Models of Computation for Massive Data

Jeff M. Phillips

August 28, 2013

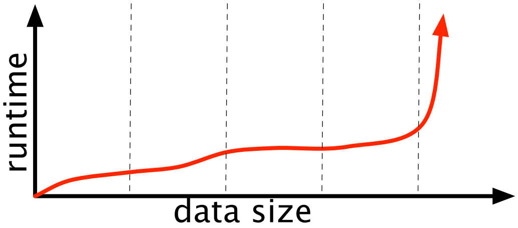
Outline

Sequential:

- External Memory / (I/O)-Efficient
- Streaming

Parallel:

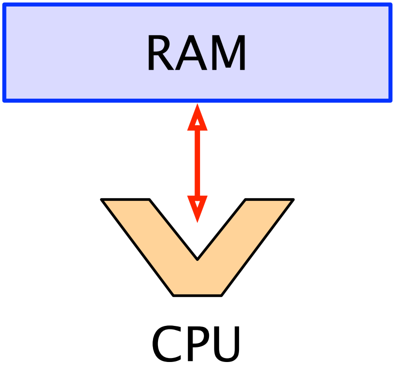
- PRAM and BSP
- MapReduce
- GP-GPU
- Distributed Computing



RAM Model

RAM model (Von Neumann Architecture):

- CPU and Memory
- CPU Operations ($+$, $-$, $*$, $\ldots$) constant time
- Data stored as *words*, not *bits*.
- READ, WRITE take constant time.

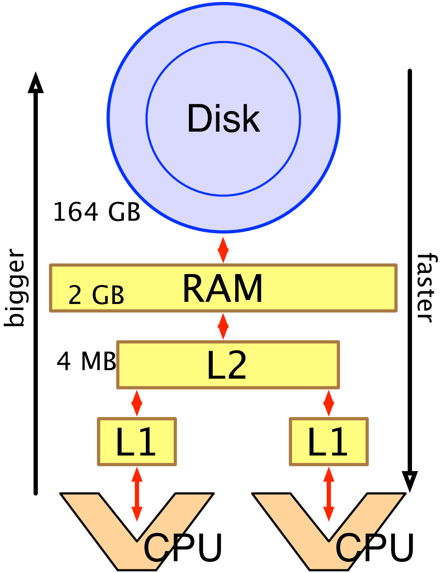


Today's Reality

What your computer actually looks like:

- 3+ layers of memory hierarchy.
- Small number of CPUs.

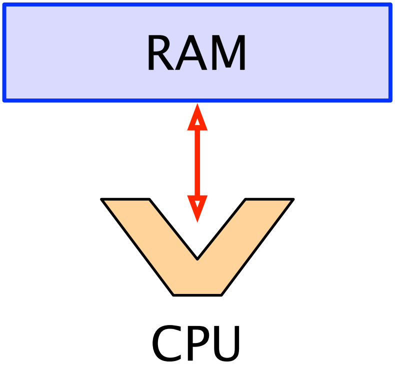
Many variations!



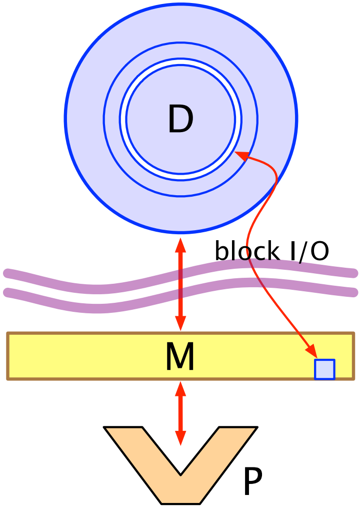
RAM Model

RAM model (Von Neumann Architecture):

- CPU and Memory
- CPU Operations $(+, -, *, \ldots)$ constant time
- Data stored as *words*, not *bits*.
- READ, WRITE take constant time.

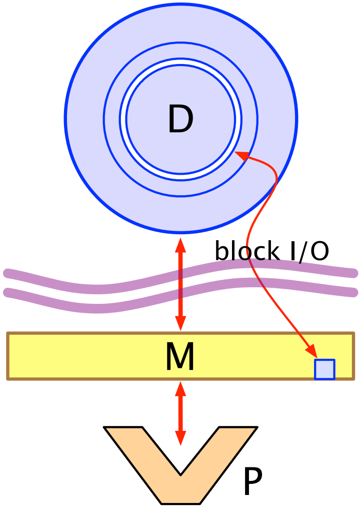


External Memory Model



- N = size of problem instance
- B = size of disk block
- M = number of items that fits in Memory
- T = number of items in output
- I/O = block move between Memory and Disk

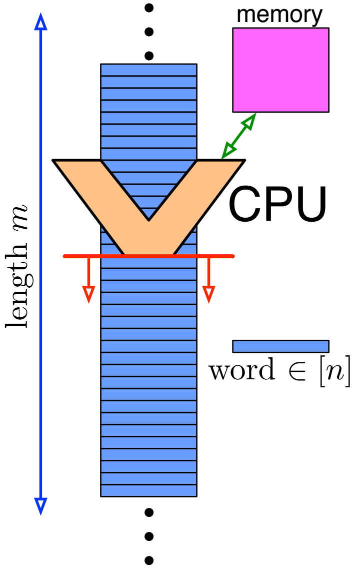
External Memory Model



- N = size of problem instance
- B = size of disk block
- M = number of items that fits in Memory
- T = number of items in output
- I/O = block move between Memory and Disk

Advanced Data Structures: Sorting, Searching

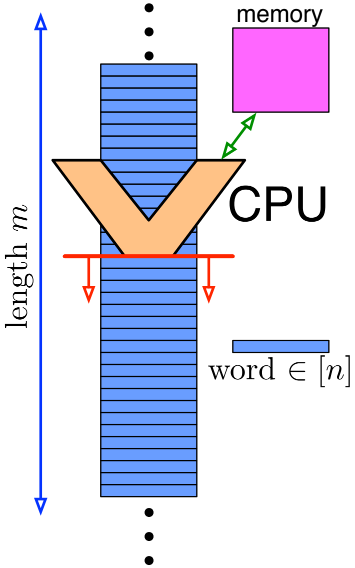
Streaming Model



CPU makes "one pass" on data

- Ordered set $A = \langle a_1, a_2, \ldots, a_m \rangle$
- Each $a_i \in [n]$, size $\log n$
- Compute $f(A)$ or maintain $f(A_i)$ for $A_i = \langle a_1, a_2, \ldots, a_i \rangle$.
- Space restricted to $S = O(\text{poly}(\log m, \log n))$.
- Updates $O(\text{poly}(S))$ for each a_i.

Streaming Model



CPU makes "one pass" on data

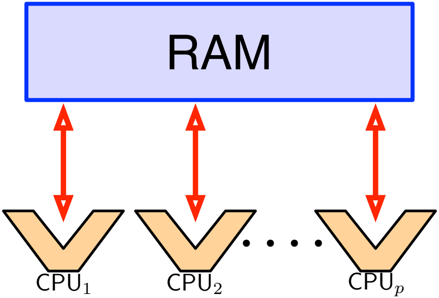
- Ordered set $A = \langle a_1, a_2, \ldots, a_m \rangle$
- Each $a_i \in [n]$, size $\log n$
- Compute $f(A)$ or maintain $f(A_i)$ for $A_i = \langle a_1, a_2, \ldots, a_i \rangle$.
- Space restricted to $S = O(\mathrm{poly}(\log m, \log n))$.
- Updates $O(\mathrm{poly}(S))$ for each a_i.

PRAM

Many (p) processors. Access shared memory:

- EREW : Exclusive Read Exclusive Write
- CREW : Concurrent Read Exclusive Write
- CRCW : Concurrent Read Concurrent Write

Simple model, but has shortcomings...
...such as Synchronization.

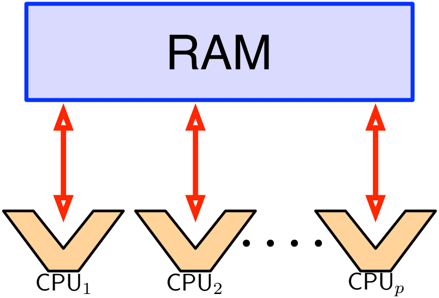


PRAM

Many (p) processors. Access shared memory:

- EREW : Exclusive Read Exclusive Write
- CREW : Concurrent Read Exclusive Write
- CRCW : Concurrent Read Concurrent Write

Simple model, but has shortcomings...
...such as Synchronization.

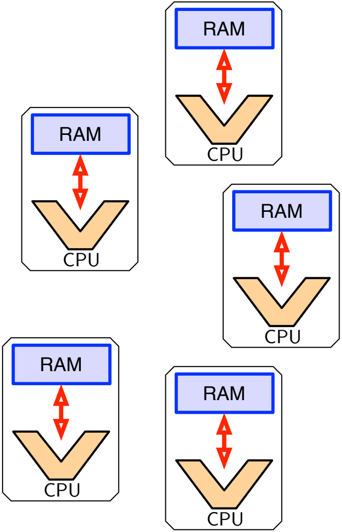


Bulk Synchronous Parallel

Each Processor has its own Memory
Parallelism Procedes in Rounds:

1. Compute: Each processor computes on its own Data: w_i.
2. Synchronize: Each processor sends messages to others: $s_i = \textsc{MessSize} \times \textsc{CommCost}$.
3. Barrier: All processors wait until others done.

Runtime: $\max w_i + \max s_i$



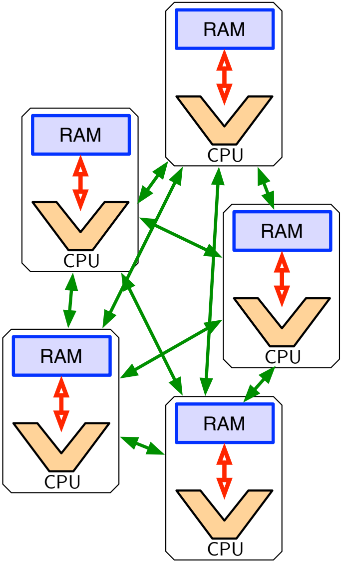
Pro: Captures Parallelism *and* Synchronization
Con: Ignores Locality.

Bulk Synchronous Parallel

Each Processor has its own Memory
Parallelism Procedes in Rounds:

1. Compute: Each processor computes on its own Data: w_i.
2. Synchronize: Each processor sends messages to others: $s_i = \textsc{MessSize} \times \textsc{CommCost}$.
3. Barrier: All processors wait until others done.

Runtime: $\max w_i + \max s_i$



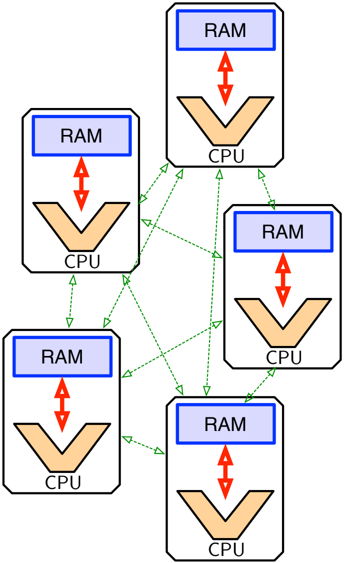
Pro: Captures Parallelism *and* Synchronization
Con: Ignores Locality.

Bulk Synchronous Parallel

Each Processor has its own Memory
Parallelism Procedes in Rounds:

1. Compute: Each processor computes on its own Data: w_i.
2. Synchronize: Each processor sends messages to others: $s_i = \textsc{MessSize} \times \textsc{CommCost}$.
3. Barrier: All processors wait until others done.

Runtime: $\max w_i + \max s_i$



Pro: Captures Parallelism *and* Synchronization
Con: Ignores Locality.

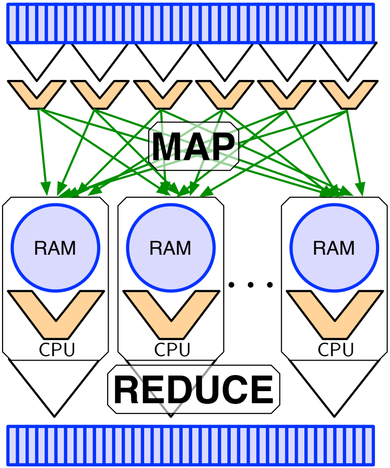
MapReduce

Each Processor has full hard drive, data items $< \text{KEY}, \text{VALUE} >$.
Parallelism Procedes in Rounds:

- Map: assigns items to processor by KEY.
- Reduce: processes all items using VALUE. Usually combines many items with same KEY.

Repeat M+R a constant number of times, often only one round.

- Optional post-processing step.



Pro: Robust (duplication) and simple. Can harness Locality
Con: Somewhat restrictive model

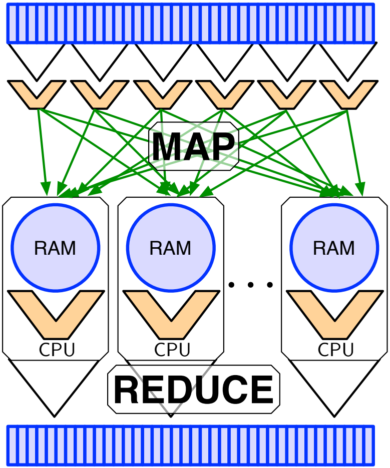
MapReduce

Each Processor has full hard drive, data items $< \text{KEY}, \text{VALUE} >$.
Parallelism Procedes in Rounds:

- Map: assigns items to processor by KEY.
- Reduce: processes all items using VALUE. Usually combines many items with same KEY.

Repeat M+R a constant number of times, often only one round.

- Optional post-processing step.

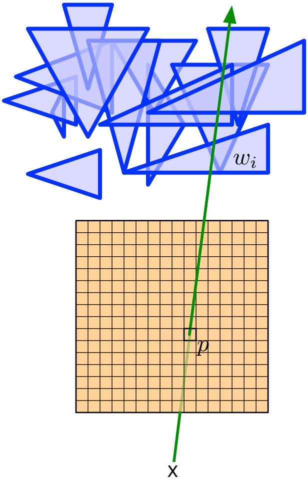


Pro: Robust (duplication) and simple. Can harness Locality
Con: Somewhat restrictive model

General Purpose GPU

Massive parallelism on your desktop.
Uses **G**raphics **P**rocessing **U**nit.
Designed for efficient video rasterizing.
Each *processor* corresponds to pixel p

- depth buffer:
  $D(p) = \min_i ||x - w_i||$
- color buffer: $C(p) = \sum_i \alpha_i \chi_i$
- ...

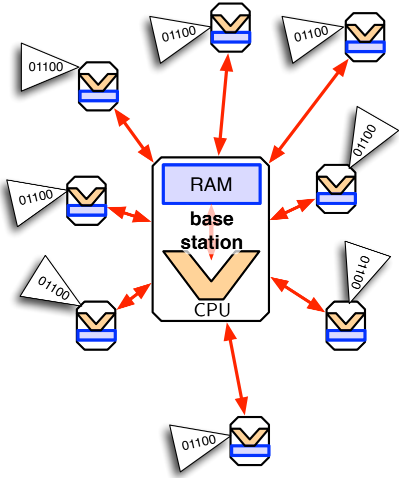


Pro: Fine grain, massive parallelism. Cheap. Harnesses Locality.
Con: Somewhat restrictive model, hierarchy. Small memory.

Distributed Computing

Many small slow processors with data.
Communication very expensive.

- Report to *base station*
- Merge tree
- Unorganized (peer-to-peer)

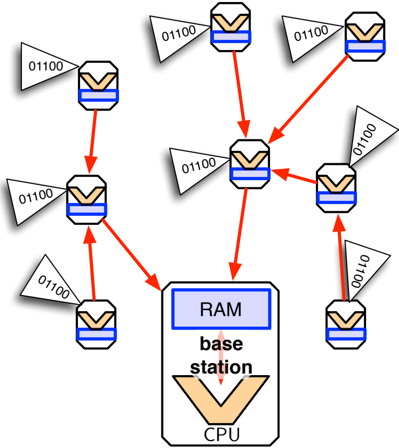


Data collection or Distribution

Distributed Computing

Many small slow processors with data.
Communication very expensive.

- Report to *base station*
- Merge tree
- Unorganized (peer-to-peer)

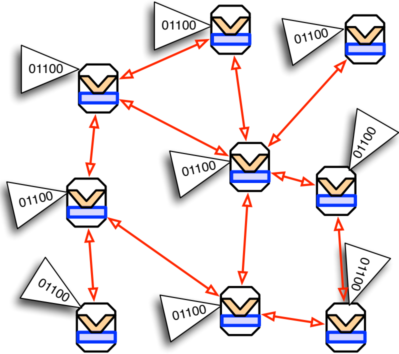


Data collection or Distribution

Distributed Computing

Many small slow processors with data.
Communication very expensive.

- Report to *base station*
- Merge tree
- Unorganized (peer-to-peer)

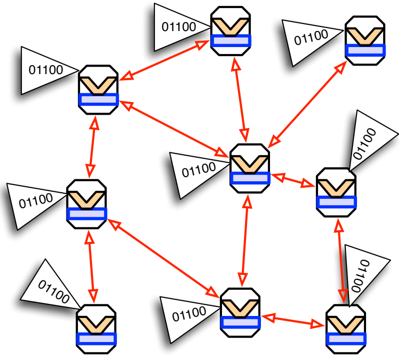


Data collection or Distribution

Distributed Computing

Many small slow processors with data.
Communication very expensive.

- Report to *base station*
- Merge tree
- Unorganized (peer-to-peer)



Data collection or Distribution

Advanced Algorithms: Approximate, Randomized

Themes

What are course goals?

- How to analyze algorithms in each model
- Taste of how to use each model
- When to use each model

Themes

What are course goals?

- How to analyze algorithms in each model
- Taste of how to use each model
- When to use each model

Work Plan:

- 1-3 weeks each model.
  - Background and Model.
  - Example algorithms analysis in each model.

I/O	Stream	Parallel	MapReduce	GPU	Distributed
4	5	4	4	3	3

Class Work

1 Credit Students:

- Attend Class. (some Fridays less important)
- Ask Questions.
- If above lacking, may have quizzes.
- Scribing Notes, Video-taping Lectures, or Giving Lectures.

3 Credit Students:
Must also do a project!

- Project Proposal (Aug 30).
  Approved or Rejected by Sept 4.
- Intermediate Report (Oct 23).
- Presentations (Dec 11 or 13).

Sequential Review

Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)

- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)

- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?

- Scanning (max):

Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)

- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?

- Scanning (max):
  TM: $O(n)$ VNA: $O(n)$

Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)

- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?

- Scanning (max):
  TM: $O(n)$ VNA: $O(n)$
- Sorting:

Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)

- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?

- Scanning (max):
  TM: $O(n)$ VNA: $O(n)$
- Sorting:
  TM: $O(n^2)$ VNA: $O(n \log n)$

Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)

- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?

- Scanning (max):
  TM: $O(n)$ VNA: $O(n)$
- Sorting:
  TM: $O(n^2)$ VNA: $O(n \log n)$
- Searching:

Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)

- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?

- Scanning (max):
  TM: $O(n)$ VNA: $O(n)$
- Sorting:
  TM: $O(n^2)$ VNA: $O(n \log n)$
- Searching:
  TM: $O(n)$ VNA: $O(\log n)$

Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max (n)
- Merge-Sort ($n \log n$)
- Bubble-Sort (n^2) ... or Dynamic Programming

$n =$	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	1
Search	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.00

Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max (n)
- Merge-Sort ($n \log n$)
- Bubble-Sort (n^2) ... or Dynamic Programming

$n =$	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	1
Search	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.00
Max	0.000003	0.000005	0.000006	0.000048	0.000387	0.003988	0.040698	9.193987	$>$15

Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max (n)
- Merge-Sort ($n \log n$)
- Bubble-Sort (n^2) ... or Dynamic Programming

$n =$	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	1
Search	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.00
Max	0.000003	0.000005	0.000006	0.000048	0.000387	0.003988	0.040698	9.193987	$>$15
Merge	0.000005	0.000030	0.000200	0.002698	0.029566	0.484016	7.833908	137.9388	

Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max (n)
- Merge-Sort ($n \log n$)
- Bubble-Sort (n^2) ... or Dynamic Programming

$n =$	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	1
Search	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.00
Max	0.000003	0.000005	0.000006	0.000048	0.000387	0.003988	0.040698	9.193987	$>$15
Merge	0.000005	0.000030	0.000200	0.002698	0.029566	0.484016	7.833908	137.9388	
Bubble	0.000003	0.000105	0.007848	0.812912	83.12960	$\sim$2 hour	$\sim$9 days	-	

Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max (n)
- Merge-Sort ($n \log n$)
- Bubble-Sort (n^2) ... or Dynamic Programming

$n =$	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	1
Search	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.00
Max	0.000003	0.000005	0.000006	0.000048	0.000387	0.003988	0.040698	9.193987	>15
Merge	0.000005	0.000030	0.000200	0.002698	0.029566	0.484016	7.833908	137.9388	
Bubble	0.000003	0.000105	0.007848	0.812912	83.12960	∼2 hour	∼9 days	-	

Complexity Theory:

- LOG: $\text{poly}\log(n) = \log^c n$ (... need to load data)

Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max (n)
- Merge-Sort ($n \log n$)
- Bubble-Sort (n^2) ... or Dynamic Programming

$n =$	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	1
Search	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.00
Max	0.000003	0.000005	0.000006	0.000048	0.000387	0.003988	0.040698	9.193987	$>$15
Merge	0.000005	0.000030	0.000200	0.002698	0.029566	0.484016	7.833908	137.9388	
Bubble	0.000003	0.000105	0.007848	0.812912	83.12960	$\sim$2 hour	$\sim$9 days	-	

Complexity Theory:

- LOG: $\text{poly}\log(n) = \log^c n$ (... need to load data)
- P : $\text{poly}(n) = n^c$ (many cool algorithms)

Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max (n)
- Merge-Sort ($n \log n$)
- Bubble-Sort (n^2) ... or Dynamic Programming

$n =$	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	1
Search	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.00
Max	0.000003	0.000005	0.000006	0.000048	0.000387	0.003988	0.040698	9.193987	$>$15
Merge	0.000005	0.000030	0.000200	0.002698	0.029566	0.484016	7.833908	137.9388	
Bubble	0.000003	0.000105	0.007848	0.812912	83.12960	$\sim$2 hour	$\sim$9 days	-	

Complexity Theory:

- LOG: $\operatorname{poly}\log(n) = \log^c n$ (... need to load data)
- P : $\operatorname{poly}(n) = n^c$ (many cool algorithms)
- EXP: $\exp(n) = c^n$ (usually hopeless ... but 0.00001^n not bad)

Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max (n)
- Merge-Sort ($n \log n$)
- Bubble-Sort (n^2) ... or Dynamic Programming

$n =$	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	1
Search	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.00
Max	0.000003	0.000005	0.000006	0.000048	0.000387	0.003988	0.040698	9.193987	$>$15
Merge	0.000005	0.000030	0.000200	0.002698	0.029566	0.484016	7.833908	137.9388	
Bubble	0.000003	0.000105	0.007848	0.812912	83.12960	$\sim$2 hour	$\sim$9 days	-	

Complexity Theory:

- LOG: $\text{poly}\log(n) = \log^c n$ (... need to load data)
- P : $\text{poly}(n) = n^c$ (many cool algorithms)
- EXP: $\exp(n) = c^n$ (usually hopeless ... but 0.00001^n not bad)
- NP: verify solution in P, find solution conjectured EXP
  (If EXP number parallel machines, then in P time)

Data Group

Data Group Meeting

Thursdays @ 12:15-1:30pm in LCR

(to be confirmed)

http://datagroup.cs.utah.edu/dbgroup.php