# CS 61BL Sorting

## Summer 2021 Recurring Section 12: Tuesday August 9, 2021

## 1 Mechanical Sorts

Show each pass of the following sorts on the following unordered list of integers (duplicate items are denoted with letters):

`2, 1, 8, 4A, 6, 7, 9, 4B`

1. Insertion Sort

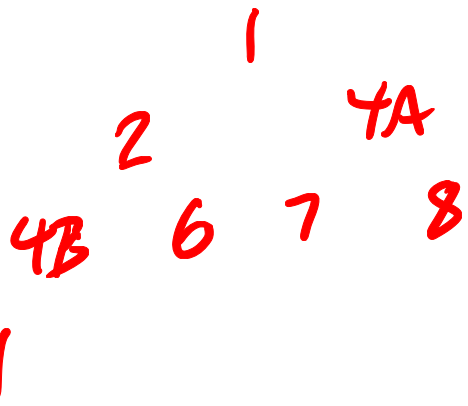
**2 | 1 8 4A 6 7 9 4B**
**1 2 | 8 4A 6 7 9 4B**
**1 2 8 | 4A 6 7 9 4B**
**1 2 4A 8 | 6 7 9 4B**
**1 2 4A 6 8 | 7 9 4B**
**1 2 4A 6 7 8 | 9 4B**
**1 2 4A 6 7 8 9 | 4B**
**1 2 4A 4B 6 7 8 9 |**

2. Selection Sort

**1 | 2 8 4A 6 7 9 4B**
**1 2 | 8 4A 6 7 9 4B**
**1 2 4A | 8 6 7 9 4B**
**1 2 4A 4B | 6 7 9 8**
**1 2 4A 4B 6 | 7 9 8**
**1 2 4A 4B 6 7 | 9 8**
**1 2 4A 4B 6 7 8 | 9**
**1 2 4A 4B 6 7 8 9 |**

3. Merge Sort

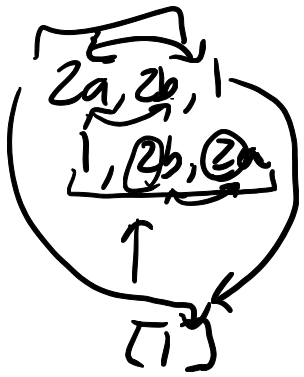
**2 1 8 4A | 6 7 9 4B**
**2 1 | 8 4A | 6 7 | 9 4B**
**2 | 1 | 8 | 4A | 6 | 7 | 9 | 4B**
**1 2 | 4A 8 | 6 7 | 4B 9**
**1 2 4A 8 | 4B 6 7 9**
**1 2 4A 4B 6 7 8 9**




4. Heapsort. Write each swap of the bottom-up heapification process and then the result of each `removeMax` call. *Note that if both children are equal, sink to the left.*

5. Quicksort

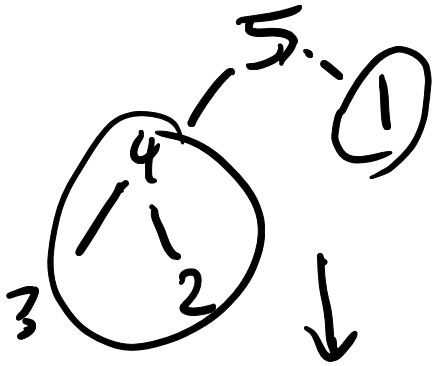
**1 | 2 | 8 4A 6 7 9 4B**
**1 | 2 | 4A 6 7 4B | 8 | 9**
**1 | 2 | 4A | 6 7 4B | 8 | 9**
**1 | 2 | 4A | 4B | 6 | 7 | 8 | 9**

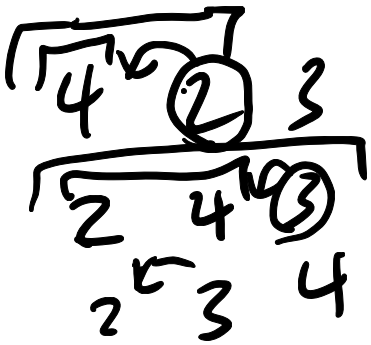


## 2 Sorting Runtimes

Fill out the best-case and worst-case runtimes for these sorts as well as whether they are stable or not in the table below.

| | Best-Case Runtime | Worst-Case Runtime | Stability |
|---|---|---|---|
| Selection Sort | N^2 | N^2 | No |
| Insertion Sort | N | N^2 | Yes |
| Heapsort | N | NlogN | No |
| Mergesort | | | |
| Quicksort | | | |
| Counting Sort | | | |
| LSD Radix Sort | | | |
| MSD Radix Sort | | | |





# 3 You Choose

1. 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?

2. Give a 5 element array such that it elicits the worst case runtime for insertion sort.

3. Give some reasons why someone would use merge sort over quicksort.

4. Which sorts never compare the same two elements twice?

5. When might you decide to use radix sort over a comparison sort, and vice versa?

# 4 Challenge: Bears and Beds

The hot new Cal startup AirBearsnBeds has hired you to create an algorithm to help them place their customers in the best possible homes to improve their experience. They are currently in their alpha stage so their only customers (for now) are bears. 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:**
Given a list of Bears with unique but unknown sizes and a list of Beds with corresponding but also unknown sizes (not necessarily in the same order), return a list of Bears and a list of Beds such that that the $i$th Bear in your returned list of Bears is the same size as the $i$th Bed in your returned list of Beds. Bears can only be compared to Beds and we can get feedback on if the Bed is too large, too small, or just right. In addition, Beds can only be compared to Bears and we can get feedback if the Bear is too large for it, too small for it, or just right for it.

**The Constraints:**
Your algorithm should run in $O(N \log N)$ time on average. It may be helpful to figure out the naive $O(N^2)$ solution first and then work from there.