**ADVANCED SORTS WORK SHEET**

**MERGE SORT**

Create three random lists with a million elements in each. For each list, sort the list with both binary merge sort and natural merge sort. Give the metrics below.

**Binary Merge Sort Natural Merge Sort**

**Comparisons Exchanges Comparisons Exchanges**

LIST 1 \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

LIST 2 \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

LIST 3 \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Does one version of merge sort out-perform the other?

Now begin with a *sorted* list of one million elements. Compare the two merge sort algorithms.

Repeat for a reverse-sorted list.

**Binary Merge Sort Natural Merge Sort**

**Comparisons Exchanges Comparisons Exchanges**

SORTED \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

REVERSED \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Which algorithm has the better performance?

**QUICK SORT**

Run QuickSort against several random lists of different lengths and record the metrics.

SIZE Comparisons Exchanges

\_\_\_\_\_1000\_\_ \_\_23888\_\_\_\_ \_\_\_1742\_\_\_\_

\_\_\_\_\_2000\_\_ \_\_\_ 57927\_\_\_\_\_\_\_\_ \_\_\_\_3873\_\_\_\_\_\_\_

\_\_\_\_ 4000\_\_ \_\_\_\_ 128895\_\_\_\_\_\_\_ \_\_\_\_8594\_\_\_\_\_\_\_

\_\_\_\_\_8000\_\_ \_\_\_ 262155\_\_\_\_\_\_\_\_ \_\_\_\_\_19557\_\_\_\_\_\_

\_\_\_\_16000\_\_ \_\_\_ 601529\_\_\_\_\_\_\_\_ \_\_\_\_\_41703\_\_\_\_\_\_

\_\_\_32000\_\_\_ \_\_\_\_\_ 1224064\_\_\_ \_\_\_92515\_\_\_\_\_\_\_\_

\_\_\_64000\_\_ \_\_\_\_\_ 2816098\_\_ \_\_\_197231\_\_\_\_\_\_\_\_

\_\_\_100000\_\_\_\_\_\_\_\_ \_\_\_ 4466010\_\_ \_\_\_\_324183\_\_\_\_\_\_\_

\_\_\_120000\_\_\_\_\_\_\_\_ \_ 5444315\_\_\_\_\_\_ \_\_\_\_ 394745\_\_\_\_\_\_\_

\_\_\_128000\_\_\_\_\_\_\_\_ \_\_\_\_segmentation fault\_\_\_\_\_\_\_ \_\_\_\_\_0\_\_\_\_\_\_

Does your data suggest that the performance of QuickSort on random lists is *n* lg *n* ?

Support your answer with math!

It has been suggested that using InsertionSort as a “clean up” sort will improve the performance of QuickSort. Test this using three different random lists and different “base cases” in which the unsorted sublist is sent to InsertionSort.

List One List Two List Three

Size Comparisons Exchanges Comparisons Exchanges Comparisons Exchanges

Insert sort sub-list = 0 Insert sort sub-list <= 14 Insert sort sub-list <= 28

2 \_\_\_\_\_8\_\_\_\_ \_\_\_\_12\_\_\_\_\_\_ \_\_\_\_\_0\_\_\_\_\_ \_\_\_\_\_1\_\_\_\_\_ \_\_\_\_\_0\_\_\_\_\_ \_\_\_\_\_1\_\_\_

4 \_\_\_\_\_14\_\_\_ \_\_\_24\_\_\_\_\_\_ \_\_\_\_\_4\_\_\_\_\_ \_\_\_\_\_\_3\_\_\_\_ \_\_\_\_\_4\_\_\_\_\_ \_\_\_\_\_\_3\_\_\_\_

6 \_\_\_\_27\_\_\_\_\_ \_\_\_\_29\_\_\_\_\_ \_\_\_\_\_7\_\_\_\_ \_\_\_\_\_5\_\_\_\_ \_\_\_\_\_7\_\_\_\_ \_\_\_\_\_5\_\_\_\_

8 \_\_\_\_37\_\_\_\_\_ \_\_\_\_66\_\_\_\_ \_\_\_\_10\_\_\_\_\_ \_\_\_\_\_\_7\_\_\_ \_\_\_\_10\_\_\_\_\_ \_\_\_\_\_\_7\_\_\_

10 \_\_\_51\_\_\_\_\_\_ \_\_\_\_89\_\_\_\_\_ \_\_\_\_\_17\_\_\_ \_\_\_\_\_9\_\_\_\_ \_\_\_\_\_17\_\_\_ \_\_\_\_\_9\_\_\_\_

12 \_\_\_70\_\_\_\_\_\_ \_\_\_132\_\_\_\_\_ \_\_\_\_\_34\_\_\_ \_\_\_\_11\_\_\_\_\_ \_\_\_\_\_34\_\_\_ \_\_\_\_11\_\_\_\_\_

14 \_\_\_\_92\_\_\_\_\_ \_\_166\_\_\_\_\_ \_\_\_\_\_56\_\_\_\_ \_\_\_\_\_21\_\_\_\_ \_\_\_\_45\_\_\_\_\_ \_\_\_\_13\_\_\_\_\_

16 \_\_\_95\_\_\_\_\_\_ \_\_\_199\_\_\_\_\_ \_\_\_\_64\_\_\_\_\_ \_\_\_\_\_23\_\_\_\_ \_\_\_\_57\_\_\_\_\_ \_\_\_15\_\_\_\_\_\_

18 \_\_\_\_129\_\_ \_\_\_\_242\_\_\_\_ \_\_\_\_77\_\_\_\_\_ \_\_\_\_\_\_57\_\_\_ \_\_\_\_\_74\_\_\_\_ \_\_\_\_\_17\_\_\_\_

20 \_\_\_112\_\_\_ \_\_\_\_286\_\_\_\_ \_\_\_\_79\_\_\_\_\_ \_\_\_\_\_61\_\_\_\_ \_\_\_\_\_85\_\_\_\_ \_\_\_\_\_19\_\_\_\_

22 \_\_\_\_143\_\_\_\_ \_\_\_340\_\_\_\_\_ \_\_\_\_85\_\_\_\_\_ \_\_\_\_\_\_90\_\_\_ \_\_\_\_97\_\_\_\_\_ \_\_\_\_\_21\_\_\_\_

24 \_\_\_176\_\_\_\_\_ \_\_\_\_436\_\_\_\_ \_\_\_\_132\_\_\_\_ \_\_\_\_140\_\_\_\_ \_\_\_\_116\_\_\_ \_\_\_\_\_23\_\_\_\_

26 \_\_\_212\_\_\_ \_\_\_\_476\_\_\_\_ \_\_\_\_161\_\_\_ \_\_\_\_136\_\_\_ \_\_\_144\_\_\_\_\_ \_\_\_\_25\_\_\_\_\_

28 \_\_\_205\_\_\_\_\_ \_\_\_521\_\_\_\_\_ \_\_164\_\_\_\_\_ \_\_\_\_\_99\_\_\_\_ \_\_\_\_193\_\_\_\_ \_\_\_\_\_35\_\_\_\_

30 \_\_\_236\_\_\_\_\_ \_\_\_603\_\_\_\_\_ \_\_\_136\_\_\_\_\_ \_\_118\_\_\_\_\_\_ \_\_\_\_189\_\_\_ \_\_\_\_\_\_37\_\_\_

32 \_\_\_242\_\_\_\_\_ \_\_\_695\_\_\_ \_\_\_191\_\_\_\_\_ \_\_\_\_205\_\_\_\_ \_\_\_\_\_205\_\_ \_\_\_\_\_\_96\_\_\_\_

34 \_\_248\_\_\_\_\_\_ \_\_\_\_752\_\_\_\_ \_\_186\_\_\_\_\_\_ \_\_255\_\_\_\_\_\_ \_\_\_237\_\_\_\_\_ \_\_\_\_97\_\_\_\_\_

36 \_\_\_310\_\_\_\_\_ \_\_\_832\_\_\_\_\_ \_\_\_\_208\_\_\_ \_\_\_\_\_174\_\_\_ \_\_\_\_276\_\_\_ \_\_\_\_\_103\_\_\_

38 \_\_336\_\_\_\_ \_\_\_941\_\_\_\_\_ \_\_\_\_244\_\_\_\_ \_\_\_209\_\_\_\_\_ \_\_\_\_236\_\_\_\_ \_\_\_84\_\_\_\_\_\_

40 \_\_\_\_\_307\_\_\_ \_\_\_\_\_972\_\_\_ \_\_\_\_\_240\_\_ \_\_\_\_\_180\_\_ \_\_\_\_256\_\_\_\_ \_\_\_\_\_119\_\_\_\_

Does this data suggest that there is an optimal choice for the size of the partition sent to InsertionSort?

Yes, I think that the data suggests that there is a certain point that optimizes what size of sub-lists should be sent to the insertion sort function. It especially makes a difference on large data sets. The insertion sort also help to reduce the number of recursive calls that quick sort has to make.

The SetMedianOfThree algorithm is irrelevant for random lists but can become important for lists with significant sort order. Using sorted lists of various sizes, compare the performance of QuickSort both with and without the SetMedianOfThree algorithm.

With SetMedianOfThree Without SetMedianOfThree

SIZE Comparisons Exchanges Comparisons Exchanges

\_\_5000\_\_\_\_ \_\_\_90067\_\_ \_\_\_\_813462\_\_\_\_ \_\_159376\_\_\_\_\_ \_\_\_11236\_\_\_\_

\_10000\_\_\_ \_\_191920\_ \_\_\_3116326 \_\_\_342072\_\_\_\_\_ \_\_\_24860\_\_\_\_\_

\_15000\_\_\_ \_\_309651\_\_\_ 7030522\_\_\_\_ \_\_\_\_57347\_\_\_\_ \_\_\_39688\_\_\_\_\_

\_20000\_\_\_\_ \_\_\_433522\_ \_12517000\_ \_\_760567\_\_\_\_\_ \_\_\_54274\_\_\_\_\_

\_25000\_\_\_ \_\_523329\_\_\_ \_20324195\_ \_\_941373\_\_\_\_ \_\_\_70232\_\_\_\_\_

\_30000\_\_\_ \_\_666096\_\_\_ \_28689319\_\_ \_\_1161348\_\_\_ \_\_86004\_\_\_\_

\_\_35000\_\_ \_\_783900\_\_ \_38348640\_\_ \_\_\_\_1405497\_\_\_\_ \_\_\_101706\_\_\_\_

\_40000\_\_\_\_ \_\_906287\_\_\_ \_50569433\_ \_\_1648895\_\_\_\_ \_116440\_\_\_\_\_

**RADIX SORT**

List the performance of binary radix sort using several different list sizes.

From this data compute the asymptotic performance of radix sort.

SIZE Comparisons Exchanges

\_\_50000\_\_\_\_\_ \_\_\_49999\_\_\_ \_\_\_300000\_\_\_\_\_

\_\_100000\_\_\_\_ \_\_\_\_99999\_\_ \_\_\_600000\_\_\_

\_\_150000\_\_\_ \_\_\_\_\_149999\_\_\_\_\_\_ \_\_\_\_\_900000\_\_\_\_\_

\_\_200000\_\_\_\_ \_199999\_\_\_\_ \_\_1200000\_\_\_\_

\_\_250000\_\_\_\_ \_\_\_249999\_\_ \_\_\_\_1500000\_\_\_

\_\_300000\_\_\_\_ \_2999999\_\_\_\_\_ \_\_\_\_\_1800000\_\_\_\_\_

\_350000\_\_\_\_ \_349999\_\_\_\_ \_\_\_\_\_2100000\_\_\_\_\_\_

\_\_\_400000\_\_\_ \_399999\_\_\_\_ \_\_\_\_\_2400000\_\_\_\_\_\_

\_500000\_\_\_\_ \_499999\_\_\_\_ \_\_\_\_2700000\_\_\_\_\_

\_550000\_\_\_\_ \_549999\_\_\_\_ \_\_3000000\_\_\_\_

What is the asymptotic performance of binary radix sort? Theta(*n*)