Question 1:

Lazy Version MST	Eager Version MST	Lazy Version PQ	Eager Version PQ
A-C	A-C	AC AB AJ	AC AB AJ
C-J	C-J	CJ CD CB AB AJ	CJ CD CB
C-D	C-D	CD JE CB AB	CD JE CB
B-D	B-D	BD JE CB AB DH	BD JE DH
B-H	B-H	BH JE DH BI	BH JE BI
H-I	H-I	HI HG JE HF BI	HI HG JE HF
H-G	H-G	HG JE HF	HG JE HF
J-E	J-E	JE HF GF	JE HF GF
E-F	E-F	EF HF GF	EF HF GF

As you can see in the table, first edge in the priority queue is same both versions of Prim's algorithm. But, in lazy version of Prim's algorithm, much space used on PQ. So, Eager version is much efficient.

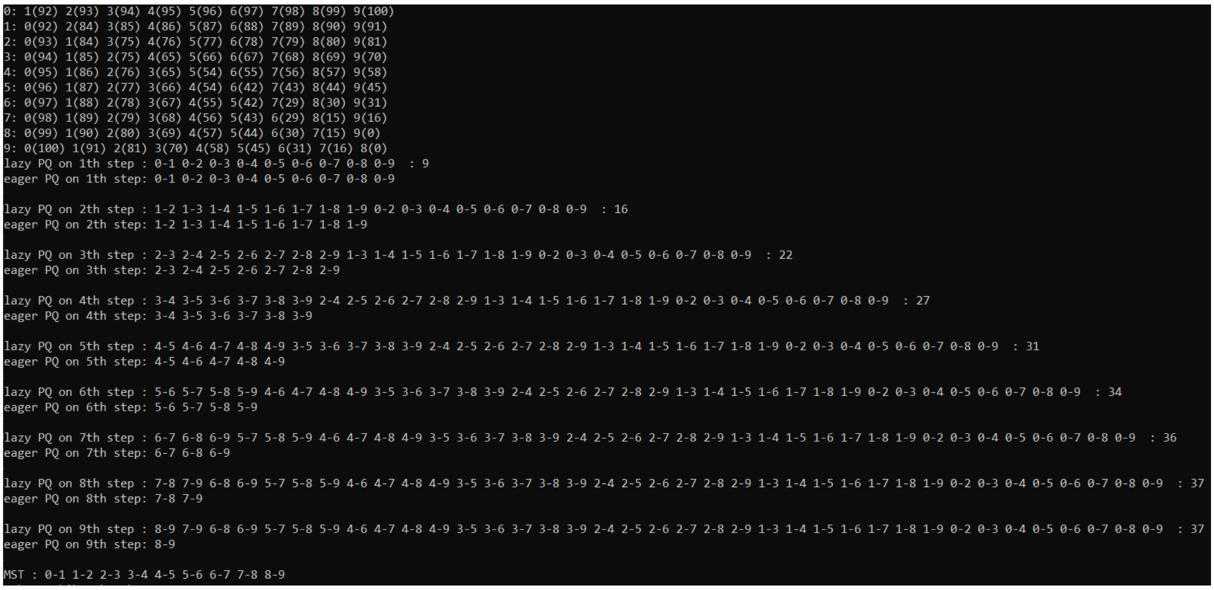
Question 2:

a. For worst case; all vertices must be connected with all other vertices and whenever we add new vertex to mst, we must refresh all edges with better ones. For example; we started on "0". Added all edges adjacent to "0". Selected minimum ("1"). Looked for adjacent edges to "1". All edges adjacent to "1" is better than edges adjecent to "0". Thus, we refreshed edges with better ones. Every time we add a new vertex to mst; we must refresh all edges on priority queue. Becasue new edges better than the old edges.

Size of PQ for eager version : 9 --> 8 --> 7 --> 6 --> 5 --> 4 --> 3 --> 2 --> 1

b. For Worst case; all vertices must be connected with all other vertices;

Size of PQ for lazy version : 9 --> 16 --> 22 --> 27 --> 31 --> 34 --> 36 --> 37 --> 37



```
0: 1(92) 2(93) 3(94) 4(95) 5(96) 6(97) 7(98) 8(99) 9(100)
1: 0(92) 2(84) 3(85) 4(86) 5(87) 6(88) 7(89) 8(90) 9(91)
2: 0(93) 1(84) 3(75) 4(76) 5(77) 6(78) 7(79) 8(80) 9(81)
3: 0(94) 1(85) 2(75) 4(65) 5(66) 6(67) 7(68) 8(69) 9(70)
4: 0(95) 1(86) 2(76) 3(65) 5(54) 6(55) 7(56) 8(57) 9(58)
5: 0(96) 1(87) 2(77) 3(66) 4(54) 6(42) 7(43) 8(44) 9(45)
6: 0(97) 1(88) 2(78) 3(67) 4(55) 5(42) 7(29) 8(30) 9(31)
7: 0(98) 1(89) 2(79) 3(68) 4(56) 5(43) 6(29) 8(15) 9(16)
8: 0(99) 1(90) 2(80) 3(69) 4(57) 5(44) 6(30) 7(15) 9(0)
9: 0(100) 1(91) 2(81) 3(70) 4(58) 5(45) 6(31) 7(16) 8(0)
lazy PQ on 1th step : 0-1 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 9
eager PQ on 1th step: 0-1 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9

lazy PQ on 2th step : 1-2 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 16
eager PQ on 2th step: 1-2 1-3 1-4 1-5 1-6 1-7 1-8 1-9

lazy PQ on 3th step : 2-3 2-4 2-5 2-6 2-7 2-8 2-9 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 22
eager PQ on 3th step: 2-3 2-4 2-5 2-6 2-7 2-8 2-9

lazy PQ on 4th step : 3-4 3-5 3-6 3-7 3-8 3-9 2-4 2-5 2-6 2-7 2-8 2-9 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 27
eager PQ on 4th step: 3-4 3-5 3-6 3-7 3-8 3-9

lazy PQ on 5th step : 4-5 4-6 4-7 4-8 4-9 3-5 3-6 3-7 3-8 3-9 2-4 2-5 2-6 2-7 2-8 2-9 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 31
eager PQ on 5th step: 4-5 4-6 4-7 4-8 4-9

lazy PQ on 6th step : 5-6 5-7 5-8 5-9 4-6 4-7 4-8 4-9 3-5 3-6 3-7 3-8 3-9 2-4 2-5 2-6 2-7 2-8 2-9 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 34
eager PQ on 6th step: 5-6 5-7 5-8 5-9

lazy PQ on 7th step : 6-7 6-8 6-9 5-7 5-8 5-9 4-6 4-7 4-8 4-9 3-5 3-6 3-7 3-8 3-9 2-4 2-5 2-6 2-7 2-8 2-9 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 36
eager PQ on 7th step: 6-7 6-8 6-9

lazy PQ on 8th step : 7-8 7-9 6-8 6-9 5-7 5-8 5-9 4-6 4-7 4-8 4-9 3-5 3-6 3-7 3-8 3-9 2-4 2-5 2-6 2-7 2-8 2-9 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 37
eager PQ on 8th step: 7-8 7-9

lazy PQ on 9th step : 8-9 7-9 6-8 6-9 5-7 5-8 5-9 4-6 4-7 4-8 4-9 3-5 3-6 3-7 3-8 3-9 2-4 2-5 2-6 2-7 2-8 2-9 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-2 0-3 0-4 0-5 0-6 0-7 0-8 0-9  : 37
eager PQ on 9th step: 8-9

MST : 0-1 1-2 2-3 3-4 4-5 5-6 6-7 7-8 8-9
```

c. O(V) space and O(E logV) time
d. O(E) space and O(E logE) time

Question 3:

```
Q3 output:
Maze:
        0       4       10      10      10
        1       8       1       1       1
        1       8       1       10      1
        1       1       1       10      1
        10      10      10      10      2
Maze distances: (Dijkstra's algorithm)
        0       4       14      24      34
        1       9       10      11      12
        2       10      11      21      13
        3       4       5       15      14
        13      14      15      25      16
Path: 0->5 5->6 6->7 7->8 8->9 9->14 14->19 19->24
```

I used Dijkstra's algorithm for calculating shortest path.
Firstly calculated distances with Dijkstra's algorithm.
Secondly started from target location and went back (lesser distance).
Thirdly, used StringBuilder to print and printed reversely.