CS61B Lectures #27

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

Merge Sorting

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

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

ernal sorting:

ık 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;
ere is data left to sort:
k so that V[k] has data and is smallest;
/[k] to output sequence;
iere is more data in sequence k, read it into V[k],
otherwise, clear V[k];
```


ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N+1$ buckets that can contain lists, initially empty.

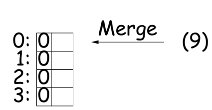
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and
bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that
f size 2^{k+1}, merge it into bucket $k+1$ and clear bucket

its are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N+1$ buckets that can contain lists, initially empty.

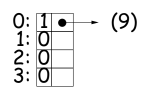
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and
bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that
f size 2^{k+1}, merge it into bucket $k+1$ and clear bucket

its are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N+1$ buckets that can contain lists, initially empty.

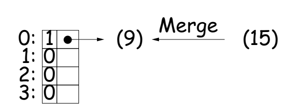
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and
bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that
f size 2^{k+1}, merge it into bucket $k+1$ and clear bucket

its are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N+1$ buckets that can contain lists, initially empty.

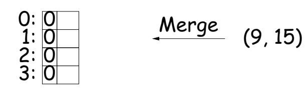
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and
bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that
f size 2^{k+1}, merge it into bucket $k+1$ and clear bucket

its are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N + 1$ buckets that can contain lists, initially empty.

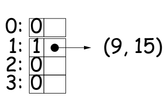
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that f size 2^{k+1}, merge it into bucket $k + 1$ and clear bucket

uts are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N + 1$ buckets that can contain lists, initially empty.

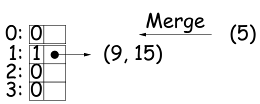
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that f size 2^{k+1}, merge it into bucket $k + 1$ and clear bucket

uts are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N + 1$ buckets that can contain lists, initially empty.

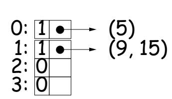
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that f size 2^{k+1}, merge it into bucket $k + 1$ and clear bucket

uts are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N + 1$ buckets that can contain lists, initially empty.

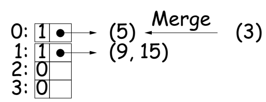
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that f size 2^{k+1}, merge it into bucket $k + 1$ and clear bucket

uts are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N + 1$ buckets that can contain lists, initially empty.

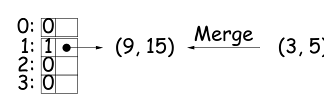
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that f size 2^{k+1}, merge it into bucket $k + 1$ and clear bucket

uts are processed, merge all the buckets into the final

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



ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

$N + 1$ buckets that can contain lists, initially empty.

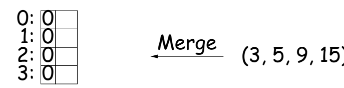
either empty or contains 2^k sorted items at any time.

m in the input list, turn it into a 1-element list, and bucket 0 (or simply put it in bucket 0 if that is empty).

merge lists of length 2^k into bucket k. Whenever that f size 2^{k+1}, merge it into bucket $k + 1$ and clear bucket

uts are processed, merge all the buckets into the final

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

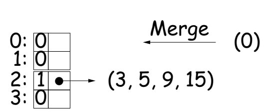


ustration of Internal Merge Sort

ting, can use a *binomial comb* to orchestrate an iterative

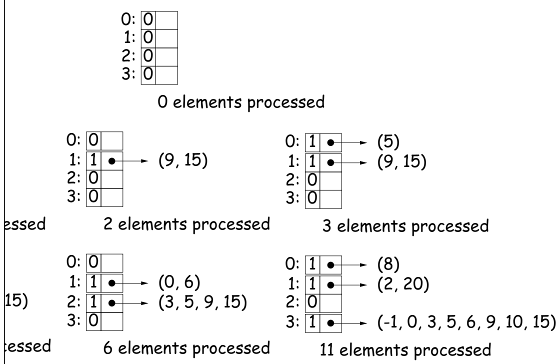
- $N+1$ buckets that can contain lists, initially empty.
- either empty or contains 2^k sorted items at any time.
- m in the input list, turn it into a 1-element list, and bucket 0 (or simply put it in bucket 0 if that is empty).
- merge lists of length 2^k into bucket k. Whenever that f size 2^{k+1}, merge it into bucket $k+1$ and clear bucket
- uts are processed, merge all the buckets into the final

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



tration of Internal Merge Sort (II)

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



all the lists into (-1, 0, 2, 3, 5, 6, 8, 9, 10, 15, 20

icksort: Speed through Probability

- a into pieces: everything $>$ a *pivot* value at the high equence to be sorted, and everything $\leq$ on the low end.
- sively on the high and low pieces.
- op when pieces are "small enough" and do insertion sort thing.
- rtion sort has low constant factors. By design, no item of its piece [why?], so when pieces are small, #inver-
- 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:

-4	-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- t 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.
- that 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 Performance

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

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

case, get $\Theta(N^2)$, as for quicksort.

non-obvious algorithm, can get $\Theta(N)$ worst-case time
e CS170).

Selection Example

just item #10 in the sorted version of array:

:

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							