Alp Eren İnceoğlu-2019400063

Deniz Ünal- 201940023

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
Ver 1	*0.0003760337829589844*	*0.00567836761474 6094*	0.010565519332885742	*0.0003742218017578125*	*0.005601 1199951 17187*	*0.0103 653430 9387207*	*0.0004 018783 5693359373*	*0.0057 99674 98779 2968*	*0.010 16592 97943 11523*	*0.00140 2950286 8652343*	0.015945 9590911 86523	*1.7177952 766418456*
Ver 2	*0.0004551887 5122070314*	*0.005739 97497558 5938*	0.009568071365356446	*0.0004634857177734375*	*0.005959 7969055 17578*	*0.0099 674224 853515 63*	*0.0005 168914 794921 875*	*0.0064 60332 87048 3398*	*0.010 16535 75897 2168*	*0.00165 1334762 5732421*	0.016744 7090148 92577	*1.7318488 597869872*
Ver 3	*0.0006504058 837890625*	*0.005759 62066650 3906*	0.014222860336303711	*0.0006364822387695312*	*0.007976 6273498 53516*	*0.0153 496265 411376 95*	*0.0006 525516 510009 766*	*0.0087 89920 80688 4765*	*0.014 94984 62677 00195*	*0.00167 7179336 5478516*	0.017143 4879302 9785	*1.7295658 588409424*
Ver 4	*0.0003668785 0952148436*	*0.001675 98724365 23438*	0.007773876190185547	*0.0003737926483154297*	*0.002018 2132720 947266*	*0.0081 723690 032958 99*	*0.0004 498004 913330 078*	*0.0037 63294 21997 0703*	*0.009 76648 33068 84765*	*0.00155 8732986 4501954*	0.015946 7220306 39647	*1.7130986 69052124*

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

Input type gets worse from 1 to 4 since each input has more repetitions from the previous one, where the last one is only one element repeated n times. The reason for this is increasing execution time is that the bigger repetitions the list contains, more unbalanced the split will get. This is caused by the implementation of the comparisons in the quick sort algorithm. In our case right pointer (pointer in the sense that it stores an index in an array not the data type pointer) will continue increasing when it reaches an element that is equal to the pivot while left pointer will stop when it reaches an element that is equal to the pivot. This results in higher right and left pointers compared to the optimal one. For example in the input type 4 since all elements are equal to pivot right pointer will increase until it is equal to the initial value of left and this means an array of n will split into two arrays such that one has n-1 elements and the other is 0 elements. This is the worst possible outcome, and will result in O(N^2) complexity. While randomized algorithms are usually designed to handle the worst cases, in this case they can only handle the case where the array is sorted but there are no repetitions and can not handle the case where more repetitions result in uneven splits. Adding repetitions mean that taking any two elements from the list would have a greater probability to be equal, thus shifting the position further to the right compared to optimal case.

Since this is average cases, randomization will not yield in any improvement to the execution times, while taking median of three guarantees a better pivot, since instead of taking a random pivot like the first three versions, in version 4 the best pivot out of three possibilities is chosen, which would be better in almost all cases. Other than that, in average cases version 1 is generally better than version 2, which is better than version 3. While their asymptotic complexities are same, since the results are experimental and not theoretical other operations beside the basic operation effect the execution times and version 3 contains a single shuffle, which has a complexity o(n), and version 2 finds a random number each loop, which consumes a little time more than version 1, but still not as much as shuffling like in version 3. This shows a shortcoming in the asymptotic analysis. While doing asymptotic analysis we ignore all statements other than basic statement which doesn't reflect time differences such as this. Besides all this, increasing the list size obviously increases the runtime for all cases.

Alp Eren İnceoğlu-2019400063

Deniz Ünal- 201940023

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	*0.0011470317840576172*	*0.10276436805725098*	1.2958014011383057	*0.0007627010345458984*	*0.057793617248535156*	*0.745506763458252*	*0.0004270076751708984*	*0.028768062591552734*	*0.346839427947998*	*0.0013616085052490234*	0.014948129653930664	*1.7068250179290771*
Ver2	*0.0005617141723632812*	*0.005718946456909180*	0.00896620750427246	*0.0005235671997070312*	*0.005809783935546875*	*0.009962320327758789*	*0.0005300045013427734*	*0.006448745727539062*	*0.009971618652343*	*0.0016179084777832031*	0.015946626663208008	*1.7318488597869872*
Ver3	*0.0006473064422607422*	*0.003545761108398437*	0.013954639434814453	*0.0006375312805175781*	*0.005442619323730469*	*0.014950275421142578*	*0.0006678104400634766*	*0.008898258209228516*	*0.015946626633208008*	*0.0016403198242187*	0.017143487930297*	*1.7295658588409424*
Ver4	*0.0003662109375*	*0.0015518665313720703*	0.0079727172851562*	*0.0003535747528076172*	*0.002018213272094726*	*0.008966445922851562*	*0.0004374980926513672*	*0.002801895141601562*	*0.009966611862182617*	*0.0015490055084228516*	0.015942096710205078	*1.6885592937469482*

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

In the worst case the effect of input type on the execution time mostly stay similar to average case, more repetitions result in uneven splits thus increasing the recursion depth and thus increasing number of statements run. This is observed clearly in the input type 4 as it becomes O(N^2) while it would become O(N*logN) otherwise. (this is the case for versions 2-3-4, version 1 is worst case O(N^2) regardless of the repetition of the elements.).

Effect of using probabilistic algorithms can be clearly seen in the worst case as it is the main reason why we use Sherwood algorithms. Taking random index as a pivot as in the version 2, or shuffling it beforehand and then taking the first element as in version 3, or taking different elements (first, last and middle elements in this case) and choosing median as in version 4 all serves the same purpose: in a biased input such as sorted list (it is the worst case for version 1 since it splits the array into two arrays: one with 0 element and the other with all remaining elements except pivot, thus increasing recursion depth.) they add an element of randomness to disregard that bias in the input. Because of this, version 2-3-4 have all O(N*log(N)) complexity and perform better than version 1 which has O(N^2) complexity. Out of the three remaining, version 3 performs worse because it adds extra operation of shuffling the list at the beginning which is a cost with no particular benefit compared to the remaining two versions. Out of remaining two, version 4 performs much better than version two. In fact in a sorted input version 4 will always choose the median as the pivot and this is the best case for quick sort algorithms. Since version 4 chooses median as the pivot in each iteration the array will be split to two arrays that are as close to even as possible. Which will reduce the recursion depth and thus executes faster.