| | 1000 | 3000 | 5000 | 7000 | 9000 | 11000 |
|---|---|---|---|---|---|---|
| qs1 | 10996.72 | 39617.142 | 71029.308 | 103930.394 | 138403.996 | 173707.299 |
| qs2 | 11725.66 | 40882.576 | 72446.146 | 105562.226 | 139515.856 | 174076.651 |
| qs3(k = 10) | 11171.118 | 40239.396 | 72061.51 | 105390.382 | 140267.16 | 175979.937 |
| qs3(k = 100) | 22629.122 | 74411.726 | 128878.508 | 185285.804 | 242794.526 | 301731.621 |
| qs4(p = .01) | 10996.746 | 39618.536 | 71034.47 | 103943.53 | 138424.906 | 173747.657 |
| qs4(p = .10) | 11278.502 | 43130.678 | 79916.844 | 122885.618 | 169516.446 | 231452.878 |
| qs4(p = .25) | 16493.414 | 99938.624 | 236079.482 | 454607.886 | 714832.814 | 1048181.75 |

| | 3 | 3.47712125 | 3.69897 | 3.84509804 | 3.95424251 | 4.04139269 |
|---|---|---|---|---|---|---|
| qs1 | 10996.72 | 39617.142 | 71029.308 | 103930.394 | 138403.996 | 173707.299 |
| qs2 | 11725.66 | 40882.576 | 72446.146 | 105562.226 | 139515.856 | 174076.651 |
| qs3(k = 10) | 11171.118 | 40239.396 | 72061.51 | 105390.382 | 140267.16 | 175979.937 |
| qs3(k = 100) | 22629.122 | 74411.726 | 128878.508 | 185285.804 | 242794.526 | 301731.621 |
| qs4(p = .01) | 10996.746 | 39618.536 | 71034.47 | 103943.53 | 138424.906 | 173747.657 |
| qs4(p = .10) | 11278.502 | 43130.678 | 79916.844 | 122885.618 | 169516.446 | 231452.878 |
| qs4(p = .25) | 16493.414 | 99938.624 | 236079.482 | 454607.886 | 714832.814 | 1048181.75 |

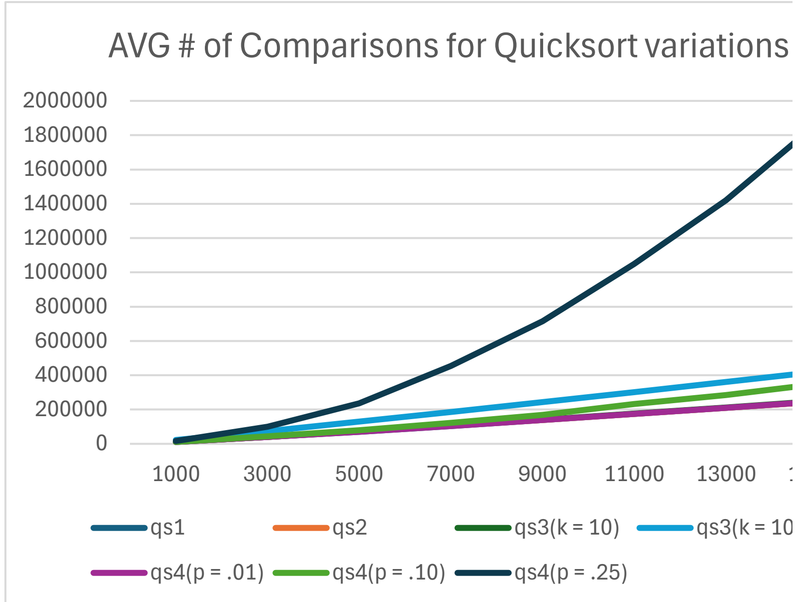
| | 1000 | 3000 | 5000 | 7000 | 9000 | 11000 |
|---|---|---|---|---|---|---|
| qs1 | 629.55926 | 1968.04088 | 3136.77685 | 4389.08507 | 5688.87745 | 7090.13237 |
| qs2 | 351.529268 | 976.039192 | 1676.00105 | 2526.1095 | 2982.80089 | 3498.0072 |
| qs3(k = 10) | 626.8603 | 1964.05029 | 3134.13707 | 4394.56348 | 5695.26214 | 7088.2555 |
| qs3(k = 100) | 1136.38994 | 2497.4235 | 3683.09574 | 5065.26027 | 6460.99512 | 7806.10196 |
| qs4(p = .01) | 629.582017 | 1969.95002 | 3143.04005 | 4400.9583 | 5726.50892 | 7134.40529 |
| qs4(p = .10) | 986.261965 | 6750.44536 | 14435.7288 | 30900.7447 | 49404.4222 | 85973.5849 |
| qs4(p = .25) | 5195.88095 | 49446.6397 | 130570.956 | 287212.706 | 437521.126 | 655476.061 |

| | 3 | 3.47712125 | 3.69897 | 3.84509804 | 3.95424251 | 4.04139269 |
|---|---|---|---|---|---|---|
| qs1 | 629.55926 | 1968.04088 | 3136.77685 | 4389.08507 | 5688.87745 | 7090.13237 |
| qs2 | 351.529268 | 976.039192 | 1676.00105 | 2526.1095 | 2982.80089 | 3498.0072 |
| qs3(k = 10) | 626.8603 | 1964.05029 | 3134.13707 | 4394.56348 | 5695.26214 | 7088.2555 |
| qs3(k = 100) | 1136.38994 | 2497.4235 | 3683.09574 | 5065.26027 | 6460.99512 | 7806.10196 |
| qs4(p = .01) | 629.582017 | 1969.95002 | 3143.04005 | 4400.9583 | 5726.50892 | 7134.40529 |
| qs4(p = .10) | 986.261965 | 6750.44536 | 14435.7288 | 30900.7447 | 49404.4222 | 85973.5849 |
| qs4(p = .25) | 5195.88095 | 49446.6397 | 130570.956 | 287212.706 | 437521.126 | 655476.061 |

Of the 7 variations of quicksort, in timing of using 1000 elements, QuickSort3 had the fastest
quicksort3 was O(nlogn) with a worst case complexity of O(n^2). When running all of the qui
chrono time standard the functions ran in speeds respectively as follows: 506306ms, 43249
520808 ms, 492269 ms, and 375993.
As you can see from the following numbers, Quicksort3 with the k=100 had the fastest variati
when passing in p=.25. This is due to the decision making of pivot selection. Quicksort4 spec

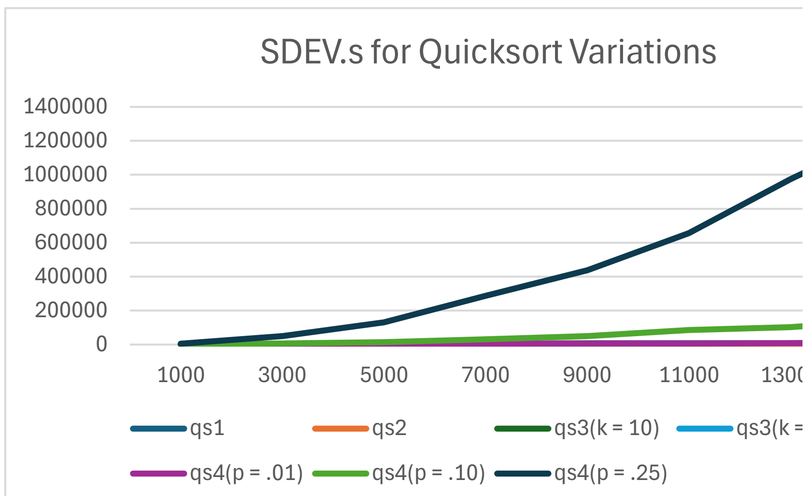
on the paramter p passed in and quicksort3 for the k respectively. In quicksort3 (our winner), elements helps with avoiding our "worst case scenarios" by comparing the last 3 elements ar before sending it to the back to be sorted by that value. In this case, if we had 8,7,9 then 8 wo sorts the other two in which the median value is chosen, this time adds up as we recursively median when we get down to the last leaves and get down to sizes of 3 for our vector. Asympt quicksort2 due to the lowest standard deviation on average as our n climbed, on the charts w numbers to a log base 10 the quicksort 2 was the fastest by this standard. Overall the basic C to the vanilla implementation.

| 13000 | 15000 |
|---|---|
| 209931.606 | 246248.62 |
| 209660.166 | 245434.028 |
| 212623.296 | 249351.874 |
| 360951.682 | 420381.17 |
| 209973.994 | 246349.116 |
| 283736.726 | 349051.184 |
| 1419401.39 | 1870341.64 |




| | |
|---|---|
| 4.11394335 | 4.17609126 |
| 209931.606 | 246248.62 |
| 209660.166 | 245434.028 |
| 212623.296 | 249351.874 |
| 360951.682 | 420381.17 |
| 209973.994 | 246349.116 |
| 283736.726 | 349051.184 |
| 1419401.39 | 1870341.64 |

| 13000 | 15000 |
|---|---|
| 9121.15676 | 9913.91485 |
| 4402.41836 | 5124.7817 |
| 9113.82216 | 9923.72086 |
| 9432.36876 | 10747.3176 |
| 9182.53363 | 10074.5154 |
| 103134.006 | 146285.567 |
| 974628.211 | 1260399.53 |

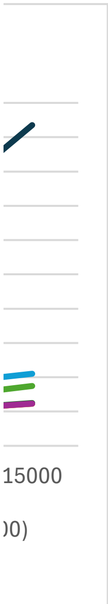


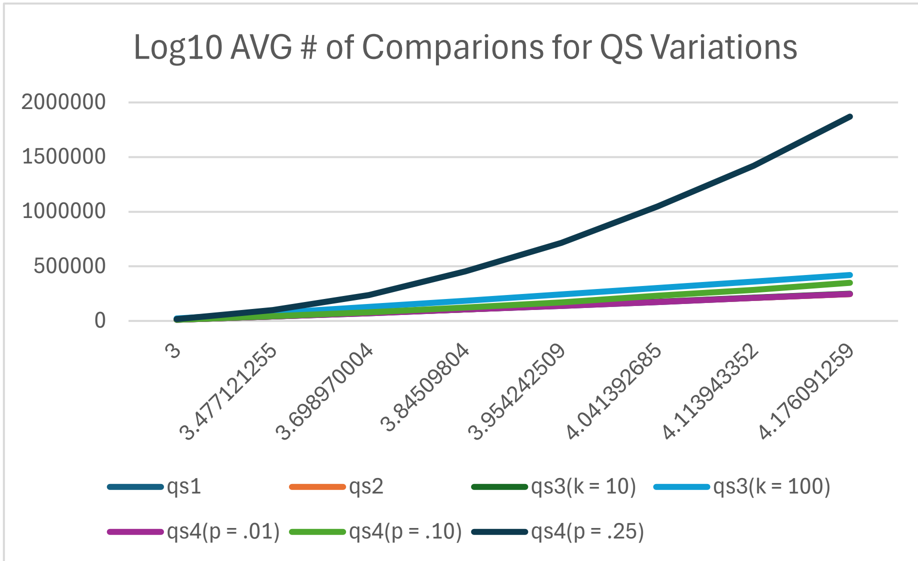

| | |
|---|---|
| 4.11394335 | 4.17609126 |
| 9121.15676 | 9913.91485 |
| 4402.41836 | 5124.7817 |
| 9113.82216 | 9923.72086 |
| 9432.36876 | 10747.3176 |
| 9182.53363 | 10074.5154 |
| 103134.006 | 146285.567 |
| 974628.211 | 1260399.53 |

: runtime. The runtime of
cksort algorithms, importing the
0 ms, 308506 ms, 205047 ms,

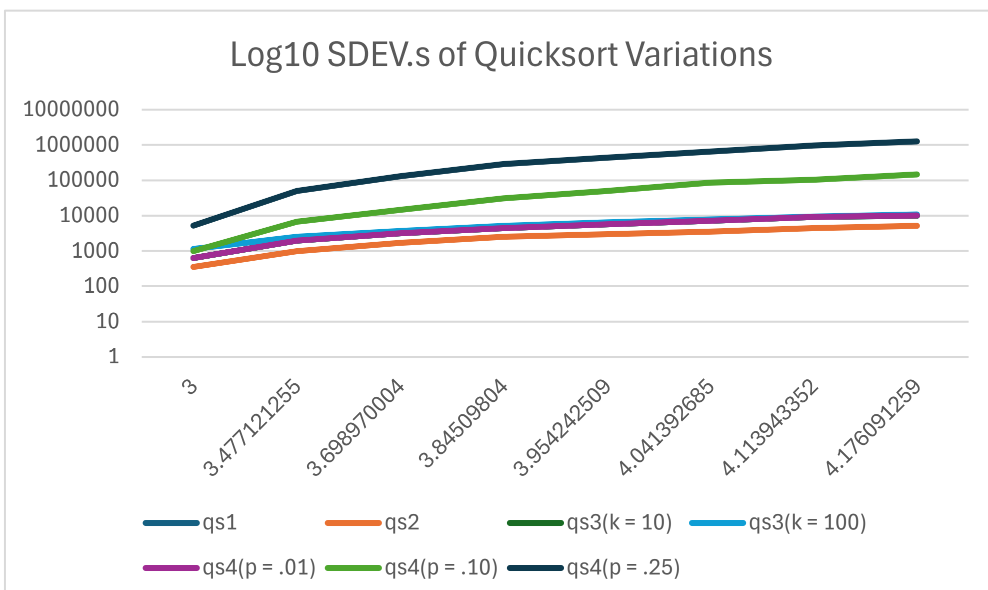
ion followed by QuickSort4
:ifically will vary widely based

, the pivot selection of the 3
nd choosing the median value
ould be chosen and already
call the function choosing a
tomatically, my fastest was
ve see when we set the
Quicksort 1 was the slowest due


15000


Log10 AVG # of Comparions for QS Variations
2000000
1500000
1000000
500000
0
3
3.477121255
3.698970004
3.84509804
3.954242509
4.041392685
4.113943352
4.176091259
qs1
qs2
qs3(k = 10)
qs3(k = 100)
qs4(p = .01)
qs4(p = .10)
qs4(p = .25)


15000


Log10 SDEV.s of Quicksort Variations
10000000
1000000
100000
10000
1000
100
10
1
3
3.477121255
3.698970004
3.84509804
3.954242509
4.041392685
4.113943352
4.176091259
qs1
qs2
qs3(k = 10)
qs3(k = 100)
qs4(p = .01)
qs4(p = .10)
qs4(p = .25)