Public Service Announcement

Monday, 2 November 2015, 7–9PM at Andersen Auditorium:

> "Join us to explore some of the most groundbreaking companies working with artificial intelligence and machine learning, from autonomous cars to virtual reality. Network with professionals from companies like Microsoft, Oculus VR, Uber and more! Come to Smart Everything and get a glimpse of the future, today."

CS61B Lectures #27

Today:

- Shell's sort, Heap, Merge sorts
- Quicksort
- Selection

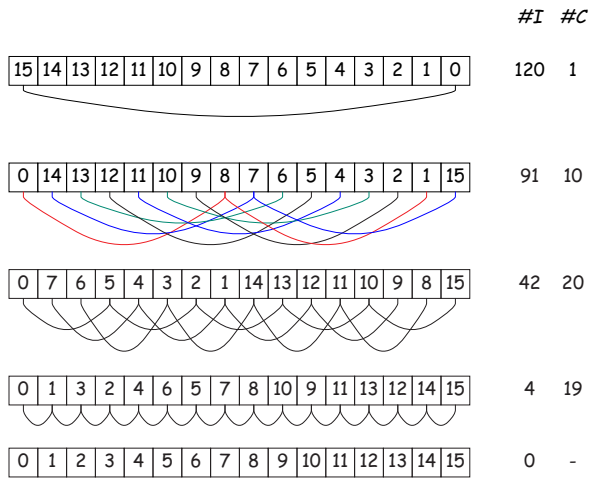
Readings: Today: *DS(IJ)*, Chapter 8; Next topic: Chapter 9.

Shell's sort

Idea: Improve insertion sort by first sorting *distant* elements:

- First sort subsequences of elements $2^k - 1$ apart:
  - sort items #$0,\ 2^k-1,\ 2(2^k-1),\ 3(2^k-1),\ \ldots$, then
  - sort items #$1,\ 1+2^k-1,\ 1+2(2^k-1),\ 1+3(2^k-1),\ \ldots$, then
  - sort items #$2,\ 2+2^k-1,\ 2+2(2^k-1),\ 2+3(2^k-1),\ \ldots$, then
  - *etc.*
  - sort items #$2^k-2,\ 2(2^k-1)-1,\ 3(2^k-1)-1,\ \ldots$,
  - Each time an item moves, can reduce #inversions by as much as 2^k+1.
- Now sort subsequences of elements $2^{k-1} - 1$ apart:
  - sort items #$0,\ 2^{k-1}-1,\ 2(2^{k-1}-1),\ 3(2^{k-1}-1),\ \ldots$, then
  - sort items #$1,\ 1+2^{k-1}-1,\ 1+2(2^{k-1}-1),\ 1+3(2^{k-1}-1),\ \ldots$,
  - ⋮
- End at plain insertion sort ($2^0 = 1$ apart), but with most inversions gone.
- Sort is $\Theta(N^{1.5})$ (take CS170 for why!).

Example of Shell's Sort

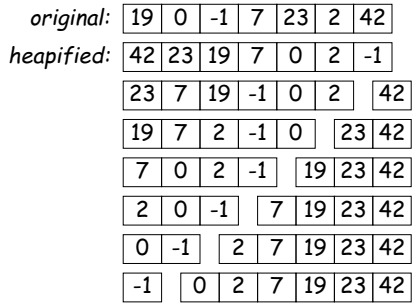


I: Inversions left.
C: Comparisons needed to sort subsequences.

Sorting by Selection: Heapsort

Idea: Keep selecting smallest (or largest) element.

- 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:



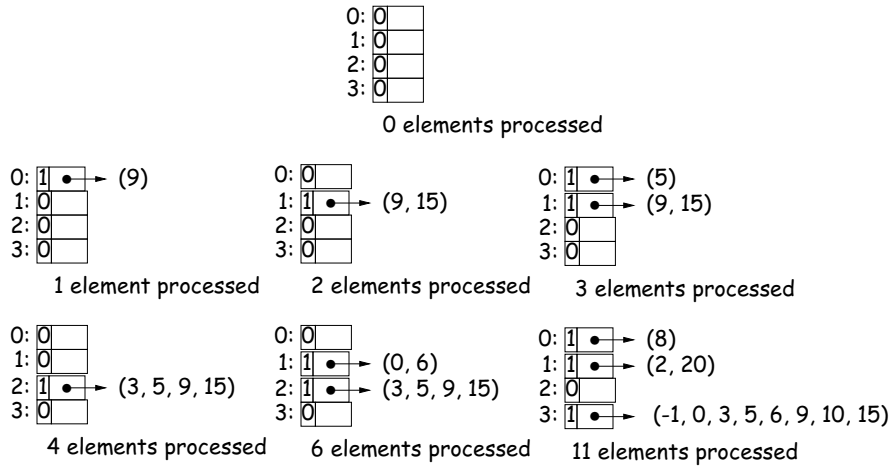
Merge Sorting

Idea: Divide data in 2 equal parts; recursively 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.
- For internal sorting, can use *binomial comb* to orchestrate:

Illustration of Internal Merge Sort

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



Quicksort: Speed through Probability

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.

Example of Quicksort

- In this example, we continue until pieces are size ≤ 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	22

- 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

Performance of Quicksort

- Probabalistic time:
  - 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!

Quick Selection

The Selection Problem: for given k, find k^{th} 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^{th} from left half of sequence.
  - If $m < k$, recursively select $(k - m - 1)^{\text{th}}$ from right half of sequence.

Selection Example

Problem: Find just item #10 in the sorted version 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

Selection Performance

- For this algorithm, if m roughly in middle each time, cost is

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

- 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).