Sorting

Discussion 12

Announcements

Week 12

- ❑ Week 12 Survey, due Monday 4/12
- ❑ Lab 13, due Friday 4/16
- ❑ Project 3 Phase 1, due Friday 4/16
- ❑ Pre-Final Form, due Monday 4/19
- ❑ Project 3 Phase 2 due Thursday, 4/27 (no slip days)

Content Review

Insertion Sort

Insertion sort iterates through the list and swaps items backwards as necessary to maintain sortedness.

3 5 1 2 4

Runtime: $O(N^2)$

Selection Sort

Selection sort finds the smallest remaining element in the unsorted portion of the list at each time step.

3 5 1 2 4

Runtime: $\Theta(N^2)$

Merge Sort

Merge sort splits the list in half, applies merge sort to each half, and then merges the two halves together in a zipper fashion.

3 5 1 2 4

Runtime: Θ(NlogN)

Quicksort

Quicksort picks a partition and uses Hoare partitioning to divide the list so that everything greater than the partition is on its right and everything less than the partition is on its left.

3 5 1 2 4

Runtime: Average case O(NlogN), worst case O(N^2)

Heap Sort

Heapsort heapifies the array into a max heap and pops the largest element off and appends it to the end until there are no elements left in the heap. You can heapify by sinking nodes in reverse level order.

3 5 1 2 4

Runtime: Θ(NlogN)

Worksheet

1A Mechanical Sorting Show the steps taken by insertion sort on this list.

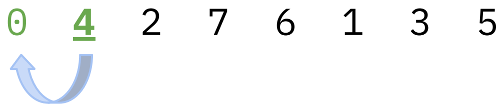
```
0   4   2   7   6   1   3   5
```

Green = sorted portion of array. **Bold and underlined** = element we're currently moving back.

1A Mechanical Sorting Show the steps taken by insertion sort on this list.

<u>**0**</u> 4 2 7 6 1 3 5

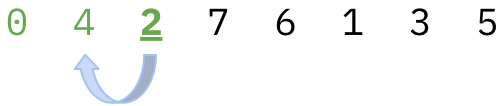
1A Mechanical Sorting Show the steps taken by insertion sort on this list.



```
0   4   2   7   6   1   3   5
```

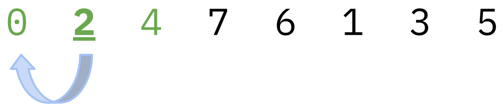
1A Mechanical Sorting

Show the steps taken by insertion sort on this list.



0 4 2 7 6 1 3 5

1A Mechanical Sorting Show the steps taken by insertion sort on this list.



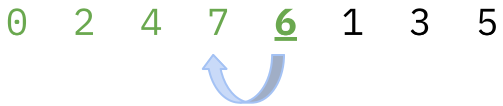
0 2 4 7 6 1 3 5

1A Mechanical Sorting

Show the steps taken by insertion sort on this list.

0 2 4 **7** 6 1 3 5

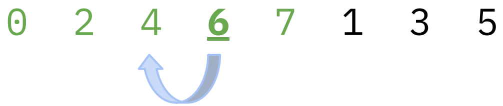
1A Mechanical Sorting Show the steps taken by insertion sort on this list.



0 2 4 7 **6** 1 3 5

1A Mechanical Sorting

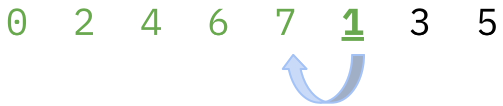
Show the steps taken by insertion sort on this list.



0 2 4 **6** 7 1 3 5

1A Mechanical Sorting

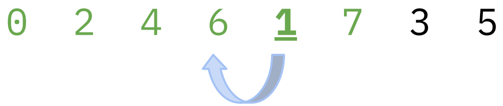
Show the steps taken by insertion sort on this list.



0 2 4 6 7 1 3 5

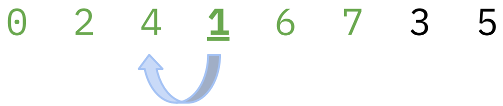
1A Mechanical Sorting

Show the steps taken by insertion sort on this list.



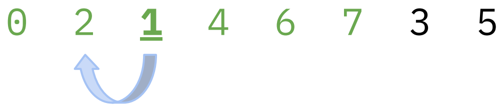
`0 2 4 6 `**`1`**` 7 3 5`

1A Mechanical Sorting Show the steps taken by insertion sort on this list.



0 2 4 **1** 6 7 3 5

1A Mechanical Sorting Show the steps taken by insertion sort on this list.

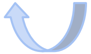


0 2 **1** 4 6 7 3 5

1A Mechanical Sorting

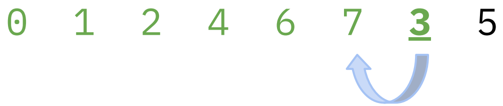
Show the steps taken by insertion sort on this list.

0 **1** 2 4 6 7 3 5



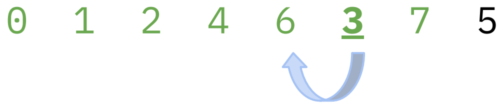
1A Mechanical Sorting

Show the steps taken by insertion sort on this list.



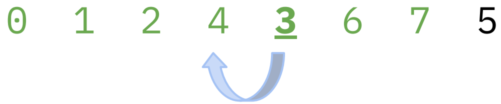
`0 1 2 4 6 7` **<u>`3`</u>** `5`

1A Mechanical Sorting Show the steps taken by insertion sort on this list.



0 1 2 4 6 **3** 7 5

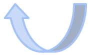
1A Mechanical Sorting Show the steps taken by insertion sort on this list.



0 1 2 4 **3** 6 7 5

1A Mechanical Sorting Show the steps taken by insertion sort on this list.

0 1 2 **3** 4 6 7 5



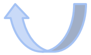
1A Mechanical Sorting

Show the steps taken by insertion sort on this list.

0 1 2 3 4 6 7 **5**

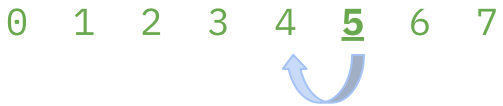
1A Mechanical Sorting Show the steps taken by insertion sort on this list.

0 1 2 3 4 6 **5** 7



1A Mechanical Sorting

Show the steps taken by insertion sort on this list.



0 1 2 3 4 **5** 6 7

1B Mechanical Sorting Show the steps taken by selection sort on this list.

```
0   4   2   7   6   1   3   5
```

Smallest element, after linear pass over all elements in the array: _____

1B Mechanical Sorting Show the steps taken by selection sort on this list.

<u>0</u> 4 2 7 6 1 3 5

Do we need to swap 0?

1B Mechanical Sorting Show the steps taken by selection sort on this list.

0 4 2 7 6 1 3 5

Next smallest element, after linear pass over all
remaining elements in the array: _____

1B Mechanical Sorting Show the steps taken by selection sort on this list.

0 4 2 7 6 <u>1</u> 3 5

Do we need to swap?

1B Mechanical Sorting

Show the steps taken by selection sort on this list.

0 **1** 2 7 6 4 3 5

Next smallest element, after linear pass over all remaining elements in the array: _____

1B Mechanical Sorting Show the steps taken by selection sort on this list.

`0   1   2   7   6   4   3   5`

Do we need to swap?

1B Mechanical Sorting Show the steps taken by selection sort on this list.

0 1 <u>**2**</u> 7 6 4 3 5

Next smallest element, after linear pass over all
remaining elements in the array: _____

1B Mechanical Sorting

Show the steps taken by selection sort on this list.

0 1 2 7 6 4 **3** 5

Do we need to swap?

1B Mechanical Sorting

Show the steps taken by selection sort on this list.

```
0   1   2   3   6   4   7   5
```

Next smallest element, after linear pass over all remaining elements in the array: _____

1B Mechanical Sorting Show the steps taken by selection sort on this list.

0 1 2 3 6 **4** 7 5

Do we need to swap?

1B Mechanical Sorting Show the steps taken by selection sort on this list.

0 1 2 3 **4** 6 7 5

Next smallest element, after linear pass over all remaining elements in the array: _____

1B Mechanical Sorting

Show the steps taken by selection sort on this list.

0 1 2 3 4 6 7 **5**

Do we need to swap?

1B Mechanical Sorting

Show the steps taken by selection sort on this list.

`0   1   2   3   4` **`5`** `  6   7`

Next smallest element, after linear pass over all
remaining elements in the array: _____

1B Mechanical Sorting Show the steps taken by selection sort on this list.

`0 1 2 3 4 5` <u>`6`</u> `7`

Do we need to swap?

1B Mechanical Sorting Show the steps taken by selection sort on this list.

0 1 2 3 4 5 **6** 7

Next smallest element, after linear pass over all remaining elements in the array: _____

1B Mechanical Sorting Show the steps taken by selection sort on this list.

0 1 2 3 4 5 6 **7**

Do we need to swap?

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

```
0   4   2   7   6   1   3   5
```

1C Mechanical Sorting Show the steps taken by mergesort on this list.

```
0 4 2 7 6 1 3 5

[ 0 4 2 7 ]          [ 6 1 3 5 ]

[ 0 4 ]   [ 2 7 ]    [ 6 1 ]   [ 3 5 ]

[ 0 ] [ 4 ] [ 2 ] [ 7 ]   [ 6 ] [ 1 ] [ 3 ] [ 5 ]
```

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 4 2 7] [6 1 3 5]

[ ] [2 7] [6 1] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 4 2 7]   [6 1 3 5]

[0]   [2 7]   [6 1]   [3 5]

[0]   [4]   [2]   [7]   [6]   [1]   [3]   [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 4 2 7]   [6 1 3 5]

[0 **4**]   [2 7]   [6 1]   [3 5]

[0] [4] [2] [7]   [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

```
                    0  4  2  7  6  1  3  5

          [ 0  4  2  7 ]                [ 6  1  3  5 ]

   [ 0  4 ]       [       ]         [ 6  1 ]       [ 3  5 ]

[ 0 ]  [ 4 ]  [ 2 ]  [ 7 ]      [ 6 ]  [ 1 ]  [ 3 ]  [ 5 ]
```

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 4 2 7]   [6 1 3 5]

[0 4]   [2   ]   [6 1]   [3 5]

[0]   [4]   [2]   [7]   [6]   [1]   [3]   [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

```
                    0   4   2   7   6   1   3   5

            [ 0   4   2   7 ]            [ 6   1   3   5 ]

      [ 0   4 ]     [ 2   7 ]          [ 6   1 ]     [ 3   5 ]

  [ 0 ]   [ 4 ]   [ 2 ]   [ 7 ]      [ 6 ]   [ 1 ]   [ 3 ]   [ 5 ]
```

1C Mechanical Sorting Show the steps taken by mergesort on this list.

```
                 0  4  2  7  6  1  3  5

         [               ]          [ 6  1  3  5 ]

    [ 0  4 ]     [ 2  7 ]          [ 6  1 ]     [ 3  5 ]

[ 0 ]  [ 4 ]  [ 2 ]  [ 7 ]     [ 6 ]  [ 1 ]  [ 3 ]  [ 5 ]
```

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0] [6 1 3 5]

[0 **4**] [**2** 7] [6 1] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2]   [6 1 3 5]

[0 **4**]   [2 **7**]   [6 1]   [3 5]

[0]   [4]   [2]   [7]   [6]   [1]   [3]   [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4] [6 1 3 5]

[0 4] [2 **7**] [6 1] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

```
                0  4  2  7  6  1  3  5

          [ 0  2  4  7 ]          [ 6  1  3  5 ]

     [ 0  4 ]     [ 2  7 ]          [ 6  1 ]     [ 3  5 ]

[ 0 ]   [ 4 ]   [ 2 ]   [ 7 ]     [ 6 ]   [ 1 ]   [ 3 ]   [ 5 ]
```

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4 7]   [6 1 3 5]

[0 4]   [2 7]   [ ]   [3 5]

[0]   [4]   [2]   [7]   [**6**]   [**1**]   [3]   [5]

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4 7] [6 1 3 5]

[0 4] [2 7] [1] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

```
                    0  4  2  7  6  1  3  5

         [ 0  2  4  7 ]              [ 6  1  3  5 ]

   [ 0  4 ]       [ 2  7 ]         [ 1  6 ]      [ 3  5 ]

[ 0 ]   [ 4 ]   [ 2 ]   [ 7 ]    [ 6 ]   [ 1 ]   [ 3 ]   [ 5 ]
```

1C Mechanical Sorting Show the steps taken by mergesort on this list.

```
                0  4  2  7  6  1  3  5

        [ 0  2  4  7 ]          [ 6  1  3  5 ]

   [ 0  4 ]    [ 2  7 ]        [ 1  6 ]    [       ]

[ 0 ]  [ 4 ]  [ 2 ]  [ 7 ]   [ 6 ]  [ 1 ]  [ 3 ]  [ 5 ]
```

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4 7] [6 1 3 5]

[0 4] [2 7] [1 6] [3]

[0] [4] [2] [7] [6] [1] [3] [**5**]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4 7] [6 1 3 5]

[0 4] [2 7] [1 6] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4 7] [ ]

[0 4] [2 7] [**1** 6] [**3** 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4 7]   [1]

[0 4] [2 7]   [1 **6**] [**3** 5]

[0] [4] [2] [7]   [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4 7]   [1 3]

[0 4]   [2 7]   [1 **6**]   [3 **5**]

[0]   [4]   [2]   [7]   [6]   [1]   [3]   [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

0 4 2 7 6 1 3 5

[0 2 4 7]   [1 3 5]

[0 4] [2 7]   [1 **6**] [3 5]

[0] [4] [2] [7]   [6] [1] [3] [5]

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

[0 4 2 7 6 1 3 5]

[0 2 4 7] [1 3 5 6]

[0 4] [2 7] [1 6] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

[ ]

[**0** 2 4 7] [**1** 3 5 6]

[0 4] [2 7] [1 6] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

[0]

[0 **2** 4 7]   [**1** 3 5 6]

[0 4]   [2 7]   [1 6]   [3 5]

[0]   [4]   [2]   [7]   [6]   [1]   [3]   [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

```
                    [ 0  1                ]

              [ 0  2  4  7 ]          [ 1  3  5  6 ]

        [ 0  4 ]     [ 2  7 ]        [ 1  6 ]     [ 3  5 ]

   [ 0 ]   [ 4 ]   [ 2 ]   [ 7 ]   [ 6 ]   [ 1 ]   [ 3 ]   [ 5 ]
```

1C Mechanical Sorting Show the steps taken by mergesort on this list.

[0 1 2]

[0 2 **4** 7]   [1 **3** 5 6]

[0 4]   [2 7]   [1 6]   [3 5]

[0]   [4]   [2]   [7]   [6]   [1]   [3]   [5]

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

[0 1 2 3]

[0 2 **4** 7] [1 3 **5** 6]

[0 4] [2 7] [1 6] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

[0 1 2 3 4]

[0 2 4 **7**]   [1 3 **5** 6]

[0 4]   [2 7]   [1 6]   [3 5]

[0]   [4]   [2]   [7]   [6]   [1]   [3]   [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

[0 1 2 3 4 5]

[0 2 4 **7**]   [1 3 5 **6**]

[0 4]   [2 7]   [1 6]   [3 5]

[0] [4] [2] [7]   [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

[0 1 2 3 4 5 6]

[0 2 4 **7**] [1 3 5 6]

[0 4] [2 7] [1 6] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting

Show the steps taken by mergesort on this list.

[0 1 2 3 4 5 6 7]

[0 2 4 7] [1 3 5 6]

[0 4] [2 7] [1 6] [3 5]

[0] [4] [2] [7] [6] [1] [3] [5]

1C Mechanical Sorting Show the steps taken by mergesort on this list.

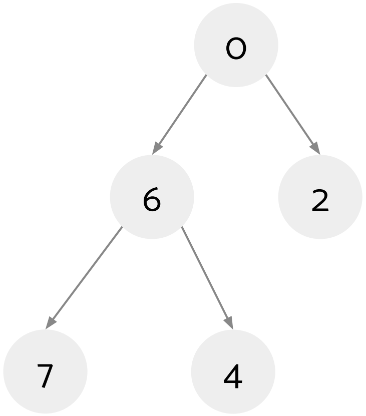
```
[ 0  1  2  3  4  5  6  7 ]

[ 0  2  4  7 ]        [ 1  3  5  6 ]

[ 0  4 ]   [ 2  7 ]        [ 1  6 ]   [ 3  5 ]

[ 0 ]  [ 4 ]  [ 2 ]  [ 7 ]     [ 6 ]  [ 1 ]  [ 3 ]  [ 5 ]
```

1D Mechanical Sorting Show the steps taken by heapsort on this list.

```
0   6   2   7   4
```

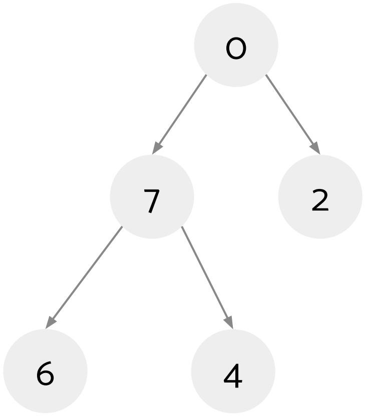
1D Mechanical Sorting Show the steps taken by heapsort on this list.

```
0   6   2   7   4
```



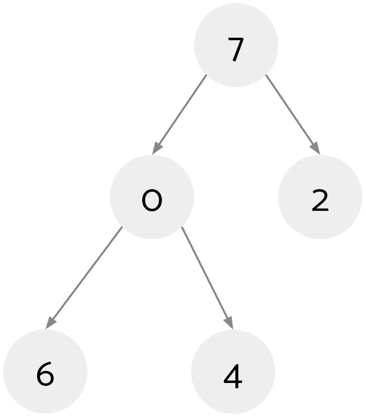
1D Mechanical Sorting Show the steps taken by heapsort on this list.

```
0   7   2   6   4
```



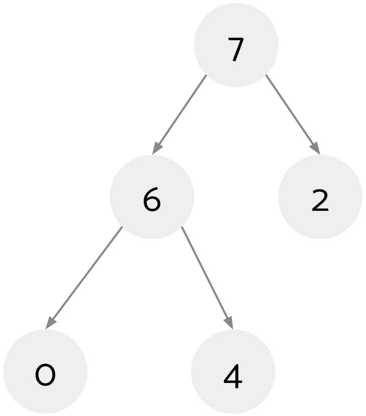
1D Mechanical Sorting Show the steps taken by heapsort on this list.

```
7  0  2  6  4
```



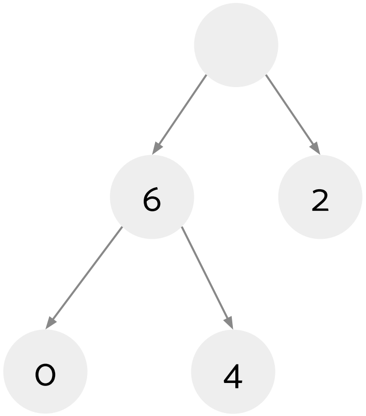
1D Mechanical Sorting Show the steps taken by heapsort on this list.

```
7   6   2   0   4
```



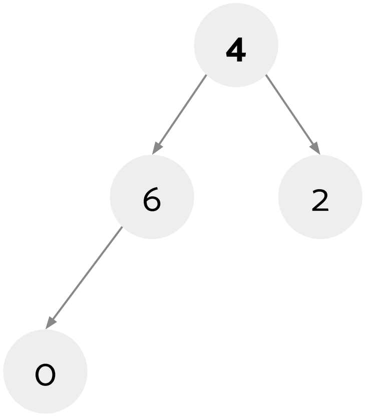
1D Mechanical Sorting Show the steps taken by heapsort on this list.

6 2 0 4 **7**



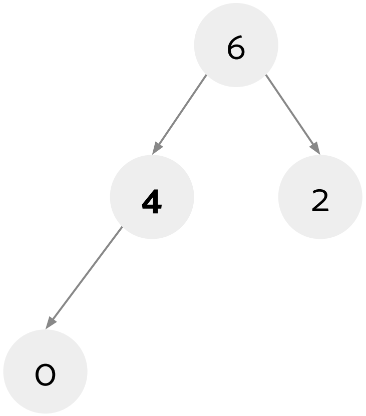
1D Mechanical Sorting Show the steps taken by heapsort on this list.

4 6 2 0 7



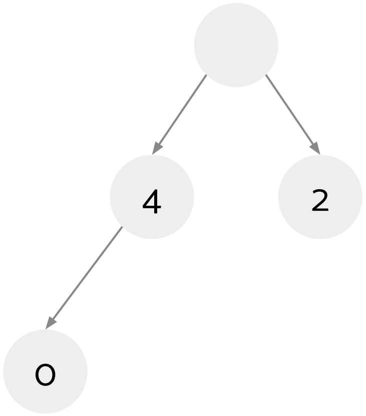
1D Mechanical Sorting Show the steps taken by heapsort on this list.

6 4 2 0 7



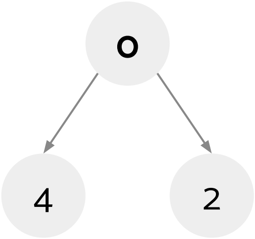
1D Mechanical Sorting Show the steps taken by heapsort on this list.

4 2 0 6 7



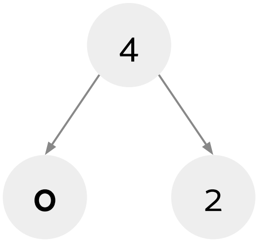
1D Mechanical Sorting Show the steps taken by heapsort on this list.

```
0   4   2   6   7
```



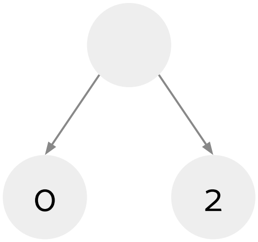
1D Mechanical Sorting Show the steps taken by heapsort on this list.

4 0 2 6 7



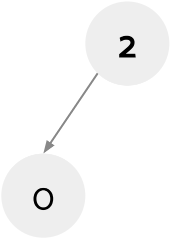
1D Mechanical Sorting Show the steps taken by heapsort on this list.

0 2 **4** 6 7



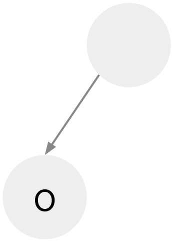
1D Mechanical Sorting Show the steps taken by heapsort on this list.

```
2 0 4 6 7
```



1D Mechanical Sorting Show the steps taken by heapsort on this list.

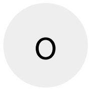
0 **2** 4 6 7



1D Mechanical Sorting

Show the steps taken by heapsort on this list.

0 **2** 4 6 7



1D Mechanical Sorting

Show the steps taken by heapsort on this list.

0 2 4 6 7

1D Mechanical Sorting Show the steps taken by heapsort on this list.

```
0  2  4  6  7
```

2 Sorta Interesting, Right?

1. What does it mean to sort ”in place”, and why would we want this?

2. What does it mean for a sort to be ”stable”? Which sorting algorithms that we have seen are stable?

3. Which algorithm would run the fastest on an already sorted list?

2 Sorta Interesting, Right?

4. Given any list, what is the ideal pivot for quicksort?

5. So far, in class, we've mostly applied our sorts to lists of numbers. In practice, how would we typically make sure our sorts can be applied to other types?

3 Zero One Two-Step

Given an array that only contains 0's, 1's and 2's, write an algorithm to sort it in linear time. You may want to use the provided helper method, `swap`.

```
public static int[] specialSort(int[] arr) {
    int front = 0;
    int back = arr.length - 1;
    int curr = 0;









}
```

```
private static void swap(int[]
arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

3 Zero One Two-Step

We just wrote a linear time sort, how cool! Can you explain in a sentence or two why we can't always use this sort, even though it has better runtime than Mergesort or Quicksort?