# Introduction to Computation Models

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

Computer science is a subject of mathematics.

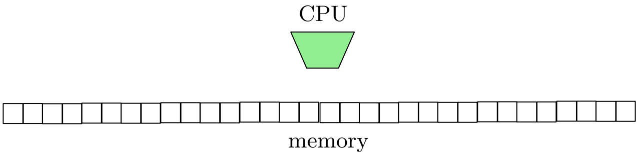
The subject studies how to apply **atomic operations** to accomplish a task with the lowest **cost**.

Different **computation models** produce different branches of the subject.

Today, we will look at several mainstream models.

## I. Random Access Machine (RAM)

This is the model you used in your undergraduate study.




**Atomic operations** ($a$ and $b$ are registers):

- $a \leftarrow 100$, $a \leftarrow b$
- $a + b$, $a - b$, $a * b$, $a/b$
- $a < b$, $a = b$, or $a > b$
- read a memory cell into $a$, or write $a$ into a memory cell.

**Time** of an algorithm = # atomic operations performed.

**The Sorting Problem:** Given an array $A$ of $n$ distinct integers, produce another array where the same integers have been arranged in ascending order.

## Merge Sort

- **Divide:** Let $A_1$ the array containing the first $\lceil n/2 \rceil$ elements of $A$, and $A_2$ be the array containing the other elements of $A$.
  Sort $A_1$ and $A_2$ recursively.
- **Conquer:** Merge the two sorted arrays $A_1$ and $A_2$ in ascending order. This can be done in $O(n)$ time.

$f(n)$: time of sorting $n$ integers

$$f(n) = 2 \cdot f(n/2) + O(n)$$

which yields $f(n) = O(n \log n)$.

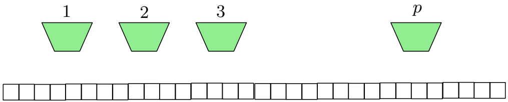
A computation model falls short when it can no longer be used to guide the design of algorithms.

Today, a modern CPU has multiple cores and supports **parallel execution**.

RAM can no longer capture an algorithm's behavior on those CPUs.

## II. Parallel Random Access Machine (PRAM)

$p = \#$ CPUs



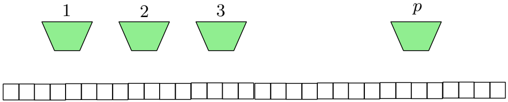

A **step**: each CPU independently performs an atomic operation ($a$ and $b$ are registers inside the CPU):

- $a \leftarrow 100$, $a \leftarrow b$
- $a + b$, $a - b$, $a * b$, $a/b$
- $a < b$, $a = b$, or $a > b$
- read a memory cell into $a$, or write $a$ into a memory cell.

> **Constraint:** In a step, if a CPU writes to a cell, no other CPU can read or write to the cell.

This is the **CREW** model (common read exclusive write).

**The Sorting Problem:** Given an array $A$ of $n \geq p$ distinct integers, produce another array where the same integers have been arranged in ascending order.




Naive: $O(n \log n)$ steps.
Optimal: $O(\frac{n}{p} \log n)$ steps. **Why?**

We will see how to do $O(\frac{n}{p} \log^2 n)$ steps.

We will assume $p = n$. Extending the discussion to $p < n$ is left to you.

**The Merging Problem:** Let $A_1$ and $A_2$ be two sorted arrays, each with $p/2$ integers. Merge $A_1$ and $A_2$ into one sorted array (with $p$ integers).

Naive: $O(p)$ steps.

We will do $O(\log p)$ steps.

| $A_1$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17 | 26 | 28 | 38 | 41 | 72 | 83 | 88 |

| $A_2$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 9 | 12 | 35 | 47 | 52 | 68 | 69 |

Consider the number 26 in $A_1$.

- It is the 2nd smallest in $A_1$.
- It is greater than 3 numbers in $A_2$.

Hence, 26 should be the $2 + 3 = 5$-th in the merged array.

$A_1$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17 | 26 | 28 | 38 | 41 | 72 | 83 | 88 |
| 4 | 5 | 6 | 8 | 9 | 14 | 15 | 16 |

$B_1$

$A_2$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 9 | 12 | 35 | 47 | 52 | 68 | 69 |
| 1 | 2 | 3 | 7 | 10 | 11 | 12 | 13 |

$B_2$

We want to generate arrays $B_1$ and $B_2$. For each number $x$ in $A_1$ (or $A_2$), the corresponding cell in $B_1$ (or $B_2$) indicates the position of $x$ in the final merged array.

Once $B_1$ and $B_2$ are ready, the merged array can be produced in $O(1)$ steps — recall that we have $p = 16$ CPUs.

$A_1$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17 | 26 | 28 | 38 | 41 | 72 | 83 | 88 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 5 | | | | | | |

$B_1$

$A_2$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 9 | 12 | 35 | 47 | 52 | 68 | 69 |

How to generate $B_1$ and $B_2$?
By assigning one CPU to each number, we can generate the number's position indicator in $O(\log p)$ steps using binary search.

This gives an algorithm solving the merging problem in $O(\log p)$ steps.

We can now sort $n$ elements with $n$ CPUs in $O(\log n)$ phases.

- Phase 0: $n/2$ parallel merges, each merging two lists of size 1.
- Phase 1: $n/4$ parallel merges, each merging two lists of size 2.
- Phase 2: $n/8$ parallel merges, each merging two lists of size 4.
- ...

Each phase takes $O(\log n)$ steps.
Overall, $O(\log^2 n)$ steps.

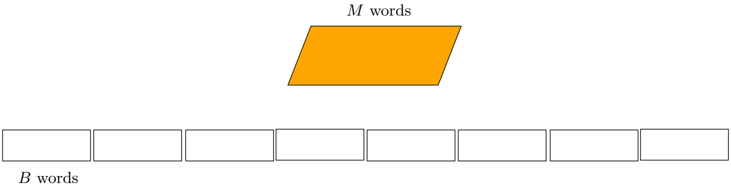
## III. External Memory (EM)

RAM and PRAM assume that the data fit in memory.

In practice, a dataset often needs to be stored in the disk. In those scenarios, CPU operations are rarely the performance bottleneck. Typically, an algorithm's efficiency depends on how many **disk I/Os** are performed.

This motivates the **external memory** model.

$M$ = the number of words in memory
$B$ = the number of words in a disk block




A **I/O**

- reads a disk block into memory, or
- writes $B$ memory words into a disk block.

**Cost** of an algorithm: # I/Os performed.

> CPU calculation is for free.

**The Sorting Problem:** Given a file $A$ of $n \geq B$ distinct integers stored in $O(n/B)$ blocks, produce another file of $O(n/B)$ blocks where the same integers have been arranged in ascending order.

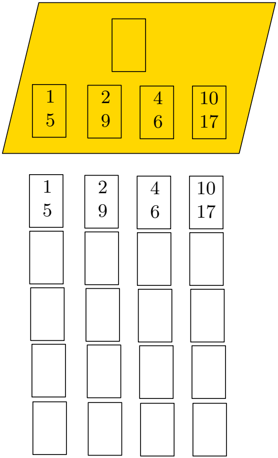
Naive: $O(n \log n)$ I/Os.
We will do: $O(\frac{n}{B} \log_{M/B} \frac{n}{B})$ I/Os.

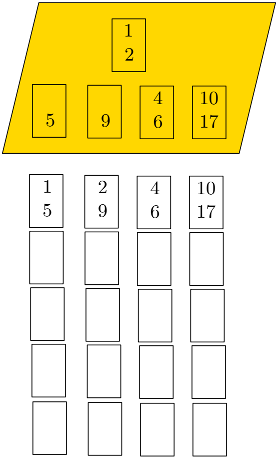
**The Merging Problem:** Let $A_1$, $A_2$, ..., $A_{\frac{M}{B}-1}$ be $\frac{M}{B}-1$ files such that

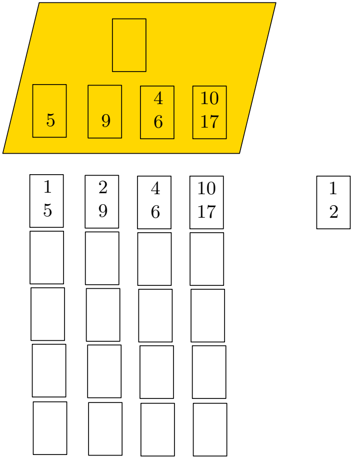
- each file stores $t_i \geq B$ integers in ascending order with $O(t_i/B)$ block;
- the integers in all the files are distinct.

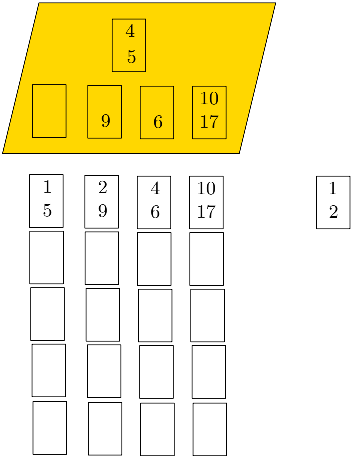
Let $t = \sum_{i=1}^{M/B-1} t_i$. Produce another file of $O(t/B)$ blocks where all the $t$ integers have been arranged in ascending order.

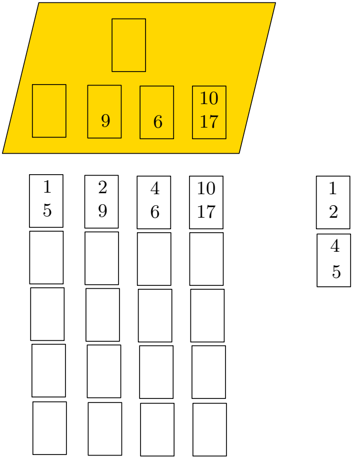
We will see how to solve the problem in $O(t/B)$ I/Os.

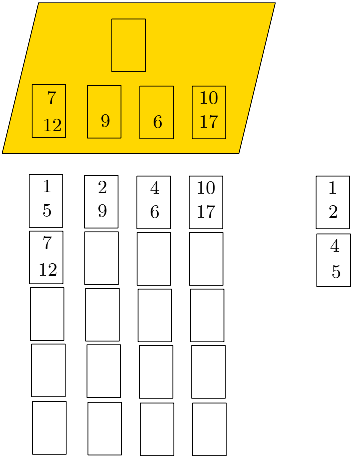

1
5
2
9
4
6
10
17
1
5
2
9
4
6
10
17


1
2
5
9
4
6
10
17
1
5
2
9
4
6
10
17


5
9
4
6
10
17
1
5
2
9
4
6
10
17
1
2


4
5
9
6
10
17
1
5
2
9
4
6
10
17
1
2


9
6
10
17
1
5
2
9
4
6
10
17
1
2
4
5




The total cost is $O(t/B)$ because

- there are $O(t/B)$ blocks in the input files;
- there are $O(t/B)$ blocks in the output file.

Think: equipped with the above merging algorithm, how can we solve the sorting problem in $O(\frac{n}{B} \log_{M/B} \frac{n}{B})$ I/Os?

IV. Massively Parallel Computation (MPC)

Today, many computational tasks are performed on datasets whose sizes are at the tera-byte order or higher. Massively-parallel systems (such as Spark) are often deployed to carry out those tasks. Such a system involves dozens or hundreds of machines interconnected by a broadband network. CPU time and I/O cost are no longer the performance bottleneck. Instead, the efficiency of an algorithm is determined by **network communication**.

This motivates the **massively parallel computation** (MPC) model.

$p = \#$ machines

A **superstep** includes two steps.

- Step 1: each machines performs local computation.
- Step 2: each machine sends a message to every other machine.

**Cost of a superstep:** the maximum number of words communicated (i.e., sent and received) by a machine in Step 2.

- Step 1 is for free.

**Cost of an algorithm:** total cost of all the supersteps.

**Example:**

$p = 3$ machines. Suppose that in a superstep

- machine 1 sends 50m and 30m words to machine 2 and 3, respectively;
- machine 2 sends 0m and 40m words to machine 1 and 3, respectively;
- machine 3 sends 10m and 5m words to machine 1 and 2, respectively.

Cost of the superstep = 85m words.

**The Sorting Problem:** Let $S$ be a set of $n$ numbers such that each machine stores $O(n/p)$ numbers originally. Re-arrange the numbers such that

- each machine still stores $O(n/p)$ numbers;
- for any $1 \leq i < j \leq p$, all the numbers on machine $i$ are less than those on machine $j$.

We will consider $n \geq p^3$ (a reasonable assumption in practice).

Naive: cost $O(n)$.
We will do: cost $O(n/p)$ in two supersteps.

$S_i$ = the set of numbers on machine $i \in [1, p]$

## Superstep 1

- **Step 1:** Machine $i$ samples each integer in $S_i$ with probability $\frac{p}{n} \ln(np)$ independently.
- **Step 2:** Each machine sends the samples to all other machines.

| machine 1 | machine 2 | machine 3 |
|---|---|---|
| $(10, 33, 57, 60, 80)$ | $(5, 22, 48, 56, 90)$ | $(15, 35, 38, 88, 96)$ |
| $(5, 33, 38, 48, 80, 88)$ | $(5, 33, 38, 48, 80, 88)$ | $(5, 33, 38, 48, 80, 88)$ |

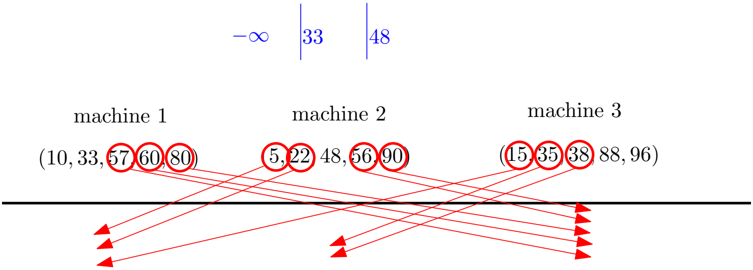
Let $s$ be the total number of samples (from all machines)

## Superstep 2

- **Step 1:** Each machine divides $\mathbb{R}$ into $p$ disjoint **segments**, each covering at most $s/p$ samples. Let these segments be $\sigma_1, \sigma_2, ..., \sigma_p$ in ascending order.

$(5, \boxed{33}, 38, \boxed{48}, 80, 88) \quad (5, \boxed{33}, 38, \boxed{48}, 80, 88) \quad (5, \boxed{33}, 38, \boxed{48}, 80, 88)$

- **Step 2:** Machine $i$ sends $S_i \cap \sigma_j$ to machine $j$, for each $j \in [1, p]$.



We can prove that the cost of the algorithm is $O(n/p)$ with probability at least $1 - O(1/n)$.

The proof is omitted from this talk.