CS 61B Fall 2025

Sorting II

Exam-Level Discussion 12: November 17, 2025

1 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.

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

0.2 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?

0.3 Give a 5 integer array that elicits the worst case runtime for insertion sort.

0.4 (T/F) Heapsort is stable.

0.5 Compare mergesort and quicksort in terms of (1) runtime, (2) stability, and (3) memory efficiency for sorting linked lists.

0.6 Describe how you might use a particular sorting algorithm to find the median of a list of N elements in worst case $\Theta(N \log N)$, without fully sorting the list.

0.7 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) Merge Sort
(C) Selection Sort
(D) Insertion Sort
(E) Heapsort
(F) None of the above

For each of the statements below, list all letters that apply. Each option may be used multiple times or not at all. Note that all answers refer to the entire sorting process, not a single step of the sorting process, and assume that N indicates the number of elements being sorted.

_____ bounded by $\Omega(N \log N)$ lower bound.

_____ Worst case runtime that is asymptotically better than quicksort's worst case runtime.

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

_____ Never compares the same two elements twice.

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

2 Sorted Runtimes

We want to sort an array of N **unique** numbers in ascending order. Determine the best case and worst case runtimes of the following sorts:

(a) Once the runs in merge sort are of size $<= \frac{N}{100}$, we perform insertion sort on them.

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

(b) We use a linear time median finding algorithm to select the pivot in quicksort.

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

(c) We implement heapsort with a min-heap instead of a max-heap. You may modify heapsort but must maintain constant space complexity.

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

(d) We use any algorithm to sort the array knowing that:

- There are at most N inversions.

  Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

- There is exactly 1 inversion.

  Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

- There are exactly $\frac{N(N-1)}{2}$ inversions.

  Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

3 Bears and Beds

In this problem, we will see how we can sort "pairs" of things without sorting out each individual entry. The hot new Cal startup AirBearsnBeds has hired you to create an algorithm to help them place their bear customers in the best possible beds to improve their experience. Now, a little known fact about bears is that they are very, very picky about their bed sizes: they do not like their beds too big or too little - they like them just right. Bears are also sensitive creatures who don't like being compared to other bears, but they are perfectly fine with trying out beds.

The Problem:

- **Inputs:**
  - A list of `Bears` with unique but unknown sizes
  - A list of `Beds` with unique but unknown sizes
  - *Note: these two lists are not necessarily in the same order*
- **Output:** a list of `Bears` and a list of `Beds` such that the `i`th `Bear` is the same size as the `i`th `Bed`
- **Constraints:**
  - `Bears` can only be compared to `Beds` and we can get feedback on if the `Bear` is too large, too small, or just right for it.
  - Your algorithm should run in $O(N \log N)$ time on average