CS4311
Design and Analysis of Algorithms

Introduction to External Memory Algorithms

About this tutorial

- Introduce External Memory (EM) Model
- How do we perform sorting ?

Our slides are based on the slides by

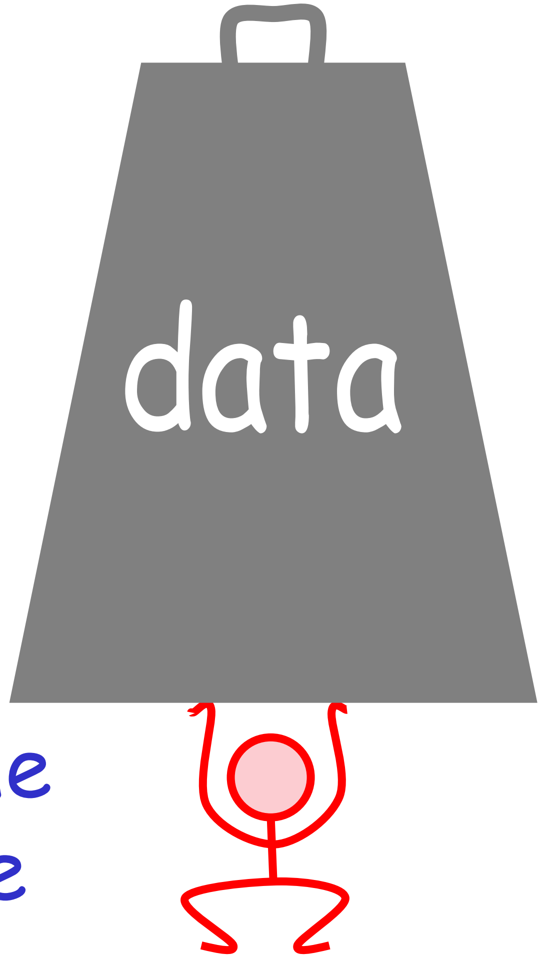
G. Brodal and R. Fagerberg

See their web page for more info:

http://www.daimi.au.dk/~gerth/emF03

Dealing with Massive Data

- In some applications, we need to handle a lot of data
  - so much that our RAM is not large enough to handle
- Ex 1: Sorting most recent 8G Google search requests
- Ex 2: Finding LCS between the DNAs of human & mouse



Dealing with Massive Data

- Since RAM is not large enough, we need the hard-disk to help the computation
- Hard-disk is useful:
  1. can store input data (obvious)
  2. can store intermediate result
- However, there are new concern, because accessing data in the hard-disk is much slower than accessing data in RAM

EM Model [Aggarwal-Vitter, 88]

- Computer is divided into three parts:
  CPU, RAM, Hard-disk
- CPU can work with data in RAM directly
  - But not directly with data in hard-disk
- RAM can read data from hard-disk, or write data to hard-disk, using the I/O (input/output) operations

EM Model [Aggarwal-Vitter, 88]

- Size of RAM = M items
- Hard-disk is divided in contiguous pages
  - Size of a disk page = B items
- In one I/O operation, we can
  - read or write one page
- Complexity of an algorithm =

number of I/Os used

➔ That means, CPU processing is free !

Test Our Understanding

- Suppose we have a set of N numbers, stored contiguously in the hard-disk

- How many I/Os to find max of the set?

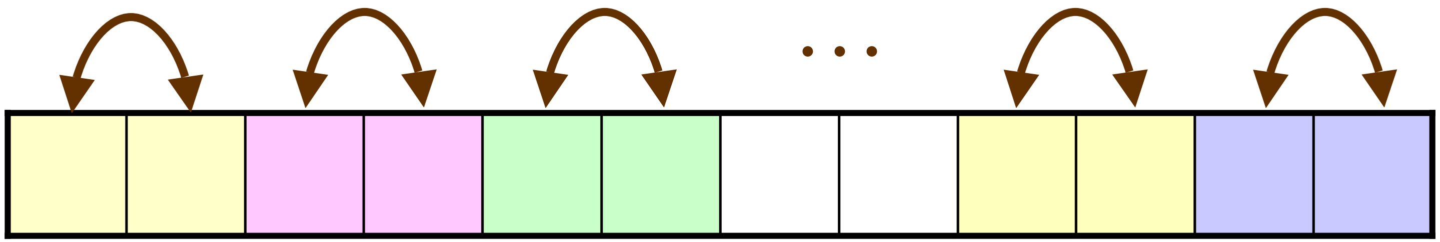
  Ans. $O(N/B)$ I/Os

- Is this optimal ?

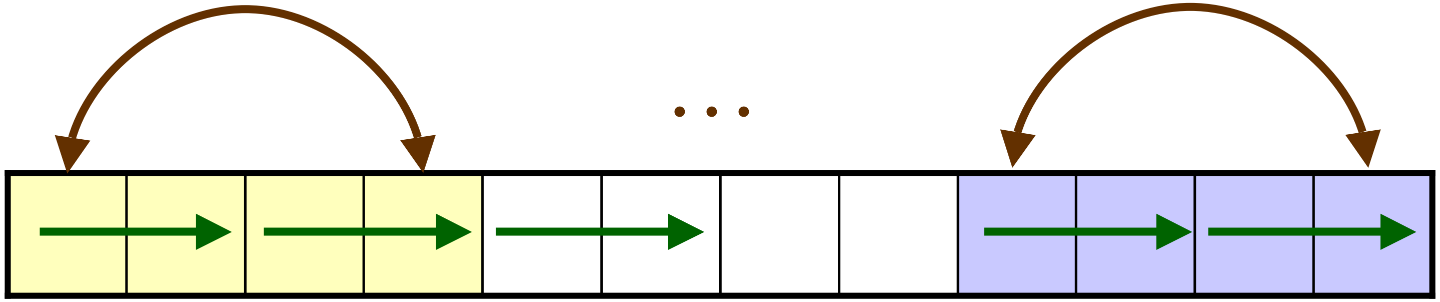
  Ans. Yes. We must read all #s to find max, which needs at least N/B I/Os

Sorting in External Memory

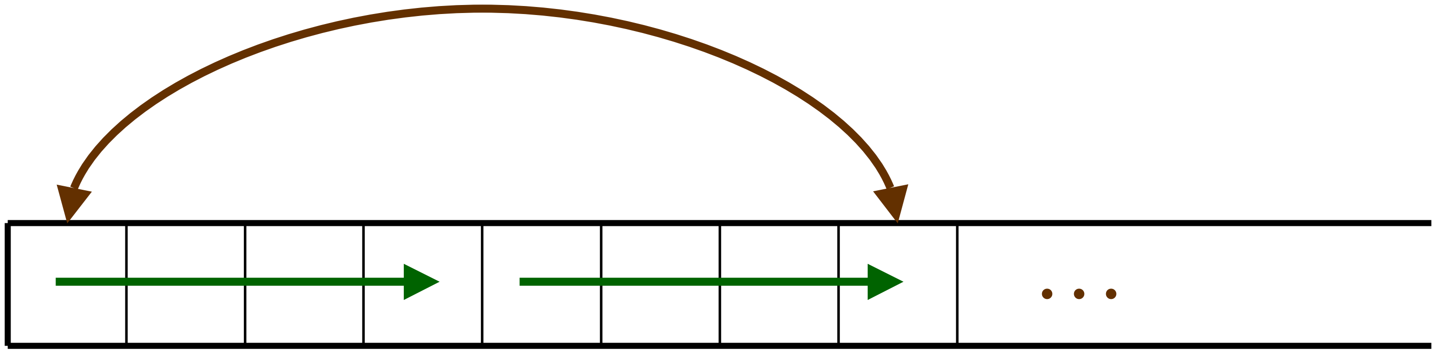
- We shall use the idea of Merge Sort to sort N numbers in the external memory
- Recall: We can perform Merge Sort in a bottom-up manner:



Round 1: Sort every 2 numbers



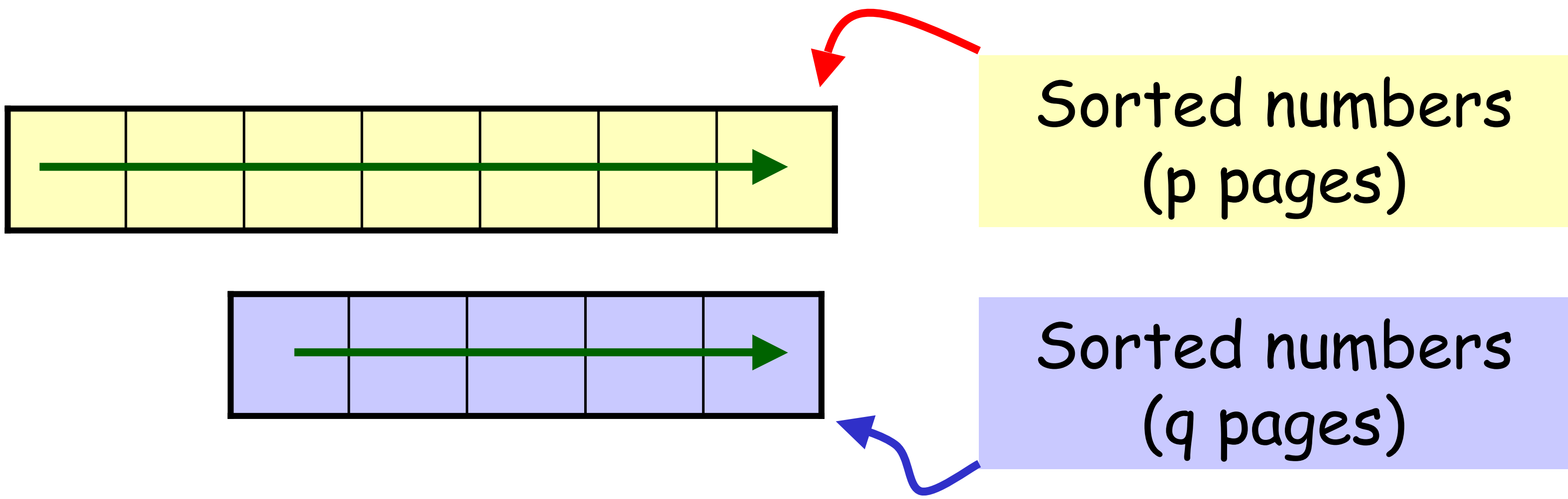
Round 2: Sort every 4 numbers by merging pairs of 2 numbers



Round k: Sort every 2^k numbers by merging pairs of 2^{k-1} numbers

Sorting in External Memory

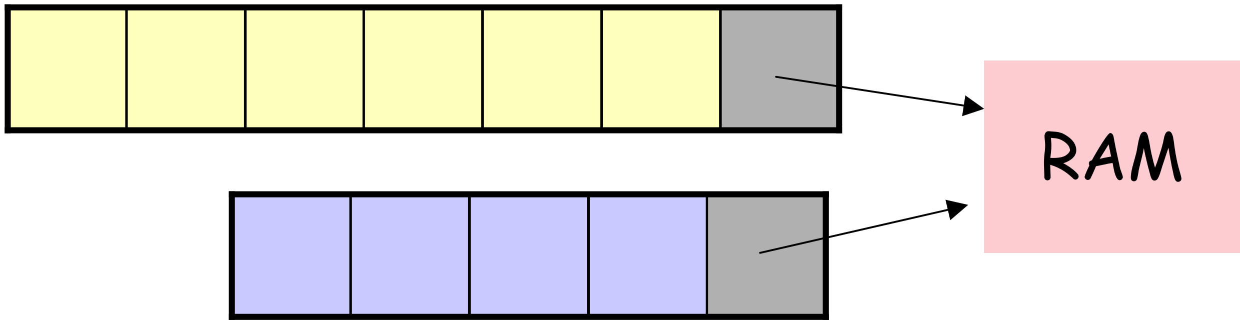
- Now, let us see if we can use Merge Sort directly to sort things in external memory
- Suppose we have two sorted lists of #s, which are placed in p pages and q pages:



Sorting in External Memory

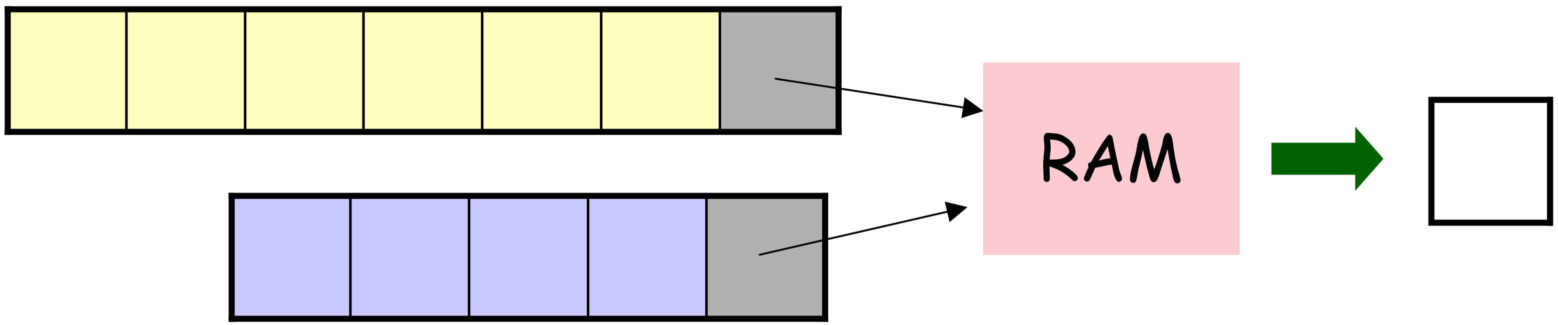
- How can we merge them ?
- Method:
  - Load 1st page from each list

    → Must contain B smallest numbers



Sorting in External Memory

- Method (cont):
  - CPU sorts the numbers in RAM
  - RAM outputs B smallest #s in a pages



- Next, we should read another page for merging... But which one ?

Sorting in External Memory

Lemma :

Suppose largest # in RAM is from List 1.

Then, all the next B smallest numbers are not contained in the next page of List 1.

Based on the above lemma, we know which page should be read next …

➔ we can repeatedly sort the remaining

➔ In total, O(p+q) I/Os

Sorting in External Memory

Thus, using Merge Sort,

- Each round takes O(N/B) I/Os
- there are O(log N) rounds

➔ Total I/O = O((N/B) log N)

Question: Can we improve it?

Recall: Our RAM can hold M pages …

Sorting in External Memory

At Round 1, instead of sorting 2 numbers, let us sort M numbers together ! (How ??)

Then, Round 1 still takes $O(N/B)$ I/Os, but we begin with N/M sorted lists

→ Only needs $\log (N/M)$ more rounds

→ Total I/O = $O((N/B) \log (N/M))$

Question: Can we further improve it?

Sorting in External Memory

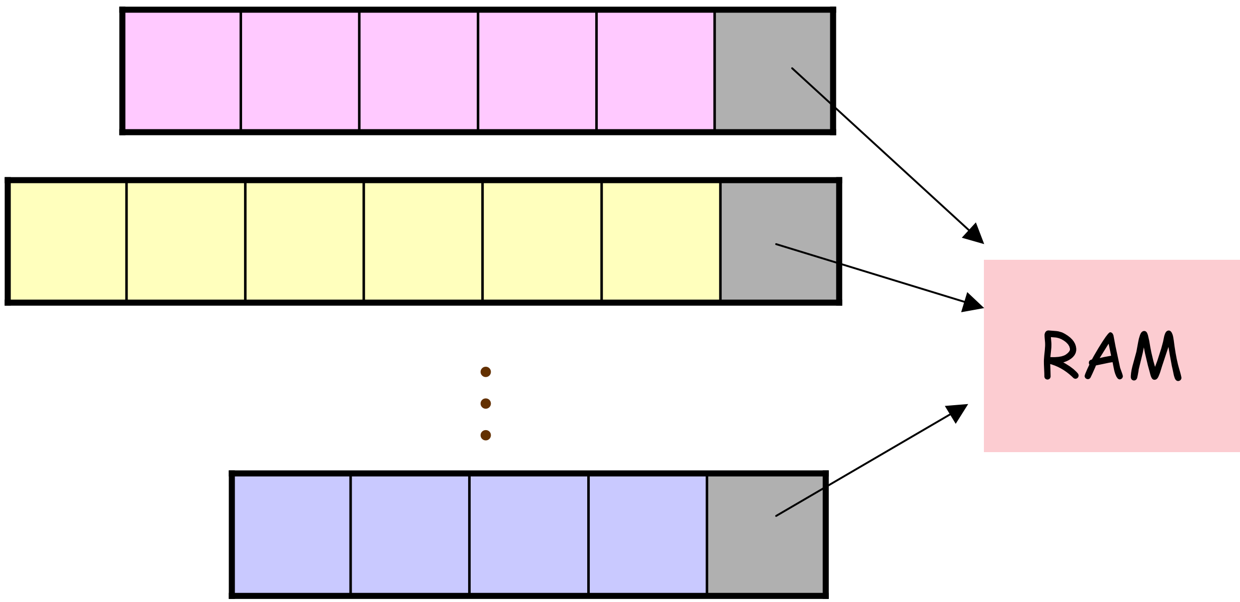
- In current Merge Sort, we are merging two lists at a time…
- What if we merge more lists at a time?

- Precisely, suppose we have k sorted lists, where List i occupies p_i pages

- How can we merge them ?

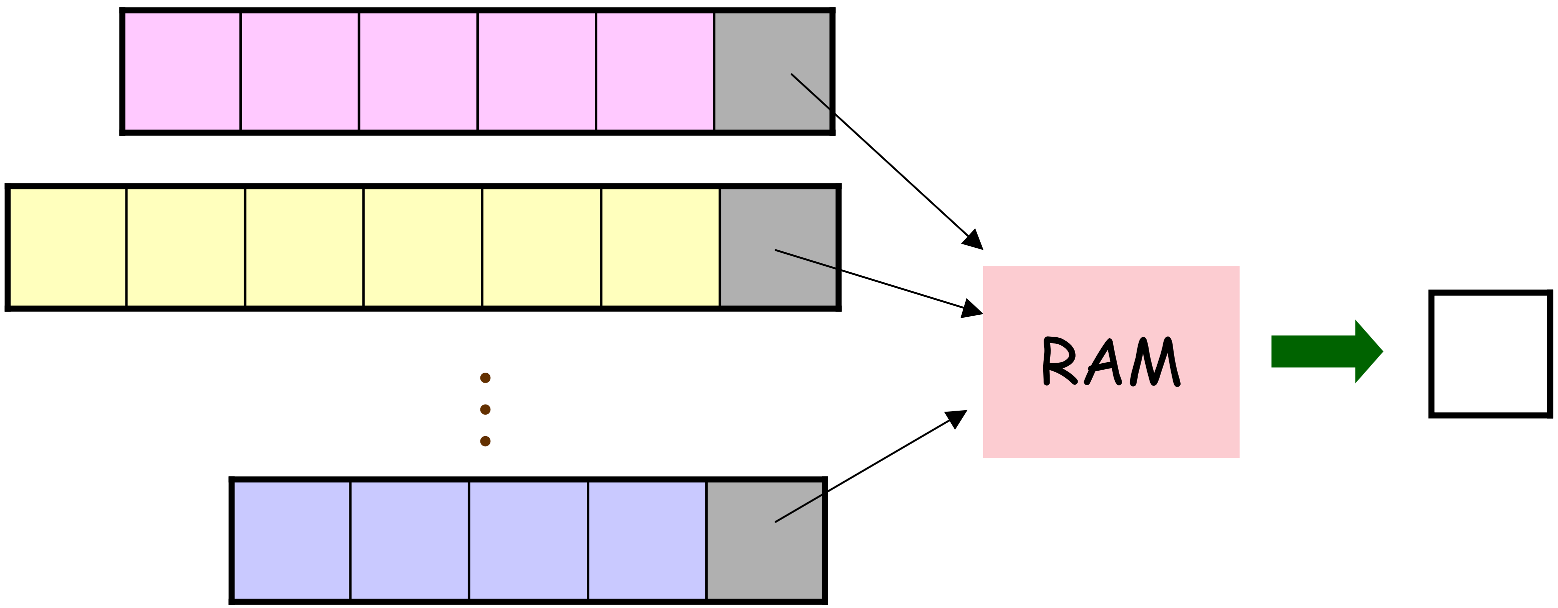
Sorting in External Memory

- Method :
  - Load 1st page from each list
    → Must contain B smallest numbers



Sorting in External Memory

- Method (cont):
  - Next, outputs B smallest #s in a pages



- Now, do we read a page? Or output more ?

Sorting in External Memory

Consider the following minor change :

- Suppose we maintain an extra page, called output buffer, in RAM
- We try to fill the output buffer with the correct smallest elements, and once the buffer is full, we output it
- When we fill the output buffer, as soon as some list L has run out of #s in RAM, we read the next page from L

Sorting in External Memory

Lemma:

- When the output buffer is full, it always contains the next smallest B #s
- Apart from the #s in output buffer, each list has at most B #s in RAM

Sorting in External Memory

- The previous lemma implies that we can repeatedly read pages from the k lists, fill the output buffer, and get a sorted list eventually

  → If List i contains p_i pages,

  Total I/O = $O(p_1 + p_2 + \ldots + p_k)$

- Also, it implies that RAM has at most $k+1$ pages at any time → $M \geq (k+1)B$ is enough

Sorting in External Memory

- So, we can perform sorting with k-way merging as follows:
  1. Create N/M sorted lists of length M
  2. At round $j = 1, 2, \ldots$

     Merge k sorted list of length $k^{j-1} M$, forming a sorted list of length $k^j M$

➔ # rounds = $\log_k (N/M)$

➔ Total I/O = $O((N/B) \log_k (N/M))$

Sorting in External Memory

- The larger the k, the smaller the term:

$$O(\ (N/B)\ \log_k (N/M)\ )$$

- Since the only restriction on k is that:

$$M \geq (k+1)B$$

- Thus, we can sort the N numbers in :

$$O(\ (N/B)\ \log_{(M/B\ -\ 1)} (N/M)\ )\ \ \text{I/Os}$$

Sorting in External Memory

- Usually, $M \gg B$, so that

$$\log(M/B - 1) = \Theta(\log(M/B))$$

- Then, sorting I/O becomes:

$$O((N/B) \log_{M/B} (N/M)) \text{ I/Os}$$
$$= O((N/B) \log_{M/B} (N/B)) \text{ I/Os}$$

➔ Better than 2-way Merge Sort: $O((N/B) \log (N/M))$

Sorting in External Memory

- In fact, we can show that if we can only use comparison to deduce the relative order between the input numbers,

  then, sorting in external memory requires

  $$\Omega(\ (N/B)\ \log_{M/B} (N/B)\ )\ \text{I/Os}$$

  in the worst case

→ (M/B)-way Merge Sort is optimal !!