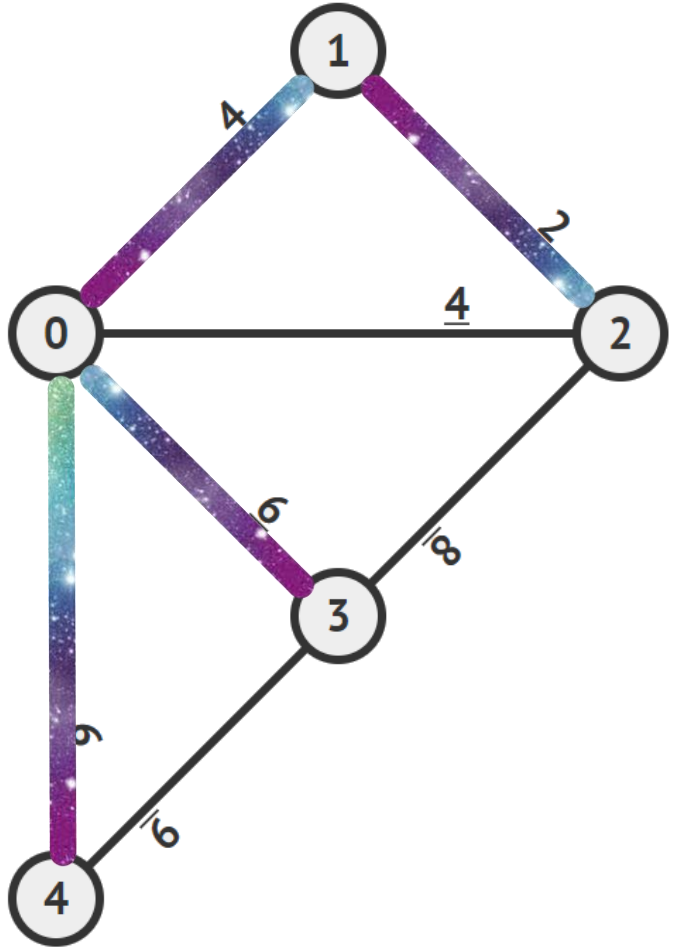
From V3, we can go to

(V3,V2,8)
(V3,V4,9)

(V3,V2,8) would cause a cycle and would not be added.

(V0,V4,6)
(V2,V3,8)
(V3,V4,9)

Prim's would pick (V0,V4,6) and remove it from the list.

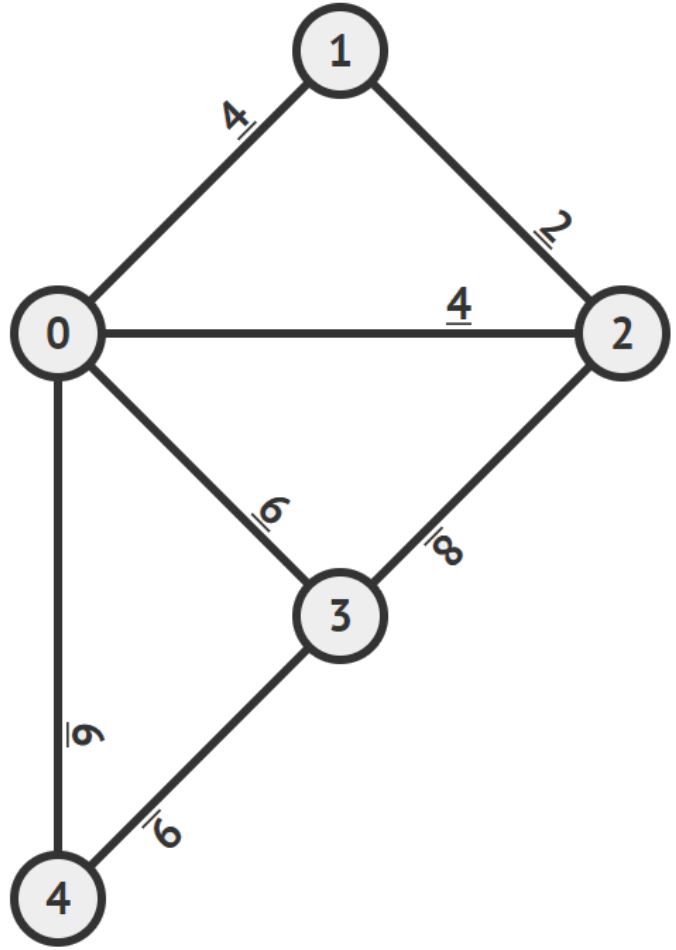


FEQ4 – Kruskal's Algorithm

Use Kruskal's Algorithm to find the MST.

Kruskal will order all edges by weight.

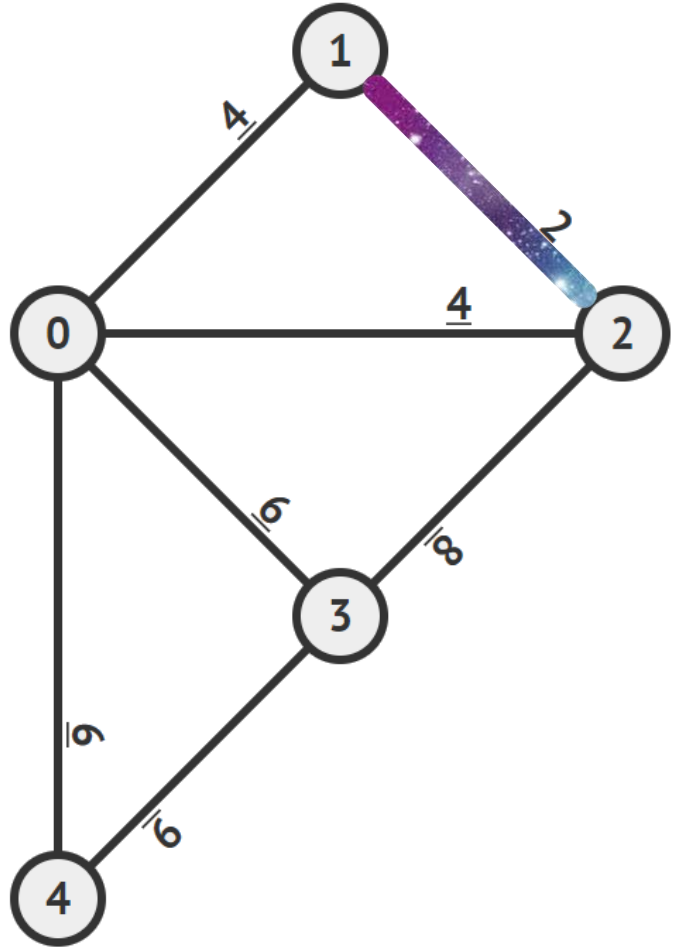
(V1,V2,2)
(V0,V1,4)
(V0,V2,4)
(V0,V3,6)
(V0,V4,6)
(V2,V3,8)
(V3,V4,9)



FEQ4 – Kruskal's Algorithm

Kruskal's then picks the edges with the lowest/least weight/cost.

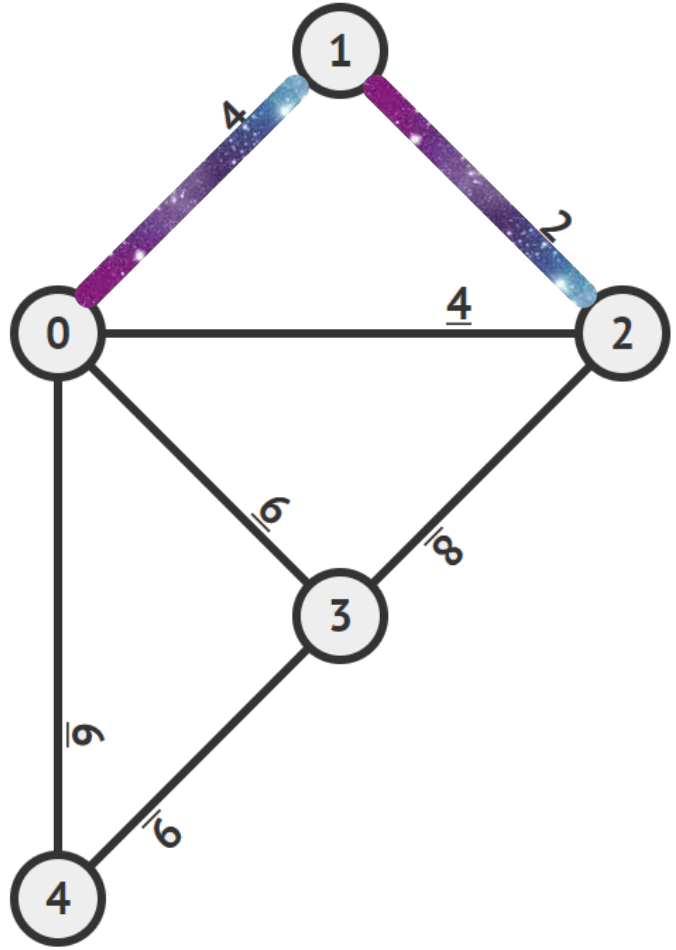
(V1,V2,2)
(V0,V1,4)
(V0,V2,4)
(V0,V3,6)
(V0,V4,6)
(V2,V3,8)
(V3,V4,9)



FEQ4 – Kruskal's Algorithm

Kruskal's then picks the edges with the lowest/least weight/cost.

~~(V1,V2,2)~~
(V0,V1,4)
(V0,V2,4)
(V0,V3,6)
(V0,V4,6)
(V2,V3,8)
(V3,V4,9)

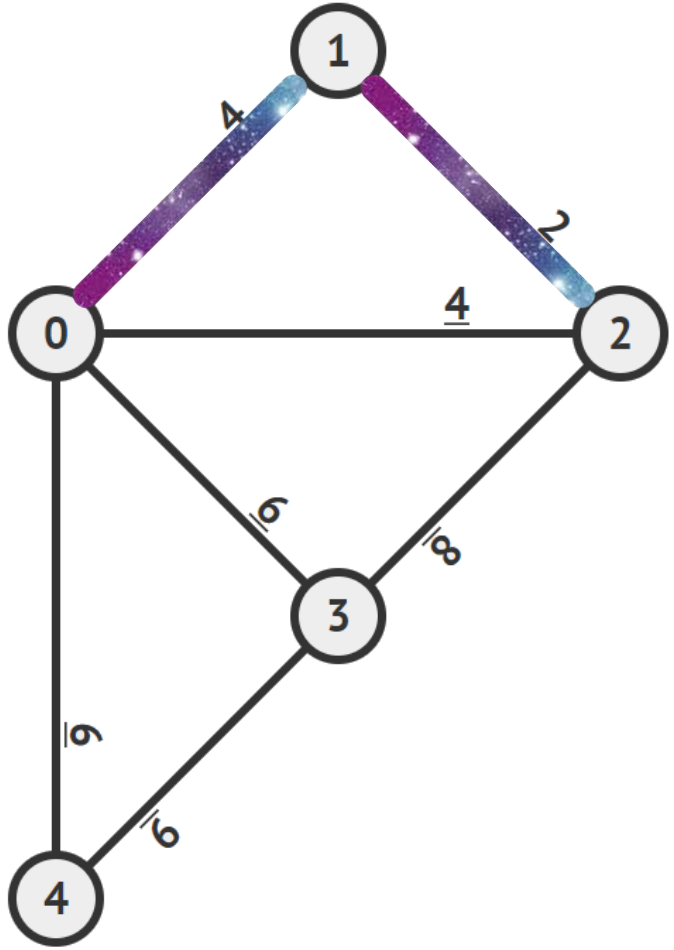


FEQ4 – Kruskal's Algorithm

Kruskal's then picks the edges with the lowest/least weight/cost.

~~(V1,V2,2)~~
~~(V0,V1,4)~~
(V0,V2,4)
(V0,V3,6)
(V0,V4,6)
(V2,V3,8)
(V3,V4,9)

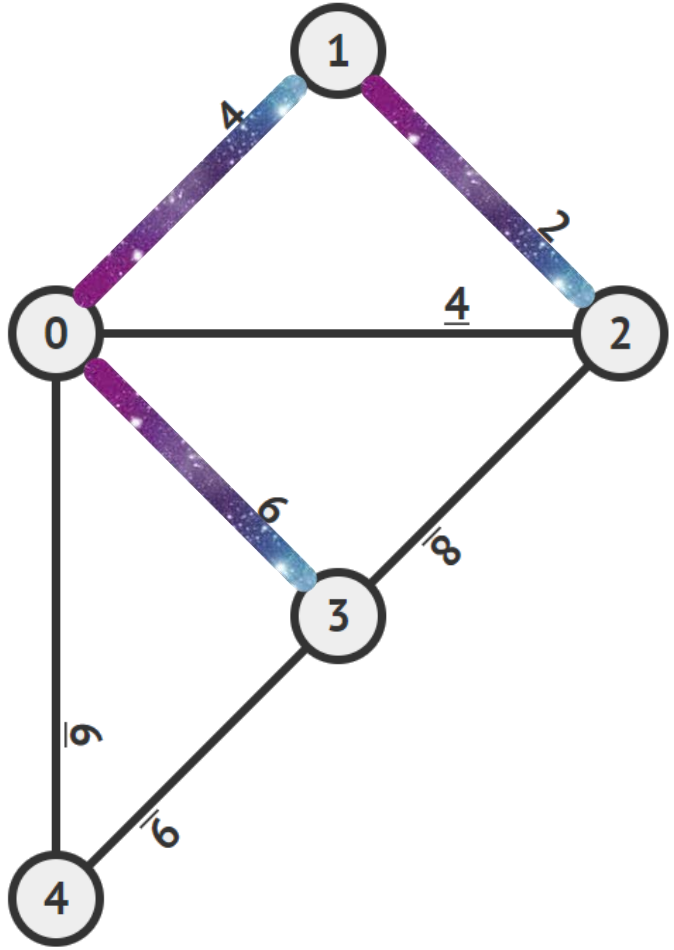
Picking (V0,V2,4) would cause a cycle so it is crossed off



FEQ4 – Kruskal's Algorithm

Kruskal's then picks the edges with the lowest/least weight/cost.

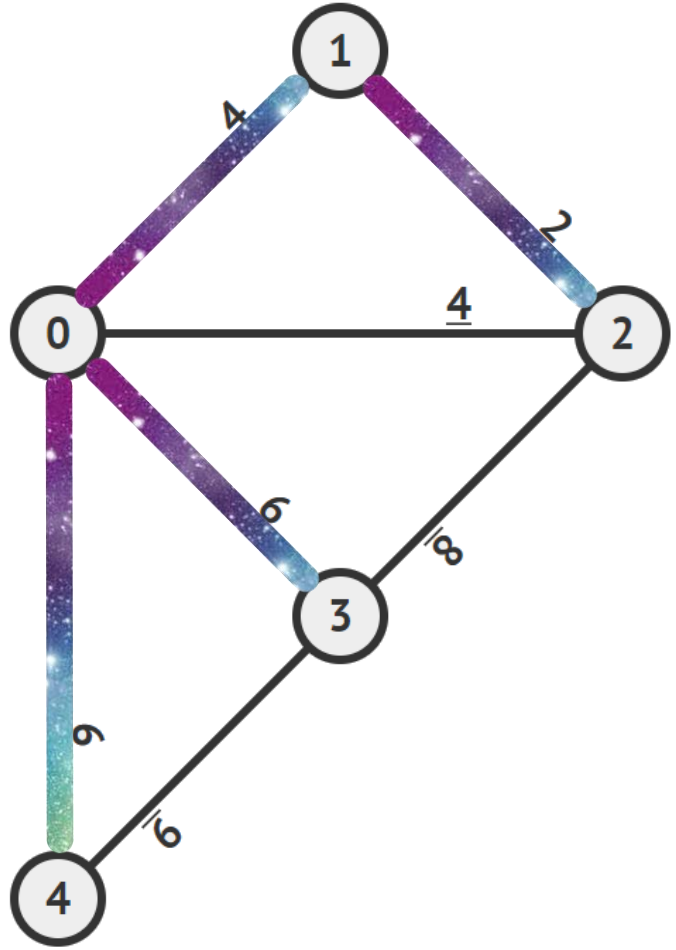
~~(V1,V2,2)~~
~~(V0,V1,4)~~
~~(V0,V2,4)~~
(V0,V3,6)
(V0,V4,6)
(V2,V3,8)
(V3,V4,9)



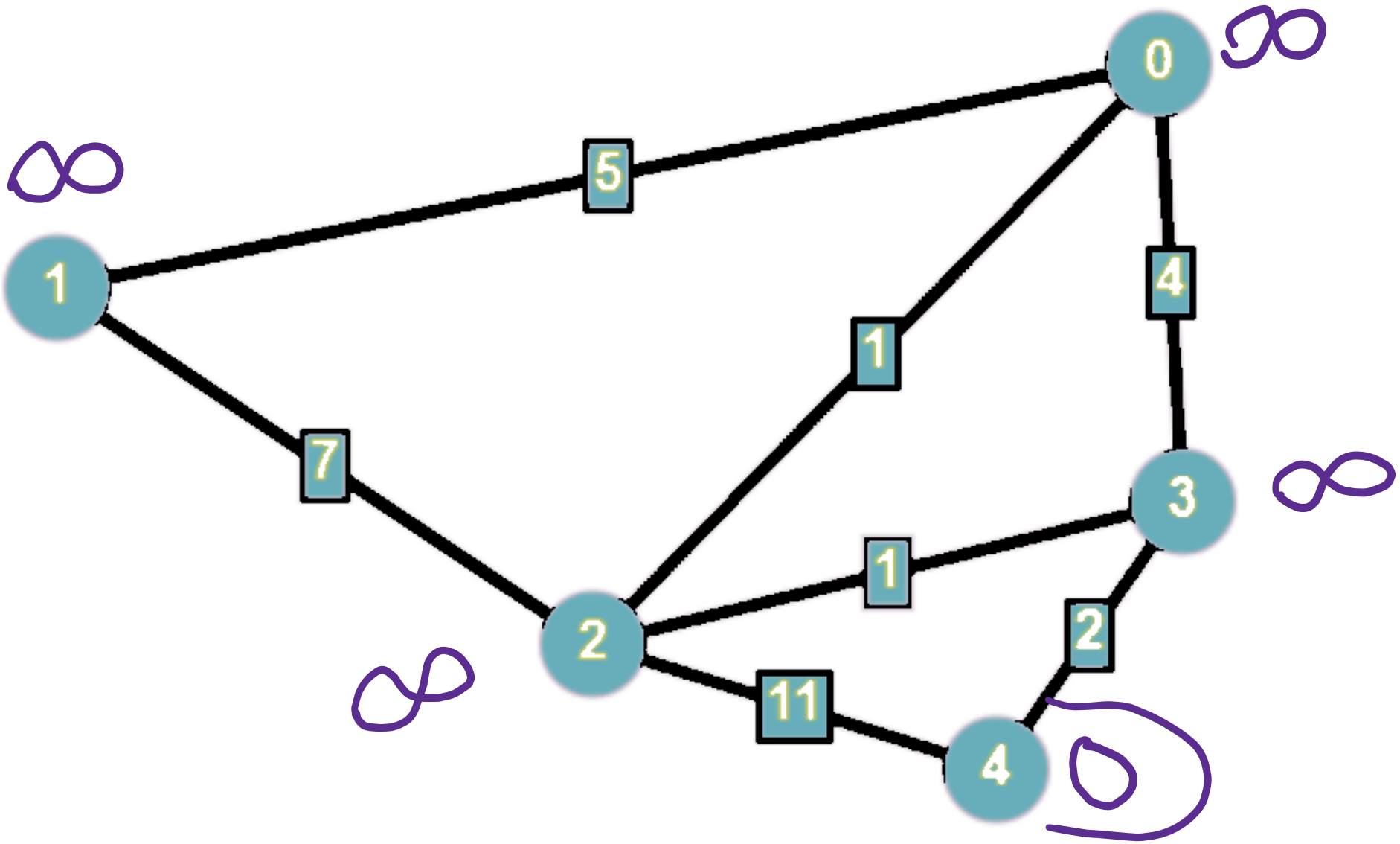
FEQ4 – Kruskal's Algorithm

Kruskal's then picks the edges with the lowest/least weight/cost.

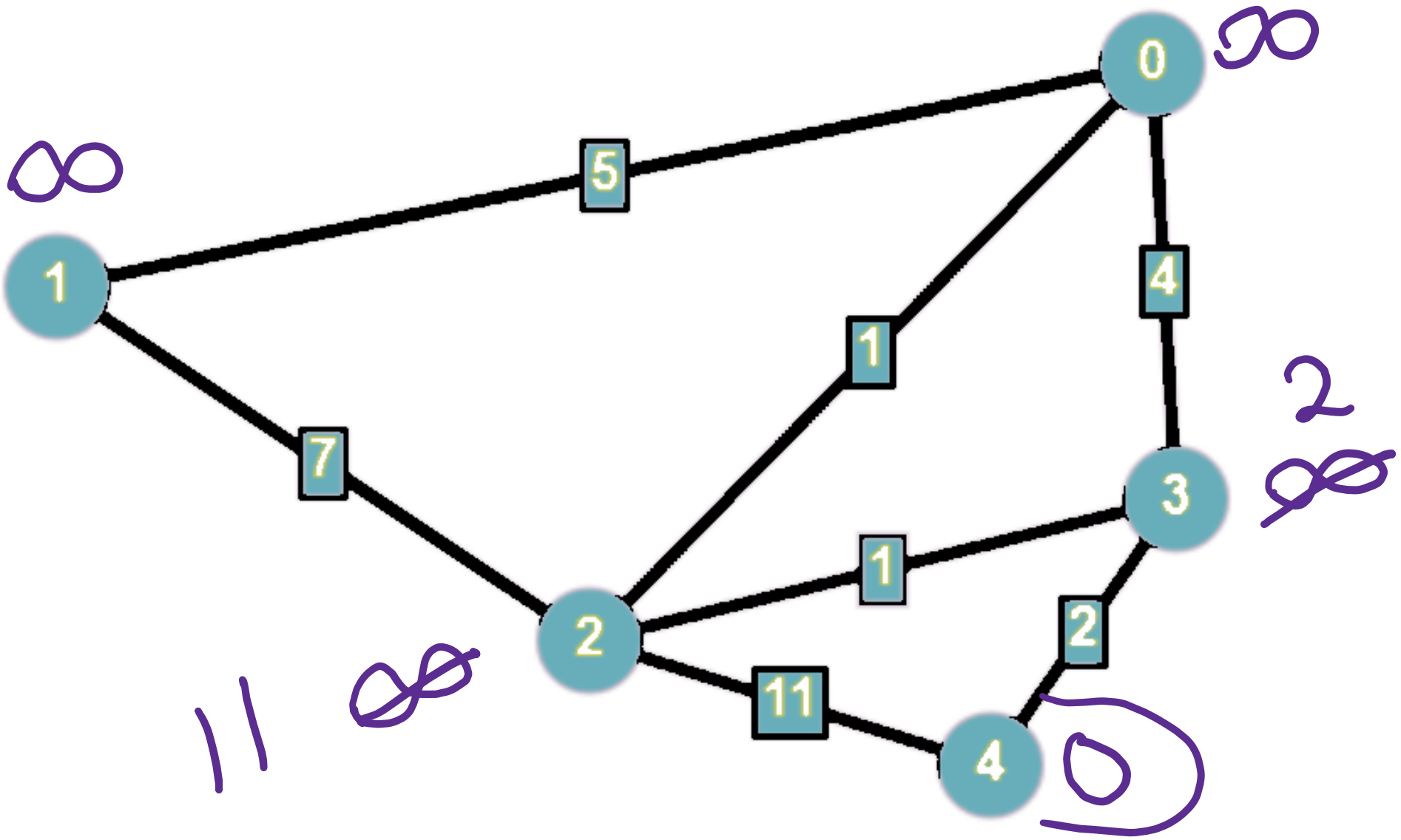
~~(V1,V2,2)~~
~~(V0,V1,4)~~
~~(V0,V2,4)~~
~~(V0,V3,6)~~
(V0,V4,6)
(V2,V3,8)
(V3,V4,9)



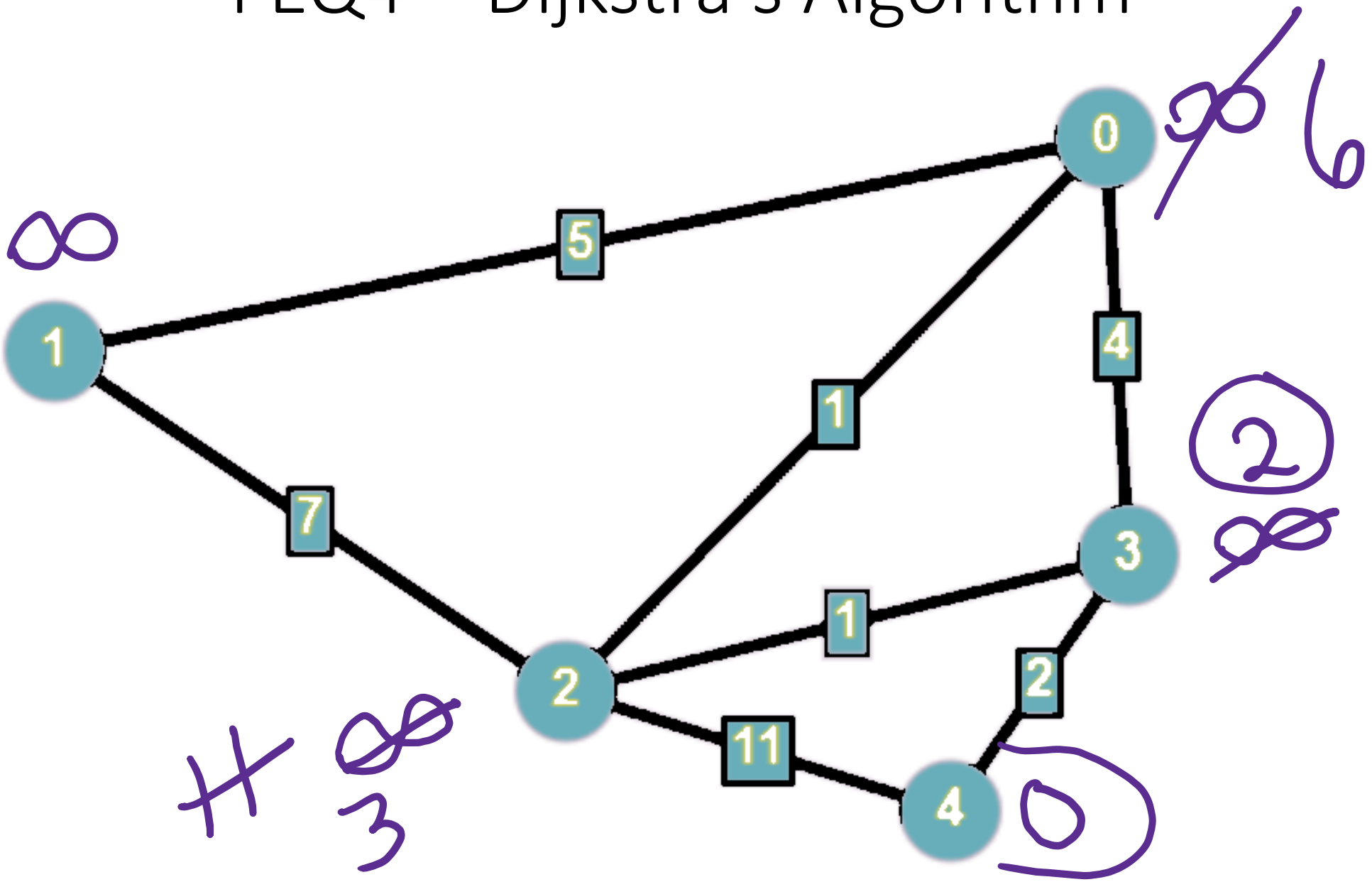
FEQ4 – Dijkstra's Algorithm



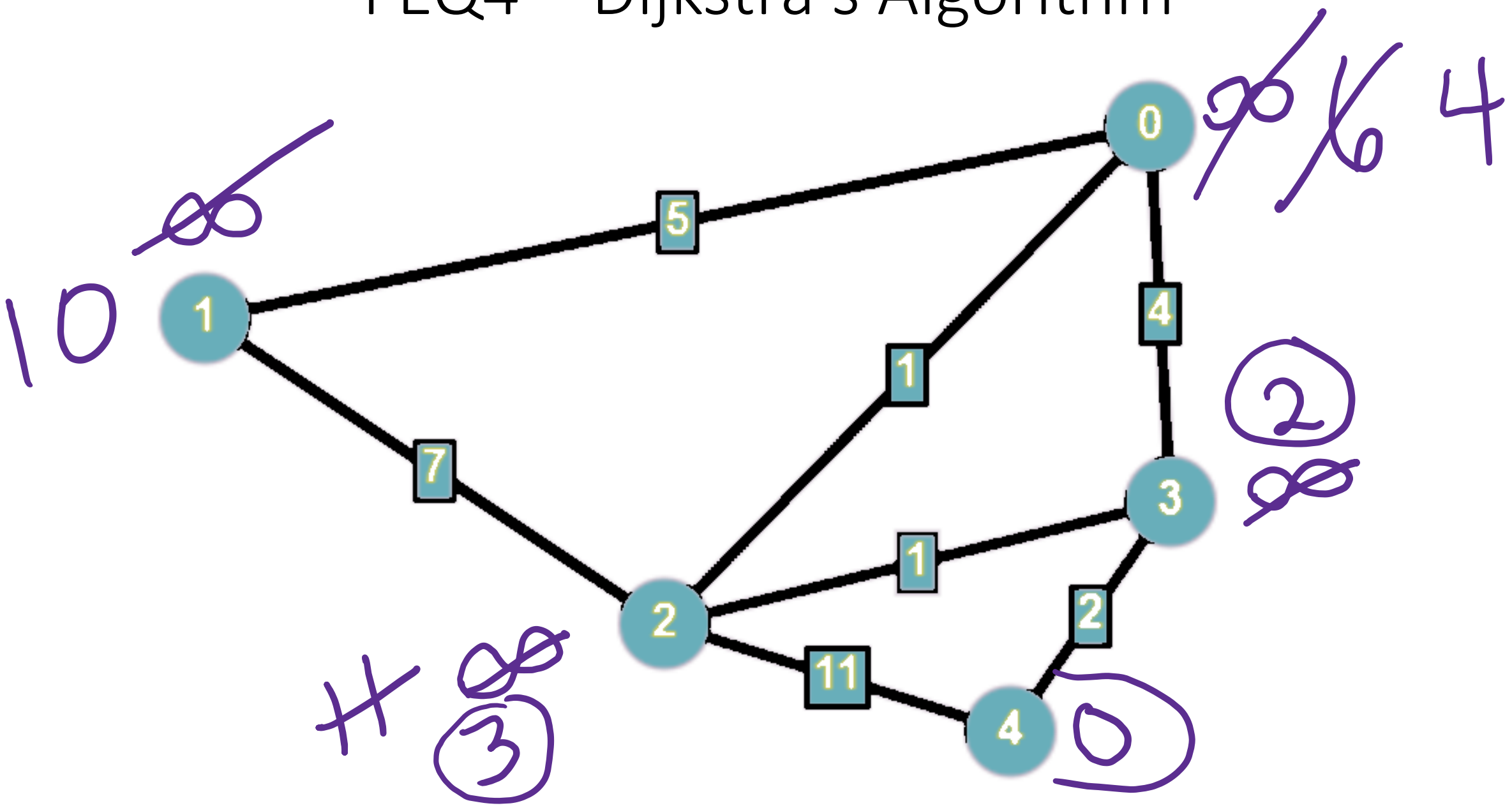
FEQ4 – Dijkstra's Algorithm



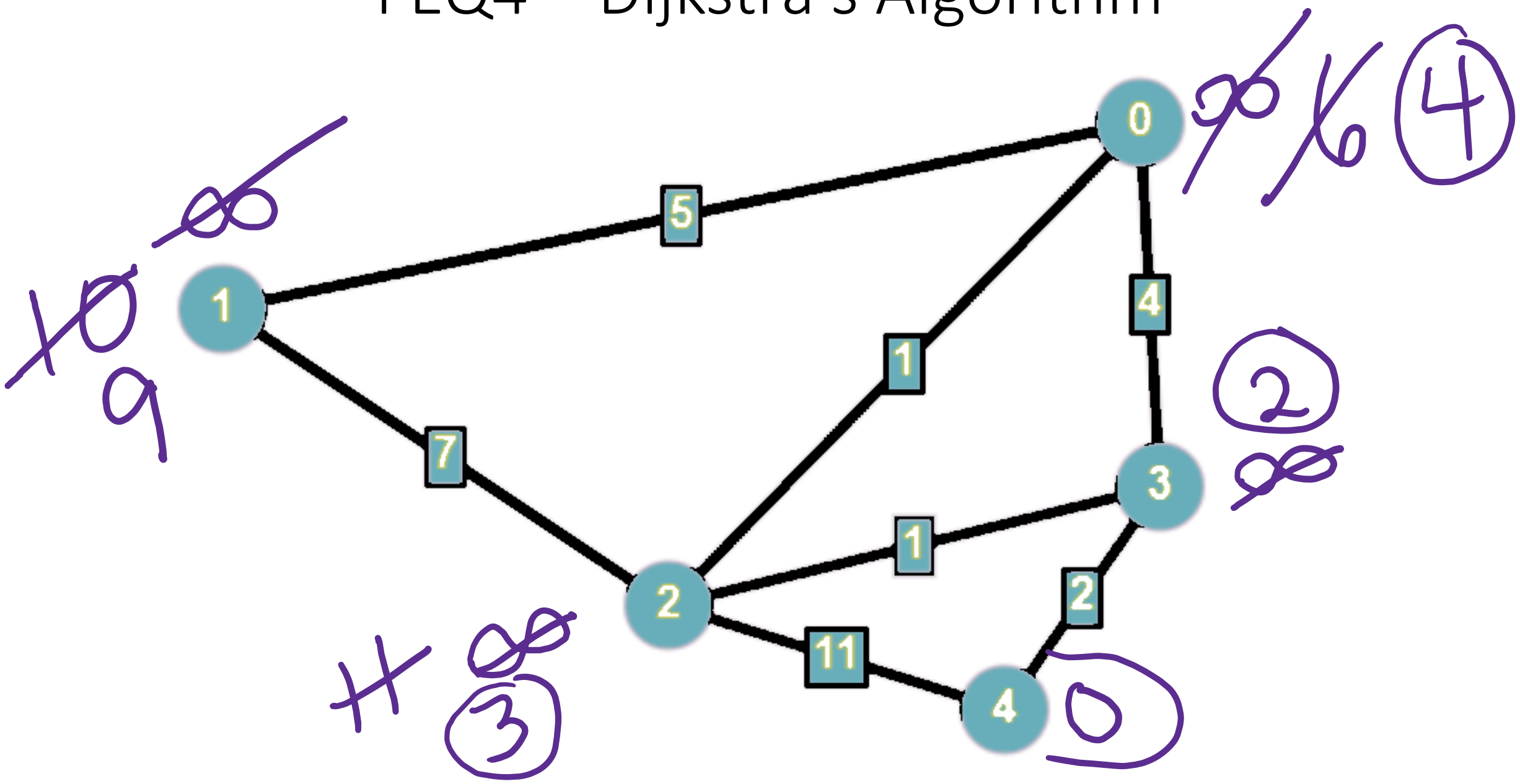
FEQ4 – Dijkstra's Algorithm



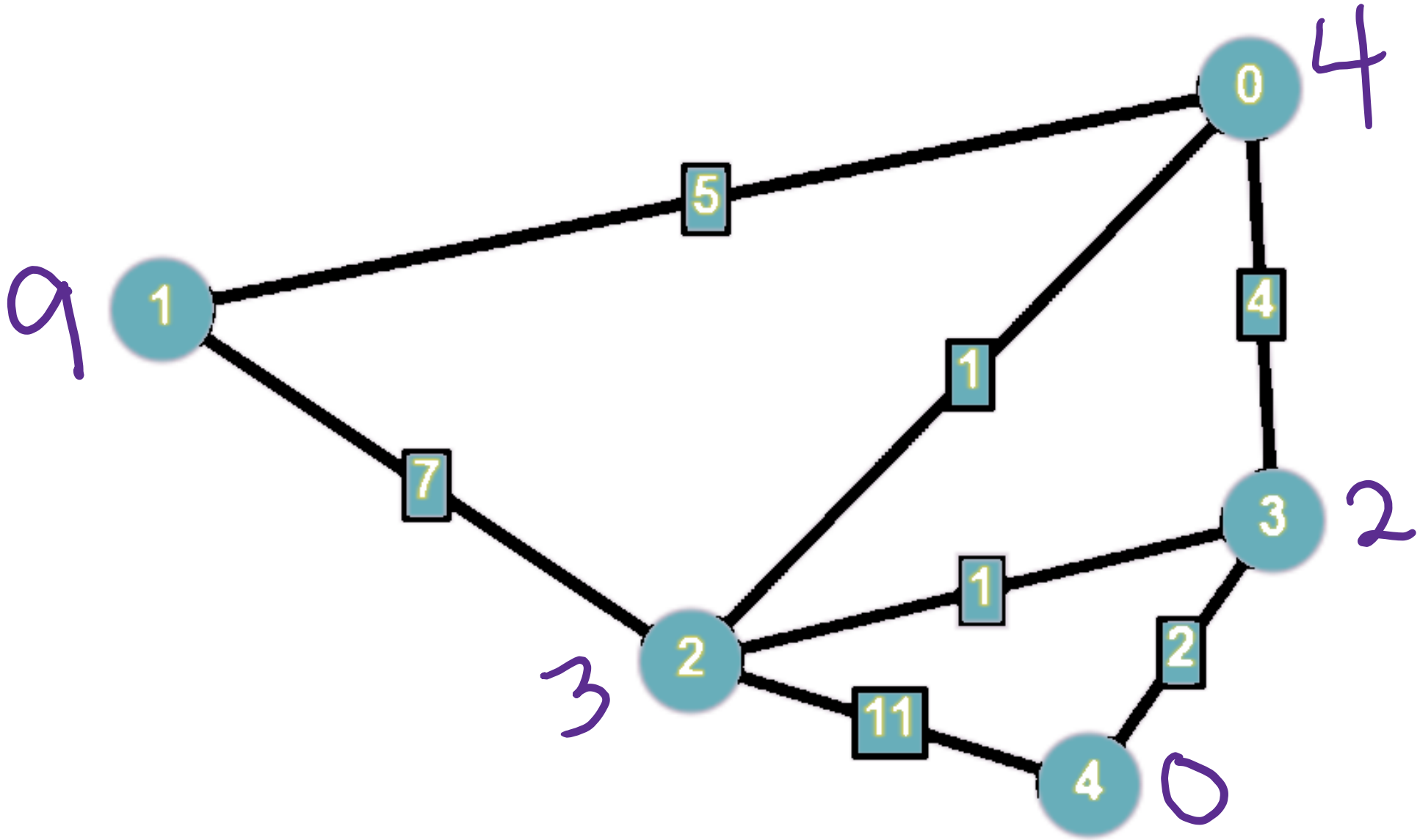
FEQ4 – Dijkstra's Algorithm



FEQ4 – Dijkstra's Algorithm

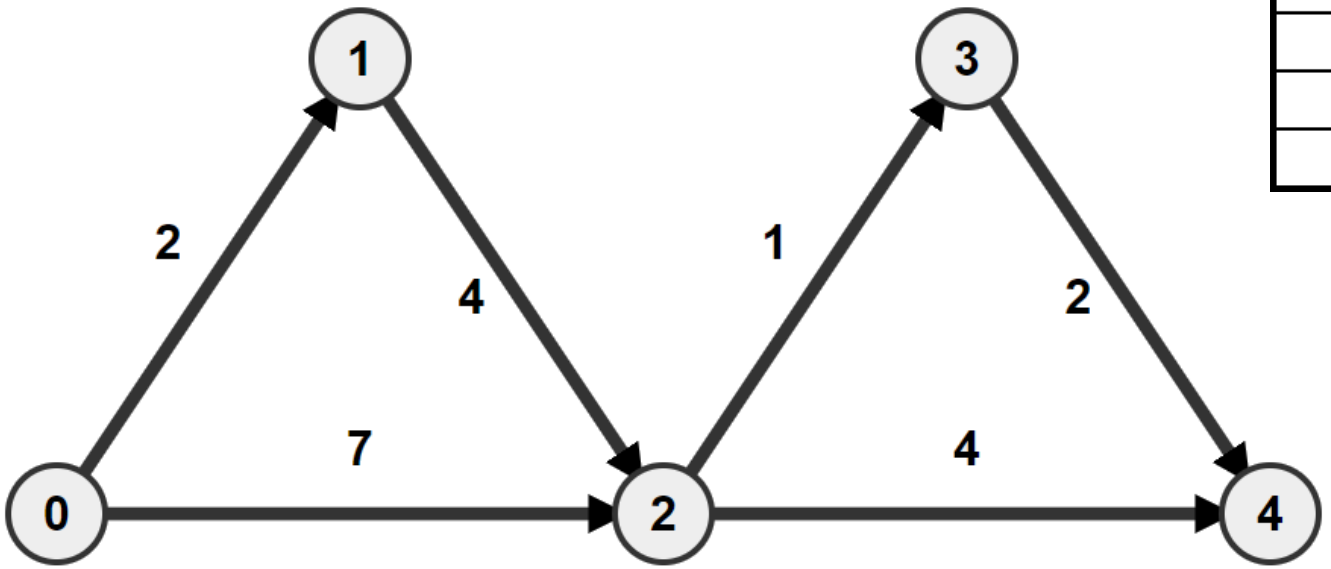


FEQ4 – Dijkstra's Algorithm



FEQ5 – Adjacency List and Matrix

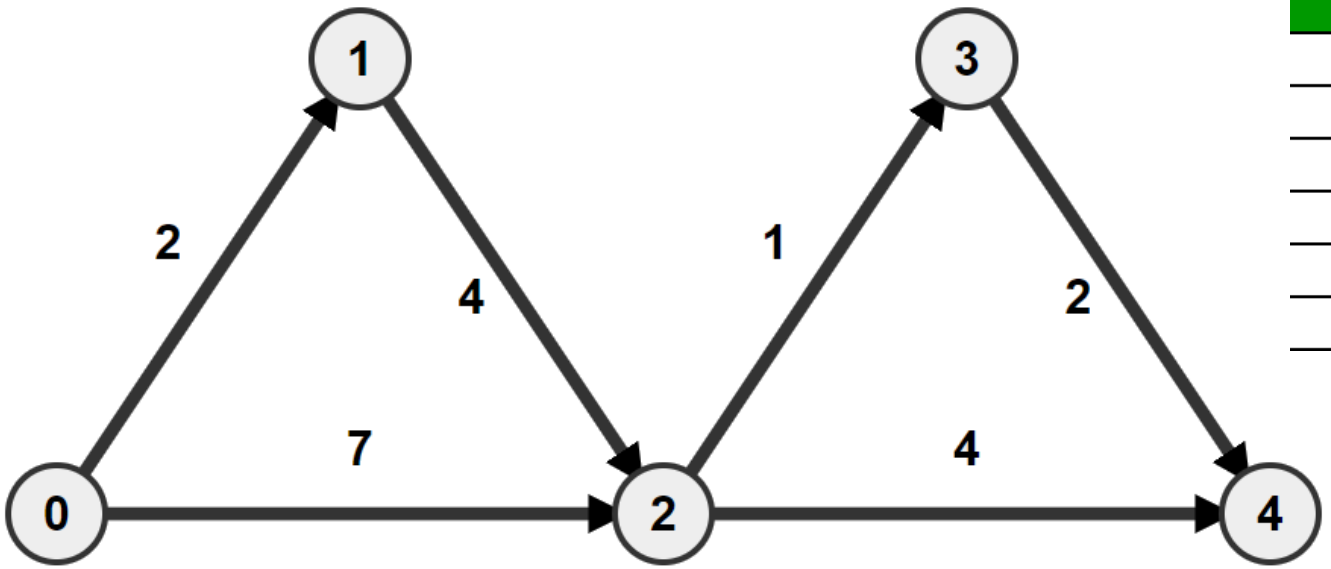
Given a graph, create the adjacency list.



Adjacency List		
0:	(1, 2)	(2, 7)
1:	(2, 4)	
2:	(3, 1)	(4, 4)
3:	(4, 2)	
4:		

FEQ5 – Adjacency List and Matrix

Given a graph, create the adjacency matrix.



Adjacency Matrix					
	0	1	2	3	4
0	-1	2	7	-1	-1
1	-1	-1	4	-1	-1
2	-1	-1	-1	1	4
3	-1	-1	-1	-1	2
4	-1	-1	-1	-1	-1

FEQ5

Question 1

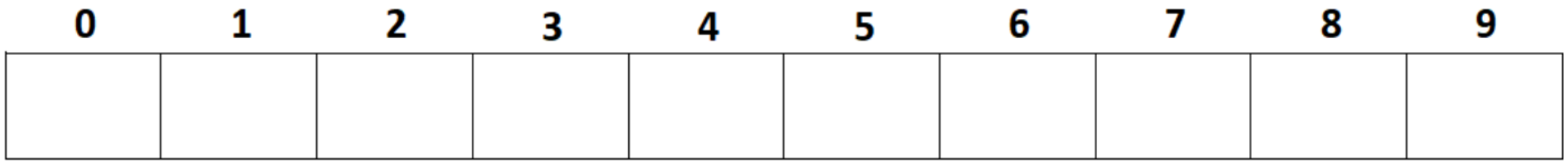
1 pts

Given a list of items to be stored in the hash table and the hashing function, show what the array will look like after all items have been hashed and stored when using open addressing. The array in use for this question has 11 elements.

List : Jan, Tim, Mia and Sue.

Hash function : add ASCII values and MOD with array size to get an array index.

Jan(281), Tim(296), Mia(279), Sue(301)



Jan [Choose]

Sue [Choose]

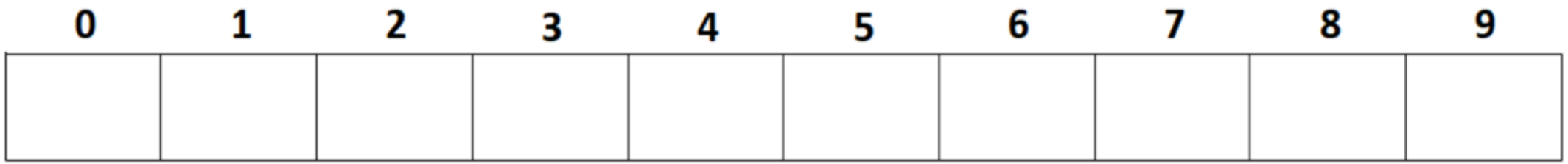
Tim [Choose]

Mia [Choose]

- [Choose]
- Array element 2
- Array element 5
- Array element 9
- Array element 7
- Array element 6
- Array element 3
- Array element 8
- Array element 0
- Array element 1
- Array element 4

FEQ5

Jan(281), Tim(296), Mia(279), Sue(301)



Jan	Array element 1
Sue	Array element 2
Tim	Array element 6
Mia	Array element 9

FEQ5

Question 1

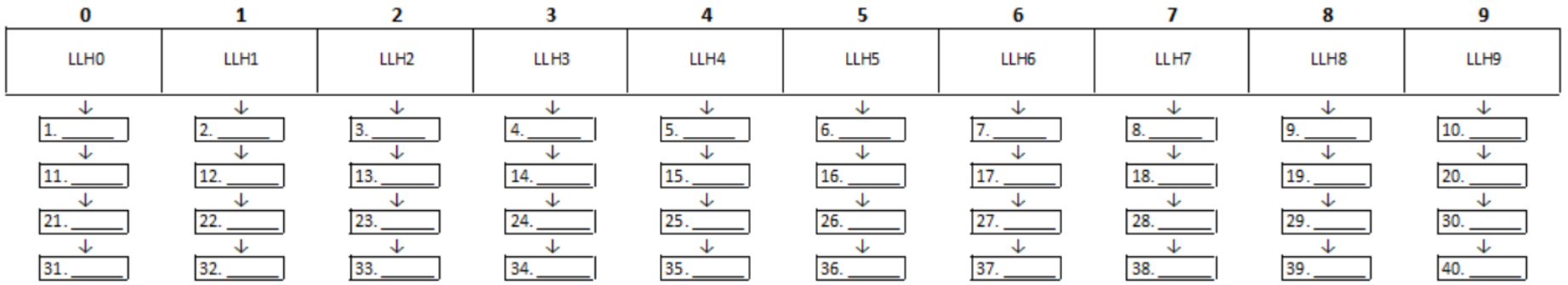
1 pts

Given a list of items to be stored in the hash table and the hashing function, show what the array will look like after all items have been hashed and stored when using separate chaining. The array in use for this question has 11 elements.

List : Jan, Tim, Mia and Sue.

Hash function : add ASCII values and MOD with array size to get an array index.

Jan(281), Tim(296), Mia(279), Sue(301)



Jan [Choose]

Tim [Choose]

Mia [Choose]

Sue [Choose]

Check for a scroll bar on the drop down box.

Good Websites for Algorithm Stuff

Desmos

www.desmos.com/calculator

Know Thy Complexities

https://www.bigocheatsheet.com/

Dijkstra, Breadth-first, Minimum Spanning Tree

https://graphonline.ru/en/

Good Websites for Algorithm Stuff

VisuAlgo – adjacency matrix, adjacency list, edge list, directed, undirected, weighted

https://visualgo.net/en/graphds

Tool for drawing graphs

https://app.diagrams.net/

Recursion Problems with Solutions

https://www.techiedelight.com/recursion-practice-problems-with-solutions/

Binary Heaps

http://btv.melezinek.cz/binary-heap.html