3. (a)

Test file	**Methods**	**Maximum Clique**	**Cardinality**	**Running time(seconds)**
C125.9	GUROBI	{0,1,4,6,8,10,17,18,24,28, 30,33,39,43,44,47,48,53,67, 69,70,76,78,79,98,100,109, 113,114,116,120,121,122, 124}	34	0.57
	MCR	{0,1,4,6,8,10,17,24,28,33,43, 47,48,53,59,67,69,70,76,78, 79,82,84,92,98,100,109,110, 113,114,116,120, 121,122}	34	97.93
brock200_1	GUROBI	{17,19,38,67,72,80,84,86, 89,91,92,93,101,107,133, 134,135,141,149,177,185}	21	79.68
	MCR	{67,133,134,135,72,141, 80,17,19,84,149,86,89,91, 92,93,101,38,107,177,185}	21	123.64
brock200_2	GUROBI	{26,47,54,69,104,119,120, 134,144,148,157,182}	12	13.76
	MCR	{69,134,104,47,144,148,182, 119,120,54, 26,157}	12	0.41
p_hat300_1	GUROBI	{53,123,180,218,246,267, 270,286}	8	13.76
	MCR	{34,234,106,17,148,24, 250,255}	8	0.19
p_hat300_3	GUROBI	{3,18,19,20,23,32,39,47, 48,55,74,75,88,97,138,148, 159,165,173,175,189,198, 204,218,220,225,234,238, 244,246,251,254,272,289, 297,298}	36	287.07
	MCR	{3,137,138,272,17,148,20, 27,158,32,289,292,165,39, 296,297,298,173,175,48, 55,189,198,75,204,88,218, 220,97,225,234,238,244, 246,251,254}	36	Almost 5 hours Note: It can find a clique whose cardinality is 33 in 1 hour

(b)

Test file	**Running time(GUROBI)**	**Running time(MCR)**	**MCR/GUROBI**
C125.9	0.57 s	97.93 s	171.81
brock200_1	79.68 s	123.64 s	1.55
brock200_2	13.76 s	0.41 s	0.03
p_hat300_1	13.76 s	0.19 s	0.01
p_hat300_3	287.07 s	More than 1 hour	N/A

Thus, for C125.9, GUROBI is faster, MCR/GUROBI = 171.81.
For brock200_1, GUROBI is faster, MCR/GUROBI = 1.55.
For brock200_2, MCR is faster, MCR/GUROBI = 0.03.
For p_hat300_1, MCR is faster, MCR/GUROBI = 0.01.
For p_hat300_3, GUROBI is faster, MCR/GUROBI = N/A, because MCR doesn't get result in 1 hour.

For solving by GUROBI, as the number of vertices in the graph increases, GUROBI solves slower. In the case of the same number of vertices in two graphs, the larger the density of the graph, the slower GUROBI solves.

For solving by MCR, the above two rules still hold. As the number of vertices in the graph increases, MCR solves slower. In the case of the same number of vertices in two graphs, the larger the density of the graph, the slower MCR solves.

(c)
For using GUROBI, we can only change the objective from sum of X_i for i belongs to V to sum of ($weight_i$ * X_i) for i belongs to V to get the maximum weight clique.

For using MCR, when we want to get the upper bound and prune in branch and bound, we shouldn't consider the degree of each vertex now, we should consider the total weight of its all neighbors (WN). Hence, we should sort vertices by WN not by degree anymore. This applies to both the main problem and the sub problem.

(d)
It depends. Although both GUROBI and MCR are both sensitive to the number of vertices and the density of the graph, MCR is more sensitive to the density of the graph.

I will choose MCR if the density is not very high (< 0.5). Even if the number of vertices is large, MCR can solve quickly, and this is only written in python, if it is written in C++, it will be even faster.

I would choose GUROBI if density of graph is high (> 0.5), in this case GUROBI will be much faster than MCR. But I don't know how the number of vertices specifically affects the solving speed of GUROBI, maybe as the number of vertices is quite large, GUROBI will not be able to solve it, which requires further research. Because GUROBI uses linear programming, if the number of vertices and density are high at the same time, there are many constraints, which makes it difficult to solve.