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	0.00019	0.00338	0.04340	0.00026	0.00489	0.04352	0.00016	0.00619	0.04383	0.00015	0.00537	0.03979
Ver2	0.00025	0.00344	0.04401	0.00036	0.00329	0.04825	0.00023	0.00345	0.04167	0.00023	0.00294	0.03706
Ver3	0.00027	0.00371	0.05808	0.00049	0.00370	0.04887	0.00026	0.00361	0.05614	0.00025	0.00342	0.04836
Ver4	0.00023	0.00320	0.03949	0.00040	0.00367	0.03798	0.00021	0.00320	0.03854	0.00021	0.00297	0.03709

Comments:

Ver1. The classical deterministic algorithm. The pivot is chosen as the first element of the list.

Ver2. The randomized algorithm. The pivot is chosen randomly. This is the algorithm “Quicksort (1st version)” in the course slides.

Ver3. The randomized algorithm. This is the algorithm “Quicksort (2nd version)” in the course slides.

Ver4. The deterministic algorithm. The pivot is chosen according to the “median of three” rule.

The strength of the quicksort algorithm is that is divides the data in half and then sorts each half and merges the results. This behavior is what gives it an O(N log N) complexity in the average case for the deterministic algorithm.

Number of pivot operations needed decreases through the input types because number of different elements also decreases. In a sorted list having same elements leads to the algorithm faster. For the Input type 4, worst case and average case does not differentiate due to all elements being same. So sorting them does not change anything. Worst and average case running times will give similar results for input type 4.

Comparison of the results at the average case analysis table:

- For the Input type 1 , type 2 ,type 3 and type 4 , Classical deterministic algorithms (Version 4) and (Version 1) is the fastest ones. Because list is unsorted and it divides the data in half and sorts each half. Therefore, the fastest runtime (unlike in the worst case) can be achieved without the need for random pivot selection by probabilistic algorithms. It has a O(N log N) complexity in the average case.

- We could not see any significant difference between probabilistic algorithms (ver 2 and ver 3) .They have the same speed and their behavior is (N log N) complexity in the average case. As we already know from the literature, in the probabilistic algorithm usually worst case, average and best case running times are close to each other expected results (Bexp , Aexp , Wexp) are (N log N) . Maybe due to permutation operation in the list, version 3 is slightly slower (not significantly) than version 2 algorithm.

- When we examine the version 4 algorithm (deterministic algorithm which uses “median of three rule”) , it is observed that the average case running time is very close to version 1 (which pivot is chosen as the first element of the list) . For the Input type 1 and size=10000 it is faster than the version 1. So using the median of three approach can improve the running time on large size lists with mostly distant integers. When much more duplicate elements in the list, all algorithm versions will approach the same results.

- As a result, if we compare the results of the average case, all of the algorithms have very close results, we did not see any significant difference. It can be said that the fastest running algorithm is version 4 (deterministic algorithms) since it gives faster results with most of the input types. Probabilistic algorithms (ver. 2 and ver. 3) also have worked at very close speed when compare with deterministic ones and there is no difference between them in terms of complexity. So it can be said that all algorithms for average case work at close speeds.

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	0.00073	0.09080	8.02893	0.00047	0.07027	5.51239	0.00043	0.03702	3.71974	0.00016	0.00224	0.03188
Ver2	0.00025	0.00318	0.04151	0.00023	0.00304	0.06689	0.00023	0.00297	0.03721	0.00027	0.00291	0.03937
Ver3	0.00027	0.00351	0.06310	0.00026	0.00343	0.07038	0.00026	0.00362	0.04384	0.00026	0.00343	0.04334
Ver4	0.00049	0.03615	4.12371	0.00036	0.02473	2.82178	0.00031	0.01812	1.90339	0.00022	0.00283	0.03809

Comments:

Ver1. The classical deterministic algorithm. The pivot is chosen as the first element of the list.
Ver2. The randomized algorithm. The pivot is chosen randomly. This is the algorithm “Quicksort (1st version)” in the course slides.
Ver3. The randomized algorithm. This is the algorithm “Quicksort (2nd version)” in the course slides.
Ver4. The deterministic algorithm. The pivot is chosen according to the “median of three” rule.

From the lecture / literature we learned , the worst-case running time (expected worst case) for the probabilistic quicksort algorithm is θ (N log N) and the classical deterministic quicksort algorithm is O(n^2). This occurs when the array is already sorted in ascending or descending order and the pivot is always chosen as the first element of the array. But in probabilistic algorithms , we can hardly see this possibility because it chooses the pivot randomly.
The probabilistic quicksort algorithms have a lower running time compared to the classical deterministic quicksort algorithm because it uses a random pivot selection strategy. This helps to avoid the worst-case running time of O(n^2) more often than the classical deterministic quicksort algorithm, which always uses the first element as the pivot. In the probabilistic algorithm usually worst case, average and best-case running times are close to each other (they are equal) , so expected results (Bexp , Aexp , Wexp) are θ (N log N) .
It's worth noting that the probabilistic quicksort algorithm is not a deterministic algorithm because it uses a random pivot selection strategy. This means that the running time of the algorithm can vary from one execution to another, even when given the same input.
The strength of the quicksort algorithm is that is divides the data in half and then sorts each half and merges the results. This behavior is what gives it an O(N log N) complexity in the average case for the deterministic algorithm. However, this is based on the fact when we divide that data in half, we actually divide it in half. However, this is not always true. We aren't just dividing the array, we are partitioning into two pieces but comparing each element to the partition element (say P). If an element is less than or equal to P, then it goes in the left sub-array otherwise is goes in the right sub-array. So, the size of the sub-arrays is heavily dependent on the partition element. If P is equal to the largest or smallest value in the array, then one of the sub-arrays will be empty and the quicksort will gain nothing from the "divide phase". If this continues each time, then the running time of the algorithm becomes O(N^2) in the worst case. The "median of three" methods prevent the partition element from being the largest or smallest element of the array by making it the middle-valued element of three chosen elements. It makes the worst case much more unlikely to occur in any actual sort.
Comparison of the results at the worst-case analysis table:

- When we compare the results at the above table, for Input Type 1 classical deterministic algorithms (ver 1 and ver 4) work slow but probabilistic algorithms (ver 2 and ver3) work significantly faster than other two algorithms. If the input type change (for type 2 ,type 3) , in this case there will be duplicate elements in the list. So, the difference between the running time starts to decrease. When there are duplicate elements in the list, deterministic algorithms starts running faster. They still can't run faster than probabilistic algorithms. In input type 4, all elements in the list are the same. That's why deterministic algorithms(ver 1 and ver 4) start working very fast. It gives close results with probabilistic algorithms. Even version 1 can give results faster than probabilistic algorithms.

- When we compare the worst cases of the two deterministic algorithms, we observe that the pivot selection method significantly affects the algorithm running time. In the version 1 , the pivot is chosen as the first element of the list and in the version 4 , the pivot is chosen according to the median of three rule. Selecting the element in the middle of the three elements from the beginning, ending and the middle of the list as the pivot speeds up the algorithm considerably. This is because in version 1, the leftmost element of the list was consistently selected as the pivot, and this element was the smallest number in the list. This was causing the recursive algorithm to be called much more and causing the algorithm complexity (for ver 1) to approach O(n^2) .

- If we compare the two probabilistic algorithms (version 2 and version 3) with each other, we do not observe that there is a significant difference between them. We observe that version 3 works a little bit slower than (not significantly) version 2. This may be because the list permutation is done in version 3.

- As a result, it can be said that the slowest working algorithm for the worst case is the classic deterministic algorithm (version 1) . But if the input type is changed and lists with more duplicate elements are given, this algorithm also speeds up. For example, in input type 4, while all elements are equal to 1 , classical deterministic algorithms produced same results with probabilistic algorithms.

In general, it can be said that the fastest working algorithms are probabilistic algorithms. They are fast regardless of input type.