CS-331 Project#1 Sort

Description

Following algorithms are implemented in this project:

- Insertion sort
- MergeSort
- QuickSort1 (Regular Quicksort)
- QuickSort2 (QuickSort1 / Insertion sort combo)
- QuickSort3 (Randomised partition value)

Testing environment

Hardware: Intel Core i5@2.6GHz, 8GB RAM
OS: Mac OSX 10.10 w/ single user
Language: PHP 5.6.14

Answers from the data

- **What does your program show about the average execution time of the five algorithms for different sizes of input**

Avg. Time Random List (ms)

ListSize	InsertSort	MergeSort	QuickSort1	QuickSort2	QuickSort3
4	0.00181	0.00830	0.00626	0.00209	0.00635
8	0.00588	0.02352	0.01439	0.00563	0.01445
16	0.02344	0.05806	0.03755	0.02367	0.03890
32	0.08527	0.13657	0.12662	0.12467	0.10235
64	0.23968	0.32084	0.30329	0.30551	0.24166
128	1.17260	0.71295	0.65619	0.65757	0.57725
256	4.81077	1.63321	1.44774	1.45261	1.34601
512	20.10533	3.47755	3.11939	3.13876	2.98513
1024	75.62964	7.85899	6.48726	6.55335	6.74150
2048	312.69912	18.30447	16.24322	16.10810	15.33353
4096	1250.69448	37.85400	46.55866	46.90987	32.76193
8192	5359.77387	84.43879	91.29804	91.07281	73.81394
16384	22397.37010	182.53117	179.06670	184.44883	170.66673
32768	99972.96095	398.34755	408.89991	403.51596	368.39650
65536	424512.98881	898.33799	822.53776	820.14342	789.99446

Avg. Time Ordered List (ms)

	InsertSort	MergeSort	QuickSort1	QuickSort2	QuickSort3
4	0.00140	0.00847	0.00568	0.00161	0.00596
8	0.00254	0.02203	0.01771	0.00276	0.01798
16	0.00487	0.05191	0.05688	0.00500	0.04853
32	0.00921	0.11946	0.26218	0.20166	0.11558
64	0.01835	0.27009	0.74217	0.74171	0.27428
128	0.03707	0.59918	2.91244	2.91020	0.63413
256	0.07479	1.36611	11.40798	11.58394	1.45987
512	0.14988	2.95445	45.83194	45.40913	3.28670
1024	0.30923	6.51405	185.12407	180.92249	7.34466
2048	0.61980	14.04879	753.53394	736.68900	16.20750
4096	1.30353	30.55144	3044.10998	2909.38997	34.45307
8192	2.71286	65.21391	11732.78499	11738.10291	77.31835
16384	5.90182	141.02126	58826.32303	48507.20119	166.81375
32768	12.62369	298.73566	N/A	N/A	359.58693
65536	24.47220	656.37093	N/A	N/A	762.61401

Notes: Due to some unknown reason (I believe it's a bug or limitation of PHP dealing with large memory), PHP failed finishing 2 test cases for QuickSort1 and QuickSort2 (marked as N/A in table)

- **Does Quicksort3 significantly improve the performance of Quicksort1 on sorted array?**

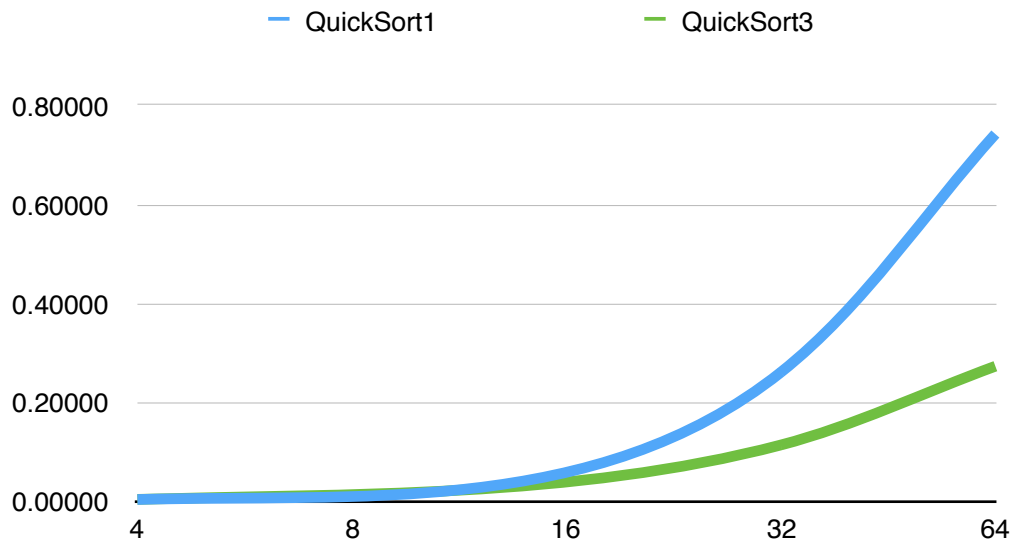
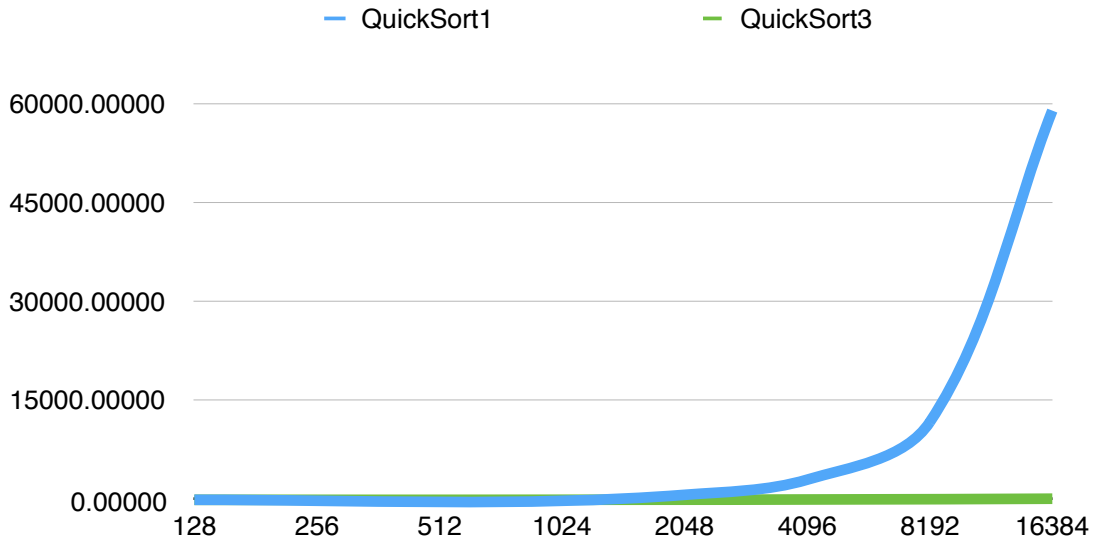


Figure 1



When array size n <= 16, Performance of QuickSort1 and QuickSort2 are same, because they use the same algorithms of Insertion Sort.
When n > 16, the performance of Quicksort3 is much better than that of Quicksort1 on sorted arrays.

- **Is Quicksort2/Quicksort3 alway faster than Quicksort1?**

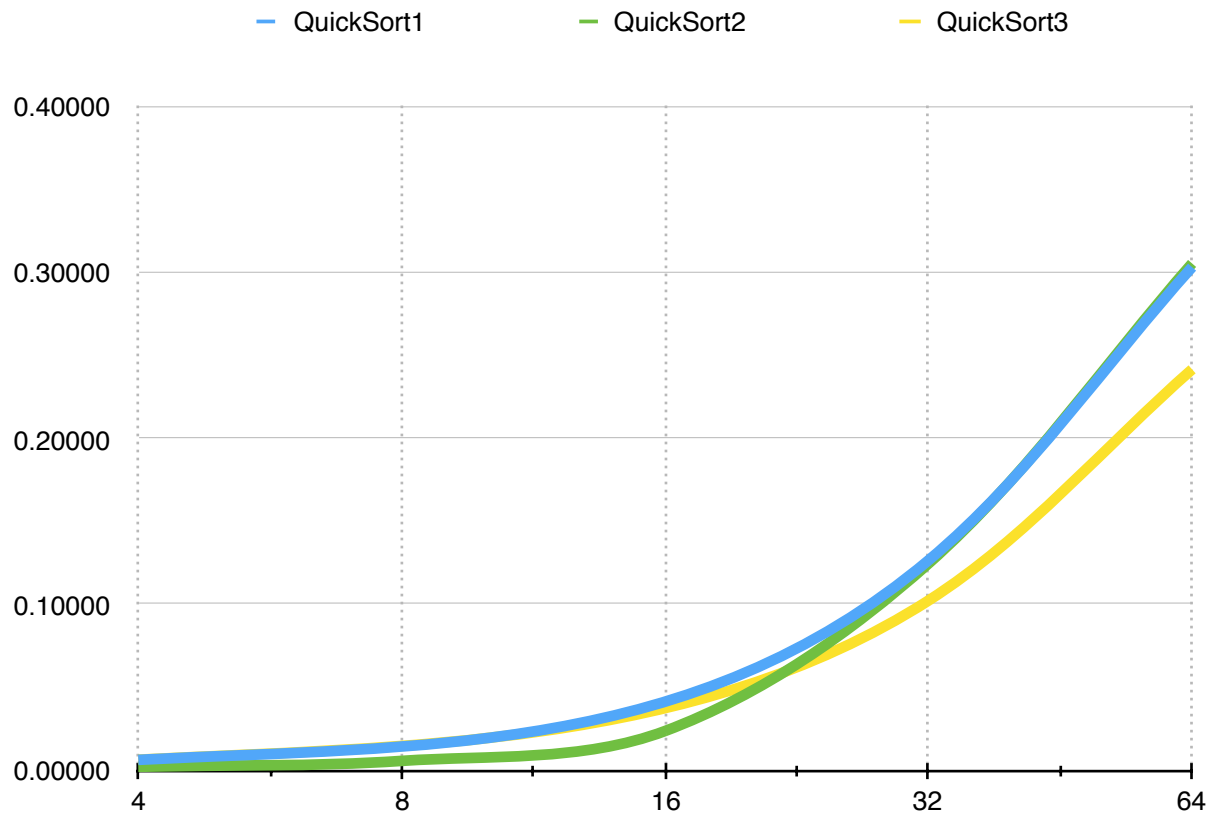


Figure 2

When the size of array n < 16, QuickSort2 is faster than QuickSort1/QuickSort3. Because QuickSort2 leverages the Insertion sort algorithm.

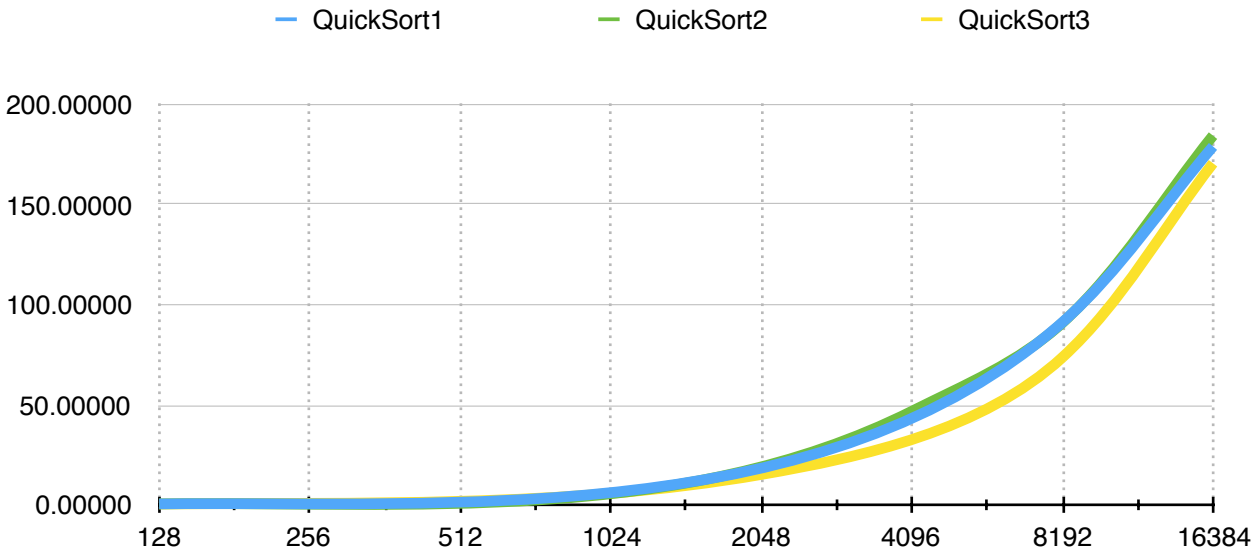


Figure 3

When n > 16, QuickSort3 is slightly faster than QuickSort2 and QuickSort1. Because QuickSort3 takes the advantage of randomised partition value.

- **What is the relation between the average computing times of Quicksort2 and Quicksort3?**

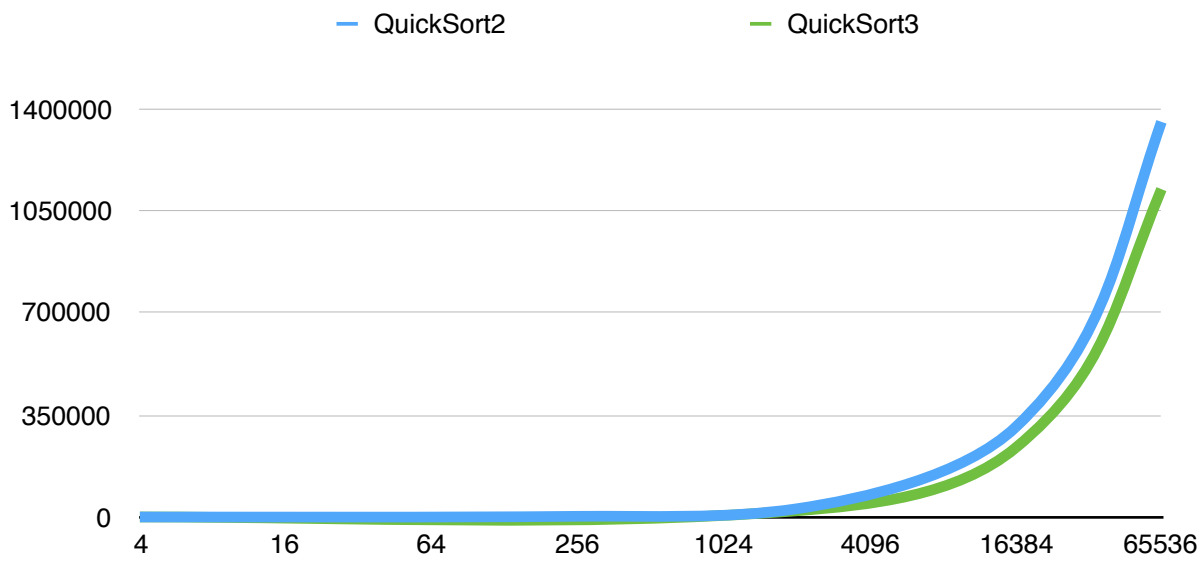


Figure 4

Average computing times of QuickSort3 is LESS than QuickSort2

- **What are the theoretical complexity comparisons of the five algorithms in worst-case and best-case?**
  Worst case:
  Insertion Sort = $O(n^2)$
  Merge Sort = O(nLogn)
  Quick Sort1 = $O(n^2)$
  Quick Sort2 = $O(n^2)$
  Quick Sort3 = $O(n^2)$
  MergeSort < QuickSort3 = QuickSort2 = QuickSort1 = InsertionSort

  Best case:
  Insertion Sort = Ω(n)
  Merge Sort = O(nLogn)
  Quick Sort1 = O(nLogn)
  Quick Sort2 = {Ω(n) n<16, O(nLogn) n>=16}
  Quick Sort3 = {Ω(n) n<16, O(nLogn) n>=16}
  InsertionSort < QuickSort2 = QuickSort3 < QuickSort1 = MergeSort

- **Does the experimental result match with the theoretic analysis of the algorithms?**
  For complexity relations, the experimental result matches with the theoretic analysis.
  On random list:
  QuickSort3 < QuickSort2 = QuickSort1 = MergeSort < InsertSort
  On ordered list:
  InsertSort < MergeSort < QuickSort3 < QuickSort2 = QuickSort1

About the measurements …

- How to verify the correctness of my code?
  - Static check: Manually verify the result by running on variant test cases, which include
    A. odd and even size of elements in a list

    B. ordered list
    C. random list
  - Runtime check: Before measurement, the program will run my sort implementation and the PHP native sort, then compare the results

- Consideration of accuracy
  There are factors can easily interfere with the accuracy of data:
  - Other processes: During the test, keep an eye on the CPU load to make sure there is only 1 CPU core under 100% load (busy running the test)
  - Too fast to measure: Given the high speed of computer, the running time might be reported as zero or a very small number. To overcome this, my strategy is that ensure sufficient loop count to reach a total time of 10 seconds per measurement, then calculate an average time
  - Overhead: Theoretically the complexity of sort algorithms focuses on key comparisons. While in real world, overheads, such as invoking a function, creating a class or copying a list, may take more time than the key comparisons do. So I tried my best to keep overheads affect as little as possible:
    A. Straight-forward functions instead of fancy OOP (In PHP world, this is a big one)
    B. Exclude the time of list copy from the measurement

Conclusion

Pros:
- The program has the “intelligent” to choose a proper loop count to improve accuracy
- Utilities to help reduce workload (batch script, automatically generate test cases)
- The result matches with expectation

Cons:
- Fail running 2 test cases for QuickSort1 and QuickSort2