	N	5	10	30	50	80	100
O(N^6)version	Iterations(K)	10000	1000	100	10	1	1
	Ticks	42	120	5335	11091	18749	66946
	Total time(sec)	0.042	0.12	5.335	11.091	18.749	66.946
	Duration(sec)	4.2*10^-6	1.2*10^-4	0.05335	1.109	18.749	66.946
O(N^4)version	Iterations(K)	10000	1000	100	100	1	1
	Ticks	16	16	103	792	49	110
	Total time(sec)	0.016	0.016	0.103	0.792	0.049	0.11
	Duration(sec)	1.6*10^-6	1.6*10^-5	0.00103	0.00792	0.049	0.11
O(N^3)version	Iterations(K)	10000	10000	1000	1000	100	100
	Ticks	13	63	111	582	239	405
	Total time(sec)	0.013	0.063	0.111	0.582	0.239	0.405
	Duration(sec)	1.3*10^-6	6.3*10^-6	0.000111	0.000582	0.00239	0.00405

Analysis:

The O(N^6) version of function calculates the sum of every possible submatrix to find the maximum sum. It searches for every submatrix by locating its first and last elements, thus having a time complexity of O(N^6) and a space complexity of O(1). The O(N^4) version of function saves the current sum into an array. For each element in the matrix, it takes N^2 to find the biggest sum. It has a time complexity of O(N^4) and a space complexity of O(N). The third version is an upgraded and advanced version of the O(N^4) version with a time complexity of O(N^3) and space complexity of O(N^2).

Comments:

The three algorithms our team design fits the demand of time and space complexity. They all can correctly find the maximum matrix sum. They are well designed and perfectly functional. As the table illustrated, when N is small enough, three versions are almost as fast as each other. When N is bigger, the gap between the efficiency of the algorithms becomes more evident, which is because the time complexity of the last two algorithms grows way more dramatically than the first one.