Average Case Analysis
(Fill in the table cells with execution times)

	InpType1			InpType2			InpType3			InpType4		
	n=100	*n*=1000	*n*=10000	*n*=100	*n*=1000	*n*=10000	*n*=100	*n*=1000	*n*=10000	*n*=100	*n*=1000	*n*=10000
Ver1	0.05122	0.76646	9.97988	0.04952	0.7535	10.8068	0.04734	0.75118	10.787	0.04408	0.61464	8.44554
Ver2	0.07978	1.05118	13.04498	0.07502	1.01684	13.24274	0.07326	0.974	12.68394	0.07112	0.86206	11.36128
Ver3	0.07062	1.0833	12.58428	0.07958	0.9632	12.31948	0.06538	0.9371	11.80506	0.06214	0.92312	10.62822
Ver4	0.0739	0.7514	10.28702	0.05186	0.74746	10.08738	0.05146	0.75308	10.12868	0.05052	0.71662	9.53884

Comments:
(Write your detailed comments about the average case running times)
(Worst case results and comments are on the next page)

*The execution times are measured as milliseconds.

I implemented quick sort algorithm in the class we covered.

In the averaged cases inputs are produced randomly. First of all, the greater number of inputs will exhibit complexities better. Therefore, we will comment about 10000 length cases. For all input types, version 2 and 3 are slower, and the others are close. Version 2 has random integer selection in each base selection in the quick sort. Therefore, this should cause algorithm to slower little. Version 3 has shuffling before sorting so it is also slow.

We can observe that, in random lists, base selection is not so much important. This is because it will not be a very bad base as it will not be a very good base probably. Therefore, between versions there is not so much difference. The difference is a little due to details outside of sorting and it is not effective near sorting.

Between input types, we observe that the number of distinct values is less while type number is increasing, which enables our algorithm to be faster. In the quick sort algorithm, the groups should have equal size as much as possible. In other words, if the base is median of the array, this has good effect on algorithm execution time. In the code we see “left” and “right” variables. These variables represent indexes. If these indexes are equal at length of list/2 the list is divided into equal parts, so if each element is equal in the list, left and right variable will encounter at the median. Thus, the execution time should be less with quick sort algorithm in the class. These clarifies why when distinction is more, algorithm gets slower.

In the input type 4 wherever you select root, there is not any difference, all values are equal and all bases are equal. Therefore, the difference in the version 2 and 3 arise from random selection and shuffling.

Lastly, we will mention input sizes. Let us look at version 1, which is basic quick sort. When input size is 1000 the time is 0.75 approximately, when input size is 100 the time is 0.05. The ratio between them is 15, which is less than n^2 complexity algorithm and more than n complexity algorithm. $(n + 1)(2 \ln(n + 1) - 3)$ is the formula for the algorithm for a(n) of quick sort. The ratio of time does not seem wrong with this formula.

To conclude, in the average case version 2 and 3 are a little bit slower than version 1 and 4. Version 4 will have better base selection than version 1, but in these cases we do not encounter with important difference. Also, all versions take advantage of the usage of less distinction.

Worst Case Analysis
(Fill in the table cells with execution times)

	InpType1			InpType2			InpType3			InpType4		
	n=100	*n*=1000	*n*=10000	*n*=100	*n*=1000	*n*=10000	*n*=100	*n*=1000	*n*=10000	*n*=100	*n*=1000	*n*=10000
Ver1	0.1834	20.0971	2132.621	0.1243	11.389	1408.9696	0.0908	7.9469	912.2566	0.0438	0.614	8.4679
Ver2	0.0757	0.9243	11.7673	0.0698	0.9199	11.5914	0.0682	0.9077	11.3779	0.0718	0.8597	11.3608
Ver3	0.067	1.0216	12.4942	0.066	0.9097	11.7437	0.0649	0.952	11.8235	0.0621	0.8568	11.3344
Ver4	0.0441	0.5563	7.5703	0.0455	0.5924	7.6255	0.0485	0.6654	9.7865	0.0506	0.7087	9.3646

Comments:
(Write your detailed comments about the worst case running times)

*The execution times are measured as milliseconds.

I implemented quick sort algorithm in the class we covered.

In the worst cases inputs are sorted. First, the greater number of inputs will exhibit complexities better. Therefore, we will comment about 10000 length cases. For all input types except input type 4, version 1 is much slower, and the others are close. Version 1 is much slower because, its base selection is very bad. It takes the smallest element, and this causes bad partition. The other 9999 elements are in the next partition. These will be reduced by 1 in each quick sort call. Thus, the worst case of input type 1 and version 1 quick sort has Θ (n^2) complexity. Version 2 takes random base, so the base selection will not be as bad version 1. Version 3 will not have much difference with average case, because they are both randomized. The critical fact is that version 4 is faster in sorted array. In version 4, the algorithm compares smallest, largest and median of list. Surely, in every iteration the median of parted list will be selected. This means, the algorithm parts list with equal sizes in each iteration. Therefore version 4 has better execution time than average case. Also, its base selection does not take much time.

In version 1, the distinction is much important as we see from the table. While distinction is less, the time is lowered. This can be expressed as, the effects of being ordered is less seen. As we explained in average complexity part, if all elements are same the base selection will be good definitely. Therefore, at input type 4 the execution time of version 1 is so much lowered. In this case, there will be better base selection while the distinction is less.

Version 2 and version 3 do not have much difference between input types. They have similar values with average cases. This is because they are still randomized even if the list sorted. In the version 4 the algorithm gets slower when the distinction is less. This may be due to interchanging elements. In the sorted array, there will much less interchanging with more distinction. This is because, base is median in both cases but when all elements are same also there will be interchange in the list. This does not affect much, but our assumption is that with this observation. Also input type 4 does not care of the order of list, since all elements are same.

Lastly, we mention about input sizes. We see that for input type 1 and version 1 the execution times have ratio approximately n^2 while n is increasing, which is parallel with our expression.

To conclude, version 1 is much slower in the worst case and version 4 executes the best performance among them except for input type 4. As overall, version 4 has better end efficient base selection, so it has less execution time generally.