

Section 08: Sorting out Sorting

1. Relaxation Zone

A timeless classic: <https://www.youtube.com/watch?v=YvTW7341kpA>

2. Mechanical 1: Sorting Algorithm Steps

Here is your input array: 32, 15, 2, 17, 19, 26, 41, 17, 17.

Show the steps taken on this array for each sort as we have learned in lecture. Sort elements so that the result is ordered from smallest to largest. Also remark on whether each sort is **stable** or not.

- (a) Insertion sort:
- (b) Selection sort:
- (c) **In-place** Heapsort (You may want to draw out the heap. Make sure the first step you do is the buildHeap step!):
- (d) Merge sort:
- (e) Quicksort (assume that we always choose the left-most element as the pivot):

3. Mechanical 2: More Sorting Algorithm Steps

Show the steps taken for each sort as we have learned in lecture. Sort elements so that the result is ordered from smallest to largest.

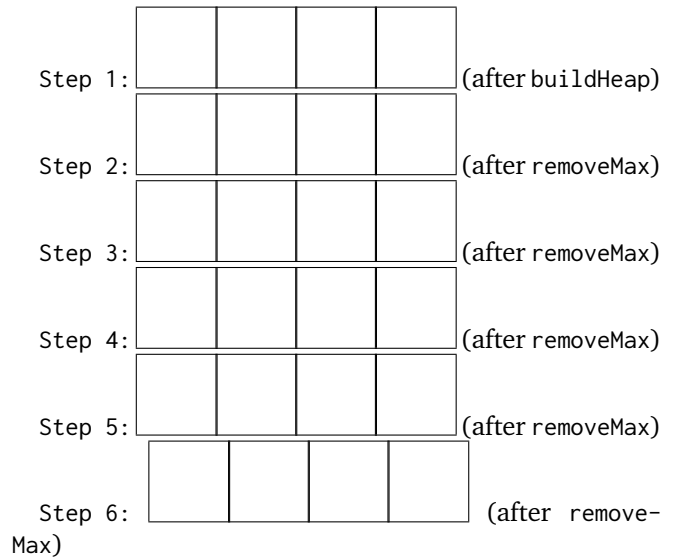
- (a) Insertion sort on 0, 4, 2, 7, 6, 1, 3, 5.
- (b) Selection sort on 0, 4, 2, 7, 6, 1, 3, 5.
- (c) **In-place** Heapsort on 0, 6, 2, 7, 4. (You may want to draw out the heap. Make sure the first step you do is the buildHeap step!)
- (d) Merge sort on 0, 4, 2, 7, 6, 1, 3, 5.
- (e) Quicksort on 18, 7, 22, 34, 99, 18, 11, 4. (Assume that we always choose the first element as the pivot. Show the steps taken at each partitioning step.)

4. Mechanical 3: More Heapsort

List out the steps of sorting the array [5, 0, 1, 3] into ascending order using **in-place** heap sort with a **max heap**.

The first step should be the array after the initial build-Heap, and each successive step should be the array after removeMax. You may not need every line provided to write your solution.

(Tip: Draw out the heap. Make sure your first step is buildHeap!)



5. Runtime 1: Insertion Sort Worst-Case Input

Give the worst possible order of input for insertion sort with the following integers: 1, 2, 3, 4, 5, 6, 7. Assume that the result of sorting should be in ascending order.

--	--	--	--	--	--	--	--

6. Runtime 2: All Worst-Case Inputs

When choosing an appropriate algorithm, there are often several trade-offs that we need to consider. Inspect the chart for the following sorting algorithms. Then, for each sort, provide an example of **both** a best-case and worst-case input.

	Runtime (best)	Runtime (worst)	Stable? (Y/N)	Notes
Selection Sort	$\Theta(N^2)$	$\Theta(N^2)$	No	
Insertion Sort	$\Theta(N)$	$\Theta(N^2)$	Yes	
Heapsort	$\Theta(N)$	$\Theta(N \log N)$	No	
Merge Sort	$\Theta(N \log N)$	$\Theta(N \log N)$	Yes	
Quicksort	$\Theta(N \log N)$	$\Theta(N^2)$	No	

- (a) Suppose we have an array where we expect the majority of elements to be sorted “almost in order”. What would be a good sorting algorithm to use?
- (b) You are writing code to run on the next Mars rover to sort the data gathered each night. (Think about sorting with limited memory and computational power.)
- (c) You’re writing the backend for the website `SortMyNumbers.com`, which sorts numbers given by users.
- (d) Your artist friend says for a piece she wants to make a computer sort every possible ordering of the numbers $1, 2, \dots, 15$. Your friend says something special will happen after the last ordering is sorted, and you’d like to see that ASAP.