CS61B Lectures #27

rts, heap sort

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

Sorting by Selection: Heapsort

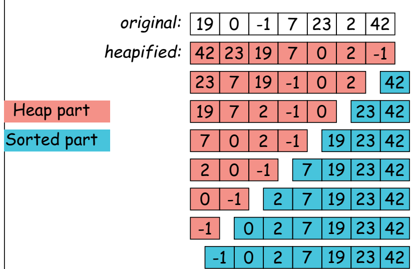
lecting smallest (or largest) element.

ea on a simple list or vector.

eady seen it in action: use heap.

$N)$ algorithm (N remove-first operations).

nove items from end of heap, we can use that area to esult:



ing By Selection: Initial Heapifying

ng heaps before, we created them by insertion in an y heap.

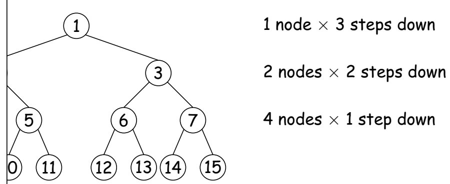
an array of unheaped data to start with, there is a dure (assume heap indexed from 0):

```
ify(int[] arr) {
 = arr.length;
int k = N / 2; k >= 0; k -= 1) {
or (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;
```

e procedure for re-inserting an element after the top he heap is removed, repeated $N/2$ times.

f being $\Theta(N \lg N)$, it's just $\Theta(N)$.

Cost of Creating Heap



orst-case cost for a heap with $h+1$ levels is

$h + 2^1 \cdot (h-1) + \ldots + 2^{h-1} \cdot 1$

$+ 2^1 + \ldots + 2^{h-1}) + (2^0 + 2^1 + \ldots + 2^{h-2}) + \ldots + (2^0)$

$- 1) + (2^{h-1} - 1) + \ldots + (2^1 - 1)$

$- 1 - h$

$^h) = \Theta(N)$

he rest of heapsort still takes $\Theta(N \lg N)$, this does not symptotic cost.

Merge Sorting

lata in 2 equal parts; recursively sort halves; merge re-

analysis: $\Theta(N \lg N)$.

ernal sorting:

k data into small enough chunks to fit in memory and

atedly merge into bigger and bigger sequences.

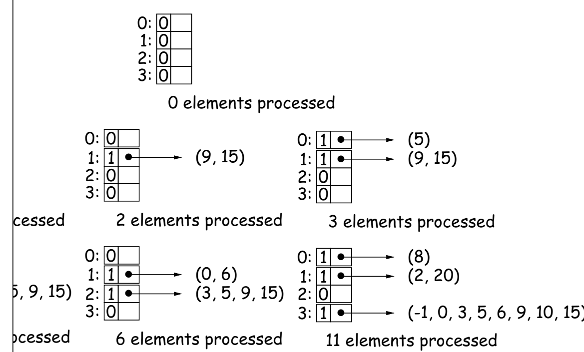
sequences of *arbitrary size* on secondary storage using e:

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


ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate:

0, 6, 10, -1, 2, 20, 8)



icksort: Speed through Probability

- ta into pieces: everything $>$ a *pivot* value at the high equence to be sorted, and everything $\leq$ on the low end.
- rsively on the high and low pieces.
- top when pieces are "small enough" and do insertion sort thing.
- rtion sort has low constant factors. By design, no item of its will move out of its piece [why?], so when pieces nversions is, too.
- ose pivot well. E.g.: *median* of first, last and middle uence.

Example of Quicksort

- ple, we continue until pieces are size ≤ 4.
- xt step are starred. Arrange to move pivot to dividing e.
- insertion sort.

18 | -4 | -7 | 12 | -5 | 19 | 15 | 0 | 22 | 29 | 34 | -1*

-1 ‖ 18 | 13 | 12 | 10 | 19 | 15 | 0 | 22 | 29 | 34 | 16*

-1 ‖ 15 | 13 | 12* | 10 | 0 ‖ 16 ‖ 19* | 22 | 29 | 34 | 18

-1 ‖ 10 | 0 ‖ 12 ‖ 15 | 13 ‖ 16 ‖ 18 ‖ 19 ‖ 29 | 34 | 22

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

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

Performance of Quicksort

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

Quick Selection

- roblem: for given k, find k^{th} smallest element in data.
- hod: sort, select element #k, time $\Theta(N \lg N)$.
- constant, can easily do in $\Theta(N)$ time:
  - h array, keep smallest k items.
- *$\Theta(N)$ time* for all k by adapting quicksort:
  - around some pivot, p, as in quicksort, arrange that pivot dividing line.
  - hat in the result, pivot is at index m, all elements $\leq$ indicies $\leq m$.
  - you're done: p is answer.
  - recursively select k^{th} from left half of sequence.
  - , recursively select $(k - m - 1)^{\text{th}}$ from right half of

Selection Example

- just item #10 in the sorted version of array:

s:

4 | 37 | 4 | 49 | 10 | 40* | 59 | 0 | 13 | 2 | 39 | 11 | 46 | 31

0 to left of pivot 40:

4 | 37 | 4* | 11 | 10 | 39 | 2 | 0 ‖ 40 ‖ 59 | 51 | 49 | 46 | 60

to right of pivot 4:

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

4

to right of pivot 31:

4 ‖ 21 | 13 | 11 | 10 ‖ 31 ‖ 39 | 37 ‖ 40 ‖ 59 | 51 | 49 | 46 | 60

9

nts; just sort and return #1:

4 ‖ 21 | 13 | 11 | 10 ‖ 31 ‖ 37 | 39 ‖ 40 ‖ 59 | 51 | 49 | 46 | 60

9

Selection Performance

- rithm, if m roughly in middle each time, cost is

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

- case, get $\Theta(N^2)$, as for quicksort.
- non-obvious algorithm, can get $\Theta(N)$ worst-case time e CS170).