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	*0.212*	*3.022*	*23.952*	*0.226*	*1.227*	*21.978*	*0.171*	*2.738*	*22.701*	*0.226*	*2.479*	*25.022*
Ver2	*0.239*	*2.280*	*32.889*	*0.132*	*1.821*	*27.449*	*0.228*	*2.547*	*26.567*	*0.204*	*3.317*	*31.982*
Ver3	*0.299*	*2.365*	*31.595*	*0.125*	*2.487*	*25.839*	*0.206*	*2.745*	*26.618*	*0.128*	*3.746*	*31.421*
Ver4	*0.208*	*3.035*	*29.963*	*0.192*	*2.354*	*22.725*	*0.222*	*2.066*	*22.863*	*0.223*	*2.034*	*33.027*

* Execution times are in milliseconds

Comments:

Input size is a significant factor that determines the execution time. It can be seen that when the input size increases, all execution time increases. The reason for the increase in execution time is that when input size increases, the number of basic operations that are executed increases.

Execution time decreases when the range of the inputs are decreased, and the input size stays the same. In other words, execution time decreases when the input type changes from type 1 to type 4. The reason is that the possibility of choosing a value that is closer to the median of the inputs as pivot increases when the numbers are chosen from a narrower range. For example, if range of 1000 numbers is 250 instead of 750, the possibility of choosing a value that is closer to the median as pivot is higher since the number of repeated elements is larger. Choosing the pivot closer to the median is important because if we choose the pivot closer to the median the list is divided into equal sublists. Equal sublists implies a recursion tree that has less depth, and the algorithm ends faster. If the input is always divided into equal sublists the complexity of the algorithm will be n*log n.

It can be seen that the average execution time of input type 4 is usually greater than the other input types. The input is divided equally in each step, but it is slower than the other input types. The main reason for the longer execution time is the repeated elements in the list. The repeated elements in the input type 4 slows down the algorithm because all elements are interchanged in the input type 4. The pivot is selected as 1 in input type 4. The algorithm checks the elements from the beginning and from the end. The algorithm do not increase the index that starts from beginning when it encounters a number that equals to pivot, which is 1. The similar situation occurs for the index that starts from the end of the list. When the index that starts from the end of the list encounters a number equal to the pivot it stops. The elements that the indices point are interchanged when the both indices stop from increasing and decreasing. This interchange operation occurs every step of the iteration for the input type 4. In conclusion the execution time for input type 4 is greater than the other input types due to the interchange operations that occurs in every step of the iteration.

The algorithm that chooses pivot affects execution time. For example, choosing a random element as a pivot for every call of quicksort function and shuffling the list bring an additional cost. Choosing the median of the three also brings an additional cost, but it compensates this cost by choosing more appropriate pivot, which is closer to the median of the inputs. This situation can be seen from the table. The execution time of the version 2 and version 3 are close to each other but greater than the first and fourth versions. The first version does not spend time on choosing the pivot. It directly chooses the first element as pivot. The fourth version spends time on choosing pivot, but the pivot is more accurate. For these reasons, the first version and the fourth version does not differ too much in terms of the execution time.

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	*1.358*	*67.257*	*3503.815*	*0.509*	*36.484*	*2143.632*	*0.373*	*27.579*	*1499.879*	*0.304*	*2.487*	*23.771*
Ver2	*0.325*	*2.312*	*29.151*	*0.113*	*1.918*	*24.147*	*0.216*	*2.843*	*24.243*	*0.322*	*3.292*	*30.554*
Ver3	*0.108*	*3.166*	*29.648*	*0.150*	*3.172*	*23.979*	*0.198*	*1.513*	*22.128*	*0.121*	*5.198*	*29.682*
Ver4	*0.097*	*2.141*	*25.452*	*0.163*	*2.227*	*23.816*	*0.224*	*2.449*	*26.625*	*0.173*	*2.690*	*26.765*

* Execution times are in milliseconds

Comments:

Input size is always an important factor for execution time. For the worst case inputs, as the size of the inputs increases, the execution time increases as in the case for average inputs. The main reason that the increase in execution time due to input size is the increasing number of basic operations.

The sorted input is the worst case if the algorithm chooses the first element as pivot. When the inputs are sorted, the algorithm that chooses the first element as pivot can remove only one element on each call. For example, if the input includes 20 sorted elements in the first call of the quicksort algorithm, then it should include 19 sorted elements in the second call of the quicksort algorithm. This implies that there are n-1 function calls until the algorithm terminates. The algorithm traverses all elements until the end of each call. The first index starts to search for an element that is greater than or equal to the pivot element. It finds at the first step because the pivot is the smallest element and all the elements in the input is greater than or equal to the pivot. The second index starts from the end of the input array and searches an element that is smaller than or equal to the pivot element, yet it cannot find until it reaches the start of the input array. This means if a quicksort function is called with input size n, it visits all n elements. The size of the input decreases one by one at each call of the function. For example, if the function is called with input size 20, the function will be called with input size 19 at the next step. The last function is called with 1 element. We can conclude that, on average, the function traverses n/2 elements at each step. We also found that there are n-1 steps at the worst case. So, the worst-case complexity of the quicksort algorithm is n^2, which occurs when the algorithm selects the first input element as pivot and the input is sorted.

The elements that are given to the versions of the algorithm are sorted in the worst-case analysis. Yet, the sorted elements are not the worst case for each version of the algorithm. The sorted elements are worst case only for the first version of the algorithm. Choosing the pivots randomly and shuffling the input list before selecting the pivot eliminates the undesired effects of the sorted input. When a pivot is selected randomly it can be any element in the list so it does not matter whether the input list is sorted or not sorted. It only affects the number of interchange operations between the elements of the list. When the algorithm shuffles the list before it selects pivot, the list becomes unsorted and we can consider this case as average case, where all inputs are ordered randomly.

Giving a sorted list as input is the best case for the fourth version of the algorithm, which is choosing the pivot as median of the three. The algorithm selects the first, the last and the middle elements in the median of three method. Then, algorithm finds the median of these three elements and sets it as pivot. If the given input list is sorted, the algorithm selects the smallest element, the largest element and the middle element, which is median. Then it calculates the median of these three elements. The median of these three elements is the median of the actual input list. So, the pivot becomes the median of the input list, which is a desired outcome. Selecting the pivot as median of the input list at each step implies that the list will be divided into equal sublists at each step. If the input list is divided into equal sublists, the algorithm finishes at minimum steps, which is log n.