

Divide and Conquer Analysis

Cody L. Strange

Utah Valley University

Divide and Conquer Analysis

This paper provides an in-depth analysis of two popular sorting algorithms Merge Sort and Quick Sort by comparing which algorithm is faster and more efficient, we will be using four case studies as evidence. Each of these case studies will be tested using both sorting algorithms, and with three different list sizes of size 210, 215, and 220. Each case study will resemble a real-life scenario and have differing distributions in their lists. We will run each test case ten times per list size and record the time it takes for each sorting algorithm to complete and record the average of the ten run times to determine which sorting algorithm is faster. We will also be recording the total number of basic operations – number of comparisons – required to complete the sorting algorithms and compare those to determine which sorting algorithm is more efficient.

Elo Simulation

Summary

The first case study will be sorting a list of players from an arbitrary competitive video game by their ELO. Where ELO represents their skill in said game.

Distribution

The list will be split into four different categories each with a lower bound and an upper bound, low(1000-2400), mid(2401-2700), high(2701-2900), and pro(2901-3000). Where 50% of the players are in the low category, 30% are in the mid, 15% are in the high, and 5% are in the pro.

Findings

Beginning with input size 2^{10} we found that quick sort beat merge sort by an average of approximately 2.65×10^{-4} seconds. For input size 2^{15} , we found that quick sort once again beat merge sort by an average of approximately 2.34×10^{-2} . For input size 2^{20} , we found that merge sort beat quick sort by an average of approximately 4.4105×10^1 seconds. What this tells us is

that for this level distribution quick sort is negligibly faster at small list sizes while merge sort is significantly faster at larger list sizes.

Data.

Merge Sort(2^{10})

```
elo_sim merge_sort
Run Times:
(1) 0.003656300000000001
(2) 0.0028258000000000033
(3) 0.0027965000000000073
(4) 0.002724199999999996
(5) 0.0028799000000000047
(6) 0.002860599999999991
(7) 0.0026248000000000105
(8) 0.003576800000000005
(9) 0.002815100000000001
(10) 0.0036471999999999893

Op Counts:
(1) 8926
(2) 8954
(3) 8960
(4) 8928
(5) 8938
(6) 8986
(7) 8934
(8) 8943
(9) 8952
(10) 8929

Average Run Time = 0.003040720000000001
Average Number of Basic Operations = 8945.0
```

Merge Sort(2^{15})

```
elo_sim merge_sort
Run Times:
(1) 0.1392779
(2) 0.15124369999999998
(3) 0.13306929999999995
(4) 0.13383499999999993
(5) 0.12562360000000006
(6) 0.14420809999999984
(7) 0.14397460000000017
(8) 0.12546579999999996
(9) 0.14516379999999995
(10) 0.15203389999999994

Op Counts:
(1) 449970
(2) 450175
(3) 450188
(4) 450053
(5) 450141
(6) 449888
(7) 450008
(8) 450042
(9) 449985
(10) 450054

Average Run Time = 0.13938957
Average Number of Basic Operations = 450050.4
```

Merge Sort(2^{20})

```
elo_sim merge_sort
Run Times:
(1) 9.180140300000001
(2) 8.217722099999998
(3) 8.0062602
(4) 7.571647400000003
(5) 8.425538999999993
(6) 9.625389999999996
(7) 9.517097899999996
(8) 8.989274599999987
(9) 9.517901199999997
(10) 9.328268299999999

Op Counts:
(1) 19644875
(2) 19643501
(3) 19642990
(4) 19644073
(5) 19643698
(6) 19643706
(10) 19643662

Average Run Time = 8.837924099999997
Average Number of Basic Operations = 19643894.2
```

Quick Sort(2^{10})

```
elo_sim quick_sort
Run Times:
(1) 0.0024826999999999905
(2) 0.0022183000000000064
(3) 0.0019184000000000007
(4) 0.0020084000000000074
(5) 0.0024765999999999955
(6) 0.0019940000000000235
(7) 0.0020638000000000045
(8) 0.0019374000000000058
(9) 0.0027195999999999887
(10) 0.004129500000000008

Op Counts:
(1) 11603
(2) 11244
(3) 11557
(4) 11313
(5) 10844
(6) 11028
(7) 11501
(8) 11170
(9) 10686
(10) 11027

Average Run Time = 0.002394870000000003
Average Number of Basic Operations = 11197.3
```

Quick Sort(2^{15})

```
elo_sim quick_sort
Run Times:
  (1) 0.14341619999999988
  (2) 0.13856559999999973
  (3) 0.2270563000000001
  (4) 0.16936630000000008
  (5) 0.13508040000000001
  (6) 0.11964329999999999
  (7) 0.18471700000000002
  (8) 0.17454900000000003
  (9) 0.15797529999999993
  (10) 0.17743139999999968
Op Counts:
  (1) 790156
  (2) 764163
  (3) 777694
  (4) 805767
  (5) 819506
  (6) 770671
  (7) 795941
  (8) 773161
  (9) 791755
  (10) 775734

Average Run Time = 0.16278007999999997
Average Number of Basic Operations = 786454.8
```

Quick Sort(2^{20})

```
elo_sim quick_sort
Run Times:
  (1) 61.848223499999999
  (2) 54.217395799999999
  (3) 45.314307499999984
  (4) 47.574325000000044
  (5) 57.647684600000005
  (6) 62.085132100000001
  (7) 58.953647600000001
  (8) 57.152927200000002
  (9) 48.967408100000006
  (10) 35.669481700000006
Op Counts:
  (1) 352800326
  (2) 354758549
  (3) 353762489
  (4) 353021359
  (5) 353408040
  (6) 353490317
  (7) 354744888
  (8) 352883889
  (9) 353879126
  (10) 353885307

Average Run Time = 52.943053310000002
Average Number of Basic Operations = 353663429.0
```

Manufacturer Simulation

Summary

The second case study will be sorting a list of manufacturer employees from some arbitrary company by their pay. The employee's pay can vary based on overtime, bonuses, and position.

Distribution

The list will be split into two different categories each with a lower bound and an upper bound, basic worker(\$30,000-\$40,000), manager(\$60,000-\$80,000). Where 90% of the employees are basic workers and 10% are managers.

Findings

Beginning with input size 2^{10} we found that merge sort beat quick sort by an average of approximately $3.02 \cdot 10^{-3}$ seconds. For input size 2^{15} , we found that merge sort beat quick sort by an average of approximately $4.028 \cdot 10^{-2}$. For input size 2^{20} , we found that quick sort beat merge sort by an average of approximately $2.44844 \cdot 10^0$ seconds. What this tells us is that for this level distribution merge sort is faster at small list sizes while merge sort is faster at larger list sizes; however, the difference is not large enough to be of much significance.

Data.

Merge Sort(2^{10})

```

manufacturer_sim merge_sort
Run Times:
(1) 0.006351499999999982
(2) 0.007322999999999996
(3) 0.004806199999999983
(4) 0.004512899999999986
(5) 0.006453399999999998
(6) 0.007850099999999999
(7) 0.008184500000000011
(8) 0.007803299999999985
(9) 0.006286999999999987
(10) 0.004212199999999999
Op Counts:
(1) 8959
(2) 8933
(3) 8924
(4) 8941
(5) 8956
(6) 8955
(7) 8946
(8) 8940
(9) 8911
(10) 8950

Average Run Time = 0.006378409999999992
Average Number of Basic Operations = 8941.5

```

Merge Sort(2^{15})

```

manufacturer_sim merge_sort
Run Times:
(1) 0.26498810000000006
(2) 0.15939410000000001
(3) 0.136003600000000045
(4) 0.15756939999999986
(5) 0.144630000000000026
(6) 0.17174129999999987
(7) 0.12153639999999921
(8) 0.15161349999999985
(9) 0.132379600000000015
(10) 0.15008679999999952
Op Counts:
(1) 449902
(2) 450093
(3) 450136
(4) 450206
(5) 450057
(6) 450213
(7) 450079
(8) 450102
(9) 450135
(10) 449920

Average Run Time = 0.15899427999999993
Average Number of Basic Operations = 450084.3

```

Merge Sort(2^{20})

```
manufacturer_sim merge_sort
Run Times:
(1) 5.712572899999941
(2) 5.620673399999987
(3) 5.553618400000005
(4) 5.65389580000001
(5) 5.702477700000031
(6) 5.557614599999965
(7) 5.927454399999988
(8) 5.624530900000082
(9) 5.748770199999967
(10) 5.674369800000022

Op Counts:
(1) 19646296
(2) 19645604
(3) 19645471
(4) 19645278
(5) 19645969
(6) 19645334
(7) 19646493
(8) 19645503
(9) 19645676
(10) 19646024

Average Run Time = 5.677597810000009
Average Number of Basic Operations = 19645764.8
```

Quick Sort(2^{10})

```
manufacturer_sim quick_sort
Run Times:
(1) 0.00494870000000028
(2) 0.00378650000000026
(3) 0.00245390000000009
(4) 0.00359309999999996
(5) 0.003381899999999932
(6) 0.003222699999999673
(7) 0.003688299999999777
(8) 0.00289359999999996
(9) 0.002717500000000393
(10) 0.00290509999999994

Op Counts:
(1) 10925
(2) 10541
(3) 10965
(4) 11028
(5) 11043
(6) 11488
(7) 10797
(8) 10489
(9) 10999
(10) 10665

Average Run Time = 0.003359129999999992
Average Number of Basic Operations = 10894.0
```


Quick Sort(2^{15})

```
manufacturer_sim quick_sort
Run Times:
(1) 0.12649679999999996
(2) 0.10177159999999993
(3) 0.134934099999999972
(4) 0.12492300000000078
(5) 0.09989299999999979
(6) 0.12117619999999985
(7) 0.12078450000000007
(8) 0.10606889999999947
(9) 0.13807830000000001
(10) 0.11302030000000052

Op Counts:
(1) 574556
(2) 586300
(3) 582251
(4) 579580
(5) 587728
(6) 563620
(7) 587562
(8) 586922
(9) 634302
(10) 595592

Average Run Time = 0.11871466999999995
Average Number of Basic Operations = 587841.3
```

Quick Sort(2^{20})

```
manufacturer_sim quick_sort
Run Times:
(1) 7.9366035000000466
(2) 8.025359499999922
(3) 8.06883449999998
(4) 8.2314844
(5) 8.012613600000009
(6) 8.135157599999957
(7) 8.360946199999944
(8) 8.132263400000056
(9) 8.145196800000008
(10) 8.211930499999994

Op Counts:
(1) 62929540
(2) 63256278
(3) 63523257
(4) 64613475
(5) 63354753
(6) 63605709
(7) 65122964
(8) 64001244
(9) 64139126
(10) 64038300

Average Run Time = 8.126038999999992
Average Number of Basic Operations = 63858464.6
```

Highschool Simulation

Summary

The third case study will be sorting a list of high school students from some arbitrary school by their age.

Distribution

The list will be split into four different categories where each category is an age group. 15yr old, 16yr old, 17yr old, 18yr old. Each category will contain 25% of the students in the list.

Findings

Beginning with input size 2^{10} we found that merge sort beat quick sort by an average of approximately $3.5581 \cdot 10^{-1}$ seconds. For input size 2^{15} , we found that merge sort beat quick sort by an average of approximately $1.721 \cdot 10^1$. For input size 2^{20} , we were unable to do the comparison as merge sort never completed within a reasonable amount of time. What this tells us is that for this level distribution merge sort is always faster and while this is negligible at very small lists on larger lists the difference is so large that quick sort is not viable as a sorting algorithm.

Data.

Merge Sort(2^{10})

```

highschool_sim merge_sort
Run Times:
(1) 0.00407540000000007
(2) 0.004158999999999968
(3) 0.005509299999999995
(4) 0.004833499999999991
(5) 0.0037499999999999756
(6) 0.0037599999999999856
(7) 0.003347600000000006
(8) 0.003147999999999984
(9) 0.003630200000000028
(10) 0.004082500000000044

Op Counts:
(1) 8218
(2) 8246
(3) 8199
(4) 8184
(5) 8206
(6) 8252
(7) 8203
(8) 8194
(9) 8215
(10) 8221

Average Run Time = 0.004029549999999998
Average Number of Basic Operations = 8213.8

```

Merge Sort(2^{15})

```

highschool_sim merge_sort
Run Times:
(1) 0.127436300000000028
(2) 0.132803799999999958
(3) 0.120246500000000037
(4) 0.111980700000000016
(5) 0.105154100000000003
(6) 0.126529800000000014
(7) 0.12592699999999999
(8) 0.143699400000000092
(9) 0.127298999999999894
(10) 0.110908400000000135

Op Counts:
(1) 406650
(2) 406798
(3) 406742
(4) 406463
(5) 406648
(6) 406592
(7) 406508
(8) 406796
(9) 406688
(10) 406492

Average Run Time = 0.12319850000000007
Average Number of Basic Operations = 406637.7

```

Merge Sort(2^{20})

```

highschool_sim merge_sort
Run Times:
(1) 5.2722754
(2) 4.752536800000001
(3) 4.508673199999999
(4) 4.497025799999999
(5) 4.740561800000002
(6) 4.486735700000005
(7) 4.456824500000003
(8) 4.672120599999995
(9) 4.4807603
(10) 4.509665800000005
Op Counts:
(1) 17596639
(2) 17597376
(3) 17598442
(4) 17598489
(5) 17600404
(6) 17595549
(7) 17597595
(8) 17596813
(9) 17595440
(10) 17598905

Average Run Time = 4.6377179900000005
Average Number of Basic Operations = 17597565.2

```

Quick Sort(2^{10})

```

highschool_sim quick_sort
Run Times:
(1) 0.018984
(2) 0.018345
(3) 0.01930759999999998
(4) 0.037233700000000036
(5) 0.040665600000000024
(6) 0.03845239999999994
(7) 0.04710749999999997
(8) 0.04733050000000005
(9) 0.04307159999999999
(10) 0.04934640000000001
Op Counts:
(1) 132094
(2) 133114
(3) 132605
(4) 132605
(5) 133115
(6) 132094
(7) 133370
(8) 132094
(9) 132095
(10) 132604

Average Run Time = 0.03598443
Average Number of Basic Operations = 132579.0

```

Quick Sort(2^{15})

```

highschool_sim quick_sort
Run Times:
(1) 16.834661500000003
(2) 18.318677400000002
(3) 17.1559545
(4) 16.875579199999997
(5) 18.1583871
(6) 16.879852
(7) 16.897435800000001
(8) 15.8273078
(9) 17.312445999999994
(10) 19.072119500000014
Op Counts:
(1) 134258685
(2) 134266876
(3) 134275069
(4) 134250495
(5) 134266876
(6) 134266876
(7) 134266876
(8) 134266878
(9) 134266878
(10) 134324215

Average Run Time = 17.33324208
Average Number of Basic Operations = 134270972.4

```

Quick Sort(2^{20})

N/A

Workorder Employee Simulation

Summary

The fourth case study will be sorting a list of service work orders by the employee that created the work order.

Distribution

The list will have an even distribution so that 100% of the work orders will be between the numbers 00000000 – 99999999.

Findings

Beginning with input size 2^{10} we found that quick sort beat merge sort by an average of approximately $3.0 \cdot 10^{-4}$ seconds. For input size 2^{15} , we found that quick sort beat merge sort by an average of approximately $5.59 \cdot 10^{-2}$. For input size 2^{20} , we found that quick sort beat quick

sort by an average of approximately $1.9072 * 10^0$ seconds. What this tells us is that for this level distribution quick sort is always faster but not by a significant difference.

Data.

Merge Sort(2^{10})

```
workorder_employee_sim merge_sort
Run Times:
(1) 0.005066200000000021
(2) 0.004209699999999983
(3) 0.003044899999999906
(4) 0.004767500000000036
(5) 0.004004200000000013
(6) 0.0032380999999999105
(7) 0.00410659999999996
(8) 0.0032868999999999815
(9) 0.003293400000000002
(10) 0.0042039999999999855

Op Counts:
(1) 8944
(2) 8923
(3) 8955
(4) 8937
(10) 10790

Average Run Time = 0.0035889899999999807
Average Number of Basic Operations = 11237.5
```

Merge Sort(2^{15})

```
workorder_employee_sim merge_sort
Run Times:
(1) 0.3209228000000053
(2) 0.23218310000001452
(3) 0.1487848000000156
(4) 0.14135459999999966
(5) 0.1649628999999777
(6) 0.17473229999998807
(7) 0.15999629999998888
(8) 0.1494692999999927
(9) 0.1415221000000031
(10) 0.16551659999998947

Op Counts:
(1) 450046
(2) 450013
(3) 449992
(4) 450014
(5) 449970
(6) 450062
(7) 450071
(8) 450180
(9) 449992
(10) 450157

Average Run Time = 0.1799444799999975
Average Number of Basic Operations = 450049.7
```

Merge Sort(2^{20})

```
workorder_employee_sim merge_sort
Run Times:
(1) 8.1242716
(2) 8.5813614
(3) 7.909700600000001
(4) 7.763017500000004
(5) 8.510405400000003
(6) 7.861182899999996
(7) 8.412631400000002
(8) 7.786585700000003
(9) 7.749161899999999
(10) 7.885946599999997

Op Counts:
(1) 19645488
(2) 19644996
(3) 19645560
(4) 19645974
(5) 19646281
(6) 19645940
(7) 19645880
(8) 19645602
(9) 19645443
(10) 19645801

Average Run Time = 8.0584265
Average Number of Basic Operations = 19645696.5
```

Quick Sort(2^{10})

```
workorder_employee_sim quick_sort
Run Times:
(1) 0.0028991999999999907
(2) 0.003993400000000036
(3) 0.003604399999999952
(4) 0.0026770000000000405
(5) 0.005410500000000096
(6) 0.003807899999999975
(7) 0.002684899999999999
(8) 0.0034359000000000472
(9) 0.0022516999999999676
(10) 0.002233900000000011
Op Counts:
(1) 11265
(2) 10688
(3) 12471
(4) 10712
(5) 11006
(6) 10594
(7) 11199
(8) 11921
(9) 10666
(10) 11431

Average Run Time = 0.003299880000000107
Average Number of Basic Operations = 11195.3
```

Quick Sort(2^{15})

```
workorder_employee_sim quick_sort
Run Times:
(1) 0.11772840000000429
(2) 0.12270949999998493
(3) 0.11767609999998285
(4) 0.12356809999999996
(5) 0.12045740000000207
(6) 0.13994809999999802
(7) 0.12241090000000554
(8) 0.11239489999999819
(9) 0.11294720000000078
(10) 0.15051669999999717
Op Counts:
(1) 589543
(2) 611272
(3) 585729
(4) 584682
(5) 584788
(6) 565382
(7) 568164
(8) 576155
(9) 582278
(10) 585640

Average Run Time = 0.12403572999999994
Average Number of Basic Operations = 583363.3
```

Quick Sort(2^{20})


```
workorder_employee_sim quick_sort
Run Times:
(1) 6.344197600000001
(2) 6.423560300000005
(3) 6.092400699999999
(4) 5.800154399999997
(5) 6.0865003000000115
(6) 6.0478934999999865
(7) 6.052337200000011
(8) 6.273100099999935
(9) 6.181628200000006
(10) 6.209786300000019

Op Counts:
(1) 25327265
(2) 26451350
(3) 26928172
(4) 25665742
(5) 25836364
(6) 25981731
(7) 26675453
(8) 26156378
(9) 26068348
(10) 25870005

Average Run Time = 6.151155860000003
Average Number of Basic Operations = 26096080.8
```