

CSE 589
Applied Algorithms
Spring 1999

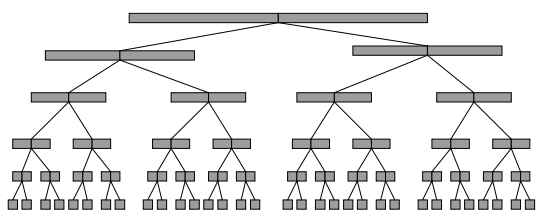
Cache Performance
Mergesort
Heapsort

Recursive Mergesort

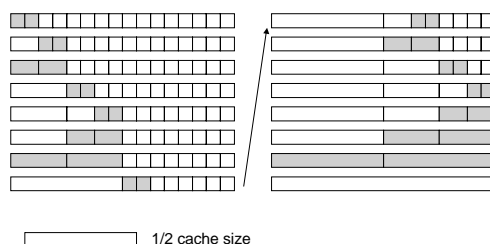
```
A[1..n] is to be sorted;
B[1..n] is an auxiliary array;

Mergesort(i,j) {sorts the subarray A[i..j]}
  if i < j then
    k := (i+j)/2;
    Mergesort(i,k);
    Mergesort(k+1,j);
    Merge A[i..k] with A[k+1..j] into B[i..j];
    Copy B[i..j] into A[i..j];
```

Mergesort Call Tree



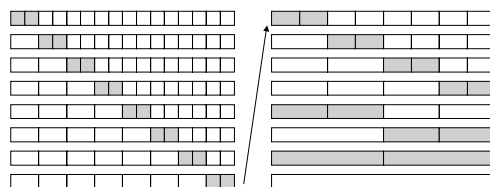
Merging Pattern of Recursive Mergesort



Notes on Recursive Mergesort

- Oblivious recursion.
 - The subarrays that are merged do not depend on the particular keys, just on the number of keys.
- Lots of copying from the auxiliary array to the source arrays.
- Recursion is elegant, but is it really needed?
- Sorting very small arrays should be done in-place.

Reorder the Merging Steps



Iterative Mergesort

- Sort small groups in-place.
- Alternate the roles of A and B as the source of the merging passes.
- Copy B to A if needed at the end.

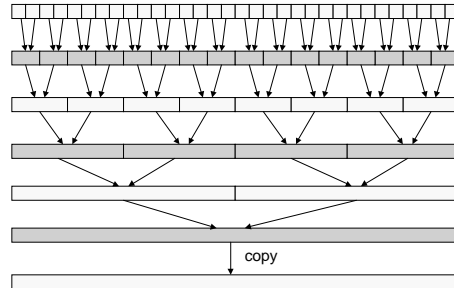
in-place sort groups of 4;
 merge sorted groups of 4 in A into sorted groups of 8 in B;
 merge sorted groups of 8 in B into sorted groups of 16 in A;
 merge sorted groups of 16 in A into sorted groups of 32 in B;

in the end if the sorted array is B then copy it to A;

CSE 589 - Lecture 8 - Spring 1999

7

Iterative Mergesort Access Pattern



CSE 589 - Lecture 8 - Spring 1999

8

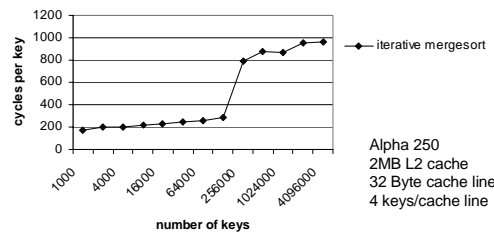
Analysis of Access Pattern

- one pass to sort into groups of 4.
 - Pass touches n key locations.
- $\log_2(n/4)$ merge passes.
 - Each pass touches $2n$ key locations, n in the source array and n in the destination array.
- One copy pass if $\log_2(n/4)$ is odd.
 - Pass touches $2n$ key locations.

CSE 589 - Lecture 8 - Spring 1999

9

Performance of Iterative Mergesort



CSE 589 - Lecture 8 - Spring 1999

10

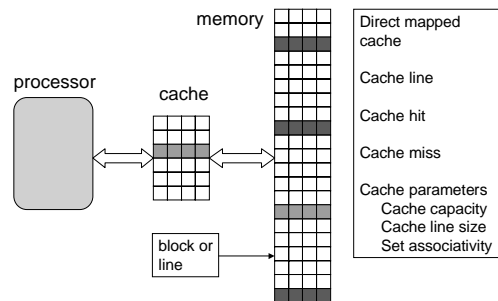
Cache Performance Matters

- Processor speeds increasing faster than memory speeds.
- Cache miss penalties can be 100 cycles and are growing.
- Algorithm design can be used to reduce cache misses and improve overall performance.

CSE 589 - Lecture 8 - Spring 1999

11

Cache Model



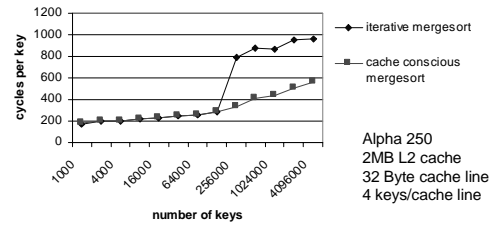
CSE 589 - Lecture 8 - Spring 1999

12

Cache Miss Terminology

- Types of misses
 - Compulsory miss: first time a memory block is read.
 - Capacity miss: accessed data does not fit in cache.
 - Conflict miss: several active memory blocks map to the same place in the cache.
- Locality reduces cache misses
 - temporal locality: a location that was recently accessed is accessed again.
 - Spatial locality: data on the same block are accessed together.

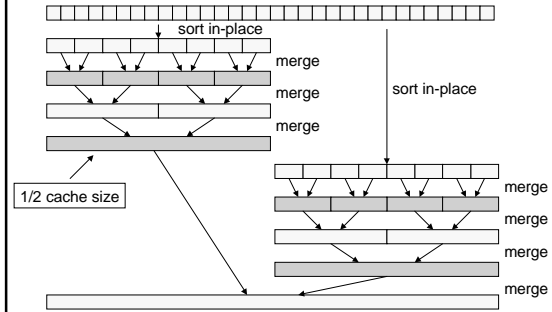
Cache Conscious Mergesort Execution Performance



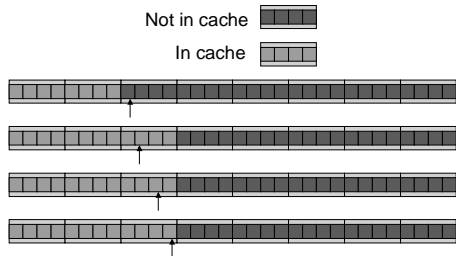
Cache Conscious Mergesort

- Partition problem into "tiles" that fit in the cache.
- Mergesort the tiles.
- Merge the tiles.
- Avoid copying by sorting in-place into groups of 2 or 4 depending on whether $\log_2(n/4)$ is odd or even.

Cache Conscious Mergesort

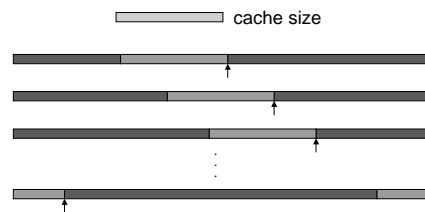


Traversal Analysis



$1/B$ misses per access where B is number of access per line

Traversal Longer than Cache



Analysis of Cache Misses

- Parameters
 - B keys per cache line
 - C cache lines in the cache
 - n keys with $n \gg BC$
- Iterative Mergesort

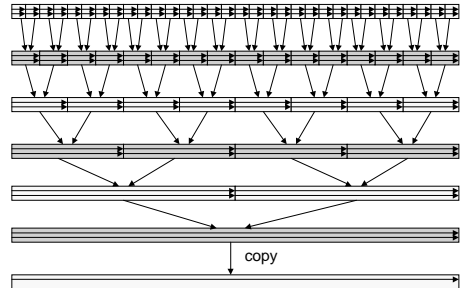
$$\frac{1}{B} + \frac{2}{B} \left\lceil \log_2 \frac{n}{4} \right\rceil + \frac{2}{B} \left(\left\lceil \log_2 \frac{n}{4} \right\rceil \bmod 2 \right) \text{ cache misses per key}$$

↑ in-place sort
 ↑ merge passes
 ↑ copy

CSE 589 - Lecture 8 - Spring 1999

19

Iterative Mergesort Cache Misses



CSE 589 - Lecture 8 - Spring 1999

20

Cache Conscious Merge Sort Analysis

$$\frac{2}{B} + \frac{2}{B} \left\lceil \log_2 \frac{2n}{BC} \right\rceil \text{ cache misses per key}$$

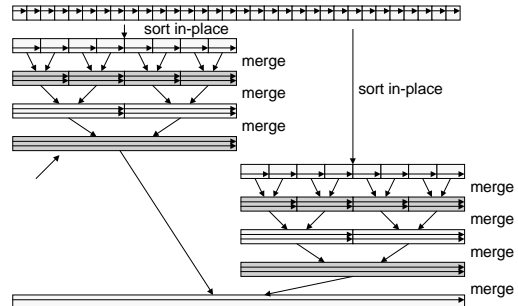
↑ sort each tile
 ↑ final merge passes

Tile size is $BC/2$.
 $n/(BC/2)$ tiles to be merged in the end.
 This take $\log_2(n/(BC/2))$ passes.

CSE 589 - Lecture 8 - Spring 1999

21

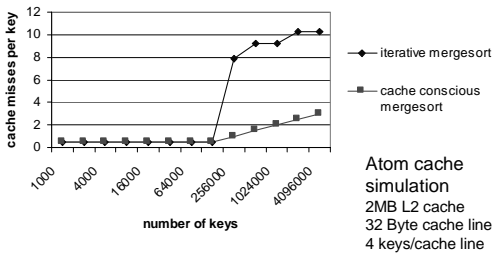
Cache Conscious Misses



CSE 589 - Lecture 8 - Spring 1999

22

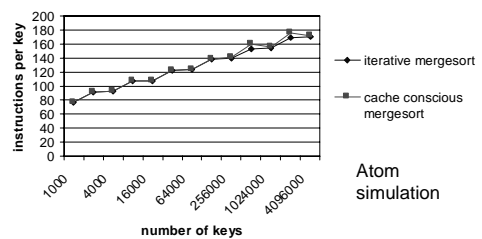
Simulated Cache Performance



CSE 589 - Lecture 8 - Spring 1999

23

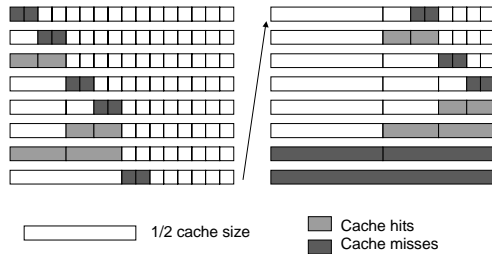
Instruction Counts



CSE 589 - Lecture 8 - Spring 1999

24

What About Recursive Mergesort?



CSE 589 - Lecture 8 - Spring 1999

25

Notes on Cache Performance

- Before trying cache conscious algorithm design you should ask if performance is really a problem.
 - if not, then don't tinker
 - if so, then check out the algorithm and data structures first. Going from an n^2 algorithm to a $n \log n$ algorithm can make a world of difference.
 - if the algorithm and data structures are basically good then consider a cache conscious design.

CSE 589 - Lecture 8 - Spring 1999

26

Some Guiding Principles

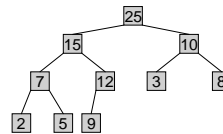
- Sacrifice instructions for better cache performance.
- Knowing architectural constants can lead to better algorithms. Cache capacity, line size.
- Small memory footprints are good.
 - Reduces capacity misses
- Block data into cache size pieces.
 - Reduces capacity misses
- Fully utilize cache lines.
 - Improves spatial locality

CSE 589 - Lecture 8 - Spring 1999

27

Heapsort

- Classic in-place, $O(n \log n)$ sorting algorithm.
- Uses the binary heap, an elegant priority queue data structure (insert and delete-max)



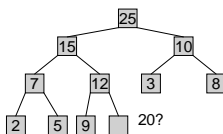
Perfectly balanced tree with the heap property. Each node is larger than its children.

CSE 589 - Lecture 8 - Spring 1999

28

Insert

- Add a new leaf and percolate up.

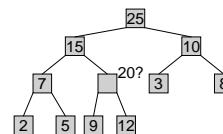


CSE 589 - Lecture 8 - Spring 1999

29

Insert (2)

- Add a new leaf and percolate up.

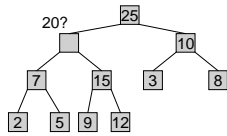


CSE 589 - Lecture 8 - Spring 1999

30

Insert (3)

- Add a new leaf and percolate up.

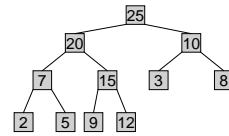


CSE 589 - Lecture 8 - Spring 1999

31

Insert (3)

- Add a new leaf and percolate up.

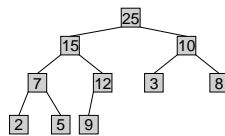


CSE 589 - Lecture 8 - Spring 1999

32

Delete-Max

- Remove the root then percolate last leaf down.

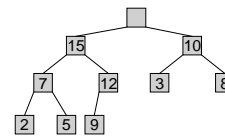


CSE 589 - Lecture 8 - Spring 1999

33

Delete-Max (2)

- Remove the root then percolate last leaf down.

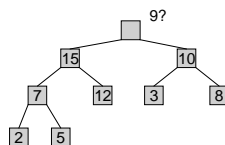


CSE 589 - Lecture 8 - Spring 1999

34

Delete-Max (3)

- Remove the root then percolate last leaf down.

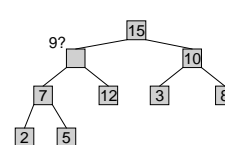


CSE 589 - Lecture 8 - Spring 1999

35

Delete-Max (4)

- Remove the root then percolate last leaf down.

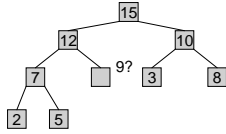


CSE 589 - Lecture 8 - Spring 1999

36

Delete-Max (5)

- Remove the root then percolate last leaf down.

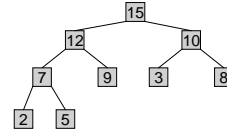


CSE 589 - Lecture 8 - Spring 1999

37

Delete-Max (5)

- Remove the root then percolate last leaf down.



CSE 589 - Lecture 8 - Spring 1999

38

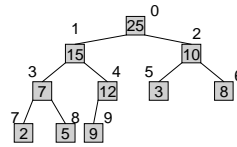
Analysis of the Heap Operation

- Insert - $O(\log n)$ worst case.
 - Each percolate up goes up at most $\log n$ levels.
 - Often $O(1)$ in practice because keys do not percolate far.
- Delete-Max - $O(\log n)$ worst case.
 - Percolates down tend to go close to the leaves of the heap.

CSE 589 - Lecture 8 - Spring 1999

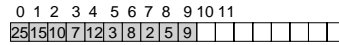
39

Implicit Pointers



parent of i is $(i-1)/2$

children of i are $2i+1, 2i+2$

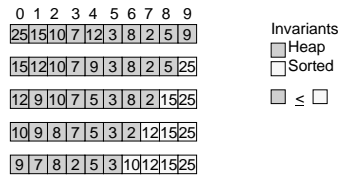


CSE 589 - Lecture 8 - Spring 1999

40

Heapsort Williams 1964

We will sort the array $A[0..n-1]$ in-place
 Build a heap in-place
 For $i = n-1$ to 1
 $A[i] := \text{delete-max};$



CSE 589 - Lecture 8 - Spring 1999

41