|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

**Question 1:**

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.

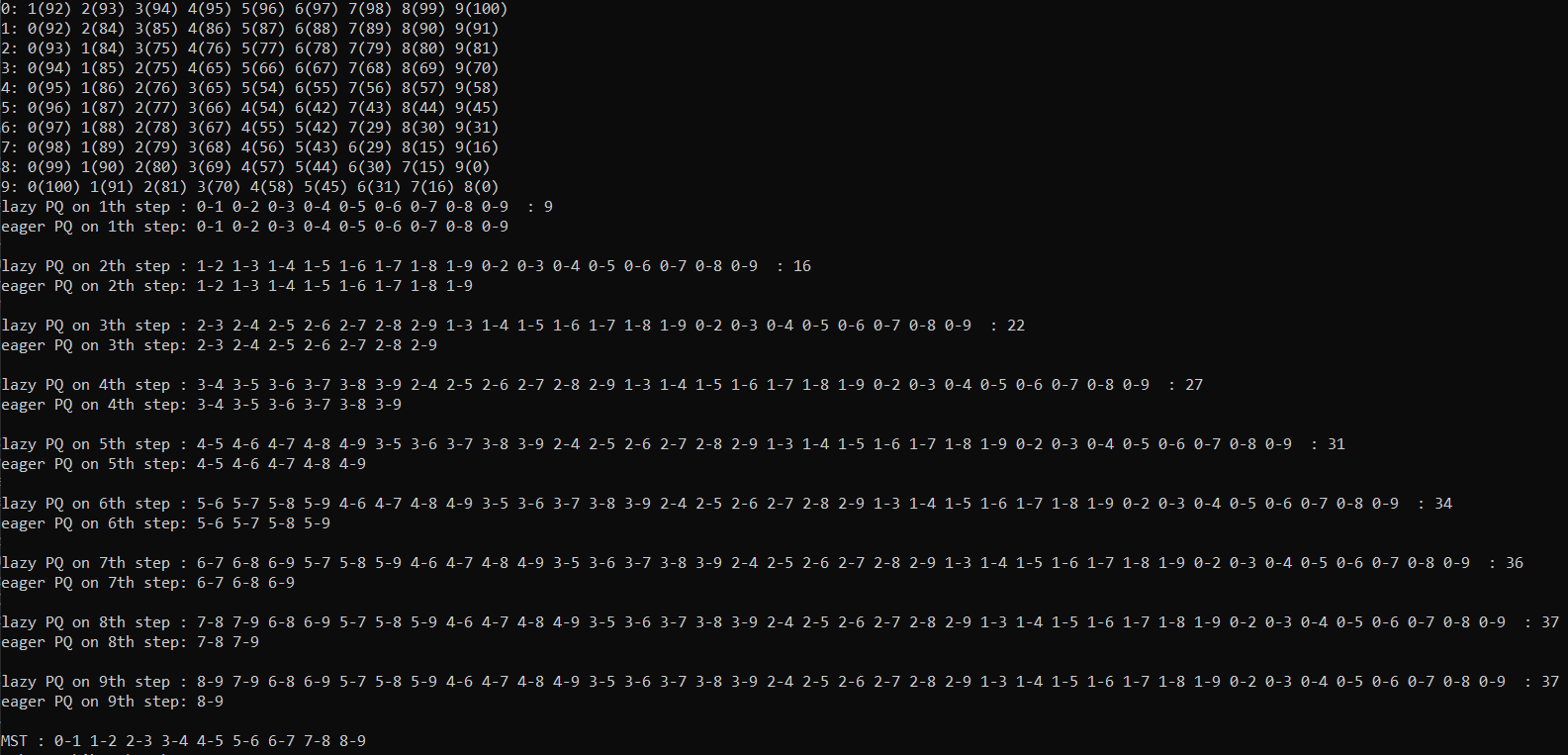
**Question 2:**

**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;

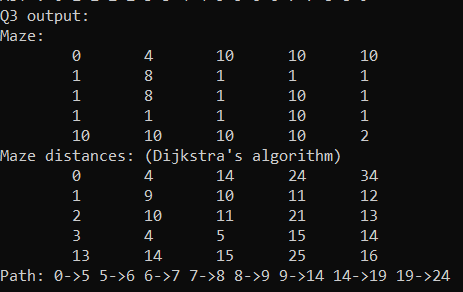
Size of PQ for lazy version : 9 --> 16 --> 22 --> 27 --> 31 --> 34 --> 36 --> 37 --> 37



**c.** O(V) space and O(E logV) time

**d.** O(E) space and O(E logE) time

**Question 3:**

****

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.