CSE 326: Data Structures
Sorting It All Out

Henry Kautz

Winter Quarter 2002

1

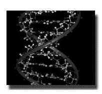
Calendar

- **Today: Finish Sorting**
  - **Read Weiss Ch 7 (skip 7.8)**
- Friday, Feb. 15th: Disjoint Sets & Union Find
  - Read Weiss Ch 8
  - Some written homework problems to be due Wednesday, Feb. 20th
- Monday, Feb. 18th: President's Day, no class
- Wednesday, Feb. 20th: Graph Algorithms
  - Weiss Ch 9 + additional material from lecture notes
  - Several lectures
- Monday, Feb 25th: Word-counting project due
- Various specialized data structures & algorithms
  - Mergeable heaps, quad-trees, Huffman codes, …
- Friday, March 8th: final written homework due
- Friday, March 15th: Last day of class
  - Final programming project – building and solving mazes – due

2

Sorting HUGE Data Sets

- *US Telephone Directory:*
  - 300,000,000 records
    - 64-bytes per record
      - Name: 32 characters
      - Address: 54 characters
      - Telephone number: 10 characters
  - About 2 gigabytes of data
  - Sort this on a machine with 128 MB RAM…
- Other examples?



3

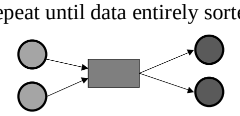
MergeSort Good for Something!

- Basis for most external sorting routines
- Can sort any number of records using a tiny amount of main memory
  - in extreme case, only need to keep 2 records in memory at any one time!

4

External MergeSort

- Split input into two "tapes" (or areas of disk)
- Merge tapes so that each group of 2 records is sorted
- Split again
- Merge tapes so that each group of 4 records is sorted
- Repeat until data entirely sorted



log N passes

5

Better External MergeSort

- Suppose main memory can hold M records.
- Initially read in groups of M records and sort them (*e.g.* with QuickSort).
- Number of passes reduced to log(N/M)

6

Sorting by Comparison: Summary

- Sorting algorithms that only compare adjacent elements are $\Theta(N^2)$ worst case – but may be $\Theta(N)$ best case
- HeapSort and MergeSort - $\Theta(N \log N)$ both best and worst case
- QuickSort $\Theta(N^2)$ worst case but $\Theta(N \log N)$ best and average case
- *Any* comparison-based sorting algorithm is $\Omega(N \log N)$ worst case
- External sorting: MergeSort with $\Theta(\log N/M)$ passes

but not quite the end of the story…

7

BucketSort

- If all keys are 1…K
- Have array of K buckets (linked lists)
- Put keys into correct bucket of array
  – linear time!
- BucketSort is a *stable* sorting algorithm:
  – Items in input with the same key end up in the same order as when they began
- Impractical for large K…

8

RadixSort

- Radix = "The base of a number system" (Webster's dictionary)
  – *alternate terminology: radix is number of bits needed to represent 0 to base-1; can say "base 8" or "radix 3"*
- Used in 1890 U.S. census by Hollerith
- Idea: BucketSort on each digit, bottom up.

9

The Magic of RadixSort

- Input list:
  126, 328, 636, 341, 416, 131, 328
- BucketSort on lower digit:
  341, 131, 126, 636, 416, 328, 328
- BucketSort result on next-higher digit:
  416, 126, 328, 328, 131, 636, 341
- BucketSort that result on highest digit:
  126, 131, 328, 328, 341, 416, 636

10

Inductive Proof that RadixSort Works

- Keys: K-digit numbers, base B
  – (that wasn't hard!)
- Claim: after i^{th} BucketSort, least significant i digits are sorted.
  – Base case: i=0. 0 digits are sorted.
  – Inductive step: Assume for i, prove for i+1.
    Consider two numbers: X, Y. Say X_i is i^{th} digit of X:
    - $X_{i+1} < Y_{i+1}$ then $i+1^{th}$ BucketSort will put them in order
    - $X_{i+1} > Y_{i+1}$, same thing
    - $X_{i+1} = Y_{i+1}$, order depends on last i digits. Induction hypothesis says already sorted for these digits because BucketSort is **stable**

11

Running time of Radixsort

- N items, K digit keys in base B
- How many passes?
- How much work per pass?
- Total time?

12

Running time of Radixsort

- N items, K digit keys in base B
- How many passes? K
- How much work per pass? N + B
  - just in case B>N, need to account for time to empty out buckets between passes
- Total time? O(K(N+B))

13

RadixSorting Strings example

	5th pass	4th pass	3rd pass	2nd pass	1st pass
String 1	z	i	p	p	y
String 2	z	a	p		
String 3	a	n	t	s	
String 4	f	l	a	p	s

NULLs are just like fake characters

14

Evaluating Sorting Algorithms

- What factors other than asymptotic complexity could affect performance?
- Suppose two algorithms perform exactly the same number of instructions. Could one be better than the other?

15

Example Memory Hierarchy Statistics

Name	Extra CPU cycles used to access	Size
L1 (on chip) cache	0	32 KB
L2 cache	8	512 KB
RAM	35	256 MB
Hard Drive	500,000	8 GB

16

The Memory Hierarchy Exploits Locality of Reference

- Idea: *small* amount of *fast* memory
- Keep *frequently* used data in the *fast* memory
- LRU replacement policy
  - Keep recently used data in cache
  - To free space, remove Least Recently Used data

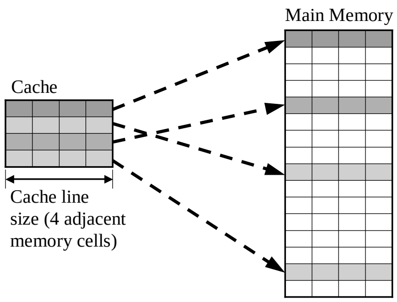
17

So what?

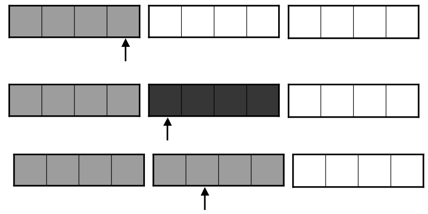
- Optimizing use of cache can make programs way faster
- One TA made RadixSort 2x faster, rewriting to use cache better!
- Not just for sorting

18

Cache Details (simplified)

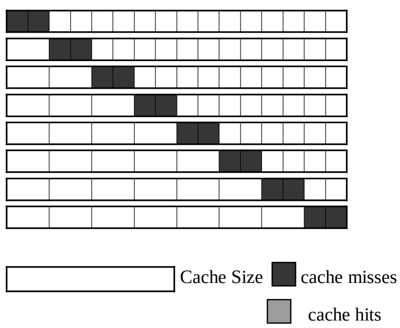


Traversing an Array

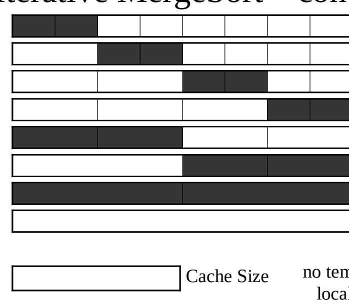


- One miss for every 4 accesses in a traversal

Iterative MergeSort

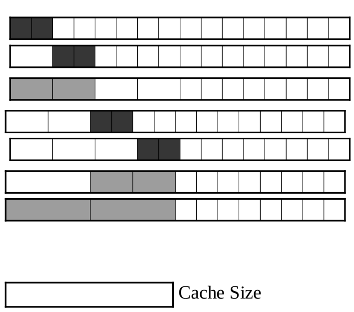


Iterative MergeSort – cont'd

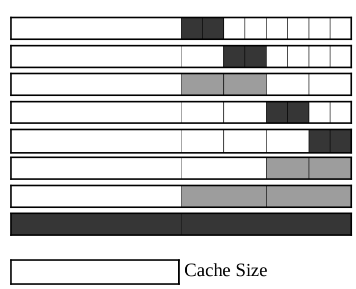


no temporal locality!

"Tiled" MergeSort – better



"Tiled" MergeSort – cont'd



QuickSort

- Initial partition causes a lot of cache misses
- As subproblems become smaller, they fit into cache
- Good cache performance

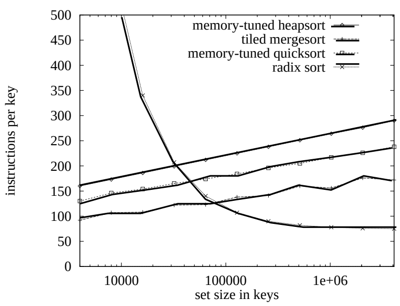
25

Radix Sort – Very Naughty

- On each BucketSort
  - Sweep through input list – cache misses along the way (bad!)
  - Append to output list – indexed by pseudo-random digit (ouch!)

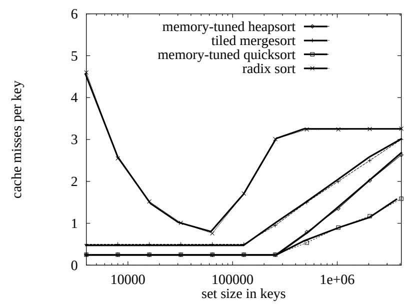
26

Instruction Count



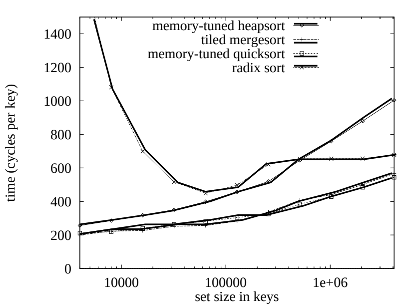
27

Cache Misses



28

Sorting Execution Time



29

Conclusions

- Speed of cache, RAM, and external memory has a huge impact on sorting (and other algorithms as well)
- Algorithms with same asymptotic complexity may be best for different kinds of memory
- Tuning algorithm to improve cache performance can offer large improvements (iterative vs. tiled mergesort)

30