Portfolio Pt.1: Intro to Machine Learning

Task 1 - - Simple sorting algorithms (insertion sort)

Consider the following list of numbers stored in an array: 12, 23, 15, 31, 21, 18

Step 1: Sort these numbers in ascending order using insertion sort. Show the contents of the array at the end of each pass.

```
Step 1: Sort these numbers in ascending order using insertion sort.
Show the contents of the array at the end of each pass.

List =[12,{23},15,31,21,18]
temp = [23]  -> compare if less than k1(12)
-> not less, therefore move on to next.

List =[12,23,{15},31,21,18]
temp = [15]  -> compare if less than k2(23); if true, repeat with k-1(12)
-> less than k2 therefore shift k2 one place to right in array
-> not less than k1, so insert into placeholder k2

List =[12,15,23,{31},21,18]
temp = [31]  -> compare if less than k3(23); if true,  repeat with k-1(15,12)
-> not less, therefore move on to next.

List =[12,15,23,31,{21},18]
temp = [21]  -> compare if less than k4(31); if true, repeat with k-1(23,15,12)
-> less than k4 and k3 therefore shift k3&k4 one place to right in array
-> not less than k2, so insert into placeholder k3

List =[12,15,21,23,31,{18}]
temp = [18]  -> compare if less than k5(31); if true, repeat with k-1(23,21,15,12)
-> less than k5 k4 & k3 therefore shift k3,k4 & k5 one place to right in array
-> not less than k2, so insert into placeholder k3

List =[12,15,18,21,23,31]
Reached end of predefined list length(k=6): end loop
```

Step 2: Write the steps of the insertion sort algorithm.

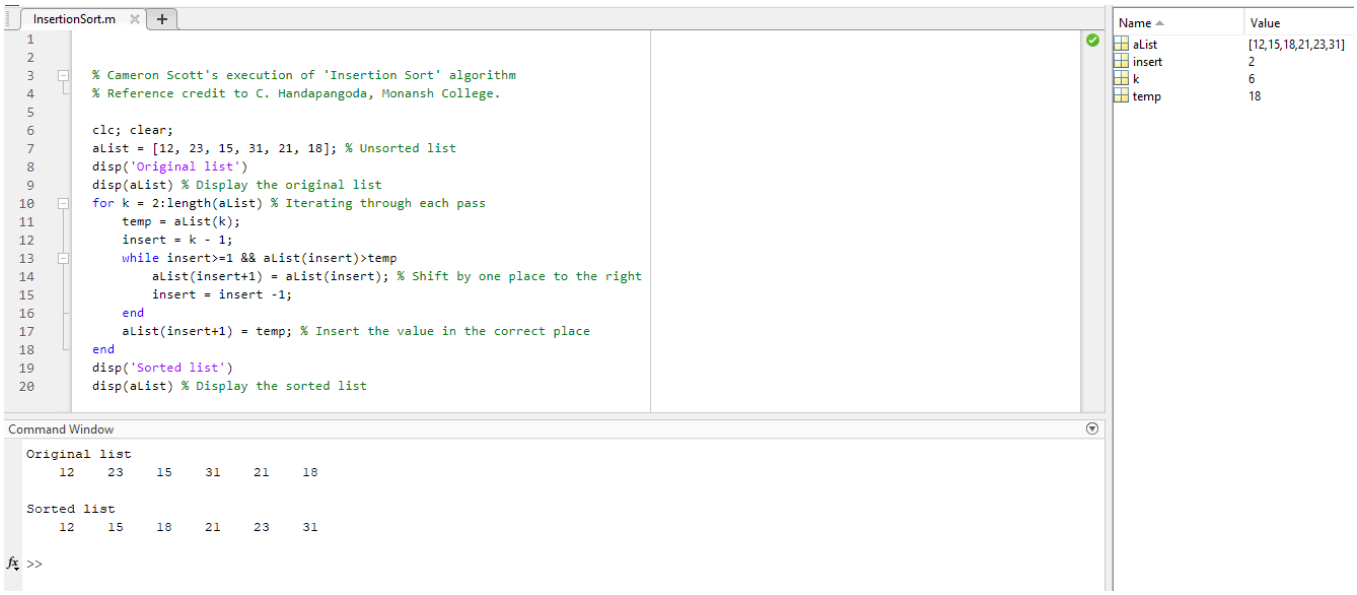
```
(S1 - begin at second item in list)

S2 - Select second number, compare with all in prior positions in list incrementally.

    S2a - Shift position of > items in array one place ahead of selected item.
    S2b - Insert into position lesser than shifted values

S3 - loop until reach end of list
```

Step 3: Implement the algorithm you wrote in step 2 above in Matlab.

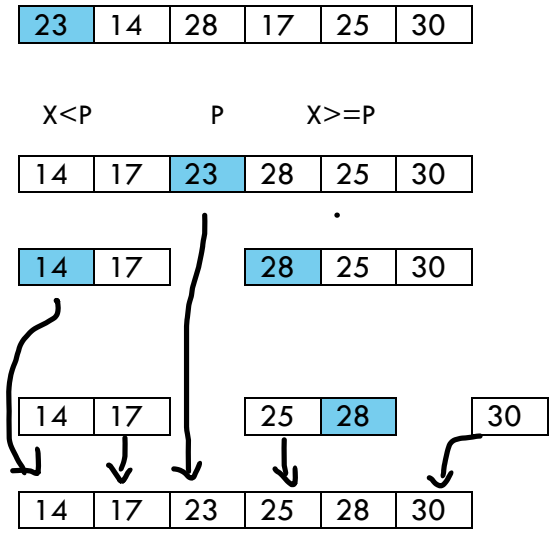


Task 2 - Complex sorting algorithm (quicksort)

Consider the following array of numbers:

23, 14, 28, 17, 25, 30

Step 1: Use a diagram to show how quicksort is applied to sort the above set of numbers in ascending order. Choose the first element as the pivot in each pass.



Step 2: What is the best case and worst case time complexity of this algorithm?

Best: $\log_2(N)$ - Pivot well split, dividing in binary fashion as Merge sort

Worst: N - pivot routinely not dissecting list, and must run item-by-item through list length.

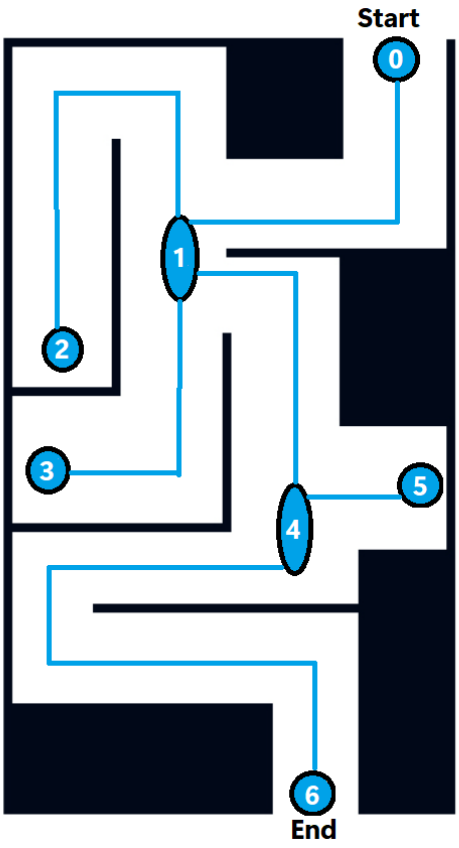
Marking feedback:

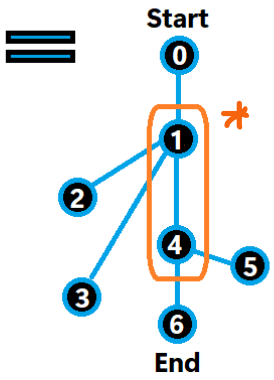
Best case is O(nlog(n)). There are log(n) number of levels (iterations), but in each level we need to do roughly n comparisons in total to partition the list. Worst case complexity is O(n^2) because we have n levels (iterations) and in each level we need to do roughly n comparisons to partition the sub-lists.

(1/3 marks lost)

Task 3 - Graphs and depth first search using stacks

Step 1: For this task, represent the above maze using a graph, placing the vertices at appropriate places and joining them by edges as appropriate.





Common "supernode"; these may be consolidated into a single node (subject to process application).

```
Step 3:
Graph represented
as adjacency list.

0 -> 1
1 -> 2,3,4*
2 -> 1
3 -> 1
4*-> 1,5,6
5 -> 4
6 -> 4
```

Marking feedback:

In the maze, vertex 1 is connected t o 0 too. You need to add 0 to the neighbours list of vertex 1

Step 2: Use *depth first search* to find the solution.

Use a stack to keep track of the path. Show the state of the stack at each step.

Iteration	i0	i1	i2	i3	i4	i5	i6	i7-i10
Output: Visited = [ ] (cumulative: i=i+1)	[ ]	[**0**]	[0,**1**]	[...1,**2**]	[...2,**3**]	[...3,**4***]	[...4,**6**]	[0,1,2,3,4,6] (end)
Neighbours = [ ]	[0]	[1]	[~~0,~~ 2,3,4*]	[~~1~~]	[~~1~~]	[~~1,~~ 5,6]	[*4*] -end-	{final}

(Pop x4)

Stack							6	
(Additions= push; removals = pop)				2	3	4	4	
			1	1	1	1	1	
		0	0	0	0	0	0	

*6 prior to 5 due to left-hand branch method

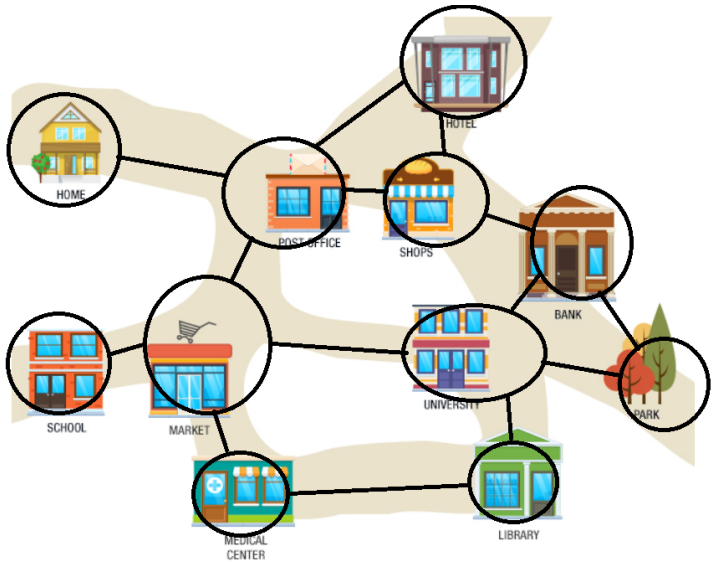
Solution as given by the stack.

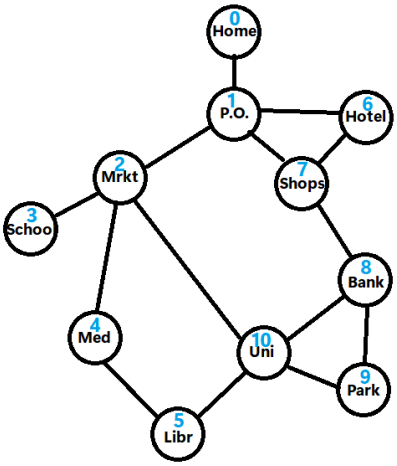
Output:
Visited = [0,1,2,3,4,6]

Stack = [ ]
Stack in i6, prior to 4x pop = [0,1,4,6]

Task 4 - Graphs and breadth first search using queues

Step 1: Graph representation





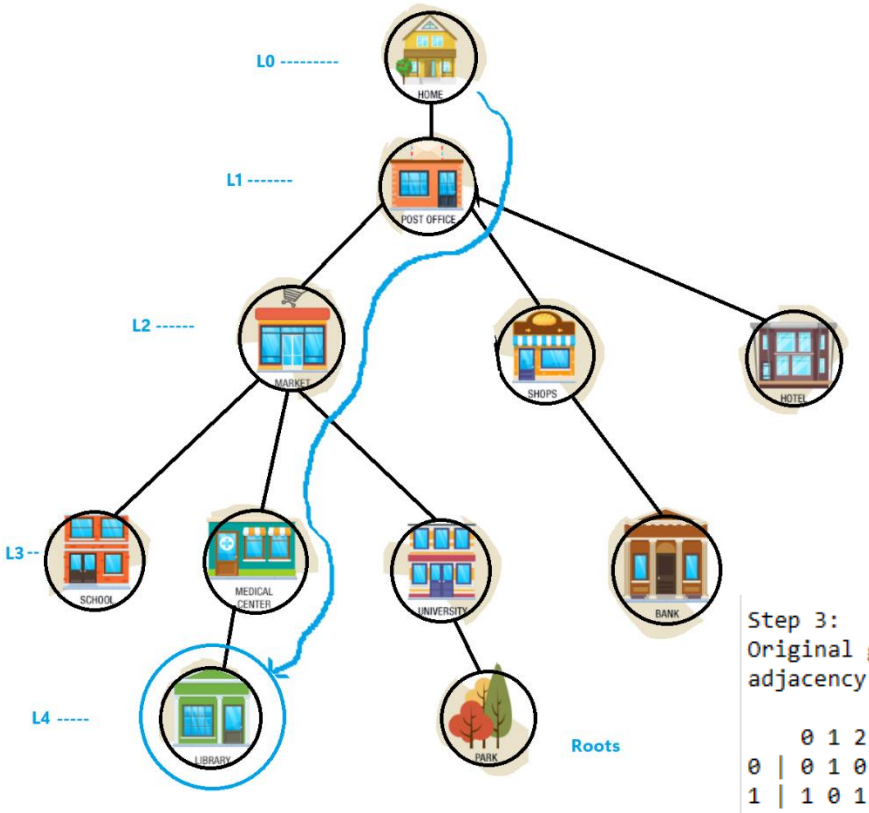
Step 2: Find a path from 'Home' to the 'Library', using breadth first search.

- Keep track of the visited places (vertices) in an array.
- Keep track of the parent vertices in a separate array.
- Use a queue to keep track of the neighbours and show the state of the queue at each step.
- Stop the search once you reach the desired destination.

Iteration	Visited = [ ]	Neighbours = [ ]	Distance = [ ]
i0	[ ]	[0]	[0]
i1	[0]	[1]	[0]
i2	[0,1]	[2,6,7]	[1]
i3	[0,1,2]	[6,7,3,4,10]	[2]
i4	[0,1,2,6]	[7,3,4,10]	[2]
i5	[0,1,2,6,7]	[3,4,10,8]	[2]
i6	[0,1,2,6,7,3]	[4,10,8]	[3]
i7	[0,1,2,6,7,3,4]	[10,8,5]	[3]
i8	[0,1,2,6,7,3,4,10]	[8,5,9]	[3]
i9	[0,1,2,6,7,3,4,10,8]	[5,9]	[3]
i10 - ***end***	[0,1,2,6,7,3,4,10,8,5]	[9]	[4]
i11	~~*[0,1,2,6,7,3,4,10,8,5,9]*~~	~~*[ ]*~~	~~*[4]*~~

Iteration	Vertex	0	1	2	3	4	5	6	7	8	9	10	
i0	Parent = [...	None	None	None	None	None	None	None	None	None	None	None	]
i1	= [	None	0	None	None	None	None	None	None	None	None	None	]
i2	= [	None	0	1	None	None	None	1	1	None	None	None	]
i3	= [	None	0	1	2	2	None	1	1	None	None	2	]
i4	= [	None	0	1	2	2	None	1	1	None	None	2	]
i5	= [	None	0	1	2	2	None	1	1	7	None	2	]
i6	= [	None	0	1	2	2	None	1	1	7	None	2	]
i7	= [	None	0	1	2	2	4	1	1	7	None	2	]
i8	= [	None	0	1	2	2	4	1	1	7	10	2	]
i9	= [	None	0	1	2	2	4	1	1	7	10	2	]
i10 ***- end***	= [	None	0	1	2	2	4	1	1	7	10	2	]
i11	**Parent = [**	**None**	**0**	**1**	**2**	**2**	**4**	**1**	**1**	**7**	**10**	**2**	**]**

Obtain the path from the source to the destination using the parent list. List the places you would pass on your way.




```
Step 3:
Original graph represented as
adjacency matrix.

    0 1 2 3 4 5 6 7 8 9 10
0 | 0 1 0 0 0 0 0 0 0 0 0    Home
1 | 1 0 1 0 0 0 1 1 0 0 0    PO
2 | 0 1 0 1 1 0 0 0 0 0 1    Mrkt
3 | 0 0 1 0 0 0 0 0 0 0 0    School
4 | 0 0 1 0 0 1 0 0 0 0 0    Med Centre
5 | 0 0 0 0 1 0 0 0 0 0 1    Library
6 | 0 1 0 0 0 0 1 0 0 0 0    Hotel
7 | 0 1 0 0 0 0 1 0 0 0 0    Shops
8 | 0 0 0 0 0 0 0 1 0 1 1    Bank
9 | 0 0 0 0 0 0 0 0 1 0 1    Park
10| 0 0 1 0 0 1 0 0 1 1 0    Uni
```