# CS 61B Spring 2022

# Sorting

Discussion 12: April 11, 2022

## 1 Mechanical Practice

Show the steps taken by each sort on the following unordered list:

`106, 351, 214, 873, 615, 172, 333, 564`

(a) Quicksort. After each partition during the algorithm, write the ordering of the list, circle the pivot that was used for that partition, and box the sub-array being partitioned. Assume that the pivot is always the first item in the sublist being sorted and that the array is sorted in place.

**106** 351 214 873 615 172 333 564

106 **351** 214 873 615 172 333 564

106 **214** 172 333 351 **873** 615 564

106 172 214 333 351 **615** 564 873

106 172 214 333 351 564 615 873

Note that depending on how the partition method is implemented, you may have different orderings of elements on either side of your pivot after a partition. The only property that must hold is that all elements less than the selected pivot go to the left, and all elements greater than the selected pivot go to the right.

(b) Merge sort. Show the intermediate merging steps.

106 351 214 873 615 172 333 564

106 351 214 873 172 615 333 564

106 214 351 873 172 333 564 615

106 172 214 333 351 564 615 873

(c) LSD radix sort. Show the ordering of the list after each round of counting sort.

106 351 214 873 615 172 333 564

351 172 873 333 214 564 615 106

106 214 615 333 351 564 172 873

106 172 214 333 351 564 615 873

# 2 Identification

Match the sorting algorithms to the sequences, each of which represents several intermediate steps in the sorting of an array of integers. Assume that for quicksort, the pivot is always the first item in the sublist being sorted. Note: these steps are not necessarily the first few intermediate steps and there may be steps which are skipped.

***Algorithms**: Quicksort, Merge Sort, Heapsort, MSD Radix Sort, Insertion Sort.*

(a) `12, 7, 8, 4, 10, 2, 5, 34, 14`
`7, 8, 4, 10, 2, 5, 12, 34, 14`
`4, 2, 5, 7, 8, 10, 12, 14, 34`

**Quicksort**. using the first element as a pivot.

(b) `23, 45, 12, 4, 65, 34, 20, 43`
`4, 12, 23, 45, 65, 34, 20, 43`

**Insertion Sort**. The first four elements are sorted.

Another solution is merge sort (with a recursive implementation), where the left half has already been fully merge-sorted.

(c) `12, 32, 14, 11, 17, 38, 23, 34`
`12, 14, 11, 17, 23, 32, 38, 34`

**MSD Radix Sort**. We have sorted by the first digit so far.

(d) `45, 23, 5, 65, 34, 3, 76, 25`
`23, 45, 5, 65, 3, 34, 25, 76`
`5, 23, 45, 65, 3, 25, 34, 76`

**Merge Sort**. We have finished 2 levels of merging stages.

(e) `23, 44, 12, 11, 54, 33, 1, 41`
`54, 44, 33, 41, 23, 12, 1, 11`
`44, 41, 33, 11, 23, 12, 1, 54`

**Heapsort**. The second line creates a max-heap.

# 3 Conceptual Sorts

Answer the following questions regarding various sorting algorithms that we've discussed in class. If the question is T/F and the statement is true, provide an explanation. If the statement is false, provide a counterexample.

(a) (T/F) Quicksort has a worst case runtime of $\Theta(NlogN)$, where $N$ is the number of elements in the list that we're sorting.

**False**, quicksort has a worst case runtime of $\Theta(N^2)$, if the array is partitioned very unevenly at each iteration.

(b) We have a system running insertion sort and we find that it's completing faster than expected. What could we conclude about the input to the sorting algorithm?

The input is small or the array is nearly sorted. Note that insertion sort has a best case runtime of $\Theta$(N), which is when the array is already sorted.

(c) Give a 5 integer array that elicits the worst case runtime for insertion sort.

A simple example is: 5 4 3 2 1. Any 5 integer array in descending order would work.

(d) (T/F) Heapsort is stable.

**False**, stability for sorting algorithms mean that if two elements in the list are defined to be equal, then they will retain their relative ordering after the sort is complete. Heap operations may mess up the relative ordering of equal items and thus is not stable. As a concrete example (taken from Stack Overflow),

consider the max heap: 21 20a 20b 12 11 8 7

(e) Give some reasons as to why someone would use mergesort over quicksort.

Some possible answers: mergesort has $\Theta$(NlogN)worst case runtime versus quicksort's $\Theta(N^2)$. Mergesort is stable, whereas quicksort typically isn't. Mergesort can be highly parallelized because as we saw in the first problem the left and right sides don't interact until the end. Mergesort is also preferred for sorting a linked list.

(f) You will be given an answer bank, each item of which may be used multiple times. You may not need to use every answer, and each statement may have more than one answer.

A. QuickSort (in-place using Hoare partitioning and choose the leftmost item as the pivot)

B. MergeSort

C. Selection Sort

D. Insertion Sort

E. HeapSort

N. (None of the above)

List all letters that apply. List them in alphabetical order, or if the answer is none of them, use N indicating none of the above. All answers refer to the entire sorting process, not a single step of the sorting process. For each of the problems below, assume that N indicates the number of elements being sorted.

_____ Bounded by $\Omega$(NlogN)lower bound.

_____ Has a worst case runtime that is asymptotically better than Quicksort's worstcase runtime.

_____ In the worst case, performs $\Theta$(N) pairwise swaps of elements.

_____ Never compares the same two elements twice.

_____ Runs in best case $\Theta$(logN)time for certain inputs

**Solution:**
A, B, C. Bounded by $\Omega$(NlogN)lower bound.

B, E. Has a worst case runtime that is asymptotically better than Quicksort's worstcase runtime.

C. In the worst case, performs $\Theta$(N) pairwise swaps of elements.

A, B, D. Never compares the same two elements twice.

N. Runs in best case $\Theta(\log N)$time for certain inputs