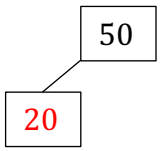
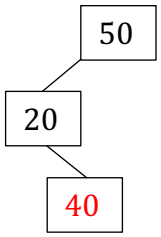
Q1a)

50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25

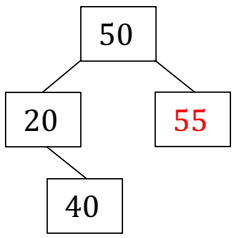
50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25



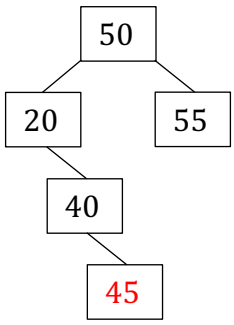
50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25



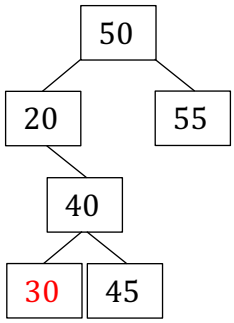
50, 20, 40, 55, 45, 30 10, 15, 35, 60, 5, 25



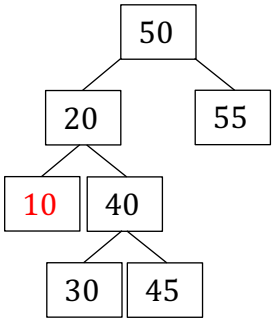
50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25



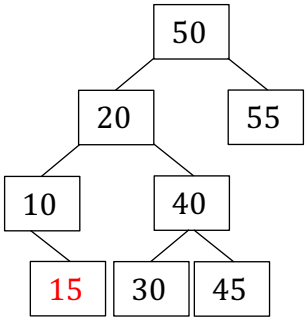
50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25



50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25

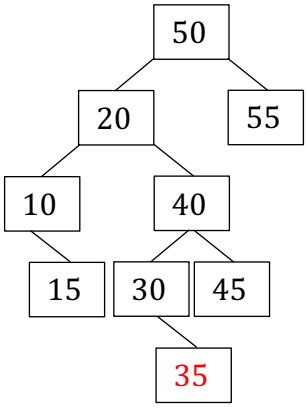


50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25

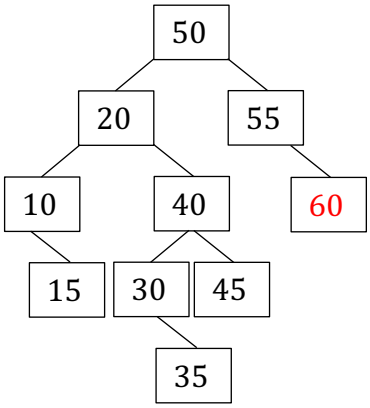


Zafir Khalid - 40152164

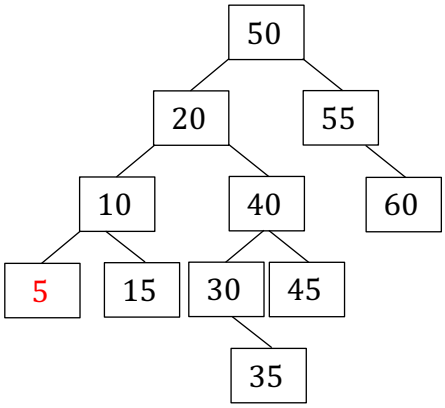
50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25



50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25

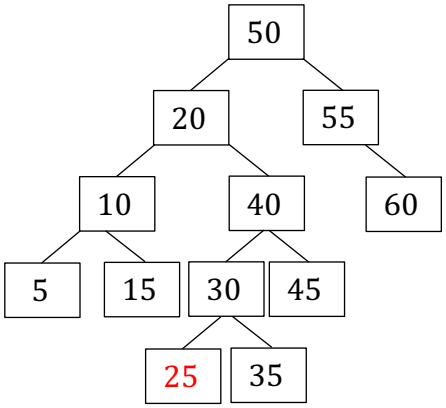


50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25



Zafir Khalid - 40152164

50, 20, 40, 55, 45, 30, 10, 15, 35, 60, 5, 25

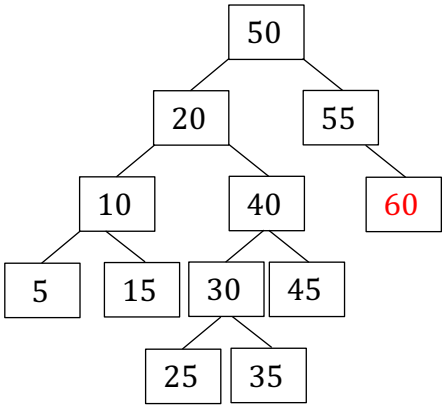


Q1b)

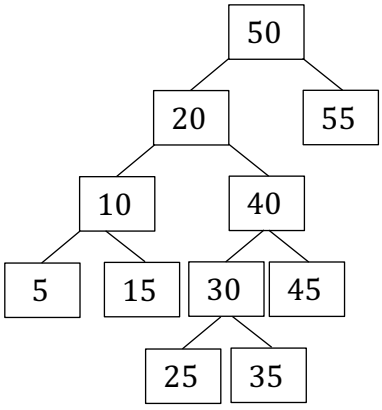
Pre-order: 50, 20, 10, 5, 15, 40, 30, 25, 35, 45, 55, 60
In-order: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60
Post-order: 5, 15, 10, 25, 35, 30, 45, 40, 20, 60, 55, 50

Q1c)

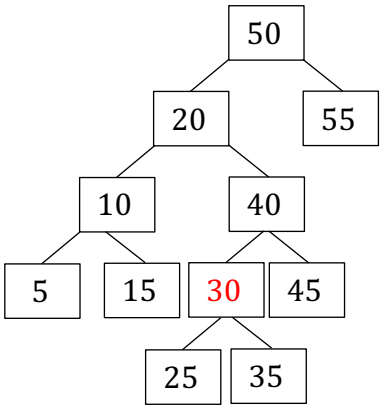
Delete 60



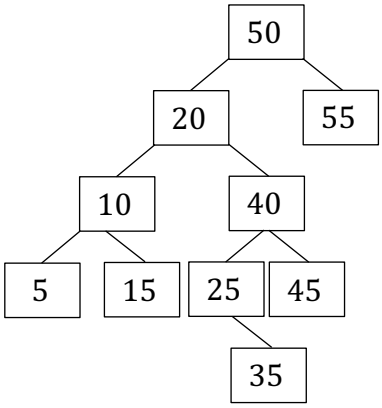
Leaf Node, directly delete



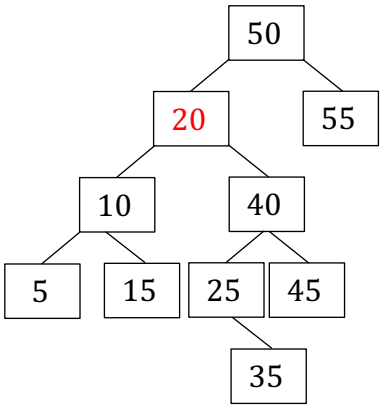
Delete 30



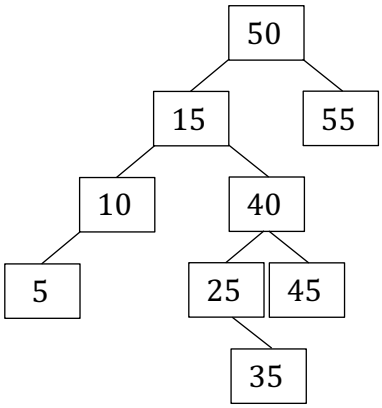
Internal Node, replace by maximum in left sub tree



Delete 20



Internal Node, replace by maximum in left sub tree

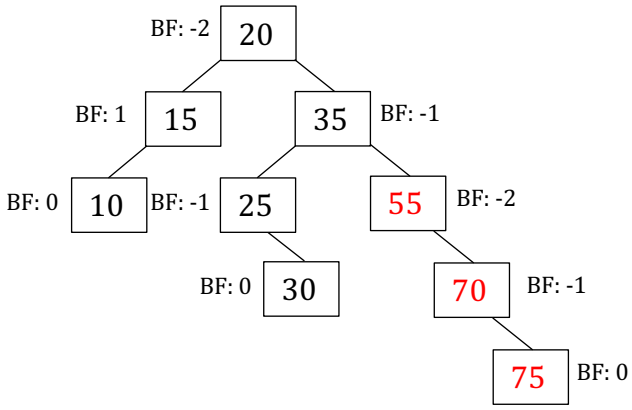


Q1d)

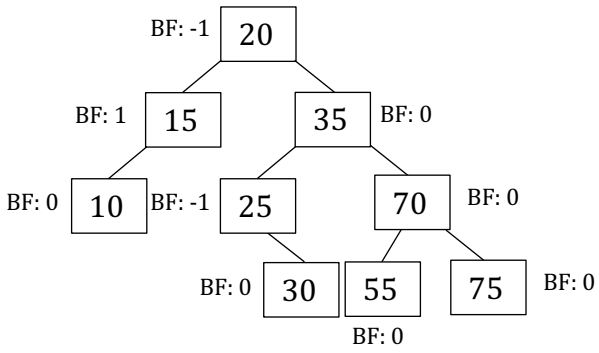
Pre-order:	50, 15, 10, 5, 40, 25, 35, 45, 55
In-order:	5, 10, 15, 25, 35, 40, 45, 50, 55
Post-order:	5, 10, 35, 25, 45, 40, 15, 55, 50

2a)

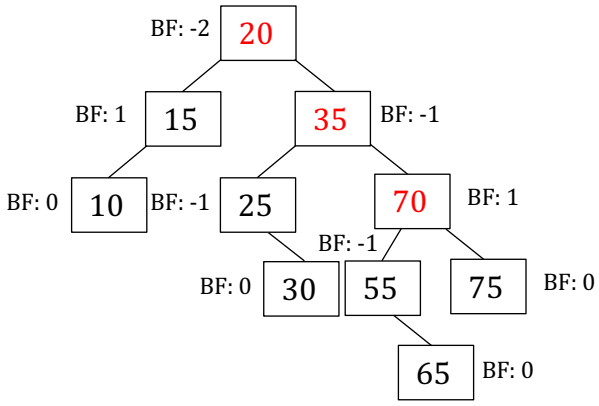
Add 75



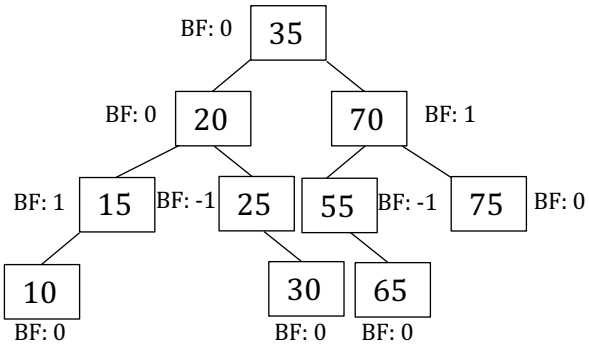
Refactor & Restore AVL property



Add 65

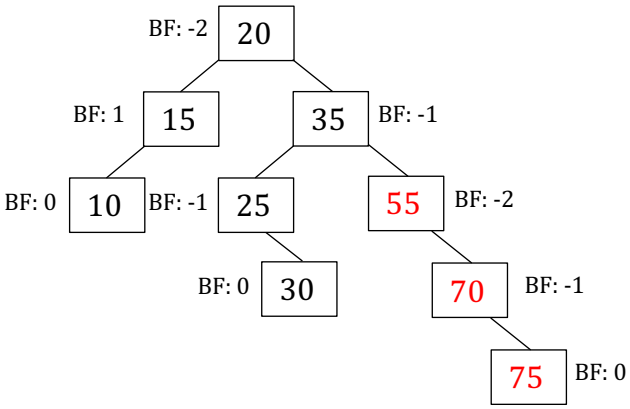


Refactor & Restore AVL Property

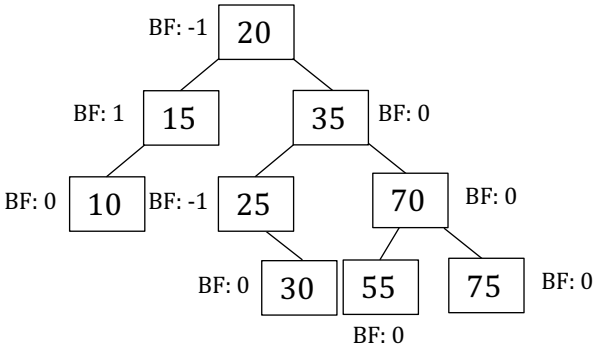


2b)

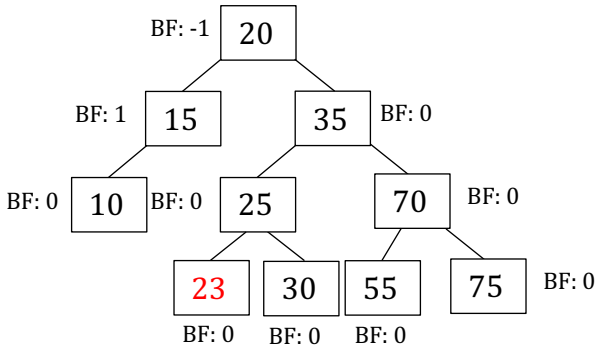
Add 75



Refactor & Restore AVL property

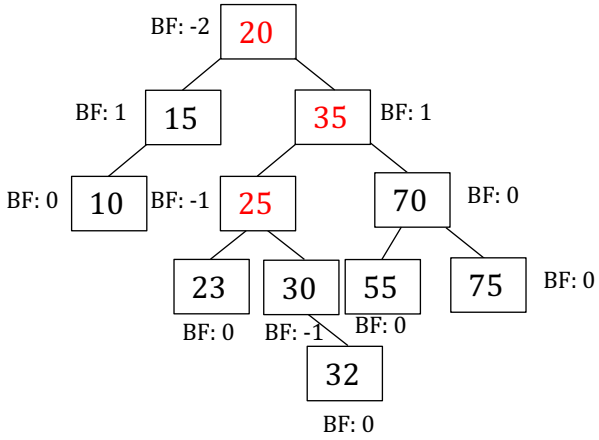


Add 23

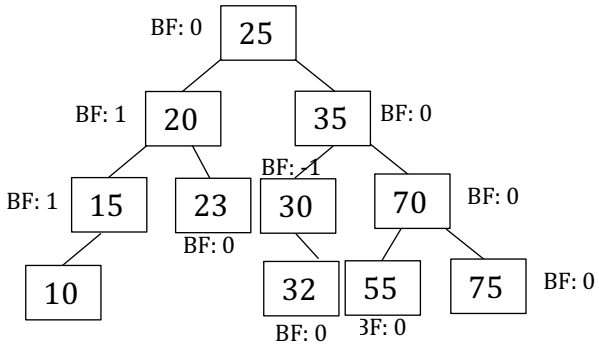


No need to Refactor & Restore AVL Property

Add 32

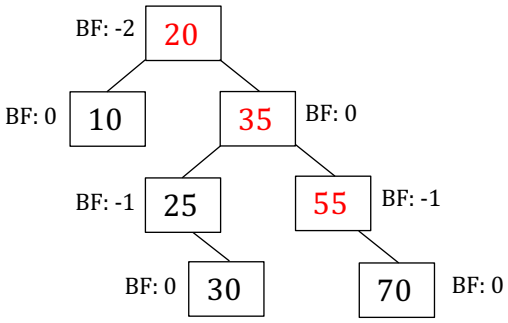


Add 32

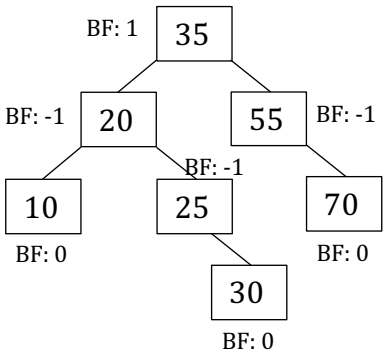


2c)

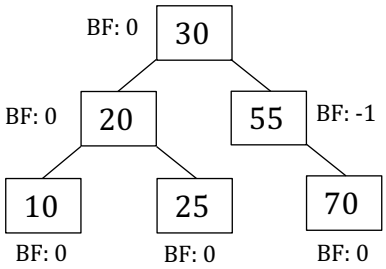
Remove 15



Refactor & Restore AVL property



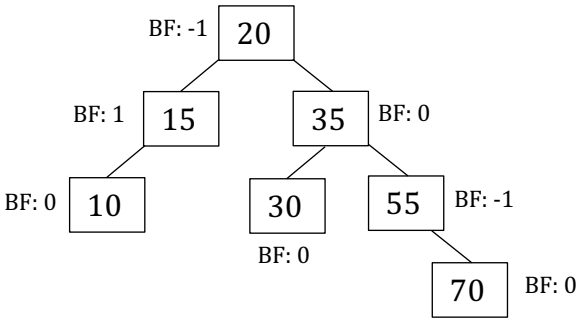
Remove 35
Replaced by largest value in left sub tree
No need to Refactor & Restore AVL property



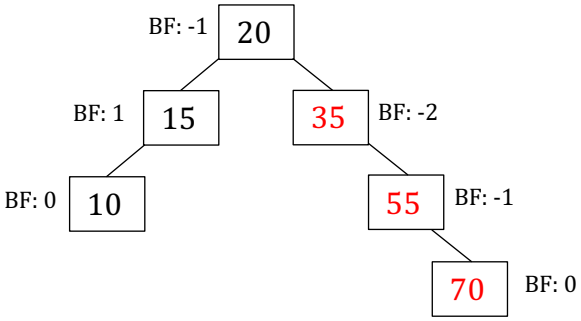
2c)

Remove 25

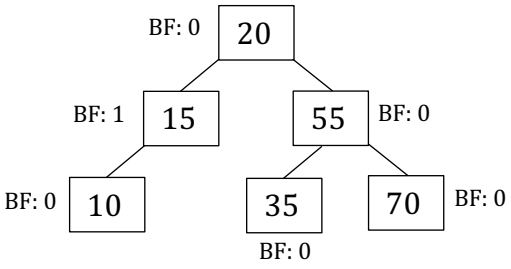
No need to Refactor and Restore AVL property



Remove 30



Refactor and Restore AVL property

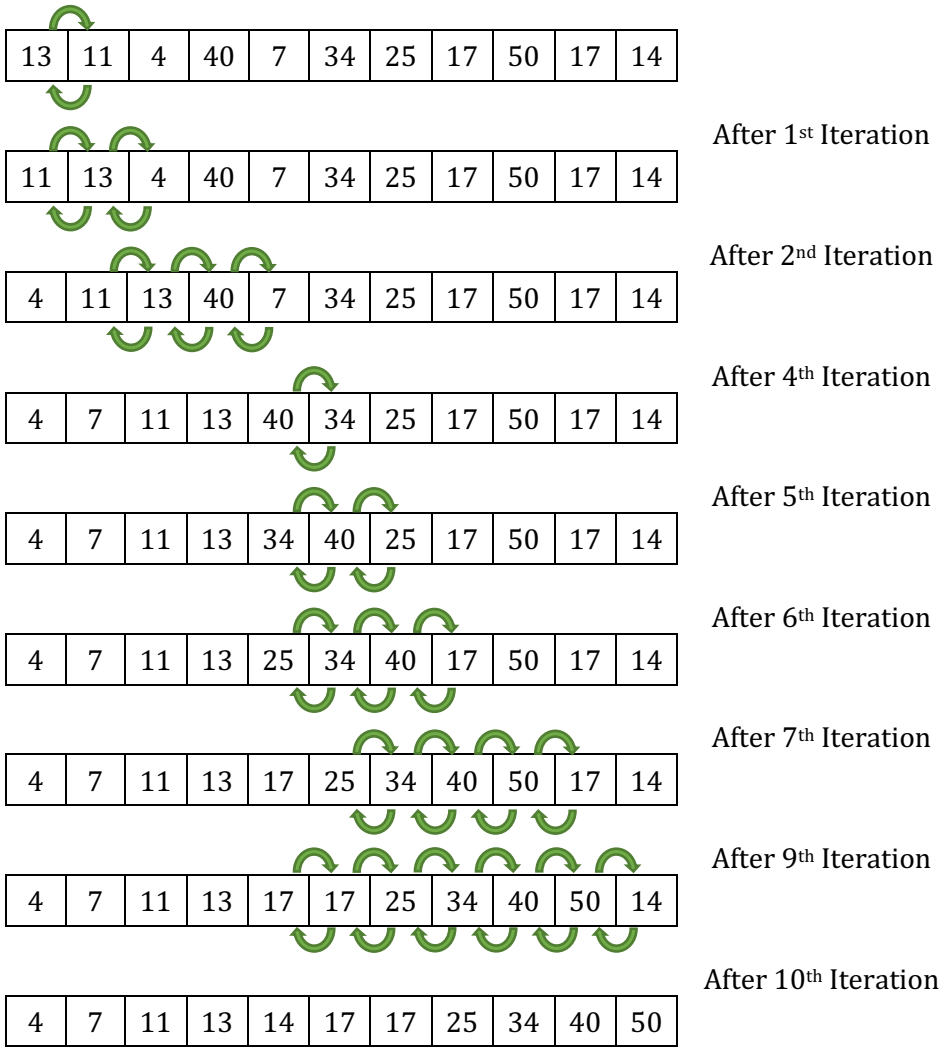


Q3 (a) Insertion sort

Note: The array is displayed only after iterations where changes are made to the array

```
1   def insertionSort(arr):
2       for i in range(1, len(arr)):
3           key = arr[i]
4           j = i-1
5           while j >= 0 and key < arr[j]:
6               arr[j + 1] = arr[j]
7               j = j - 1
8           arr[j + 1] = key
```

Note: This is the algorithm which was hand traced.
NOT COMPILED AND RUN



Total swaps made: 22 swaps

: One Swap

Assignments:	Line 3, Line 4, Line 6, Line 7, Line 8
Comparisions:	Line 5
Total Assignemnts:	74
Total Comparisions:	32
Assignments + Comparisions:	106

Q3 (b) Selection sort

```
1       def selectionSort(arr):
2               for i in range(len(arr)):
3                       minVal = i
4                       for j in range(i+1, len(arr)):
5                               if arr[minVal] > arr[j]:
6                               minVal = j
7                       arr[i], arr[minVal] = arr[minVal], arr[i]
```

Note: This is the algorithm which was hand traced.
NOT COMPILED AND RUN

13	11	4	40	7	34	25	17	50	17	14

After 1st Iteration

4	11	13	40	7	34	25	17	50	17	14

After 2nd Iteration

4	7	13	40	11	34	25	17	50	17	14

After 3rd Iteration

4	7	11	40	13	34	25	17	50	17	14

After 4th Iteration

4	7	11	13	40	34	25	17	50	17	14

After 5th Iteration

4	7	11	13	14	34	25	17	50	17	40

After 6th Iteration

4	7	11	13	14	17	25	34	50	17	40

After 7th Iteration

4	7	11	13	14	17	17	34	50	25	40

After 8th Iteration

4	7	11	13	14	17	17	25	50	34	40

After 9th Iteration

4	7	11	13	14	17	17	25	34	50	40

After 10th Iteration

4	7	11	13	14	17	17	25	34	40	50

Total swaps made: 10 swaps

Assignments:	Line 3, Line 6, Line 7
Comparisions:	Line 5
Total Assignemnts:	37
Total Comparisions:	55
Assignments + Comparisions:	92

Q3 (c) Heap sort

Note: The heapify method creates a max heap and stores it in arr, thereby rearragning the elements

```
1   def heapSort(arr):
2       n = len(arr)
3
4       for i in range(n // 2 - 1, -1, -1):
5           heapify(arr, n, i)
6
7       for i in range(n-1, 0, -1):
8           arr[i], arr[0] = arr[0], arr[i]
9           heapify(arr, i, 0)
10
11  def heapify(arr, n, i):
12      largest = i
13      l = 2 * i + 1
14      r = 2 * i + 2
15
16      if l < n and arr[i] < arr[l]:
17          largest = l
18
19      if r < n and arr[largest] < arr[r]:
20          largest = r
21
22      if largest != i:
23          arr[i],arr[largest] = arr[largest],arr[i]
24          heapify(arr, n, largest)
```

Note: This is the algorithm which was hand traced.
NOT COMPILED AND RUN

13	11	4	40	7	34	25	17	50	17	14

Initial Max-Heap formed 8 swaps performed

50	40	34	17	17	4	25	13	11	7	14

After 1st Iteration

Swap 1 swap performed

14	40	34	17	17	4	25	13	11	7	50

Reform max-heap 2 swaps performed

40	17	34	14	17	4	25	13	11	7	50

After 2nd Iteration

Swap 1 swap performed

7	17	34	14	17	4	25	13	11	40	50

Reform max-heap 2 swaps performed

34	17	25	14	17	4	7	13	11	40	50

After 3rd Iteration

Swap 1 swap performed

11	17	25	14	17	4	7	13	34	40	50

Reform max-heap 1 swap performed

25	17	11	14	17	4	7	13	34	40	50

After 4th Iteration

Swap 1 swap performed

13	17	11	14	17	4	7	25	34	40	50

Reform max-heap 2 swaps performed

17	17	11	14	13	4	7	25	34	40	50

After 5th Iteration

Swap 1 swap performed

7	17	11	14	13	4	17	25	34	40	50

Reform max-heap 2 swaps performed

17	14	11	7	13	4	17	25	34	40	50

After 6th Iteration

Swap 1 swap performed

4	14	11	7	13	17	17	25	34	40	50

Reform max-heap 2 swaps performed

14	13	11	7	4	17	17	25	34	40	50

After 7th Iteration

Swap 1 swap performed

4	13	11	7	14	17	17	25	34	40	50

Reform max-heap 2 swaps performed

13	7	11	4	14	17	17	25	34	40	50

After 8th Iteration

Swap 1 swap performed

4	7	11	13	14	17	17	25	34	40	50

Reform max-heap 1 swap performed

11	7	4	13	14	17	17	25	34	40	50

After 9th Iteration

Swap 1 swap performed

4	7	11	13	14	17	17	25	34	40	50

Reform max-heap 1 swap performed

7	4	11	13	14	17	17	25	34	40	50

After 10th Iteration

Swap 1 swap performed

4	7	11	13	14	17	17	25	34	40	50

Reform max-heap 0 swap performed

4	7	11	13	14	17	17	25	34	40	50

Total swaps made: 33 swaps

Assignments:	Line 2, Line 8, Line 12, Line 13, Line 14, Line 17, Line 20, Line 23
Comparisions:	Line 16, Line 19, Line 22
Total Assignemnts:	177
Total Comparisions:	114
Assignments + Comparisions:	291

Q3) (d) Merge sort

[13] [11] [4] [40] [7] [34] [25] [17] [50] [17] [14]
[11 13] [4 40] [7 34] [17 25] [17 50] [14]
[4 11 13 40] [7 17 25 34] [14 17 50]
[4 7 11 13 17 25 34 40] [14 17 50]
[4 7 11 13 14 17 17 25 34 50]

Algorithm followed:

1 Repeat Until: Array not sorted
2 Select pairs of arrays
3 Repeat Until: No element left in list
4 Find smallest element by comparision
5 Merge smallest element into array (Assignment)
6 Mark smallest element as added

Example:

Line 2: Select pairs of arrays
[11 13] [4 40]
Loop start
Line 4: Find smallest element by comparision (Ignore added elements)
[11 13] [4 40]
Line 5: Merge smallest element into array
[4 11 13] [40]
Line 6: Mark smallest element as added
[4 11 13] [40]
Loop back and repeat process

This process repeats until the whole array is sorted

Number of comparisions
=

$$\sum_{i=1}^{2} i + \sum_{i=1}^{4} i + \sum_{i=1}^{8} i + \sum_{i=1}^{11} i = 115$$

Total Comparisions:	115
Total Assignments:	40
Assignments + Comparisions:	155

Note: This is the algorithm which was hand traced.
NOT COMPILED AND RUN

Zafir Khalid - 40152164

Q3 (e) Quick Sort

Algorithm used:

```
1       def quickSort(arr, l, r):
2           if( l>= r):
3               return

4           p = partition(arr, l, r)
5           quickSort(arr, l, p-1)
6           quickSort(arr, p+1, r)
```

```
7       def partition(arr, l, r):
8           pivot = arr[r]
9           i = l-1
10          for j in range(l, r):
11              if arr[j] < pivot:
12                  i += 1
13                  swap(arr[i], arr[j])
14                  swap(arr[i+1], arr[r])
15          return(i+1)
```

Original List

13	11	4	40	7	34	25	17	50	17	14

1st Recursive Call – quickSort(arr, 0, len(arr)-1)

l = 0, r = 10

13	11	4	40	7	34	25	17	50	17	14

Pivot = 14 at Index 10

Array output after partition call:

13	11	4	7	14	34	25	17	50	17	40

After parition call (p = 4)

2nd Recursive Call – quickSort(arr, 0, 3)

l = 0, r = 3

13	11	4	7	14	34	25	17	50	17	40

Pivot = 7 at Index 3

Array output after partition call:

4	7	13	11	14	34	25	17	50	17	40

After parition call (p = 1)

3rd Recursive Call – quickSort(arr, 0, 0)

Base Case – No changes to array

4th Recursive Call – quickSort(arr, 2, 3)

l = 0, r = 3

4	7	13	11	14	34	25	17	50	17	40

Pivot = 11 at Index 3

Array output after partition call:

4	7	11	13	14	34	25	17	50	17	40

After parition call (p = 2)

5th Recursive Call – quickSort(arr, 2, 1)

Base Case – No changes to array

6th Recursive Call – quickSort(arr, 3, 3)

Base Case – No changes to array

7th Recursive Call – quickSort(arr, 5, 10)

l = 5, r = 10

4	7	11	13	14	34	25	17	50	17	40

Pivot = 40 at Index 10

Array output after partition call:

4	7	11	13	14	34	25	17	17	40	50

After parition call (p = 9)

8th Recursive Call – quickSort(arr, 5, 8)

l = 5, r = 8

4	7	11	13	14	34	25	17	17	40	50

Pivot = 17 at Index 8

Array output after partition call:

4	7	11	13	14	17	25	17	34	40	50

After parition call (p = 5)

9th Recursive Call – quickSort(arr, 5, 4)

Base Case – No changes to array

10th Recursive Call – quickSort(arr, 6, 8)

l = 6, r = 8

4	7	11	13	14	17	25	17	34	40	50

Pivot = 34 at Index 8

Array output after partition call:

4	7	11	13	14	17	25	17	34	40	50

After parition call (p = 8)

11th Recursive Call – quickSort(arr, 6, 7)

l = 6, r = 7

4	7	11	13	14	17	25	17	34	40	50

Pivot = 17 at Index 7

Array output after partition call:

4	7	11	13	14	17	17	25	34	40	50

After parition call (p = 6)

12th Recursive Call – quickSort(arr, 6, 5)

Base Case – No changes to array

13th Recursive Call – quickSort(arr, 7, 7)

Base Case – No changes to array

14th Recursive Call – quickSort(arr, 9, 8)

Base Case – No changes to array

15th Recursive Call – quickSort(arr, 10, 10)

Base Case – No changes to array

Final Sorted array

4	7	11	13	14	17	17	25	34	40	50

Assignments:	Line 4, Line 8, Line 9, Line 12, Line 13, Line 14
Comparisions:	Line 2, Line 11
Total Assignemnts:	51
Total Comparisions:	41
Assignments + Comparisions:	92

Q4) (1)

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0	1	0	0
2	0	1	0	1	1	1	0	1	1	0
3	0	0	1	0	0	1	0	0	0	0
4	0	0	1	0	0	1	1	0	0	0
5	0	0	1	1	1	0	1	0	0	0
6	1	0	0	0	1	1	0	0	1	1
7	0	1	1	0	0	0	0	0	1	0
8	0	0	1	0	0	0	1	1	0	1
9	1	0	0	0	0	0	1	0	1	0

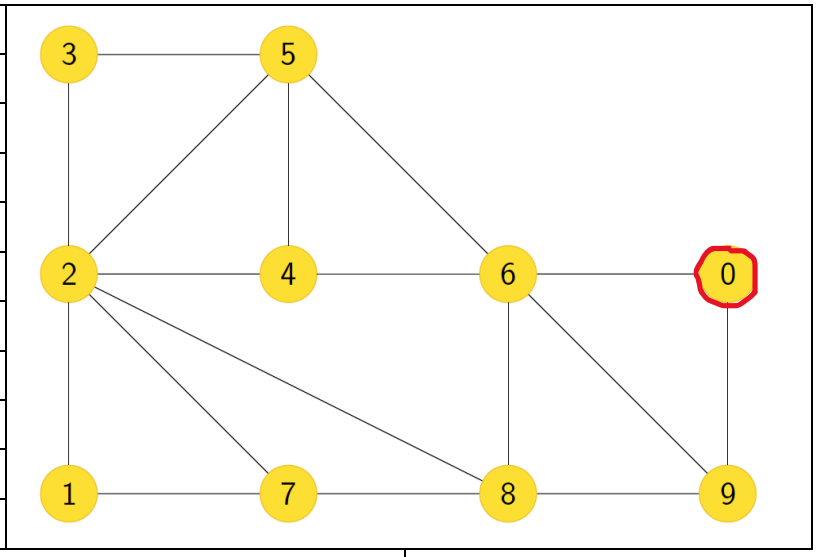
Q4) (2)

0	→	6	9				
1	→	2	7				
2	→	1	3	4	5	7	8
3	→	2	5				
4	→	2	5	6			
5	→	2	3	4	6		
6	→	0	4	5	8	9	
7	→	1	2	8			
8	→	2	6	7	9		
9	→	0	6	8			

Q4) (3)

BFS(0)

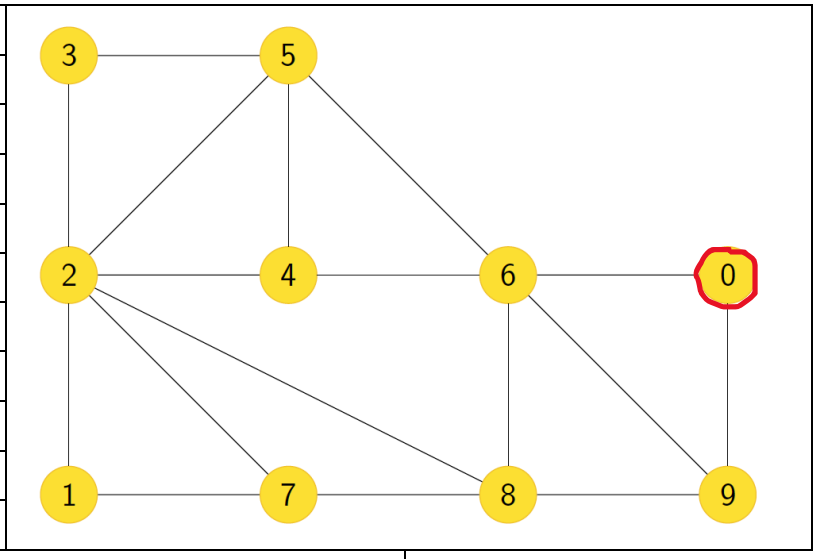
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |0|

BFS Traversal =

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			

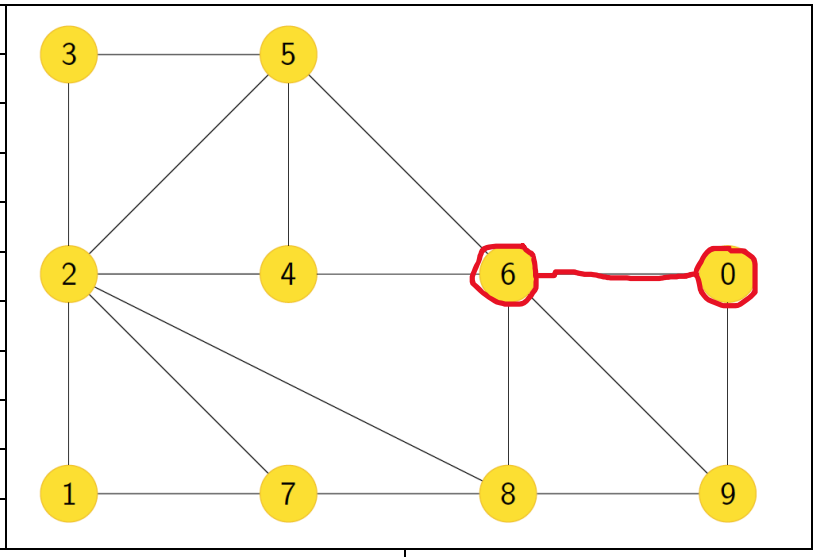


Queue = |6, 9|

BFS Traversal = 0

Zafir Khalid - 40152164

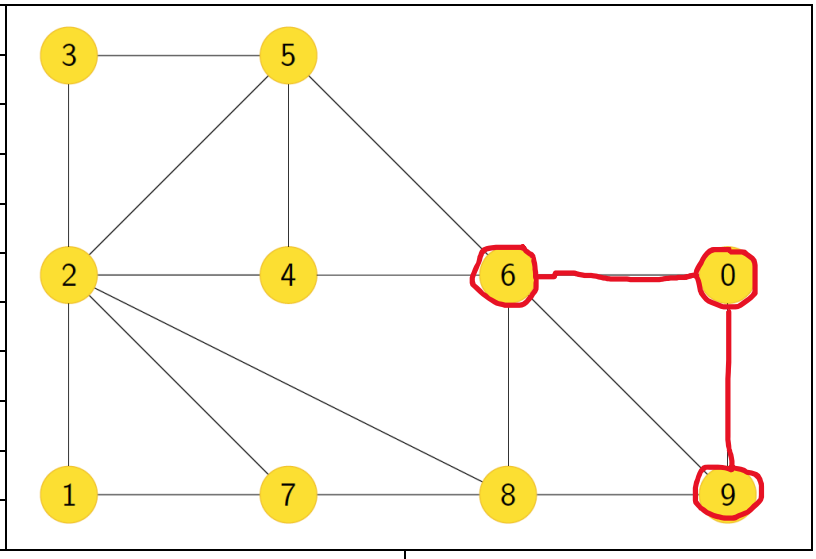
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |9, 4, 5, 8|

BFS Traversal = 0, 6

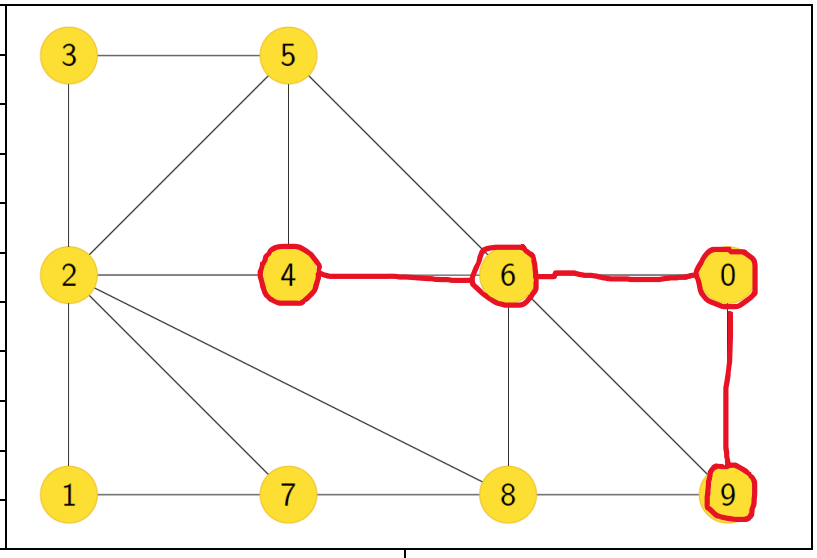
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |4, 5, 8|

BFS Traversal = 0, 6, 9

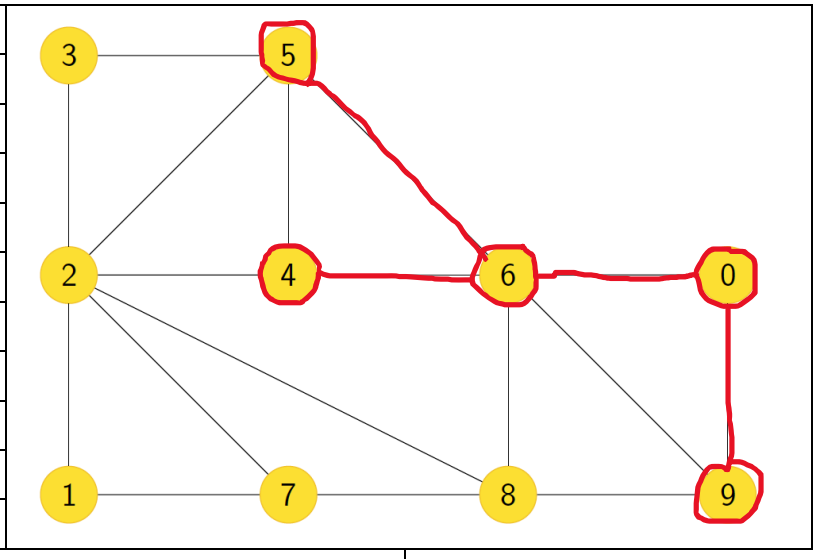
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |5, 8, 2|

BFS Traversal = 0, 6, 9, 4

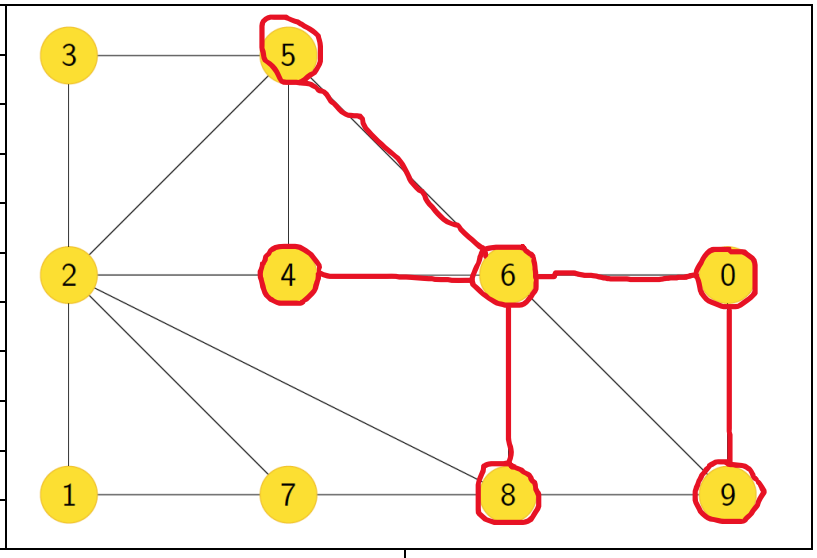
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |8, 2, 3|

BFS Traversal = 0, 6, 9, 4, 5

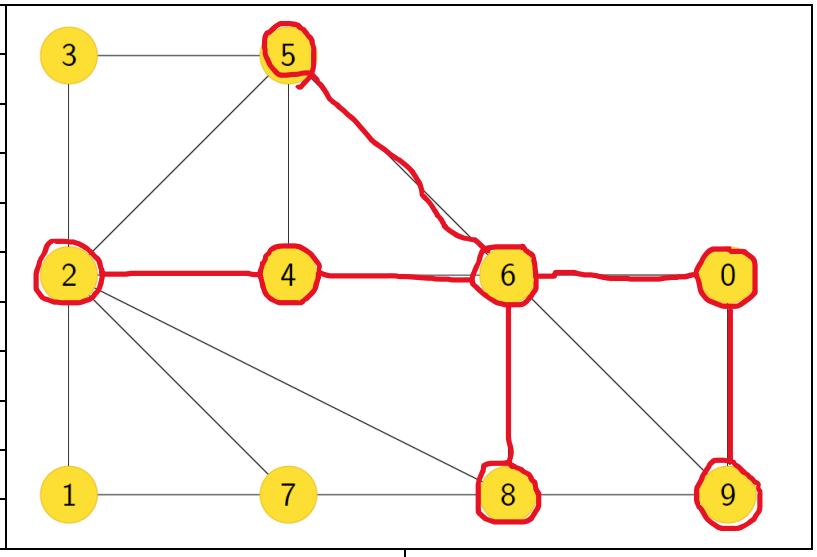
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |2, 3, 7|

BFS Traversal = 0, 6, 9, 4, 5, 8

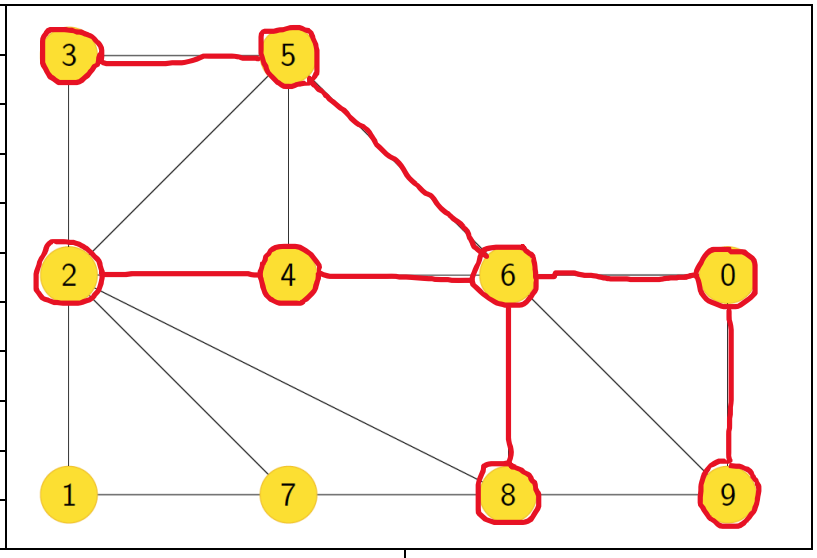
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |3, 7, 1|

BFS Traversal = 0, 6, 9, 4, 5, 8, 2

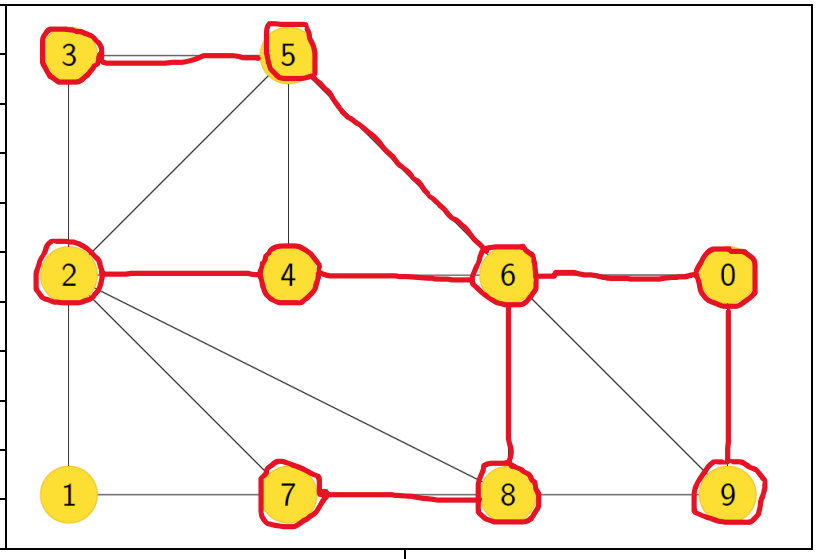
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |7, 1|

BFS Traversal = 0, 6, 9, 4, 5, 8, 2, 3

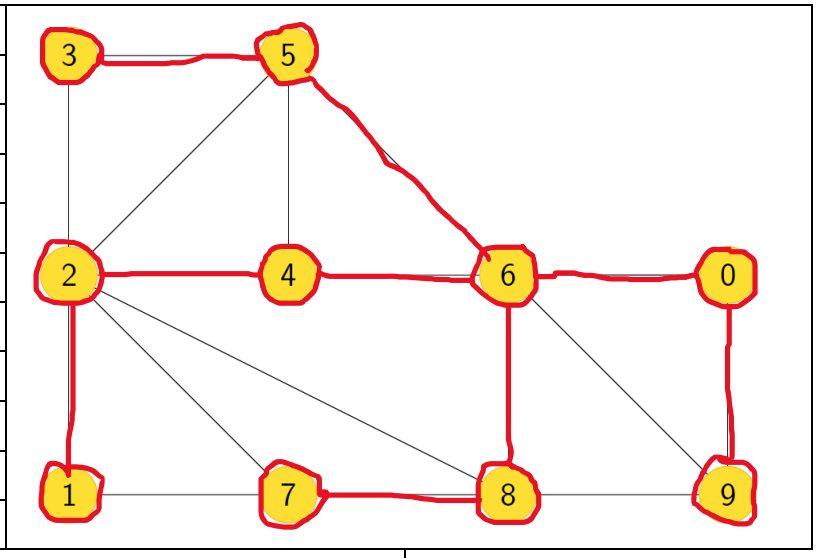
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



Queue = |1|

BFS Traversal = 0, 6, 9, 4, 5, 8, 2, 3, 7

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



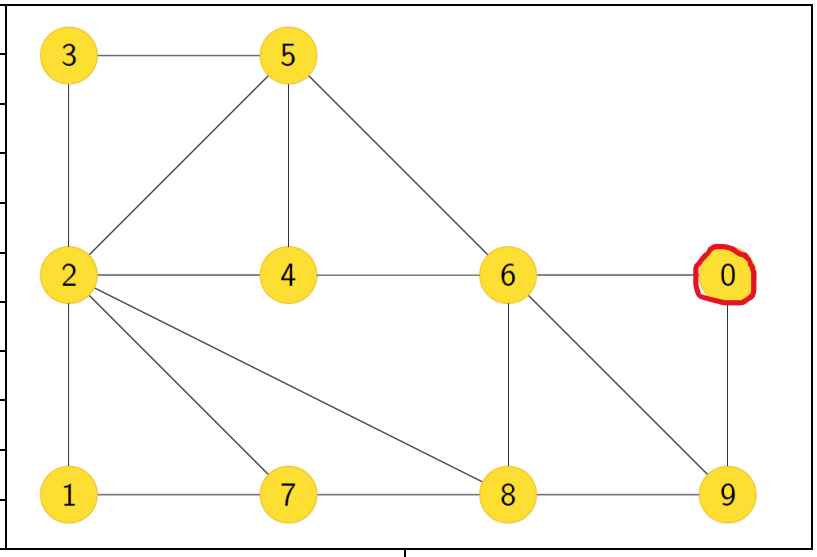
Queue = ||

BFS Traversal = 0, 6, 9, 4, 5, 8, 2, 3, 7, 1

BFS Traversal – 0, 6, 9, 4, 5, 8, 2, 3, 7, 1
Note: The tree structure is shown highlighted in red

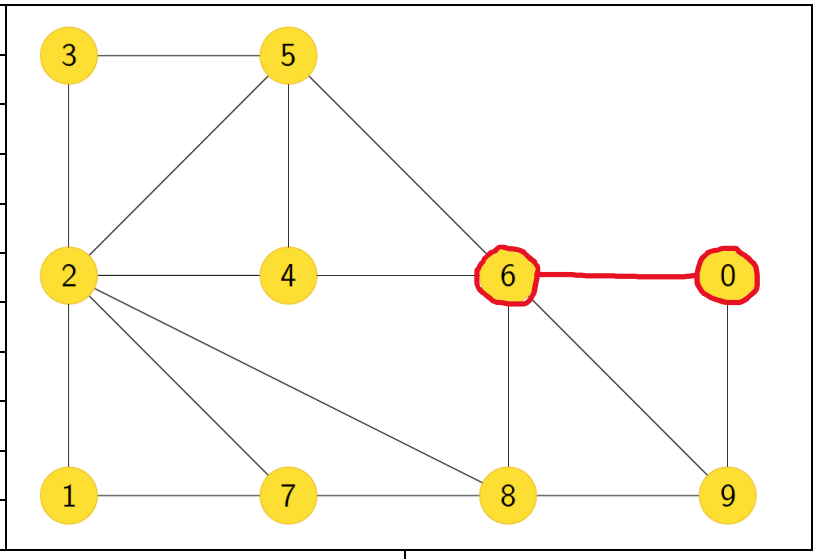
Q4 (4)

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



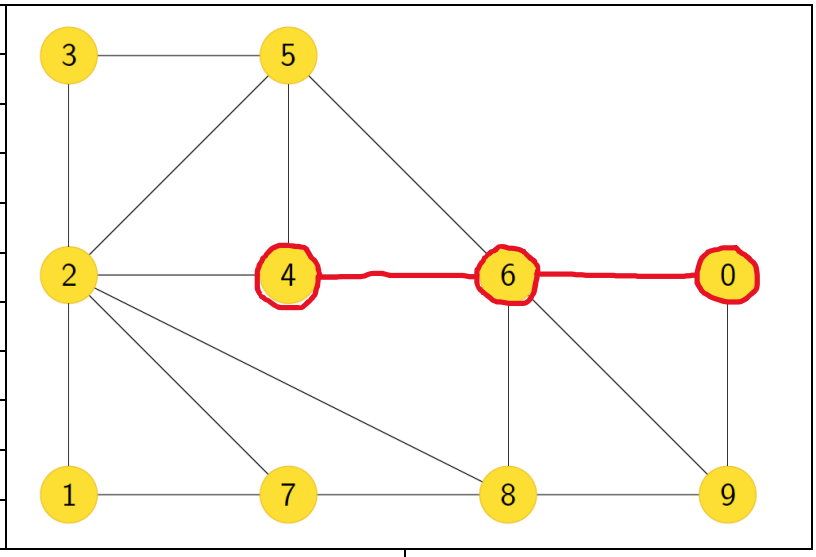
DFS Traversal = 0

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



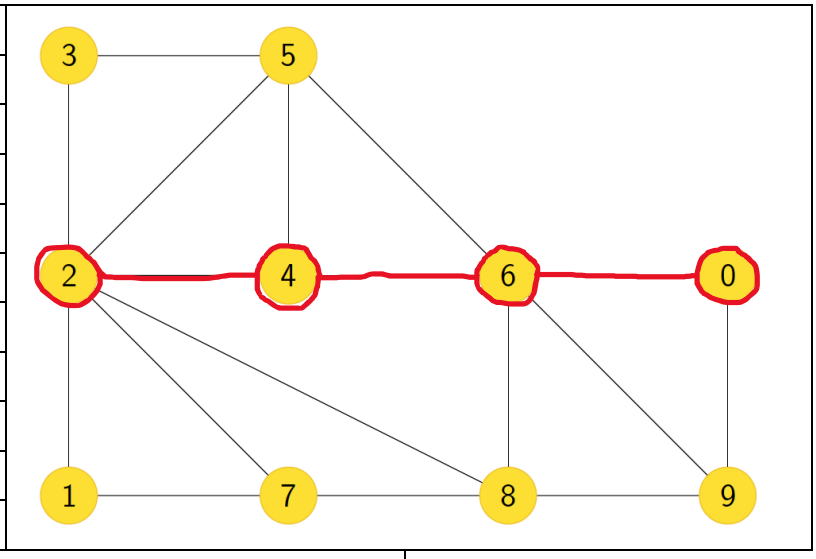
DFS Traversal = 0, 6

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



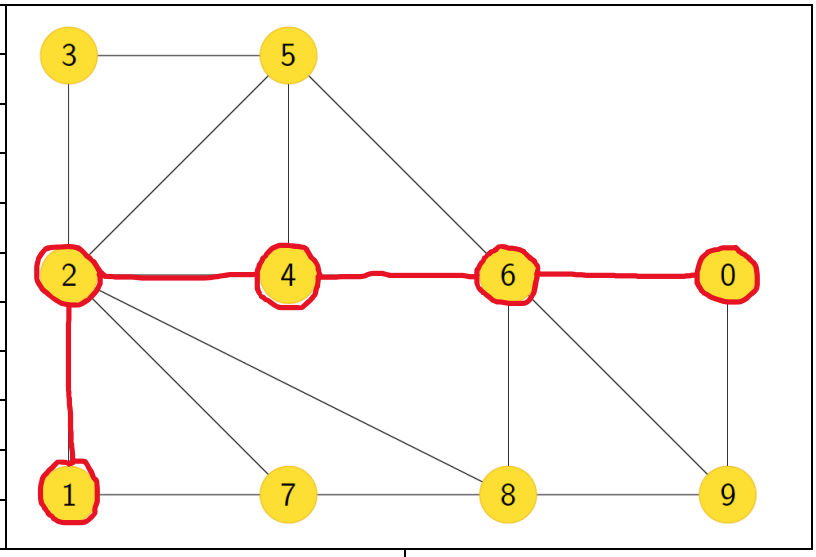
DFS Traversal = 0, 6, 4

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



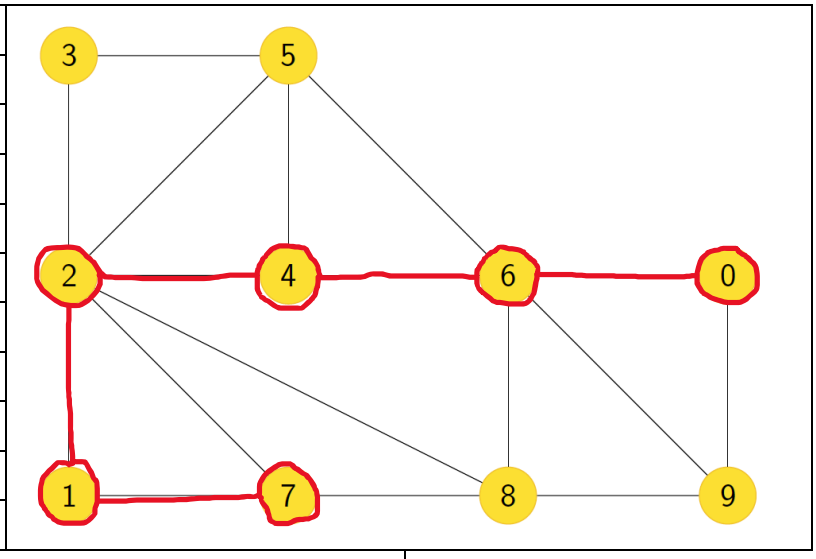
DFS Traversal = 0, 6, 4, 2

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



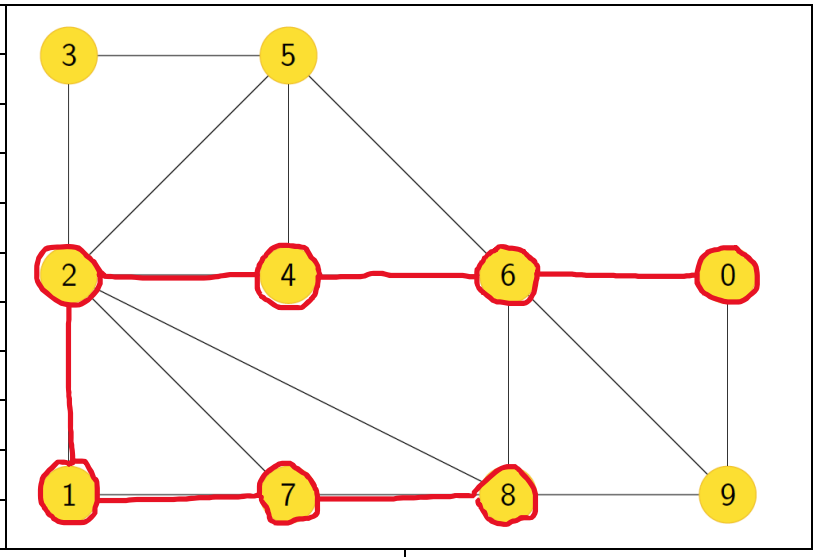
DFS Traversal = 0, 6, 4, 2, 1

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



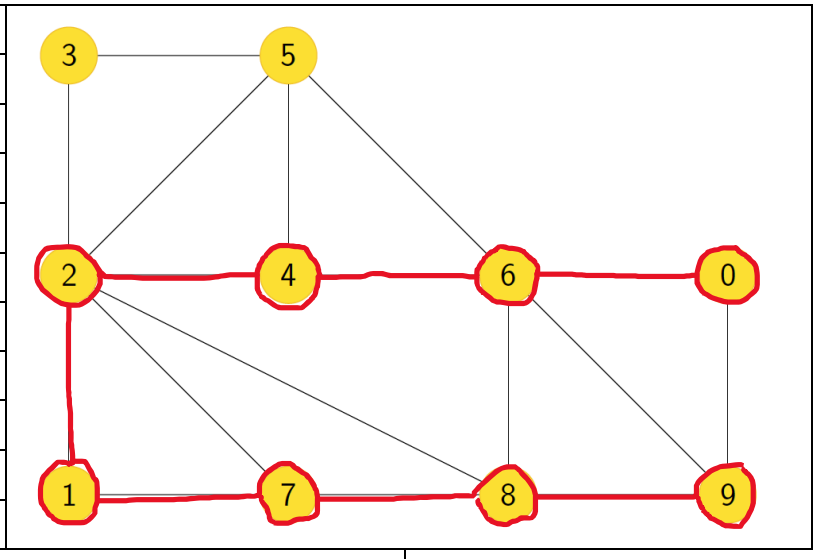
DFS Traversal = 0, 6, 4, 2, 1, 7

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



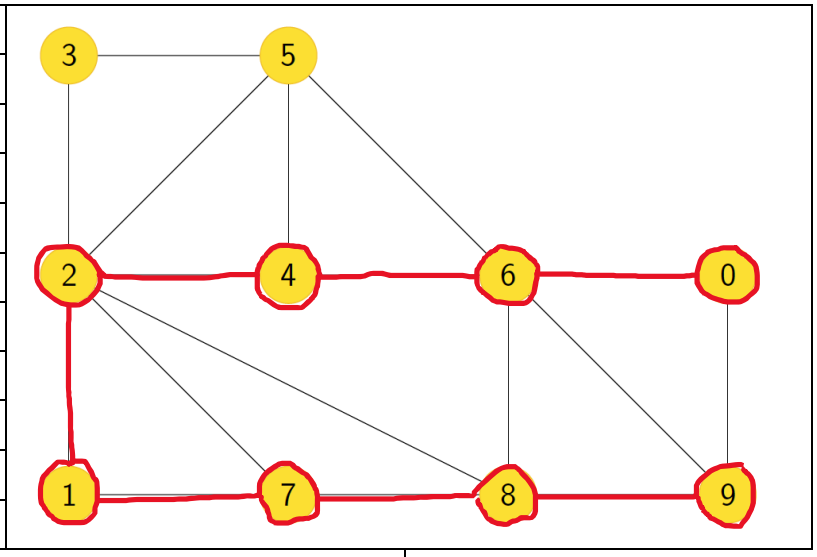
DFS Traversal = 0, 6, 4, 2, 1, 7, 8

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



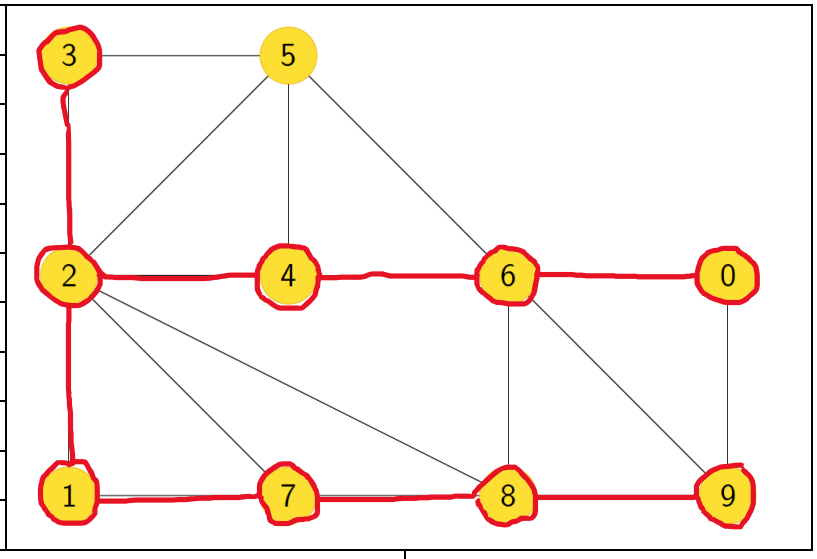
DFS Traversal = 0, 6, 4, 2, 1, 7, 8, 9

Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



DFS Traversal = 0, 6, 4, 2, 1, 7, 8, 9

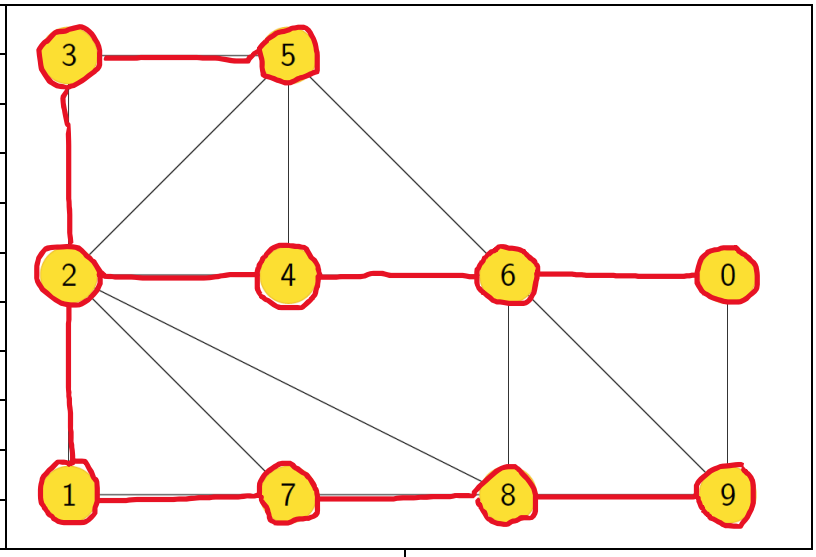
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



DFS Traversal = 0, 6, 4, 2, 1, 7, 8, 9, 3

Zafir Khalid - 40152164

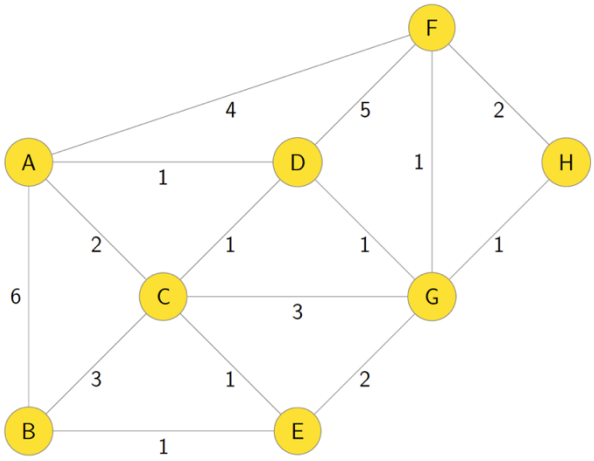
Vertex	Adjacent Vertices					
0	6	9				
1	2	7				
2	1	3	4	5	7	8
3	2	5				
4	2	5	6			
5	2	3	4	6		
6	0	4	5	8	9	
7	1	2	8			
8	2	6	7	9		
9	0	6	8			



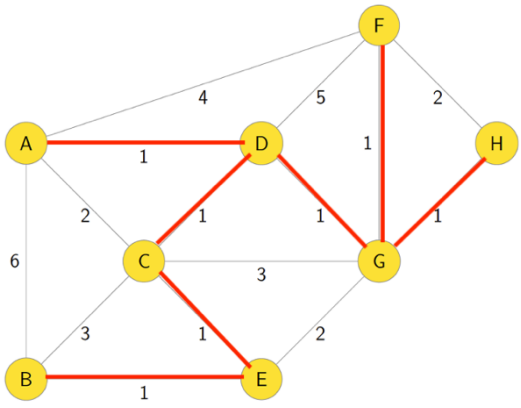
DFS Traversal = 0, 6, 4, 2, 1, 7, 8, 9, 3, 5

DFS Traversal = 0, 6, 4, 2, 1, 7, 8, 9, 3, 5
Note: The tree structure is shown highlighted in red

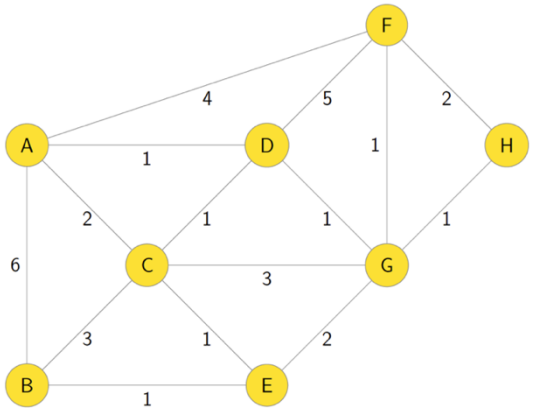
Q5) 1)



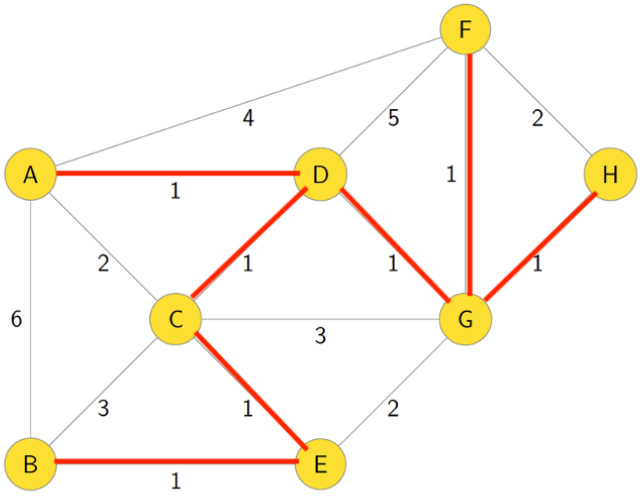
Step	Edge Considered	Cost	Accepted/Rejected
1	AD	1	Accepted
2	CD	1	Accepted
3	CE	1	Accepted
4	BE	1	Accepted
5	GD	1	Accepted
6	FG	1	Accepted
7	GH	1	Accepted
8	AC	2	Rejected
9	EG	2	Rejected
10	FH	2	Rejected
11	BC	3	Rejected
12	CG	3	Rejected
13	AF	4	Rejected
14	DF	5	Rejected
15	AB	6	Rejected



Q5) 2)

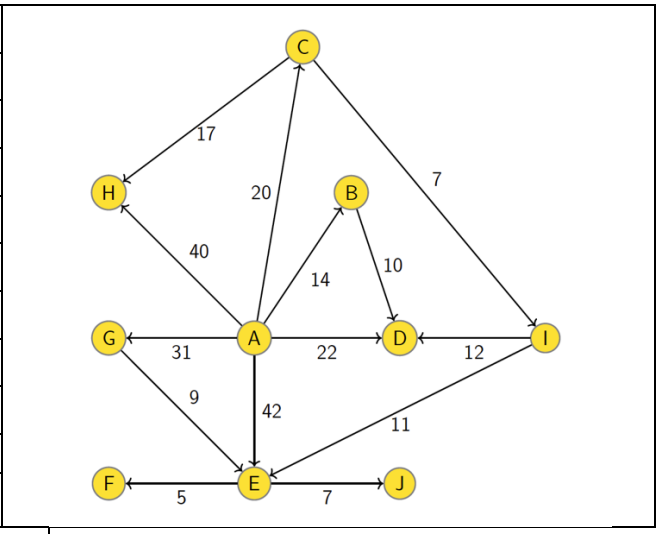


Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
D	Yes	1	A
C	Yes	1	D
E	Yes	1	C
B	Yes	1	E
F	Yes	1	G
G	Yes	1	D
H	Yes	1	G



Q6) (a) BFS topological

In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
1	B	D					
1	C	H	I				
3	D						
3	E	F	J				
1	F						
1	G	E					
2	H						
1	I	D	E				
1	J						



Queue = |A|

BFS Topological =

In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
2	D						
2	E	F	J				
1	F						
0	G	E					
1	H						
1	I	D	E				
1	J						

Queue = |B, C, G|

BFS Topological = A

In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
1	D						
2	E	F	J				
1	F						
0	G	E					
1	H						
1	I	D	E				
1	J						

Queue = |C, G|

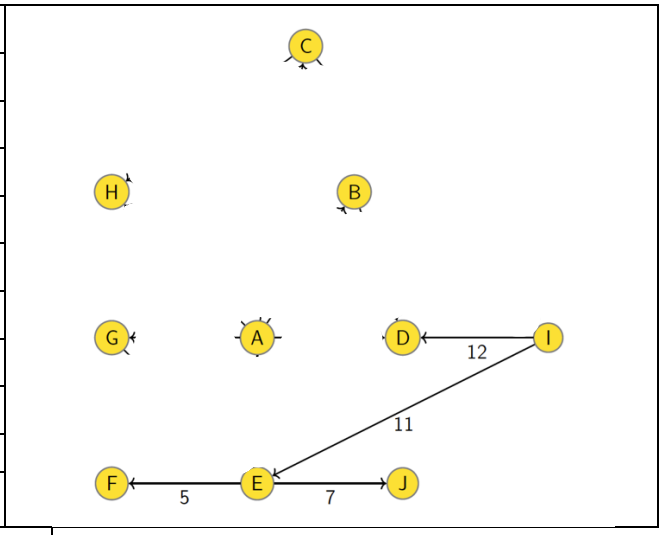
BFS Topological = A, B

In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
1	D						
2	E	F	J				
1	F						
0	G	E					
0	H						
0	I	D	E				
1	J						

Queue = |G, H, I|

BFS Topological = A, B, C

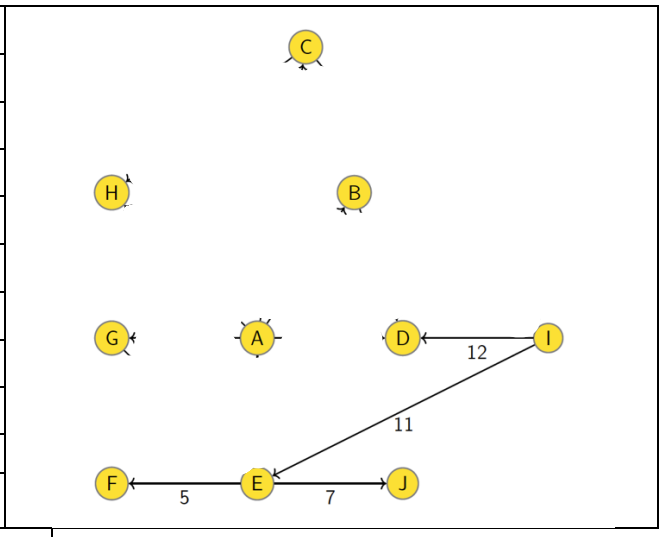
In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
1	D						
1	E	F	J				
1	F						
0	G	E					
0	H						
0	I	D	E				
1	J						



Queue = |H, I|

BFS Topological = A, B, C, G

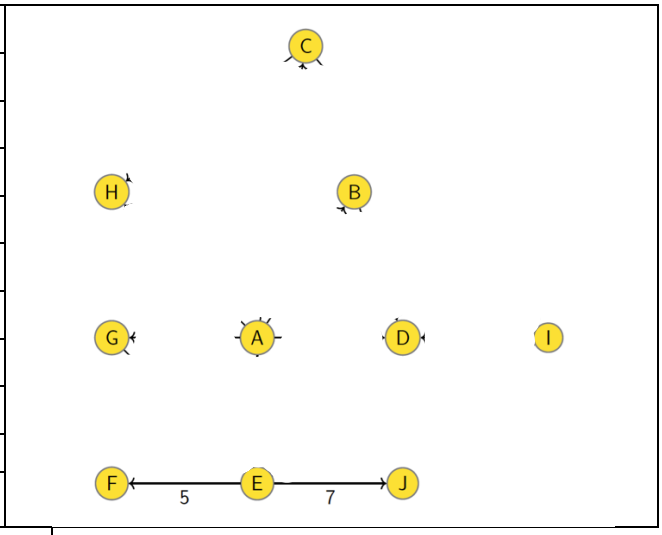
In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
1	D						
1	E	F	J				
1	F						
0	G	E					
0	H						
0	I	D	E				
1	J						



Queue = |I|

BFS Topological = A, B, C, G, H

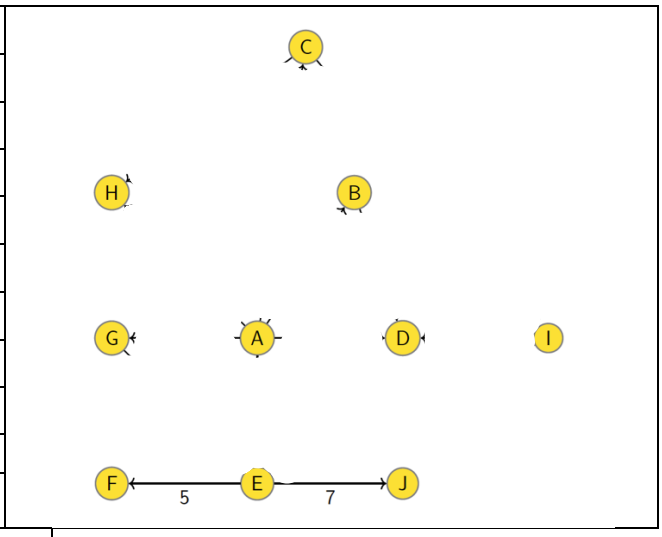
In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
0	D						
0	E	F	J				
1	F						
0	G	E					
0	H						
0	I	D	E				
1	J						



Queue = |D, E|

BFS Topological = A, B, C, G, H, I

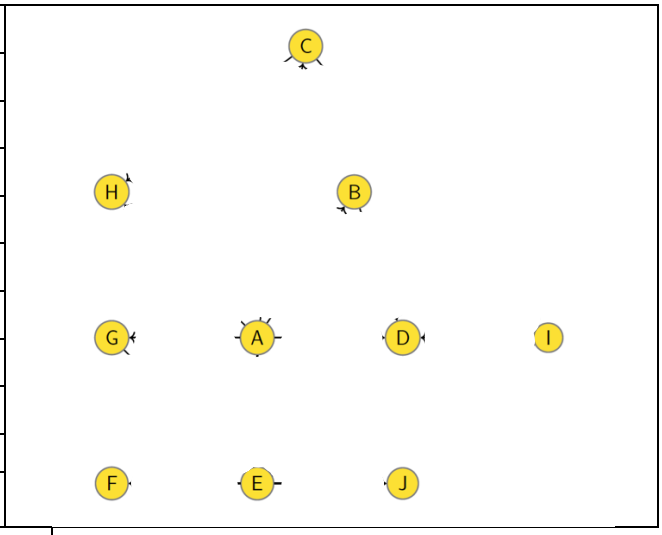
In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
0	D						
0	E	F	J				
1	F						
0	G	E					
0	H						
0	I	D	E				
1	J						



Queue = |E|

BFS Topological = A, B, C, G, H, I, D

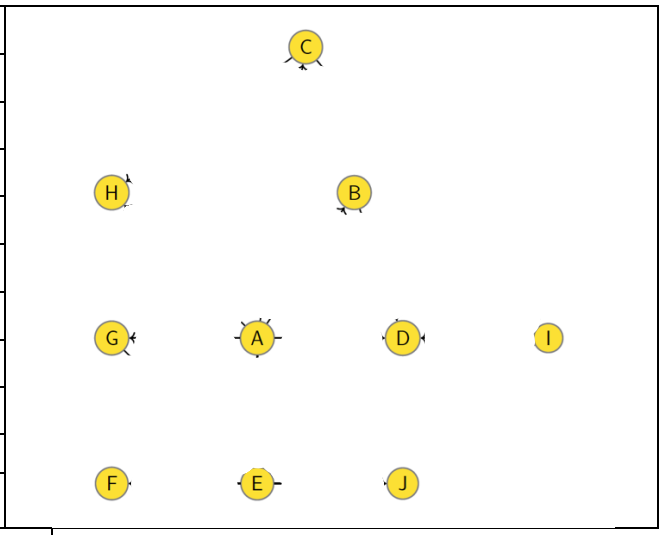
In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
0	D						
0	E	F	J				
0	F						
0	G	E					
0	H						
0	I	D	E				
0	J						



Queue = |F, J|

BFS Topological = A, B, C, G, H, I, D, E

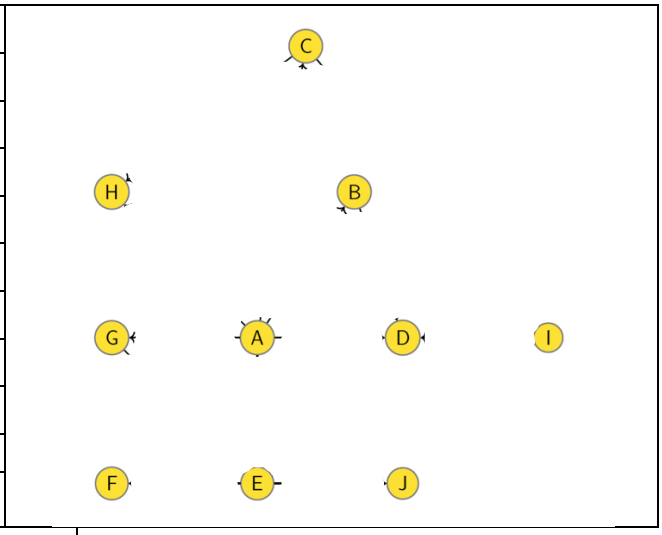
In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
0	D						
0	E	F	J				
0	F						
0	G	E					
0	H						
0	I	D	E				
0	J						



Queue = |J|

BFS Topological = A, B, C, G, H, I, D, E, F

In-Degree	V	Adjacent Vertices					
0	A	B	C	D	E	G	H
0	B	D					
0	C	H	I				
0	D						
0	E	F	J				
0	F						
0	G	E					
0	H						
0	I	D	E				
0	J						



Queue = ||
BFS Topological = A, B, C, G, H, I, D, E, F, J

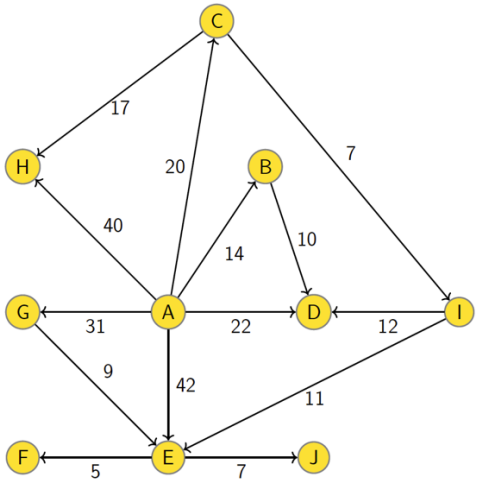
BFS Topological = A, B, C, G, H, I, D, E, F, J

Q6) (a) DFS topological

Algorithm topological Sort DFS(Graph G)
    Initialize a list L
    Mark all vertices as unvisited
    while there exists an unvisited vertex v
        visit(L, v)
    end-while

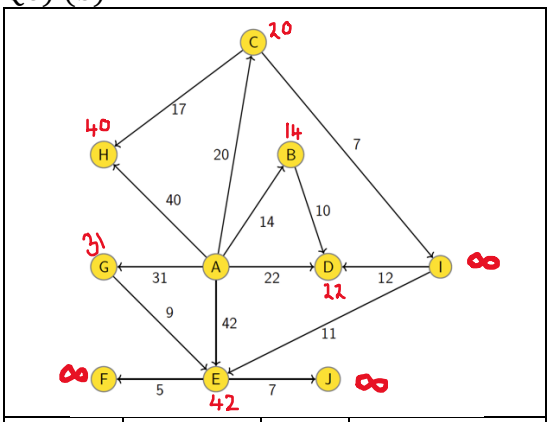
Algorithm visit(L, v)
    if v is unvisited
        for each unvisited neighbor w of v
            visit(L, w)
        end-for
        mark v visited
        add v to front of L
    end-if

- visit(L, A)
  - visit(L, B)
    - visit(L, D)
    - mark D as visited
    - L = {D}
  - mark B as visited
  - L = {B, D}
  - visit(L, C)
    - visit(L, H)
    - mark H as visited
    - L = {H, B, D}
    - visit(L, I)
      - visit(L, E)
        - visit(L, F)
        - mark F as visited
        - L = {F, H, B, D}
        - visit(L, J)
        - mark J as visited
        - L = {J, F, H, B, D}
      - mark E as visited
      - L = {E, J, F, H, B, D}
    - mark I as visited
    - L = {I, E, J, F, H, B, D}
  - mark C as visited
  - L = {C, I, E, J, F, H, B, D}
  - visit(L, G)
  - mark G as visited
  - L = {G, C, I, E, J, F, H, B, D}
- mark A as visited
- L = {A, G, C, I, E, J, F, H, B, D}



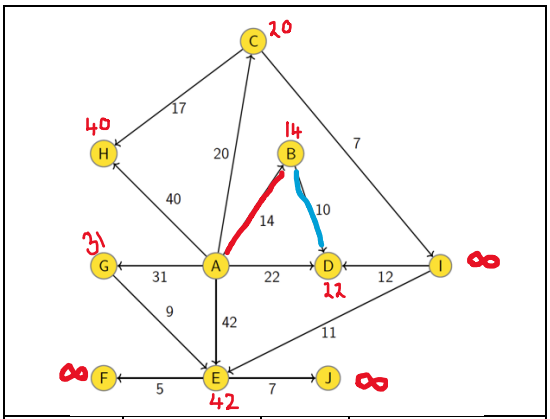
DFS Topological = {A, G, C, I, E, J, F, H, B, D}

Q6) (b)



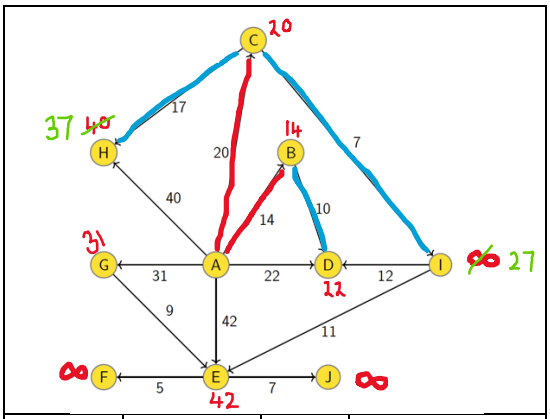
Vertex	Visited?	Cost	Predecessor
A	Yes	0	-

Options	Cost
A,B	14
A,C	20
A,D	22
A,G	31
A,E	42



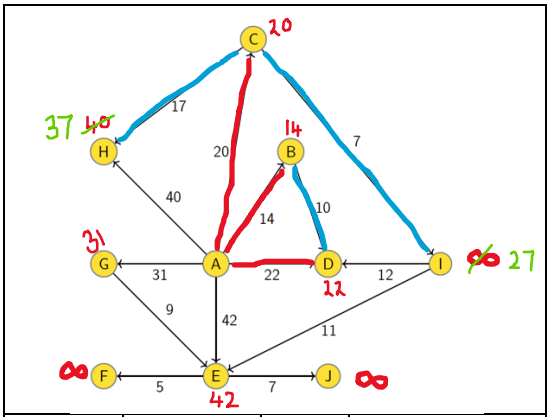
Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A

Options	Cost
A,C	20
A,D	22
A,G	31
A,E	42



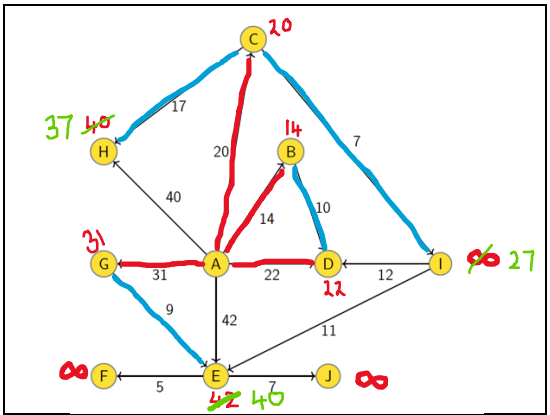
Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A
C	Yes	20	A

Options	Cost
A,D	22
A,G	31
A,E	42
C,I	27
C,H	37



Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A
C	Yes	20	A
D	Yes	22	A

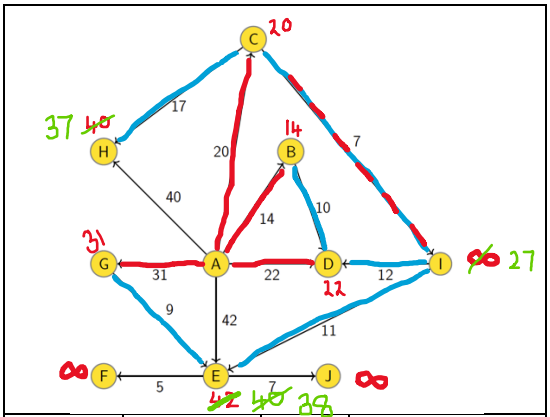
Options	Cost
A,G	31
A,E	42
C,I	27
C,H	37



Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A
C	Yes	20	A
D	Yes	22	A
G	Yes	31	A

Options	Cost
~~A,E~~	~~42~~
C,I	27
C,H	37
G,E	40

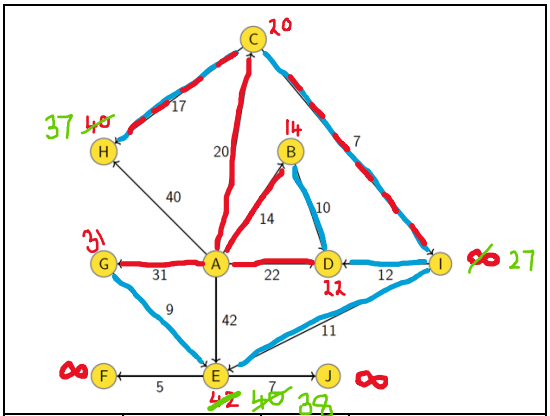
Note: (A,E) is no longer an option because E is relaxed by the edge (G, E).



Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A
C	Yes	20	A
D	Yes	22	A
G	Yes	31	A
I	Yes	27	C

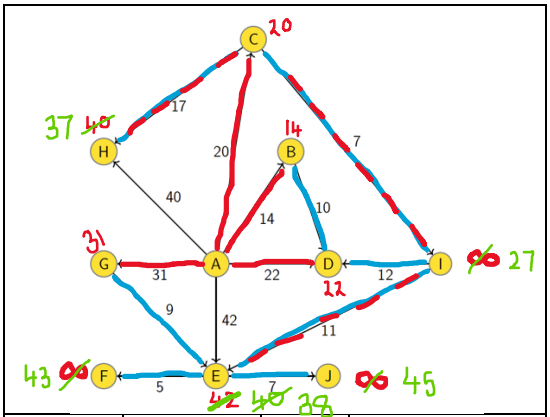
Options	Cost
~~A,E~~	~~42~~
C,H	37
~~G,E~~	~~40~~
I,E	38

Note: (G,E) is no longer an option because E is relaxed by the edge (I, E).



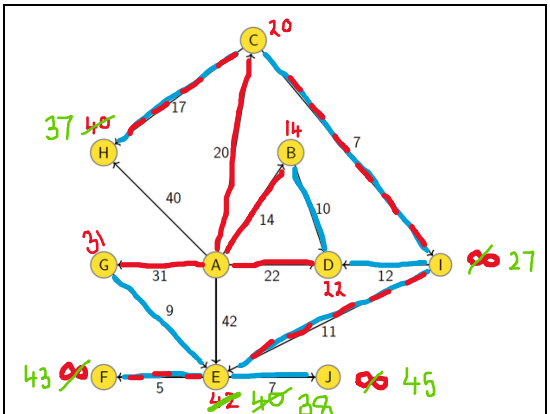
Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A
C	Yes	20	A
D	Yes	22	A
G	Yes	31	A
I	Yes	27	C
H	Yes	37	C

Options	Cost
~~A,E~~	~~42~~
~~G,E~~	~~40~~
I,E	38



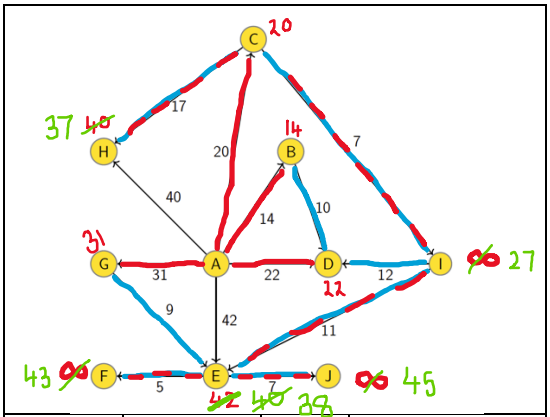
Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A
C	Yes	20	A
D	Yes	22	A
G	Yes	31	A
I	Yes	27	C
H	Yes	37	C
E	Yes	38	I

Options	Cost
~~A,E~~	~~42~~
~~G,E~~	~~40~~
E,F	43
E,J	45



Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A
C	Yes	20	A
D	Yes	22	A
G	Yes	31	A
I	Yes	27	C
H	Yes	37	C
E	Yes	38	I
F	Yes	43	E

Options	Cost
~~A,E~~	~~42~~
~~G,E~~	~~40~~
E,J	45



Vertex	Visited?	Cost	Predecessor
A	Yes	0	-
B	Yes	14	A
C	Yes	20	A
D	Yes	22	A
G	Yes	31	A
I	Yes	27	C
H	Yes	37	C
E	Yes	38	I
F	Yes	43	E
J	Yes	45	E

Options	Cost
~~A,E~~	~~42~~
~~G,E~~	~~40~~

Therefore, the shortest distance from node A to every other node is as follows: