Question2：

I programmed in python3 to implement the MDP. As the formula in the lecture note:

We have a state space S.

For $x \in S$, action set A(x)

For $x \in S$, $a \in A(x)$, we have immediate reward $r_{x,a}$.

For $x \in S$, $a \in A(x)$, we have transition to state s' with probability $p^{a}_{x,s'}$.

For $x \in S$:

$$U^*(s) = \max_{a \in A(s)} \left[r_{s,a} + \beta \sum_{s'} p^{a}_{s,s'} U^*(s') \right].$$

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \left[r_{s,a} + \beta \sum_{s'} p^{a}_{s,s'} U^*(s') \right].$$

And by value iteration,

$$\max_{s \in S} |U^*_k(s) - U^*(s)| \to 0 \text{ as } k \to \infty.$$

Hence,

$$|U^*_{k+1}(s) - U^*(s)| \le \beta \epsilon_k.$$

In my approach, I use the following convergence condition to accelerate the convergence,

$$\left|U^*_{k+1}(s) - U^*(s)\right| \le \frac{1-\beta}{\beta}\varepsilon.$$

And the representation of data is showed below.

State Set

New	Used1	Used2	Used3	Used4	Used5	Used6	Used7	Used8	Dead
0	1	2	3	4	5	6	7	8	9

Action Set

Used	Replace
0	1

Transition Set

Transition_used	Transition_replace

Transition used, (-infinite denotes can't be reach)

state	0	1-8	9
(probability, next state)	(1, 1)	(0.1 * i, i + 1), (1 – 0.1 * i, i)	(negative infinite, 0)

Transition replace

state	0	1-9
(probability, next state)	(negative infinite, 0)	(1, 0)

Reward Set

Reward_used	Reward_replace

Reward used, (0 denotes can't be reach and no reward)

0	1	2	3	4	5	6	7	8	9
100	90	80	70	60	50	40	30	20	0

Reward replace, (0 denotes can't be reach and no reward)

0	1	2	3	4	5	6	7	8	9
0	-250	-250	-250	-250	-250	-250	-250	-250	-250

So when I get U by value iteration, I can easily use it to find the best action to do on the exactly state.

a)

	state	$U^* \ (state)$
New	0	800.5305499262429,
Used1	1	778.3673954680371,
Used2	2	643.2212348213058,
Used3	3	556.1225096942011,
Used4	4	502.8349428957191,
Used5	5	475.8449436861682,
Used6	6	470.4773889386281,
Used7	7	470.4773889386281,
Used8	8	470.4773889386281,
Dead	9	470.4773889386281

b) 0 is used, 1 is replace.

	state	action
New	0	0
Used1	1	0
Used2	2	0
Used3	3	0
Used4	4	0
Used5	5	0
Used6	6	1
Used7	7	1
Used8	8	1
Dead	9	1

c)

Assume we just let the cost to be integer, so I modify the code to the file mdp_c.py to output the U of this problem.

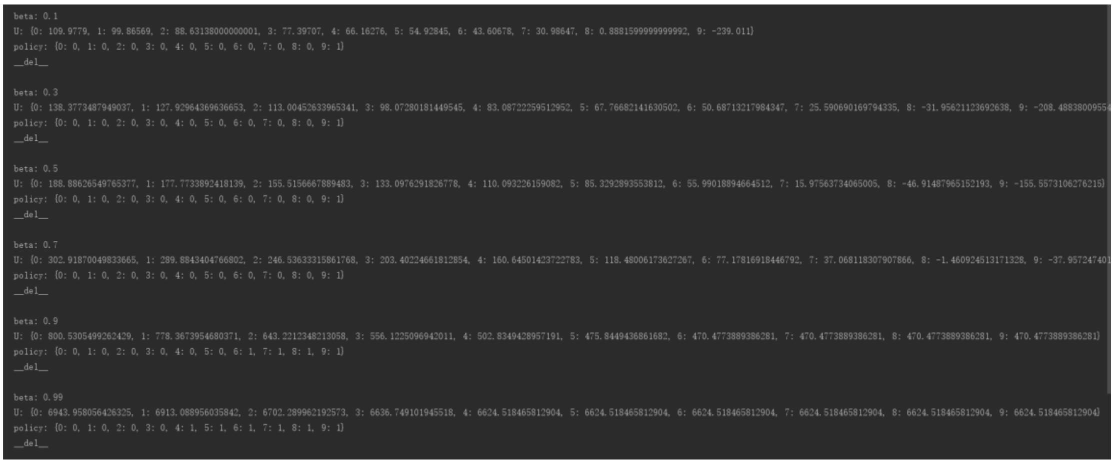
From observation, I found that when the cost is 169, all the *U*(state)* are slightly larger than the U I have got in a) with a delta = 0.8 approximately. And when the cost is 170, the U is smaller than the U in a).

So, I conclude that the highest price supposed to be 169 so that the used machine would be a rational choice.

d)

I modify the code to mdp_d.py

Just use different values of beta to run the program, so I get: when beta increases, the number of states best to *replace* is increases.



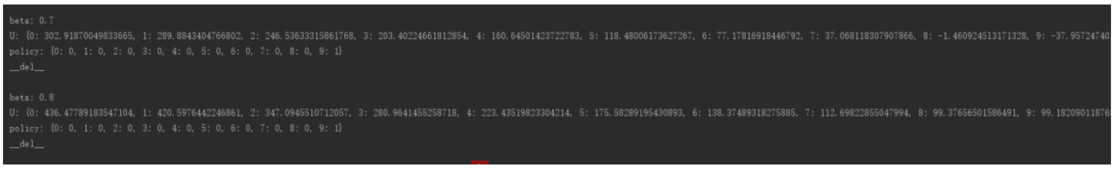
```
beta: 0.1
U: {0: 109.9779, 1: 99.86569, 2: 88.63138000000001, 3: 77.39707, 4: 66.16276, 5: 54.92845, 6: 43.60678, 7: 30.98647, 8: 0.8881599999999992, 9: -239.011}
policy: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1}
__del__

beta: 0.3
U: {0: 138.3773487949037, 1: 127.92964369636653, 2: 113.00452633965341, 3: 98.07280181449545, 4: 83.08722259512952, 5: 67.76682141630502, 6: 50.68713217984347, 7: 25.590690169794335, 8: -31.95621123692638, 9: -208.48838009554
policy: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1}
__del__

beta: 0.5
U: {0: 188.88626549765377, 1: 177.7733892418139, 2: 155.5156667889483, 3: 133.0976291826778, 4: 110.093226159082, 5: 85.3292893553812, 6: 55.99018894664512, 7: 15.97563734065005, 8: -46.91487965152193, 9: -155.5573106276215}
policy: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1}
__del__

beta: 0.7
U: {0: 302.91870049833665, 1: 289.8843404766802, 2: 246.53633315861768, 3: 203.40224661812854, 4: 160.6450142372783, 5: 118.48006173627267, 6: 77.17816918446792, 7: 37.068118307907866, 8: -1.460924513171328, 9: -37.957247401
policy: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1}
__del__

beta: 0.9
U: {0: 800.5305499262429, 1: 778.3673954680371, 2: 643.2212348213058, 3: 556.1225096942011, 4: 502.8349428957191, 5: 475.8449436861682, 6: 470.4773889386281, 7: 470.4773889386281, 8: 470.4773889386281, 9: 470.4773889386281}
policy: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 1, 8: 1, 9: 1}
__del__

beta: 0.99
U: {0: 6943.958056426325, 1: 6913.088956035842, 2: 6702.289962192573, 3: 6636.749101945518, 4: 6624.518465812904, 5: 6624.518465812904, 6: 6624.518465812904, 7: 6624.518465812904, 8: 6624.518465812904, 9: 6624.518465812904}
policy: {0: 0, 1: 0, 2: 0, 3: 0, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1}
__del__
```

So, from the outcome I could assume that there is a policy,

	state	action
New	0	Used
Used1	1	Used
Used2	2	Used
Used3	3	Used
Used4	4	Used
Used5	5	Used
Used6	6	Used
Used7	7	Used
Used8	8	Used
Dead	9	Replace

That using this policy we can have the optimal until beta is 0.7. But for beta larger than 0.7, we may not have a optimal policy for all beta.

Bonus)



```
beta: 0.7
U: {0: 302.91870049833665, 1: 289.8843404766802, 2: 246.53633315861768, 3: 203.40224661812854, 4: 160.6450142372783, 5: 118.48006173627267, 6: 77.17816918446792, 7: 37.068118307907866, 8: -1.460924513171328, 9: -37.95724740
policy: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1}
__del__

beta: 0.8
U: {0: 436.47789183547104, 1: 420.5976442246861, 2: 347.0945510712057, 3: 280.9641455258718, 4: 223.43519823304214, 5: 175.58289195430893, 6: 138.37489318275885, 7: 112.69822855047994, 8: 99.37656501586491, 9: 99.1820901187
policy: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1}
__del__
```

For beta = 0.8, I found all U is bigger than 0, and beta smaller than 0.7 there are always negative Us, so I think the long term discounted value ***x*** may be between 0.7 and 0.8. When it is small than ***x***, we will get net loss and otherwise we can get a net gain.