Average Case Analysis

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 | 59.77 microseconds | 1376.73 microseconds | 12567.27 microseconds | 57.48 microseconds | 935.48 microseconds | 13008.10 microseconds | 62.61 microseconds | 944.63 microseconds | 12728.96 microseconds | 208.77 microseconds | 18768.94 microseconds | 1986307.43 microseconds |
| Ver2 | 94.19 microseconds | 2273.24 microseconds | 15362.92 microseconds | 93.85 microseconds | 1305.88 microseconds | 15470.24 microseconds | 99.10 microseconds | 1360.06 microseconds | 16016.29 microseconds | 261.85 microseconds | 19788.83 microseconds | 2037076.92 microseconds |
| Ver3 | 83.88 microseconds | 1694.48 microseconds | 14270.51 microseconds | 85.46 microseconds | 1174.18 microseconds | 14465.63 microseconds | 83.75 microseconds | 1191.68 microseconds | 14400.33 microseconds | 240.52 microseconds | 19461.81 microseconds | 2019306.33 microseconds |
| Ver4 | 271.57 microseconds | 4105.22 microseconds | 25713.30 microseconds | 272.97 microseconds | 2866.90 microseconds | 28948.37 microseconds | 328.40 microseconds | 3535.75 microseconds | 36337.54 microseconds | 416.24 microseconds | 21156.75 microseconds | 2042177.60 microseconds |

Comments: One can analyze this table for different aspects as follows:

1. We can analyze this table in the sense of size of inputs.

For all versions, execution time increases for all input types as the size increases. This is obvious because all parts of our quick sort algorithms depend on the size of the input.

1. We can analyze this table in the sense of behaviors of all four versions.

Even though we analyze average case behaviors in this table, probabilistic algorithms are actually used for making worst case behaviors better. Therefore, we would expect that their execution times will be higher than deterministic algorithms in analyzing average case behaviors.

Version two and three are probabilistic algorithms and their way of working is very similar. Version three permutes the array and chooses the first element as pivot. On the other hand, version two picks pivot in the random index of that array. The main difference is that permuting operation has more running time than the randomly choosing operation. However, we can do permuting operation just once while we can choose a random element as pivot in every recursive call in version two. As a result, even though we found that their running time is very similar as expected, version two takes relatively longer than version three.

Version one has the best execution time. It only chooses first element in array as pivot which has O(1) time complexity. It has no other costly operation such as permuting etc.

The worst version of average case is version four. The main reason is choosing the pivot as median of three. While choosing pivot in version four, we create a temporary array and do a simple bubble sort for choosing median element of this array. While creating this temporary array, we do a deepcopy operation to make sure that bubble sorting will not affect our main array. Because of the fact that both bubble sort and deepcopy operations are very costly operations, we would expect that version four’s running time behavior will be worse.

1. We can analyze this table in the sense of types of inputs.

We know that the worst input for quick sort algorithm is a sorted list. Due to this fact, we would expect that input type four which has 1 in all indexes is kind of a worst input for all versions.

In all versions, input4 has the worst behavior as expected. In input two and three, there are repetitions in the list. Repetition means that these arrays will have less element to sort. As a result, running time is better than in input four and one. In addition, probability of having repetitions in input three is relatively higher than input two. Therefore, running times in input three is relatively lower than input two.

In input one, the probability of having repetitions is very very low, so we can assume that this array in unsorted and has unique elements. As we would expect, the running times almost are not as bad as input four.

Worst Case Analysis

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 | 489.29 microseconds | 74114.92 microseconds | 4398534.50 microseconds | 283.17 microseconds | 27192.25 microseconds | 2618060.71 microseconds | 136.33 microseconds | 11923.87 microseconds | 1152021.08 microseconds | 208.79 microseconds | 18635.42 microseconds | 1973363.42 microseconds |
| Ver2 | 92.17 microseconds | 4617.96 microseconds | 16290.92 microseconds | 97.38 microseconds | 1236.79 microseconds | 15154.21 microseconds | 96.62 microseconds | 1192.92 microseconds | 15819.08 microseconds | 263.42 microseconds | 19582.04 microseconds | 2009403.21 microseconds |
| Ver3 | 86.71 microseconds | 1175.58 microseconds | 14234.54 microseconds | 87.33 microseconds | 1185.17 microseconds | 14290.08 microseconds | 84.42 microseconds | 1085.58 microseconds | 14875.87 microseconds | 240.62 microseconds | 19274.88 microseconds | 2014017.08 microseconds |
| Ver4 | 260.17 microseconds | 2438.54 microseconds | 26010.33 microseconds | 270.37 microseconds | 2834.00 microseconds | 28584.00 microseconds | 312.50 microseconds | 3313.12 microseconds | 34928.33 microseconds | 413.63 microseconds | 21004.58 microseconds | 2025564.87 microseconds |

Comments: One can analyze this table for different aspects as follows:

1. We can analyze this table in the sense of size of inputs.

For all versions, execution time increases for all input types as the size increases. This is obvious because all parts of our quick sort algorithms depend on the size of the input.

1. We can analyze this table in the sense of behaviors of all four versions.

Even though we analyze average case behaviors in this table, probabilistic algorithms are actually used for making worst case behaviors better. Therefore, we would expect that their execution times will be lower than deterministic algorithms in analyzing worst case behaviors.

Version two and three are probabilistic algorithms and their way of working is very similar. Version three permutes the array and chooses the first element as pivot. On the other hand, version two picks pivot in the random index of that array. The main difference is that permuting operation has more running time than the randomly choosing operation. However, we can do permuting operation just once while we can choose a random element as pivot in every recursive call in version two. As a result, even though we found that their running time is very similar as expected, version two takes relatively longer than version three.

Deterministic algorithms has more execution time than probabilistic algorithms. Because probabilistic algorithms are designed to decrease gap between average and worst execution times. The worst version of average case is version four. The main reason is choosing the pivot as median of three. While choosing pivot in version four, we create a temporary array and do a simple bubble sort for choosing median element of this array. While creating this temporary array, we do a deepcopy operation to make sure that bubble sorting will not affect our main array. Because of the fact that both bubble sort and deepcopy operations are very costly operations, we would expect that version four’s running time behavior will be worse.