- Selection sorts, heap sort
- Merge sorts
- Quicksort

**Readings:** Today: *DS(IJ)*, Chapter 8; Next topic: Chapter 9.



ement.

- Really bad idea on a simple list or vector.
- But we've already seen it in action: use heap.
- Gives $O(N \lg N)$ algorithm ($N$ remove-first operations).
- Since we remove items from end of heap, we can use that area to accumulate result:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *original:* | 19 | 0 | -1 | 7 | 23 | 2 | 42 |
| *heapified:* | 42 | 23 | 19 | 7 | 0 | 2 | -1 |
| | 23 | 7 | 19 | -1 | 0 | 2 | 42 |
| | 19 | 7 | 2 | -1 | 0 | 23 | 42 |
| | 7 | 0 | 2 | -1 | 19 | 23 | 42 |
| | 2 | 0 | -1 | 7 | 19 | 23 | 42 |
| | 0 | -1 | 2 | 7 | 19 | 23 | 42 |
| | -1 | 0 | 2 | 7 | 19 | 23 | 42 |
| | -1 | 0 | 2 | 7 | 19 | 23 | 42 |

Heap part

Sorted part



- When covering heaps before, we created them by insertion in an initially empty heap.
- When given an array of unheaped data to start with, there is a faster procedure (assume heap indexed from 0):

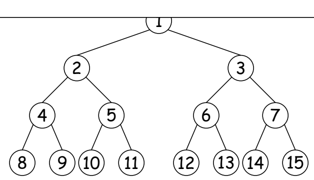
```
void heapify(int[] arr) {
    int N = arr.length;
    for (int k = N / 2; k >= 0; k -=
1) {
        for (int p = k, c = 0; 2*p +
1 < N; p = c) {
            c = 2k+1 or 2k+2, whichever
is < N
                and indexes larger value
in arr;
            swap elements c and k of arr;
        }
    }
```



is removed, repeated $N/2$ times.

- But instead of being $\Theta(N \lg N)$, it's just $\Theta(N)$.






1 node × 3 steps down

2 nodes × 2 steps down

4 nodes × 1 step down

- In general, worst-case cost for a heap with $h+1$ levels is

$$
\begin{aligned}
& 2^0 \cdot h + 2^1 \cdot (h-1) + \ldots + 2^{h-1} \cdot 1 \\
= \; & (2^0 + 2^1 + \ldots + 2^{h-1}) + (2^0 + 2^1 + \ldots + 2^{h-2}) + \ldots + (2^0) \\
= \; & (2^h - 1) + (2^{h-1} - 1) + \ldots + (2^1 - 1) \\
= \; & 2^{h+1} - 1 - h \\
\in \; & \Theta(2^h) = \Theta(N)
\end{aligned}
$$

- Alas, since the rest of heapsort still takes $\Theta(N \lg N)$, this does not improve its asymptotic cost.



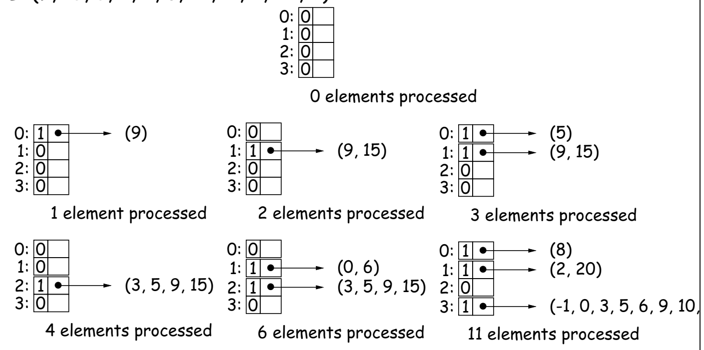
sort halves, merge results.

- Already seen analysis: $\Theta(N \lg N)$.
- Good for *external sorting:*
  - First break data into small enough chunks to fit in memory and sort.
  - Then repeatedly merge into bigger and bigger sequences.
- Can merge $K$ sequences of *arbitrary size* on secondary storage using $\Theta(K)$ storage:

```
Data[] V = new Data[K];
For all i, set V[i] to the first data item of sequence i;
while there is data left to sort:
    Find k so that V[k] is smallest;
    Output V[k], and read new value into V[k] (if present).
```

orchestrate:

L: (9, 15, 5, 3, 0, 6, 10, -1, 2, 20, 8)






**Idea:**

- *Partition* data into pieces: everything $>$ a *pivot* value at the high end of the sequence to be sorted, and everything $\leq$ on the low end.
- Repeat recursively on the high and low pieces.
- For speed, stop when pieces are "small enough" and do insertion sort on the whole thing.
- Reason: insertion sort has low constant factors. By design, no item will move out of its will move out of its piece [why?], so when pieces are small, #inversions is, too.
- Have to choose pivot well. E.g.: *median* of first, last and middle items of sequence.



size $\leq 4$.

- Pivots for next step are starred. Arrange to move pivot to dividing line each time.
- Last step is insertion sort.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 16 | 10 | 13 | 18 | -4 | -7 | 12 | -5 | 19 | 15 | 0 | 22 | 29 | 34 | -1* |
| -4 | -5 | -7 | -1 | 18 | 13 | 12 | 10 | 19 | 15 | 0 | 22 | 29 | 34 | 16* |
| -4 | -5 | -7 | -1 | 15 | 13 | 12* | 10 | 0 | 16 | 19* | 22 | 29 | 34 | 18 |
| -4 | -5 | -7 | -1 | 10 | 0 | 12 | 15 | 13 | 16 | 18 | 19 | 29 | 34 | 2 |

- Now everything is "close to" right, so just do insertion sort:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -7 | -5 | -4 | -1 | 0 | 10 | 12 | 13 | 15 | 16 | 18 | 19 | 22 | 29 | 34 |



  - If choice of pivots good, divide data in two each time: $\Theta(N \lg N)$ with a good constant factor relative to merge or heap sort.
  - If choice of pivots bad, most items on one side each time: $\Theta(N^2)$.
  - $\Omega(N \lg N)$ in best case, so insertion sort better for nearly ordered input sets.
- Interesting point: randomly shuffling the data before sorting makes $\Omega(N^2)$ time *very* unlikely!



smallest element in data.

- Obvious method: sort, select element #$k$, time $\Theta(N \lg N)$.
- If $k \leq$ some constant, can easily do in $\Theta(N)$ time:
  - Go through array, keep smallest $k$ items.
- Get *probably $\Theta(N)$ time* for all $k$ by adapting quicksort:
  - Partition around some pivot, $p$, as in quicksort, arrange that pivot ends up at dividing line.
  - Suppose that in the result, pivot is at index $m$, all elements $\leq$ pivot have indicies $\leq m$.
  - If $m = k$, you're done: $p$ is answer.
  - If $m > k$, recursively select $k^{\text{th}}$ from left half of sequence.

sion of array:

*Initial contents:*

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 51 | 60 | 21 | -4 | 37 | 4 | 49 | 10 | 40* | 59 | 0 | 13 | 2 | 39 | 11 | 46 | 31 |

0

*Looking for #10 to left of pivot 40:*

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 31 | 21 | -4 | 37 | 4* | 11 | 10 | 39 | 2 | 0 | 40 | 59 | 51 | 49 | 46 | 60 |

0

*Looking for #6 to right of pivot 4:*

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -4 | 0 | 2 | 4 | 37 | 13 | 11 | 10 | 39 | 21 | 31* | 40 | 59 | 51 | 49 | 46 | 60 |

4

*Looking for #1 to right of pivot 31:*

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -4 | 0 | 2 | 4 | 21 | 13 | 11 | 10 | 31 | 39 | 37 | 40 | 59 | 51 | 49 | 46 | 60 |

9

*Just two elements; just sort and return #1:*

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -4 | 0 | 2 | 4 | 21 | 13 | 11 | 10 | 31 | 37 | 39 | 40 | 59 | 51 | 49 | 46 | 60 |

9

Result: 39



each time, cost is

$$C(N) = \begin{cases} 1, & \text{if } N = 1, \\ N + C(N/2), & \text{otherwise.} \end{cases}$$
$$= N + N/2 + \ldots + 1$$
$$= 2N - 1 \in \Theta(N)$$

- But in worst case, get $\Theta(N^2)$, as for quicksort.
- By another, non-obvious algorithm, can get $\Theta(N)$ worst-case time for all $k$ (take CS170).