Sorting atomic items

Chapter 5

Sorting on 2-level memory model

Atomic items occupy constant-fixed number of memory cells (no variable length). Usually 4 or 8 bytes.

Sequence S of n atomic items with $n>M$

In disk model Sorting problem is equivalent to Permuting problem by the point of view of I/o complexity.

In RAM model Sorting includes Permuting since we need to determine the sorted permutation and then permute the items. Sorting is $\Theta(n\log n)$ while permuting is $\Theta(n)$.

Moving elements is difficult as Sorting in this model. It is the real bottleneck

The merge-based sorting

MergeSort (S ,i, j)

1. If (i<j) **{**

2. m=(i+j)/2;
3. MergeSort (S, i, m);
4. MergeSort (S, m+1, j);
5. Merge (S, I, m, j);
6. }

Based on Divide&Conquer. MergeSort is not in place alg.

Merging step needs auxiliary array on elements.

Merge takes O(n) time hence:

$T(n)= 2T(n/2) + O(n)$, $T(n) = \Theta(n\log n)$ time

and $\Theta(n)$ space

2-level model

The cost of merging 2 sequences of a total number of items z is $O(z/B)$ I/O's .

If $M \geq 2B$, the alg. takes in main memory 2 pages that contain items pointed by the 2 pointers scanning $S(i, j)$.

When a pointer advances into another page there is an I/O fault and another page is fetched to M.

$O(z/B)$ I/O's are also needed to write the merged sequence

Hence the I/O complexity of MergeSort is:

$$T(n) = 2T(n/2) + O(n/B) = O(n/B \log n)$$

logn levels of recursion, at each level $O(n/B)$

2-level model

Assume n>M: S is stored on disk, I/O operation takes 5ms on average.

If one comparison takes one I/O operation the running time on a massive data set S is:

5ms X Θ(nlogn).

If n is of order of few Gigabytes, such as $n \sim 2^{30}$

$5 X 2^{30} X 30 = 10^8$ ms around 1 day of computation!!

But if we run MergeSort on a PC on such a sequence S it takes only few hours.

Why?

When the recursion produces sub-arrays of size less than M the cost reduces.

2-level model

When the sub-sequence is of size $z=O(M)$, is contained into the cache. It can be handled completely inside the memory with no I/O faults.

The cost of sorting sub-sequence is of size $z=O(M)$ is not $O(z/B \log z)$ according to the previous result

But $O(z/B)$ which is the cost to load the sub-sequence into the memory.

MergeSort in 2-level model

N. of sequences	N. of items	#I/Os for Merge
2	n/2	O(n/B)
4	n/4	O(n/B)
8	n/8	O(n/B)
....	.....	....
n/M	M	O(n/B)

The last n/M sequences takes O(M/B) instead of O((M/B)logM)) to be sorted.
The total gain is O((n/B)logM)
O((n/B) logn) - O((n/B)logM) =

O((n/B) log(n/M)) Total number of I/O's

Optimize MergeSort

Stop the recursion at M:

More precisely: when the subsequence size S[i+1,j], $j-i < cM$.

c takes into account of the space occupancy of the sorter. ($c=1$ for in place sorting, $c=0.5$ for MergeSort for the extra-array for merging).

We should write cM instead of M in the previous bound,

that becomes $O((n/B)\log(n/cM))$.

c is close to 1 using a different in place alg. when sorting small subsequences, e.g. Heapsort or InsertionSort which is good enough for small values of M. e. g. when there are two levels of cache L1 and L2, L1 is small (few megabytes).

Optimize MergeSort

Problem: Merge passes over the data bottleneck!

$O((n/B) \log(n/M))$ is bigger when M is small

Solutions:

Enlarge M (physically is very expensive!)

1) Deploy M as much as possible

SnowPlow algorithm: virtually increase the memory size of a factor 2 in average.

2) Enlarge M virtually

Data compression: encode the items with integer compression which squeezes integers in fewer bits. Encoded items can be packed more in internal memory.

LATER!

SnowPlow

Is divided in phases. Each phase produces a sorted subsequence of size s, $M \le s \le 2M$.

Each phase has 4 steps:

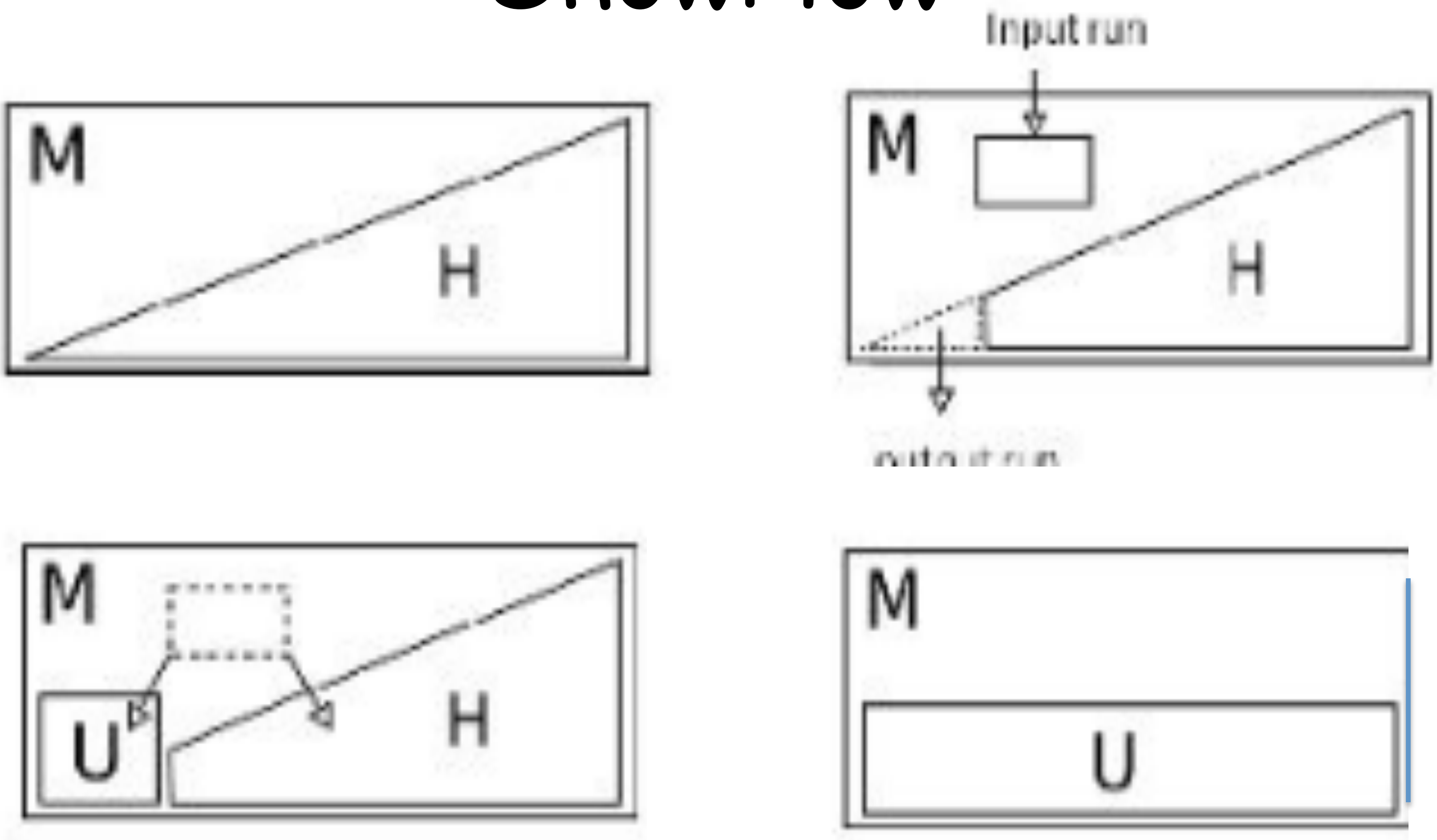
1. Form a min Heap H of the items contained in M.
2. At each step, while scanning items from S:
3. - extract min from H and output it;
4. - **if** next > min insert next in H

   else insert next in a auxiliary storage U.*

A phase terminates when H is empty and U occupies the whole M.

- The output run is non decreasing.
- At the end all the elements in H will be output, hence the number of steps of a phase is $\ge$ M.

SnowPlow



The 4 steps of a phase.

SnowPlow

SnowPlow is more efficient than MergeSort on average.

Let τ be the number of elements read in a phase
A phase ends when H is empty and $|U|=M$.
M items of the τ scanned end-up in U.
T – M goes to H and written to the output sorted run .
The length of the sorted run at the end of the phase is
$M + (\tau - M)$.

How much is τ on average?

Pr(next < min) =1/2 for uniform distribution.
So on average $\tau/2$ elements go to H and $\tau/2$ elements go to U. So $M=\tau/2$ and $\tau=2M$.

SnowPlow

SnowPlow builds O(n/M) sorted runs, each larger than M, and of length 2M in average.

Using Snowplow for generating sorted runs and a Merge-based sorting scheme we obtain:

$O((n/B)\log(n/2M))$ I/O's on average.

Multi-way MergeSort

Previous algorithms → binary Merge

Now: Multi-way Merge.

Binary merge uses 3 blocks: 2 blocks to cache items from S[x] and S[y], 1 block to cache the output items.

But M/B >> 3: Many more blocks available!

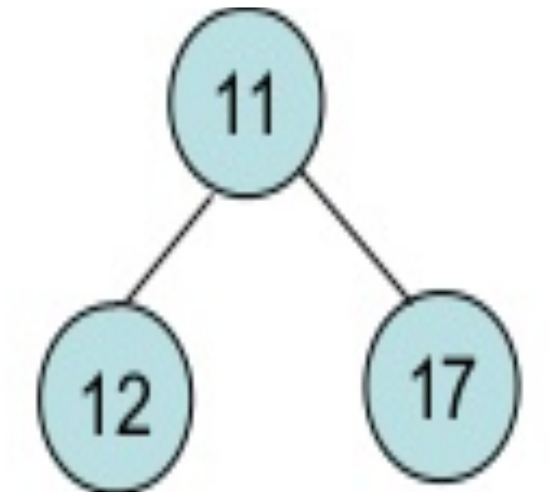
K-way Merging, set k =(M/B)-1 (1 block for the output)

Merge K-runs:

- Build minHeap H to contain the k minima from the k runs.
- Items are represented by pairs : <Ri[1], i>

Multi-way MergeSort: Example k=3

Heap H



M= 4B k=(M/B-1)

R_1 : 11, 18, 44, 63, 87 …

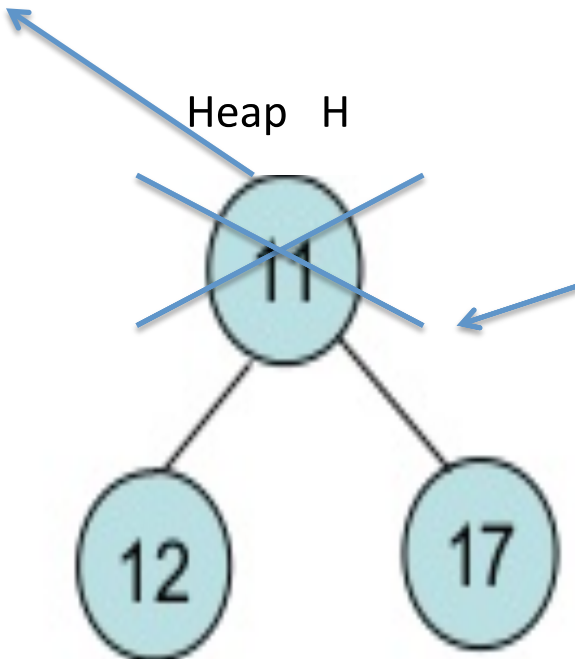
R_2: 12, 21, 32, 48, 54, …

R_3: 17, 19, 25, 33, 39, ….

At each step:

- Extract min from H
- Take another item from Ri (if not ended)

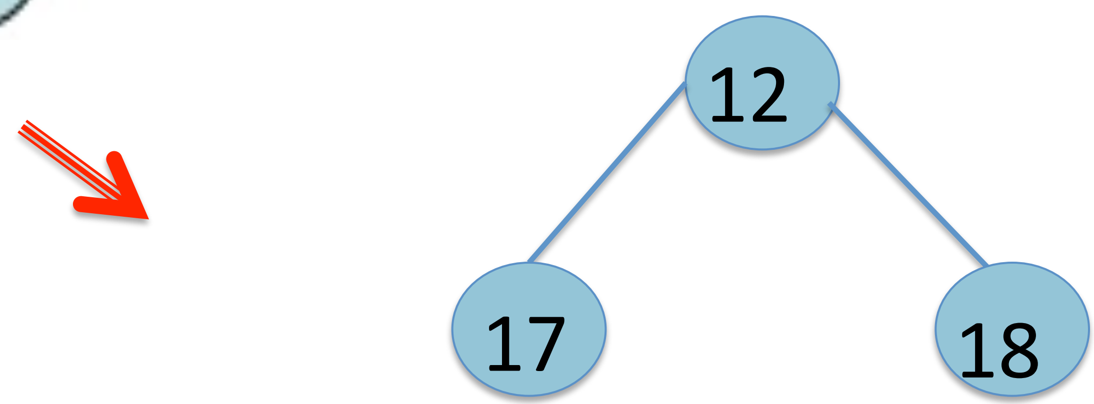
Multi-way MergeSort



R_1: 11, 18, 44, 63, 87 …

R_2: 12, 21, 32, 48, 54, …

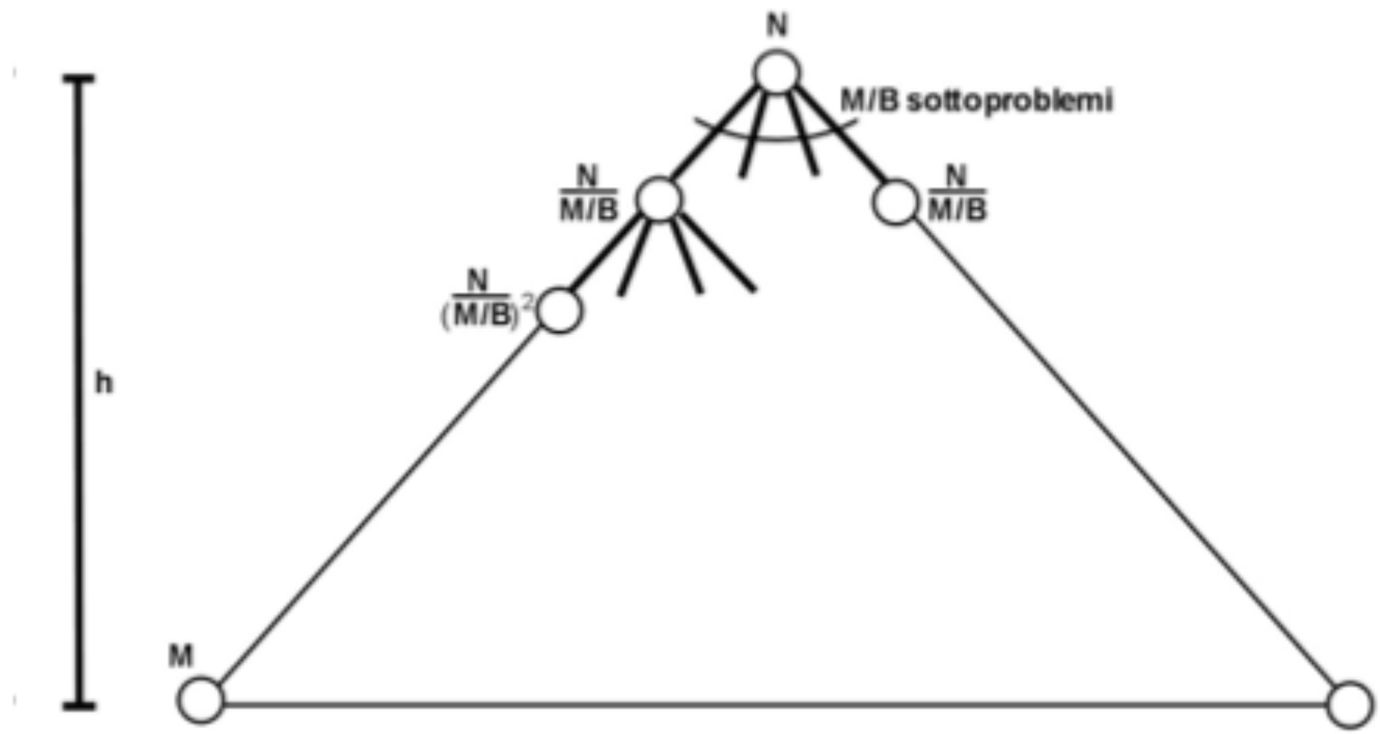
R_3: 17, 19, 25, 33, 39, ….



Multi-way MergeSort

Merging takes O(logk) time per item.
O(z/B) I/O's to merge k runs of total length z.

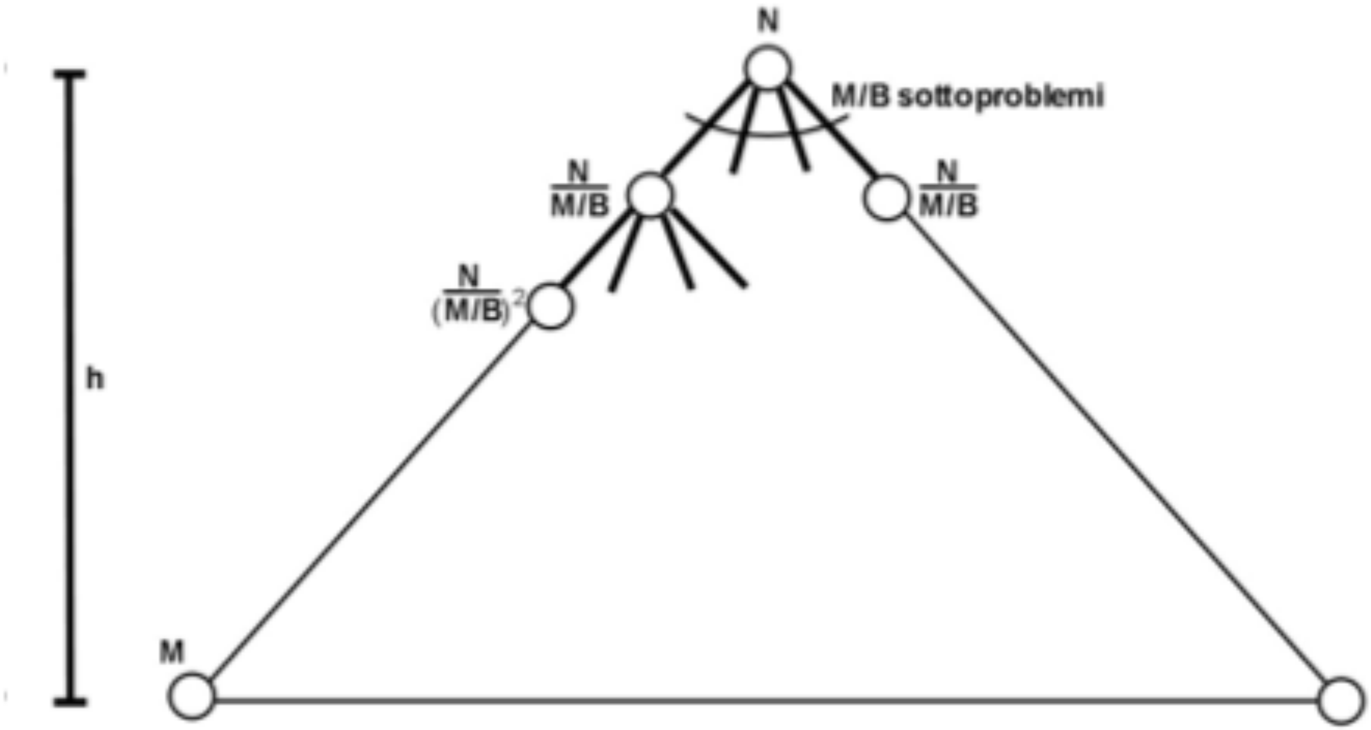
The runs can be produced e.g. by SnowPlow alg.



Multi-way MergeSort

How many levels of recursion?
$n/(M/B)^i \leq M$ $n/M \leq (M/B)^i$ $i \geq \log_{M/B}(n/M)$

Total number of I/O's = $O((n/B) \log_{M/B}(n/M))$
and $O(n\log n)$ time



Multi-way MergeSort

In practice:

The number of recursion levels is very small.

Assume B=4KB, M=4GB, $M/B= 2^{32}/2^{12}=2^{20}$

The number of levels = $\log_{M/B}$, is 1/20 less than Binary MS!

Remember:

$\mathrm{Log}_b a=\log_c a/\log_c b$ $\log_{M/B} n/M= \log (n/m)/\log(M/B)$