Hiring Problem:

Swaps (20): 2, 3, 5, 2, 4, 3, 3, 1, 4, 2

Avg (20): 2.9

Swaps (100): 1, 3, 4, 4, 5, 7, 7, 5, 3, 4

Avg (100): 4.3

Swaps (1,000): 5, 4, 7, 4, 7, 5, 11, 8, 6, 8

Avg (1,000): 6.5

Swaps (10,000): 8, 8, 5, 4, 8, 9, 8, 6, 12, 17

Avg (10,000): 8.5

Avg Sort Times

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Array Size** | **Randomized Quick Sort** | **Merge Sort** | **Heap Sort** | **Quick Sort** |
| 50,000 | 0.00846899 | 0.010362 | 0.0208923 | 0.00774279 |
| 100,000 | 0.0181605 | 0.022035 | 0.0449865 | 0.0173458 |
| 150,000 | 0.0277624 | 0.0342272 | 0.0699574 | 0.0258311 |
| 200,000 | 0.0394352 | 0.0465841 | 0.0966877 | 0.0358472 |
| 250,000 | 0.0631265 | 0.0635907 | 0.126464 | 0.045355 |
| 300,000 | 0.058315 | 0.077203 | 0.150477 | 0.0560717 |
| 350,000 | 0.0711748 | 0.0853329 | 0.178483 | 0.0686083 |
| 400,000 | 0.0780418 | 0.0985795 | 0.207183 | 0.0754197 |
| 450,000 | 0.0927765 | 0.111369 | 0.235433 | 0.0838379 |
| 500,000 | 0.100156 | 0.160639 | 0.264203 | 0.0949112 |
| 550,000 | 0.110922 | 0.138451 | 0.294107 | 0.103838 |
| 600,000 | 0.120839 | 0.152003 | 0.322944 | 0.116709 |
| 650,000 | 0.131842 | 0.165359 | 0.351069 | 0.124044 |
| 700,000 | 0.144259 | 0.179344 | 0.383632 | 0.13362 |
| 750,000 | 0.159353 | 0.192457 | 0.408684 | 0.145526 |
| 800,000 | 0.165237 | 0.206549 | 0.441216 | 0.160509 |
| 850,000 | 0.177061 | 0.219583 | 0.468525 | 0.167623 |
| 900,000 | 0.187755 | 0.233983 | 0.500083 | 0.177655 |
| 950,000 | 0.199155 | 0.246935 | 0.532095 | 0.188513 |
| 1,000,000 | 0.214694 | 0.261493 | 0.564225 | 0.197292 |

1. My code makes all arrays random using shuffle before passing to each array.
2. RQS Best: O(nlogn)

RQS Worst: O(nlogn)

MS Best: O(nlogn)

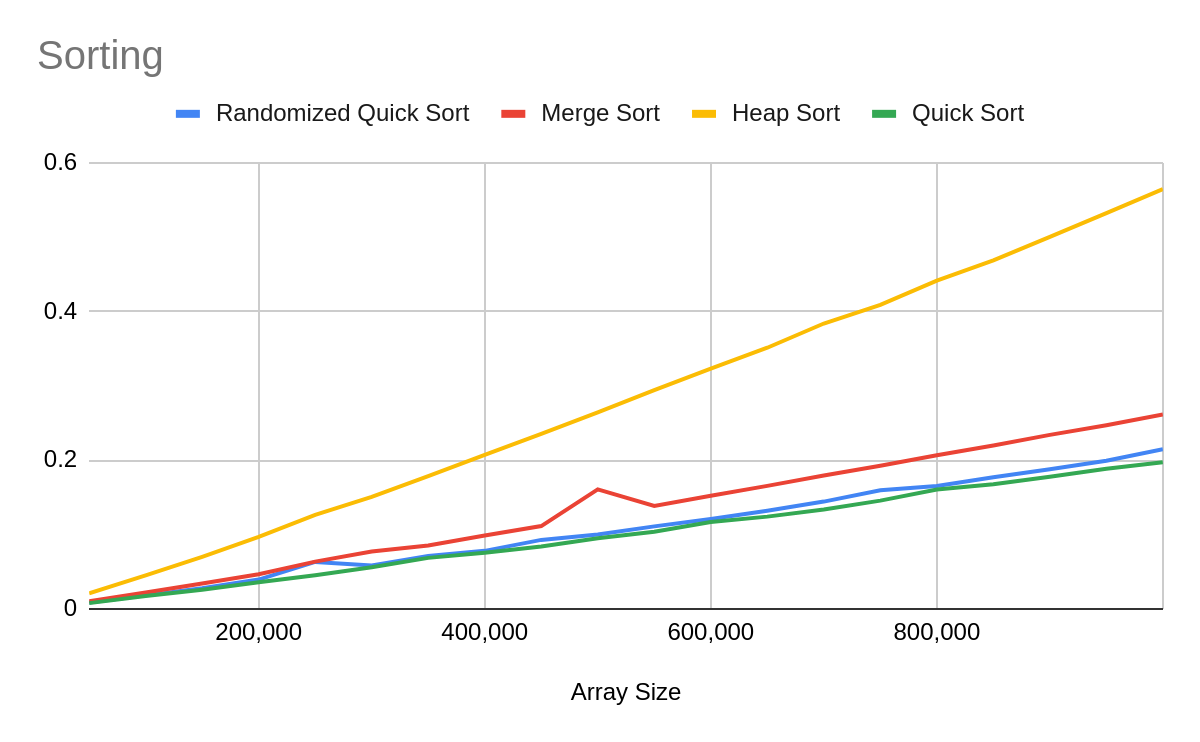
MS Worst: O(nlogn)

HS Best: O(nlogn)

HS Worst: O(nlogn)

QS Best: O(nlogn)

QS Worst: O(n^2)

1. Scales with each 50,000 by a scale of nlogn for most functions
2. Worst case would be 1 second, heap sort makes it about half way, but does not reach this case
3. Make a chart of what number appear in what positions, and see how evenly distributed the results are
4. The code could probably have a way to have shuffled arrays faster, as the array shuffling takes a lot of time to run