I/O-Efficient Algorithms and Data Structures

Steffan Christ Sølvsten

8th of September, 2023

AARHUS
UNIVERSITY

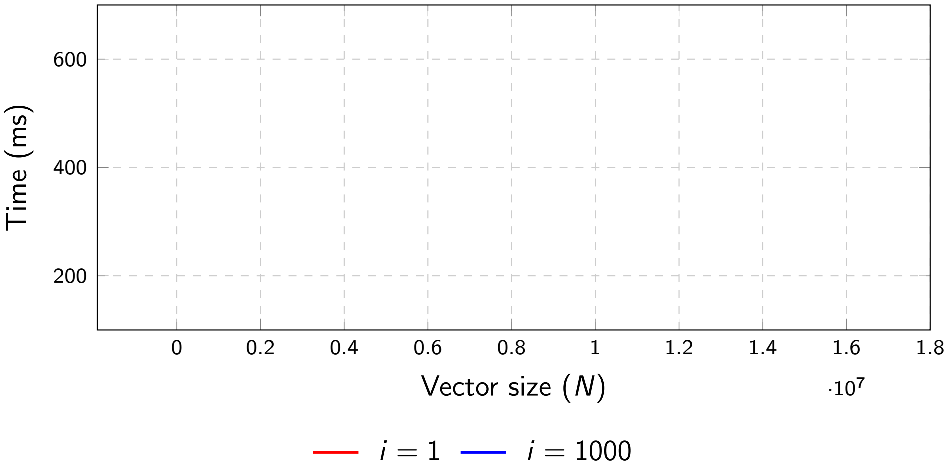
$$0,\ 1,\ 2,\ \cdots\quad i,\ i+1,\ \cdots\ 2i,\ 2i+1,\ \cdots\cdots\quad N-1$$

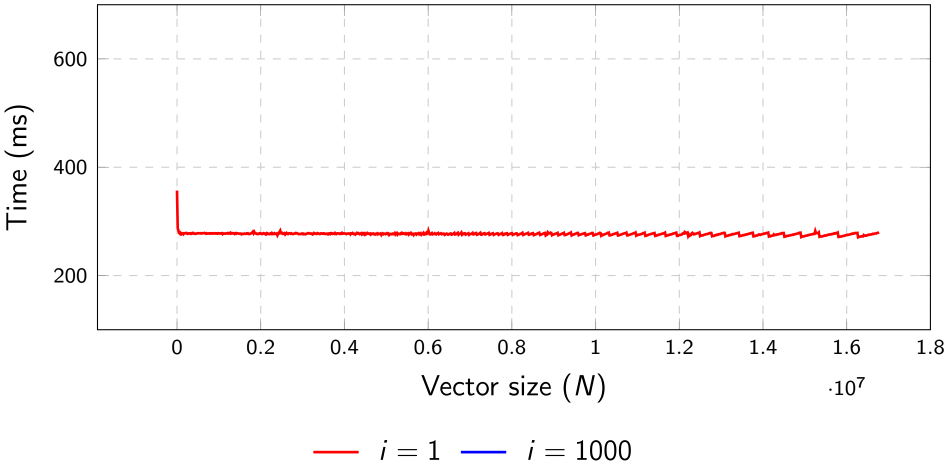
$$0,\ 1,\ 2,\ \cdots\ \ i,\ i+1,\ \cdots\ 2i,\ 2i+1,\ \cdots\cdots\ \ N-1$$

$$0,\ 1,\ 2,\ \cdots\quad i,\ i+1,\ \cdots\ 2i,\ 2i+1,\ \cdots\cdots\quad N-1$$

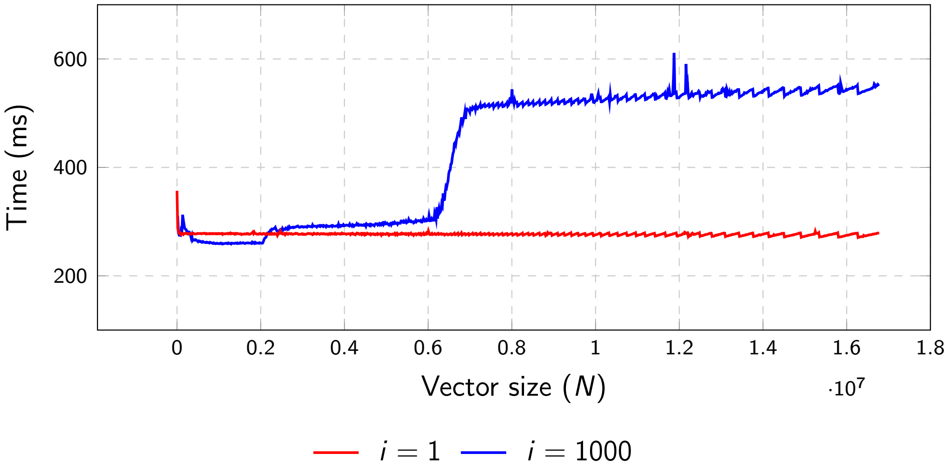
$$0,\ 1,\ 2,\ \cdots\quad i,\ i+1,\ \cdots\quad 2i,\ 2i+1,\ \cdots\cdots\quad N-1$$

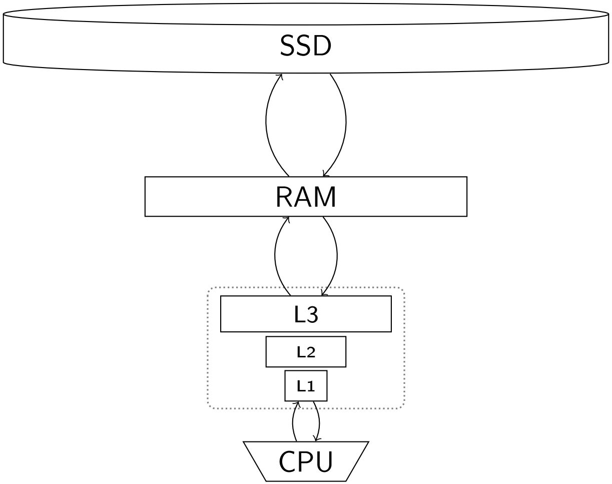
$$0,\ 1,\ 2,\ \cdots\ i,\ i+1,\ \cdots\ 2i,\ 2i+1,\ \cdots\cdots\ N-1$$

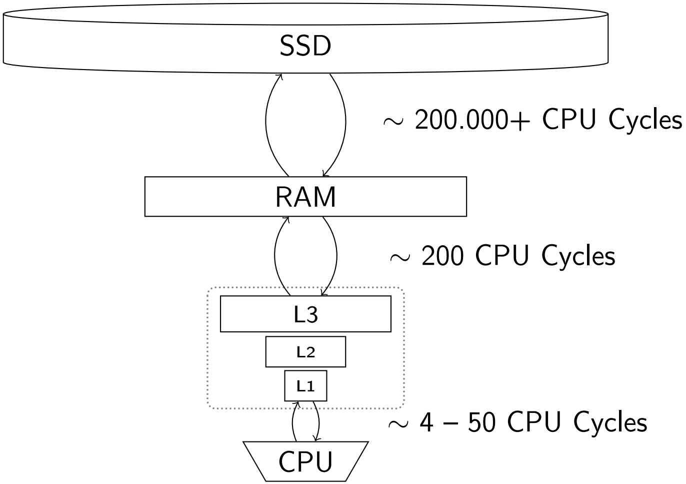

600
400
200
Time (ms)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
Vector size (N)
$\cdot 10^7$
$i = 1$
$i = 1000$

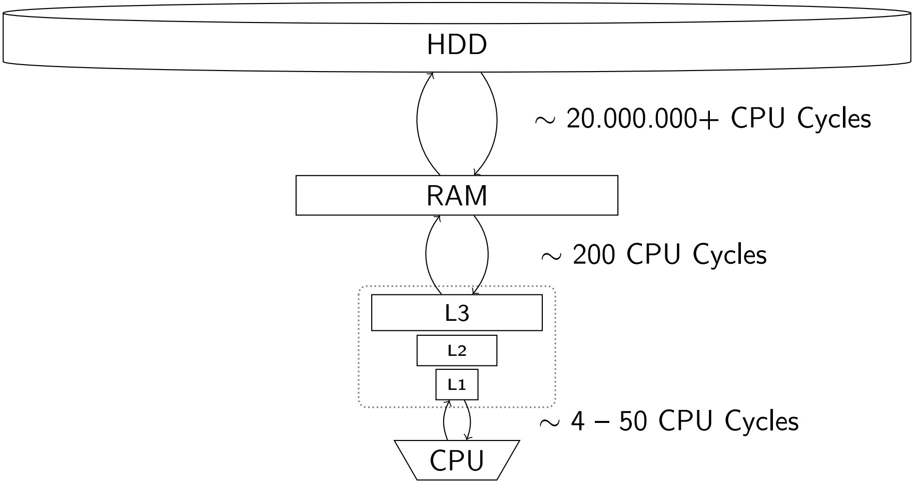


600
400
200
Time (ms)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
Vector size (N)
·10^7
i = 1
i = 1000

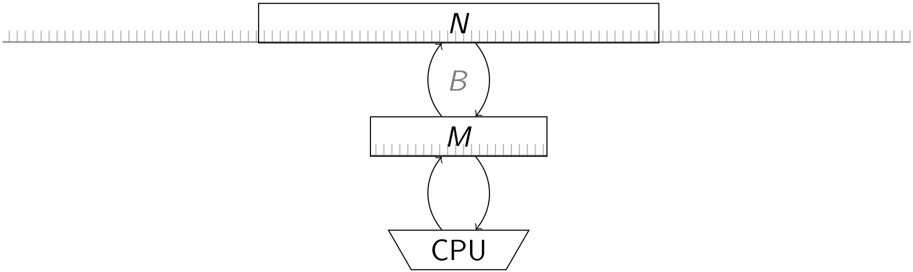

600
400
200
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
Time (ms)
Vector size (N)
$\cdot 10^7$
$i = 1$
$i = 1000$


SSD
RAM
L3
L2
L1
CPU

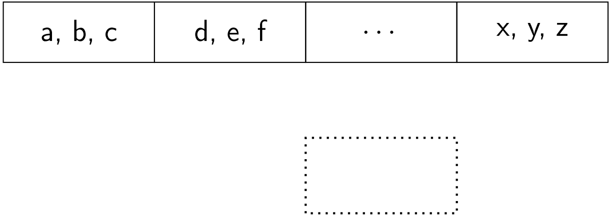

SSD
~ 200.000+ CPU Cycles
RAM
~ 200 CPU Cycles
L3
L2
L1
~ 4 – 50 CPU Cycles
CPU


HDD
~ 20.000.000+ CPU Cycles
RAM
~ 200 CPU Cycles
L3
L2
L1
~ 4 – 50 CPU Cycles
CPU

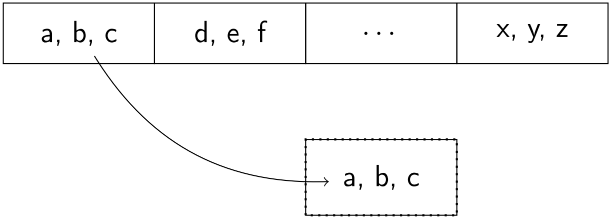
I/O Model
Aggarwal and Vitter '87



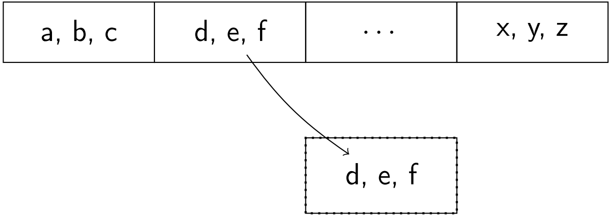
I/O Model : Sequential Access
Aggarwal and Vitter ’87



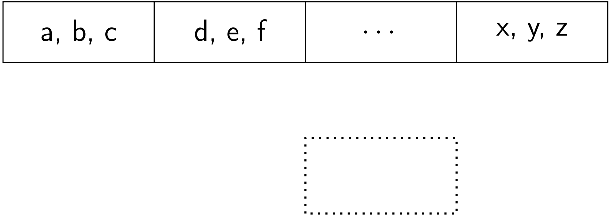
I/O Model : Sequential Access
Aggarwal and Vitter '87



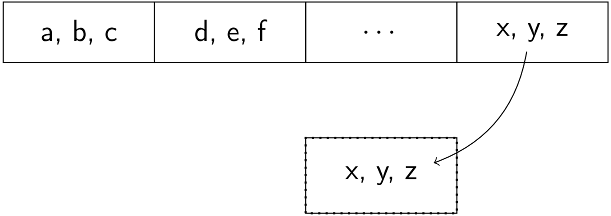
I/O Model : Sequential Access
Aggarwal and Vitter '87



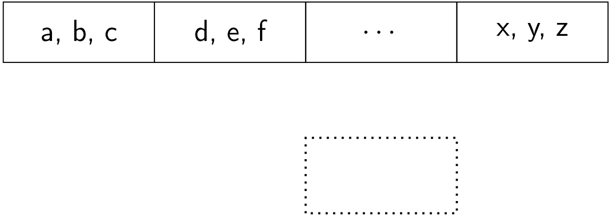
I/O Model : Sequential Access
Aggarwal and Vitter '87



I/O Model : Sequential Access
Aggarwal and Vitter '87



I/O Model : Sequential Access
Aggarwal and Vitter '87



I/O Model : Sequential Access
Aggarwal and Vitter '87

a, b, c	d, e, f	. . .	x, y, z

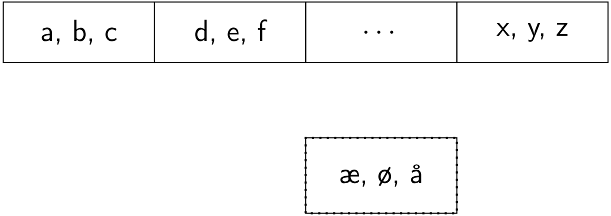
æ

I/O Model : Sequential Access
Aggarwal and Vitter '87

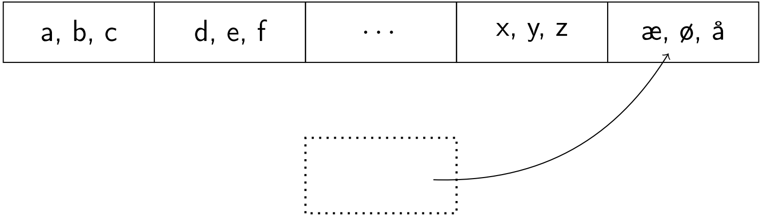
a, b, c	d, e, f	$\cdots$	x, y, z

æ, ø

I/O Model : Sequential Access
Aggarwal and Vitter ’87



I/O Model : Sequential Access
Aggarwal and Vitter '87

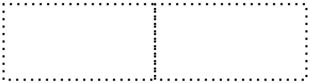


I/O Model : Sequential Access
Aggarwal and Vitter '87

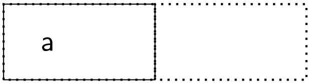
a, b, c	d, e, f	$\cdots$	x, y, z	æ, ø, å

Time	:	N
I/O	:	N/B
Memory	:	B

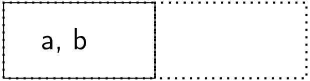
I/O Model : Stack



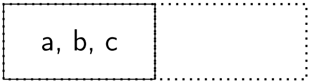
I/O Model : Stack



I/O Model : Stack



I/O Model : Stack



I/O Model : Stack

a, b, c	d

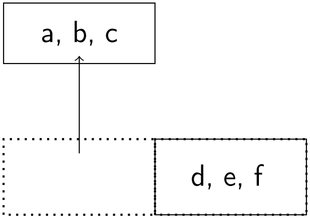
I/O Model : Stack

a, b, c	d, e

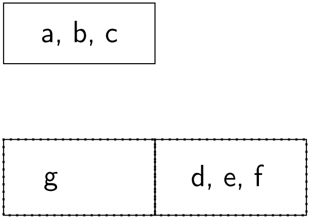
I/O Model : Stack

a, b, c	d, e, f

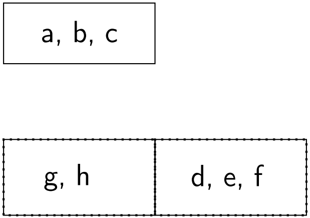
I/O Model : Stack



I/O Model : Stack



I/O Model : Stack

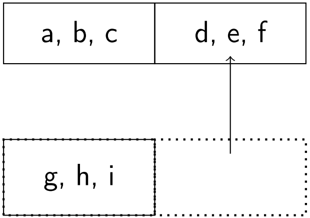


I/O Model : Stack

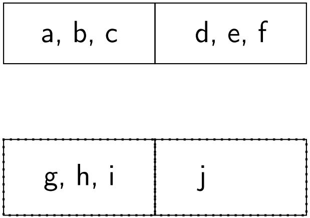
a, b, c

g, h, i | d, e, f

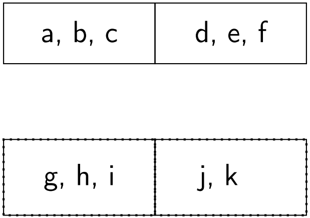
I/O Model : Stack



I/O Model : Stack



I/O Model : Stack

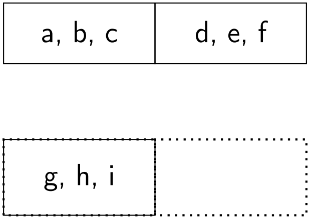


I/O Model : Stack

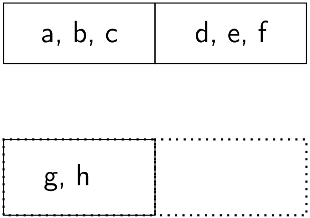
a, b, c	d, e, f

g, h, i	j

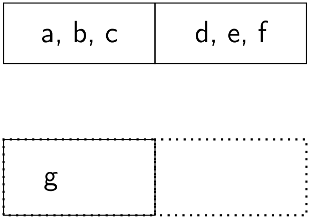
I/O Model : Stack



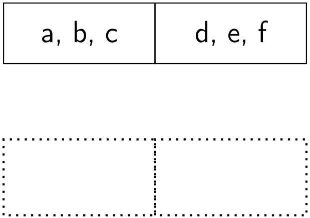
I/O Model : Stack



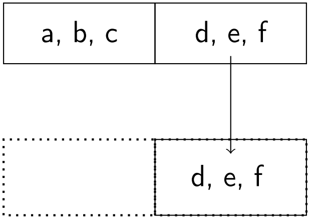
I/O Model : Stack



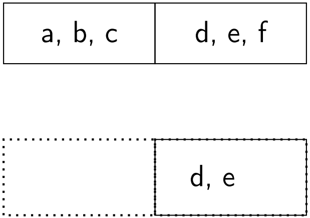
I/O Model : Stack



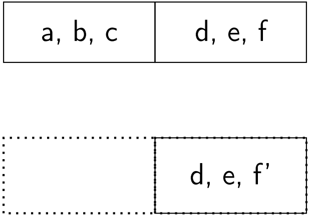
I/O Model : Stack



I/O Model : Stack



I/O Model : Stack



I/O Model : Stack

a, b, c	d, e, f
g'	d, e, f'

I/O Model : Stack

a, b, c	d, e, f

g', h'	d, e, f'

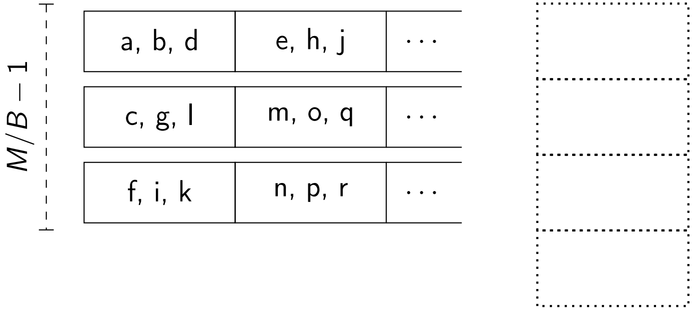
I/O Model : Stack

a, b, c	d, e, f

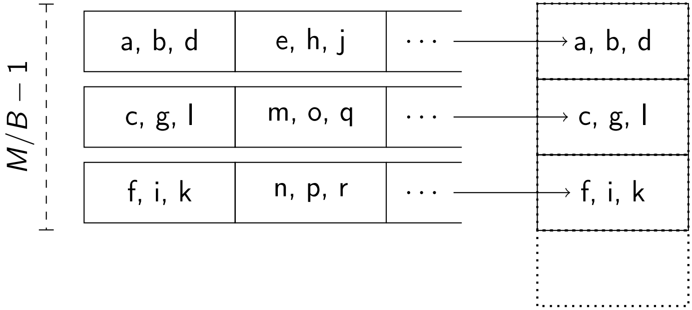
g', h'	d, e, f'

Time	:	$O(N)$
I/O	:	$O(N/B)$
Memory	:	$2B$

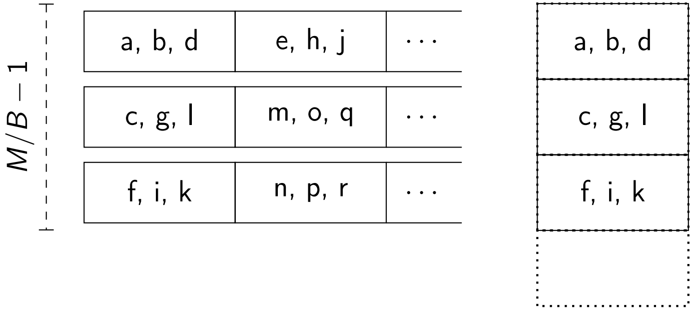
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



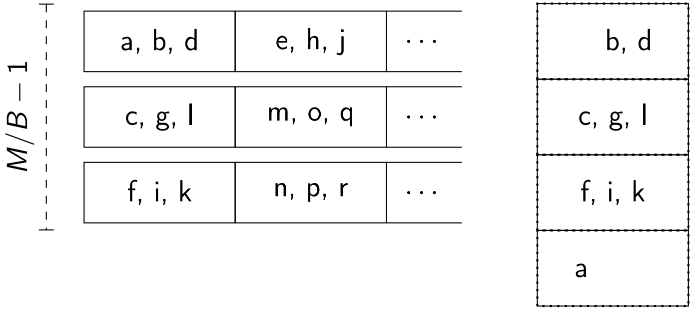
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



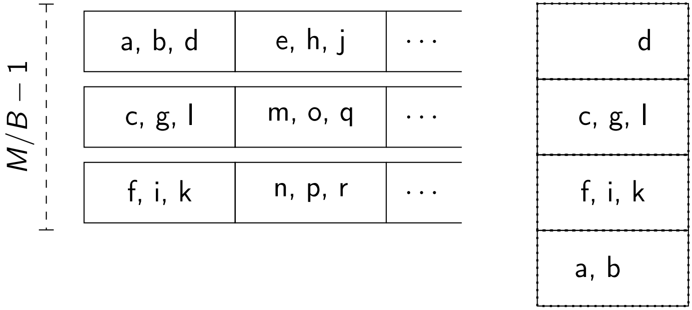
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



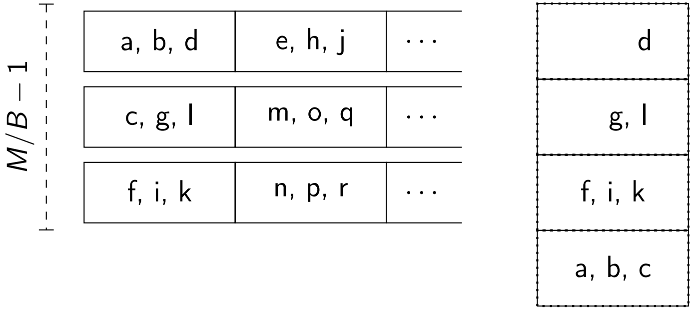
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



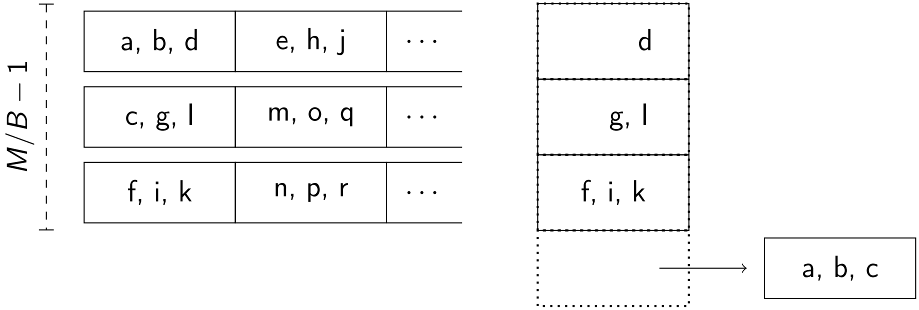
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



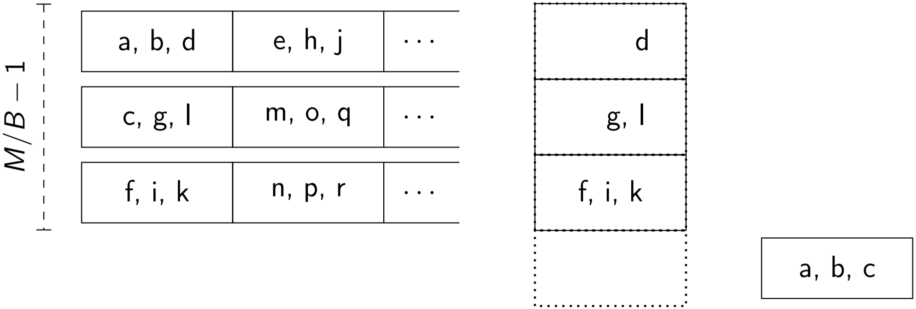
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



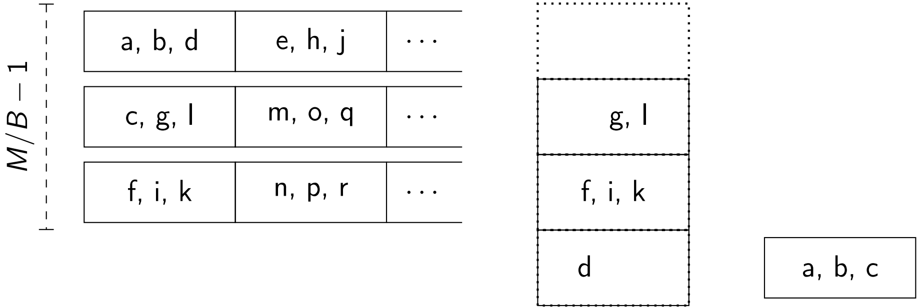
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



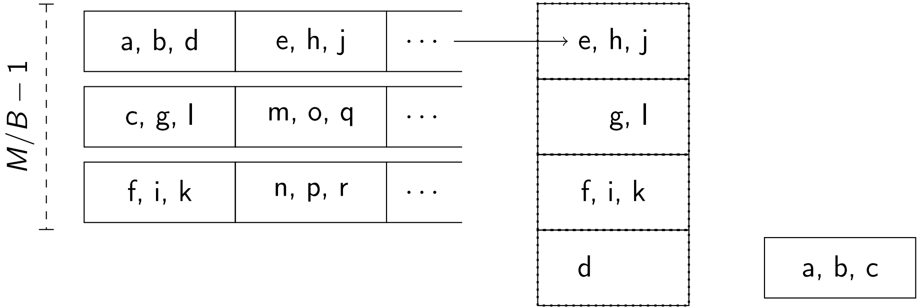
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



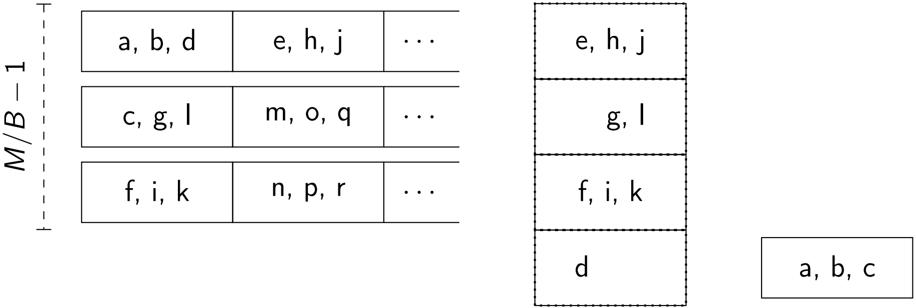
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



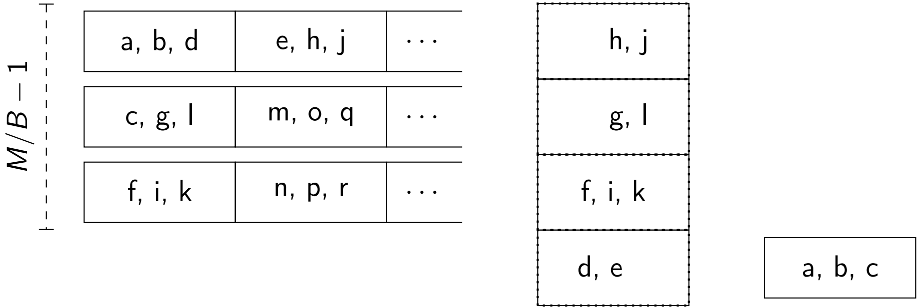
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



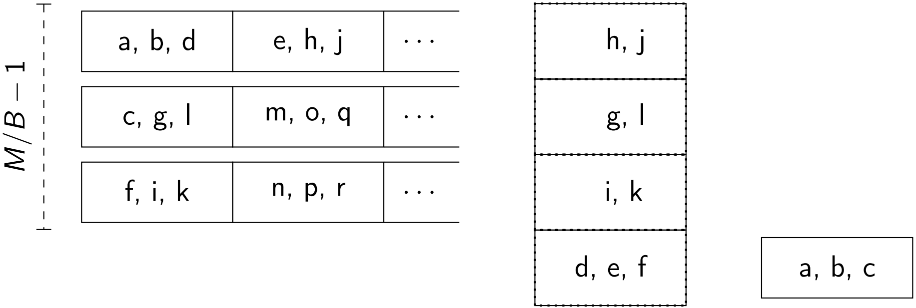
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



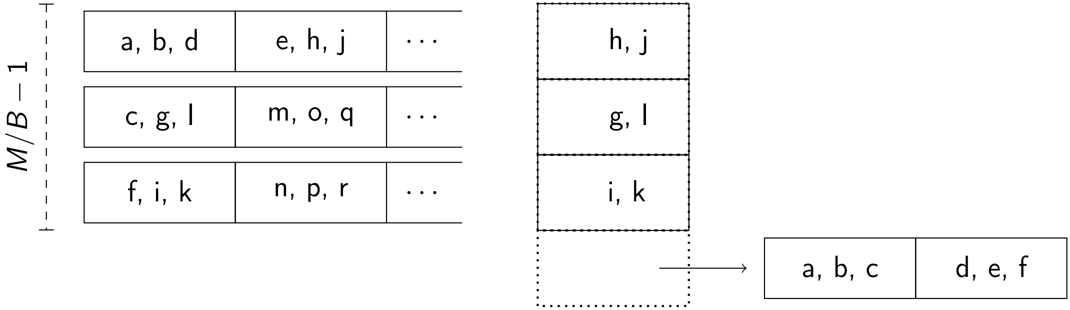
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87

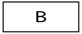


I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87

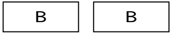


I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87

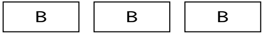
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



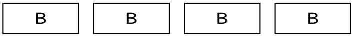
I/O Model : M/B-way Mergesort
Aggarwal and Vitter ’87



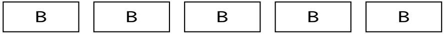
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



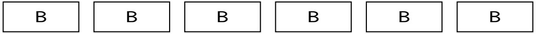
I/O Model : M/B-way Mergesort
Aggarwal and Vitter ’87



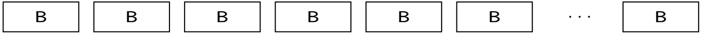
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



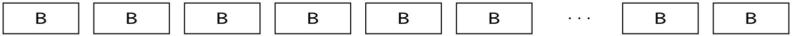
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



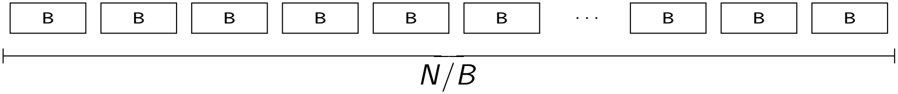
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



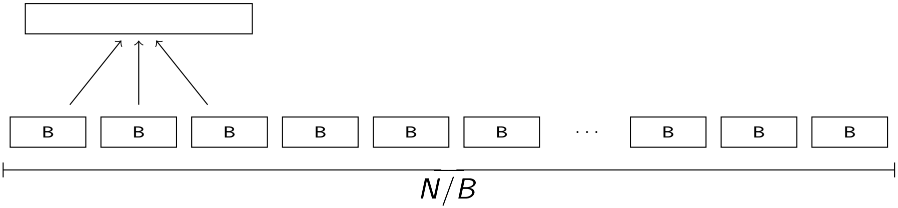
I/O Model : M/B-way Mergesort
Aggarwal and Vitter ’87



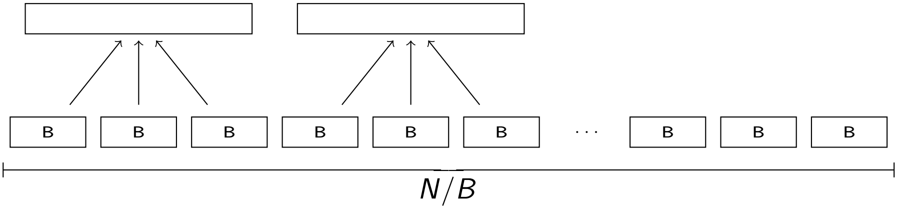
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



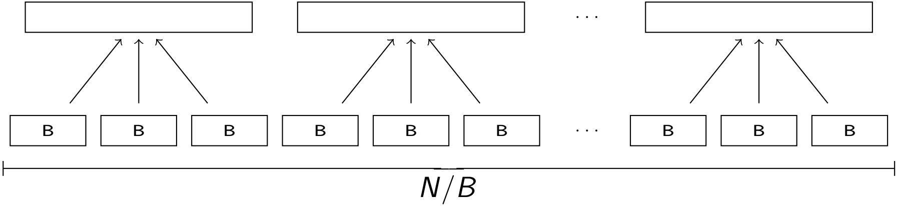
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



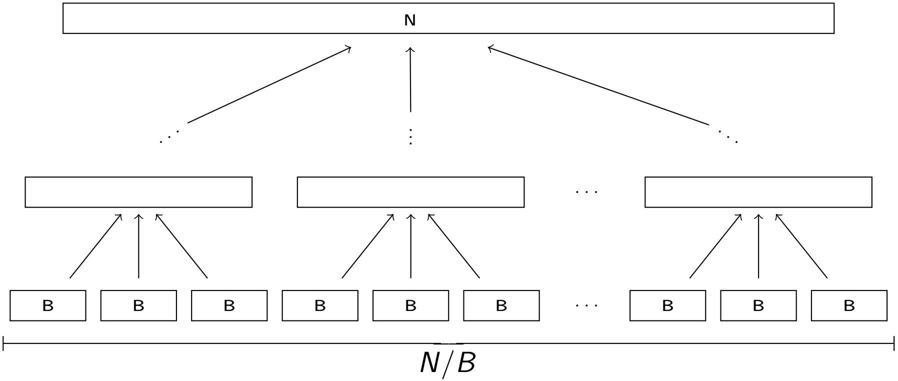
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



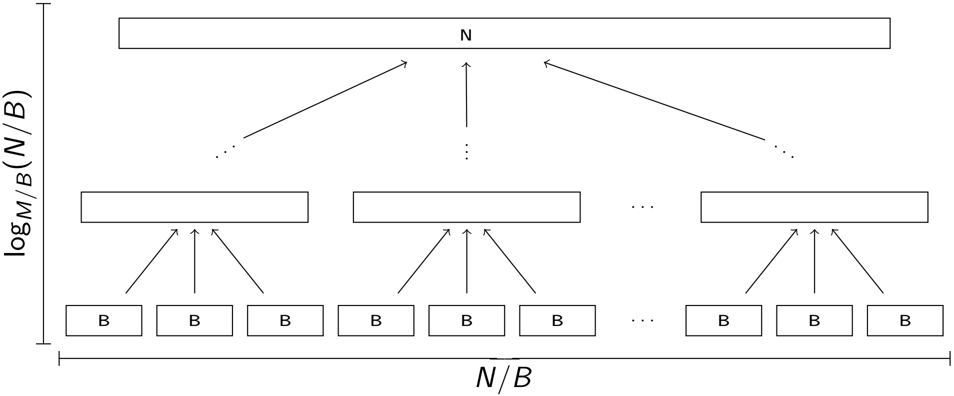
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87



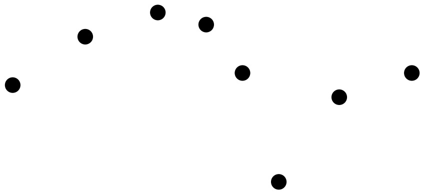
I/O Model : M/B-way Mergesort
Aggarwal and Vitter '87

Theorem
N elements can be sorted in $\Theta(N/B \cdot \log_{M/B}(N/B))$ I/Os.

Convex Hull : Graham Scan
Graham '72

Convex Hull

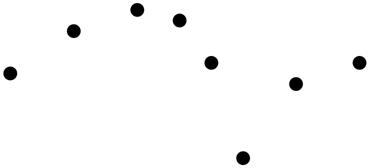
Compute the *convex hull* for N points in the plane.



Theorem
Convex Hull can be computed in $O(N/B \cdot \log_{M/B}(N/B))$ *I/Os.*

Convex Hull : Graham Scan
Graham '72

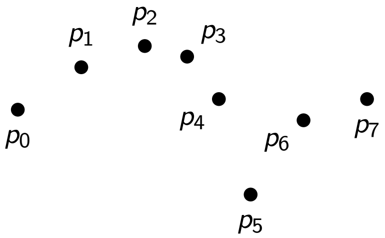
Upper Hull:



Convex Hull : Graham Scan
Graham '72

Upper Hull:

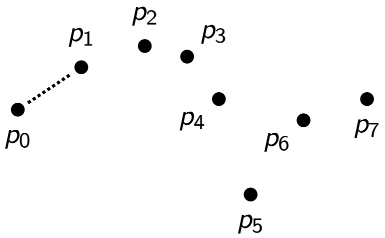
- Sort input points by x-axis



Convex Hull : Graham Scan
Graham '72

Upper Hull:

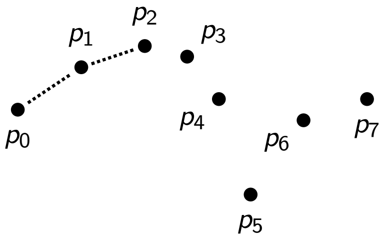
- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$



Convex Hull : Graham Scan
Graham '72

Upper Hull:

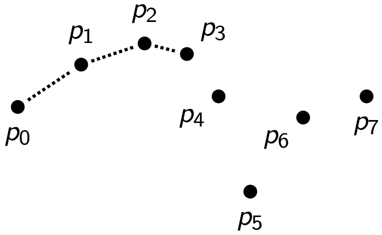
- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S



Convex Hull : Graham Scan
Graham '72

Upper Hull:

- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S

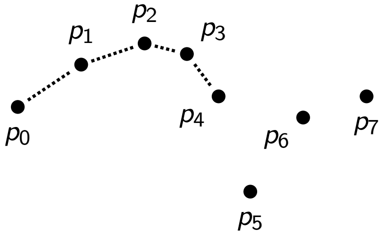


Convex Hull : Graham Scan
Graham '72

Upper Hull:

- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:

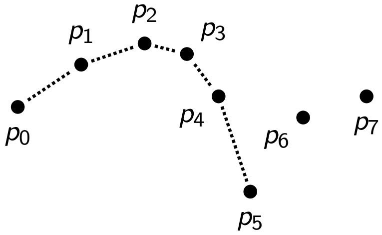
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S



Convex Hull : Graham Scan
Graham '72

Upper Hull:

- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S

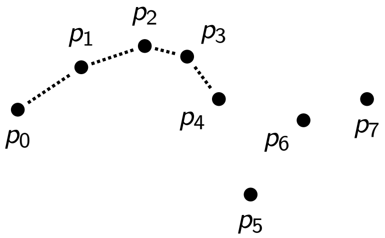


Convex Hull : Graham Scan
Graham '72

Upper Hull:

- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:

  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S

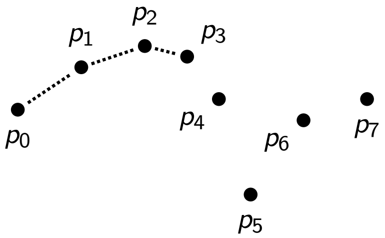


Convex Hull : Graham Scan
Graham '72

Upper Hull:

- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:

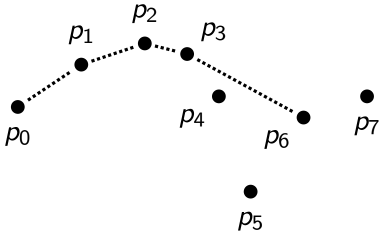
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S



Convex Hull : Graham Scan
Graham '72

Upper Hull:

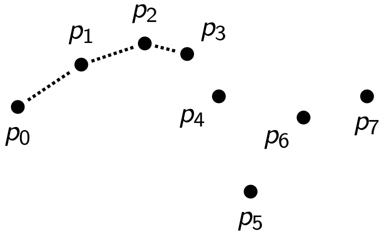
- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S



Convex Hull : Graham Scan
Graham '72

Upper Hull:

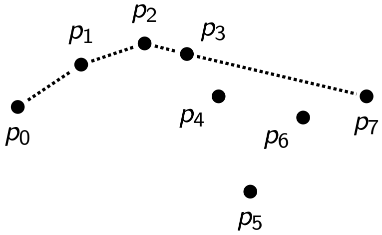
- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S



Convex Hull : Graham Scan
Graham '72

Upper Hull:

- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S



Convex Hull : Graham Scan
Graham '72

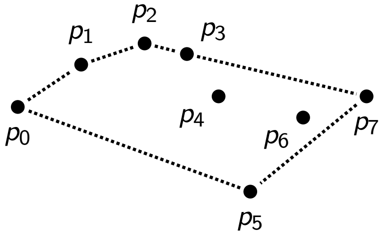
Upper Hull:

- Sort input points by x-axis
- Initialize stack $S = [p_0, p_1]$
- For remaining points $p_i \in p_2, p_3, \ldots, p_{N-1}$:

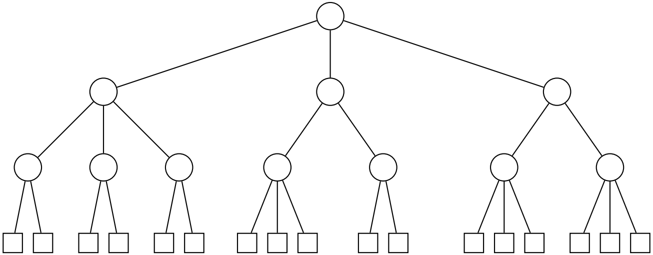
  1. Let p_s, p_t be the two top-most points of S
  2. While $p_s - p_t - p_i$ is a "left-turn":
     - Pop p_t and go-to 1
  3. Push p_i onto S

Lower Hull:

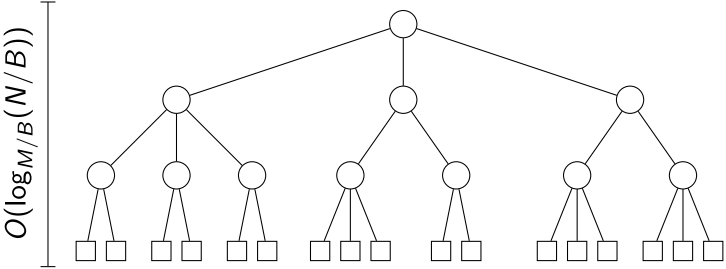
- Symmetric...



a–b Tree
Huddleston and Mehlhorn '82

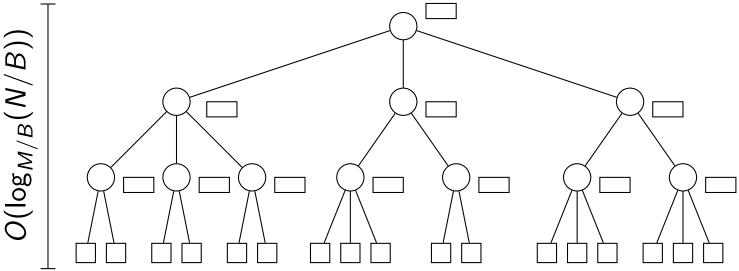


Buffer Tree
Arge '95



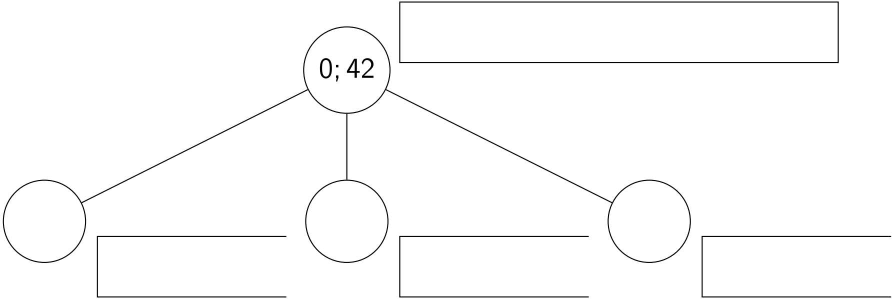
$$a = \frac{1}{4}M/B, \qquad b = M/B, \qquad \text{Leaf Size} = B$$

Buffer Tree
Arge '95

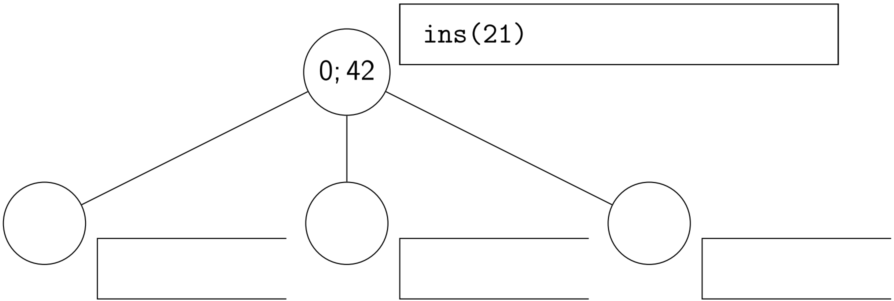


$$a = \frac{1}{4}M/B, \qquad b = M/B, \qquad \text{Leaf Size} = B$$

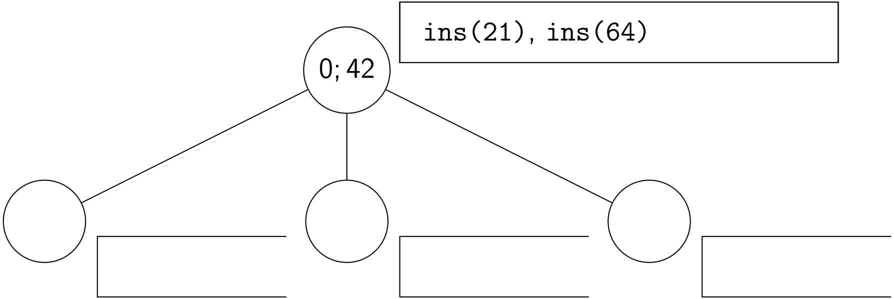
Buffer Tree
Arge '95



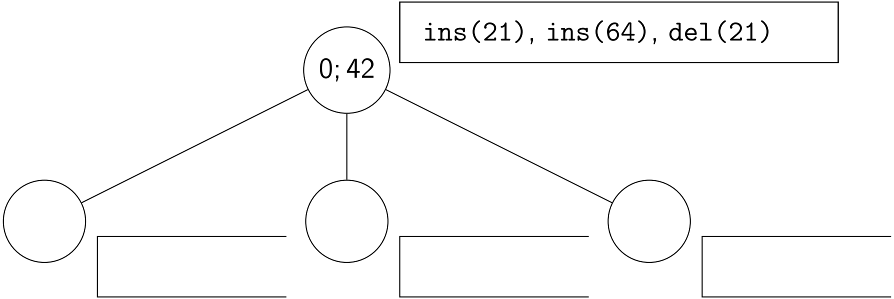
Buffer Tree
Arge '95



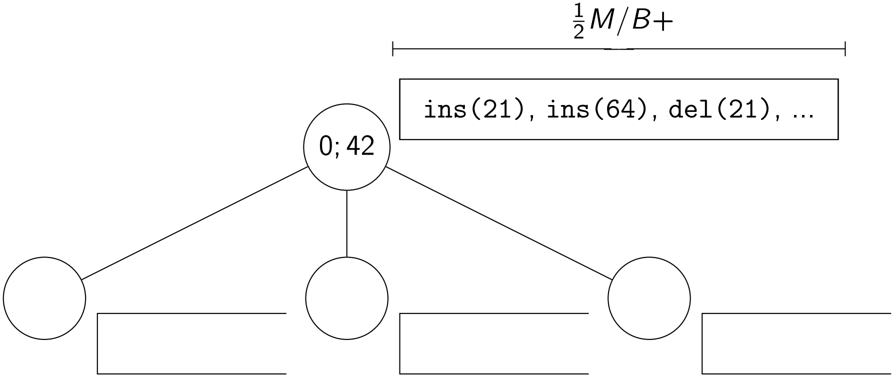
Buffer Tree
Arge '95



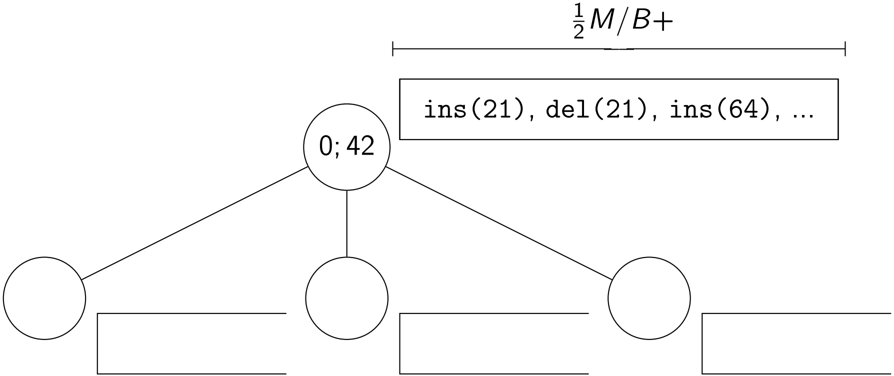
Buffer Tree
Arge '95



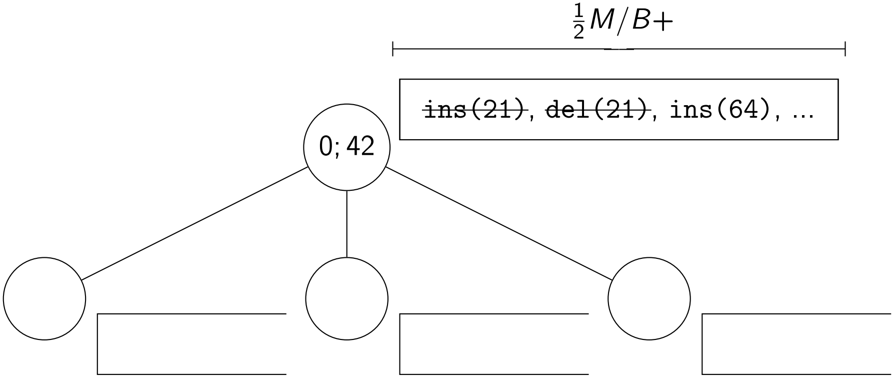
Buffer Tree
Arge '95



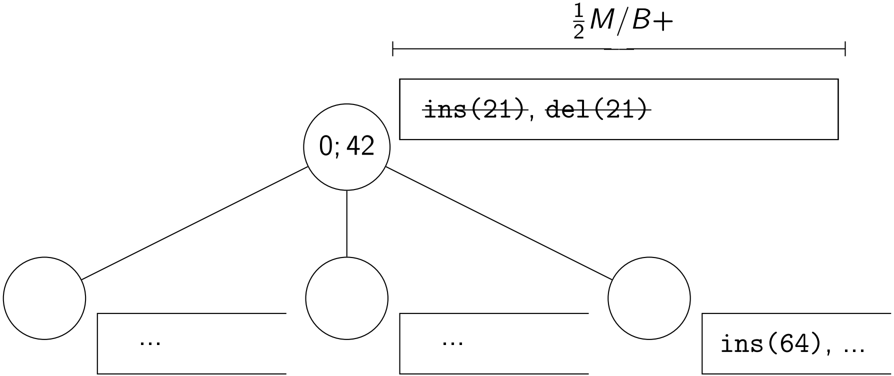
Buffer Tree
Arge '95



Buffer Tree
Arge '95



Buffer Tree
Arge '95



Buffer Tree
Arge '95

Theorem
A Buffer Tree can resolve N `inserts` *and* `deletes` *in $\Theta(N/B \cdot \log_{M/B}(N/B))$ I/Os.*

Buffer Tree
Arge '95

Theorem
A Buffer Tree can resolve N $\mathtt{inserts}$ and $\mathtt{deletes}$ in $\Theta(N/B \cdot \log_{M/B}(N/B))$ I/Os.

Theorem
A Buffer Tree with N requests can empty all its buffers, and output all remaining sorted elements, in $\Theta(N/B)$ I/Os.

Buffer Tree
Arge '95

Theorem
A Buffer Tree can resolve N $\mathtt{inserts}$ *and* $\mathtt{deletes}$ *in* $\Theta(N/B \cdot \log_{M/B}(N/B))$ *I/Os.*

Theorem
A Buffer Tree with N requests can empty all its buffers, and output all remaining sorted elements, in $\Theta(N/B)$ *I/Os.*

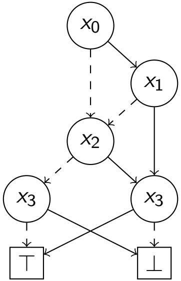
Corollary
An I/O-efficient Priority Queue can resolve N $\mathtt{push}$ *and* $\mathtt{deletemin}$ *operations in* $\Theta(N/B \cdot \log_{M/B}(N/B))$ *I/Os.*

Proof.
Use an $M/2$ sized internal memory priority queue, `pq`. If `pq` overflows, move $M/4$ the largest elements to a Buffer Tree, `t`. If `pq` underflows, obtain the $M/4$ smallest elements from `t`. □

Binary Decision Diagrams
Arge '96, Sølvsten '22

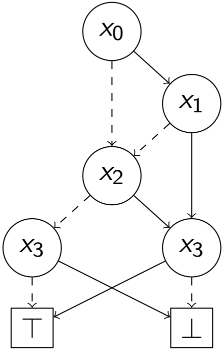
#Paths
Given a Binary Decision Diagram of N nodes, compute the number of paths from the root to the $\top$ terminal.

Theorem
#Paths can be computed in $O(N/B \cdot \log_{M/B}(N/B))$ I/Os.



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Binary Decision Diagrams
Arge '96, Sølvsten '22



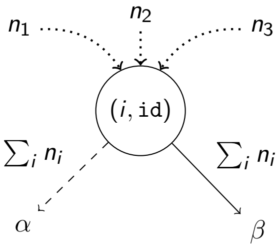
Decision Diagram

$$
\begin{array}{lcl}
[& ((0,0),(2,0),(1,0)) & , \\
 & ((1,0),(2,0),(3,1)) & , \\
 & ((2,0),(3,0),(3,1)) & , \\
 & ((3,0),\top,\bot) & , \\
 & ((3,1),\bot,\top) &]
\end{array}
$$

On-Disk Format

(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

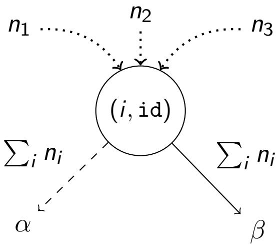
Binary Decision Diagrams
Arge '96, Sølvsten '22



Idea

Count the number of in-going paths to each node.

Binary Decision Diagrams
Arge '96, Sølvsten '22

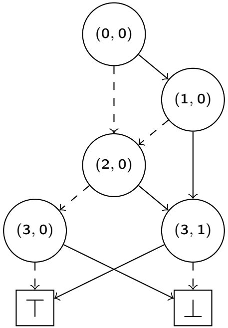


Time-Forward Processing

Defer work with Q_{count} : $\texttt{PriorityQueue}\langle(s \to t, \quad \mathbb{N})\rangle$ sorted on t in ascending order.

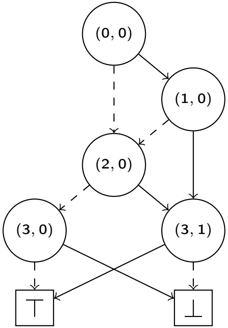
$$((i, \texttt{id}) \xrightarrow{\perp} \alpha, \quad \textstyle\sum_i n_i), \qquad ((i, \texttt{id}) \xrightarrow{\top} \beta, \quad \textstyle\sum_i n_i)$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

Binary Decision Diagrams
Arge '96, Sølvsten '22



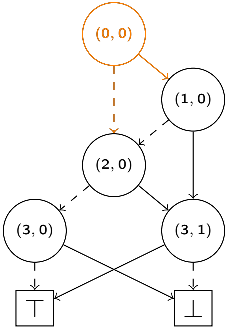
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Priority Queue: Q_{count}:

[

]

Binary Decision Diagrams
Arge '96, Sølvsten '22



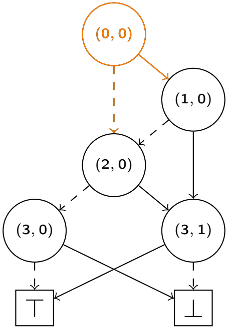
(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

Priority Queue: Q_{count}:

[

]

Binary Decision Diagrams
Arge '96, Sølvsten '22

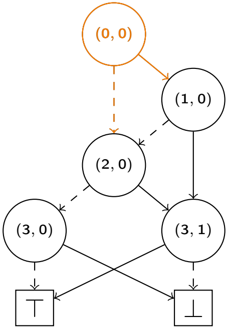


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Priority Queue: Q_{count}:

$$\begin{array}{lll} [& ((0,0) \xrightarrow{\top} (1,0), \quad 1) & , \\ & ((0,0) \xrightarrow{\bot} (2,0), \quad 1) & , \\ & &] \end{array}$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



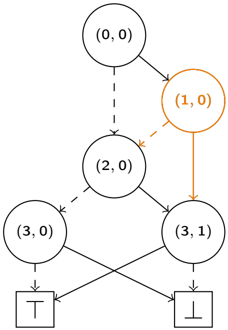
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	0	0

Priority Queue: Q_{count}:

$$[\quad ((0,0) \xrightarrow{\top} (1,0),\quad 1)\quad ,$$
$$((0,0) \xrightarrow{\perp} (2,0),\quad 1)\quad ,$$
$$]$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



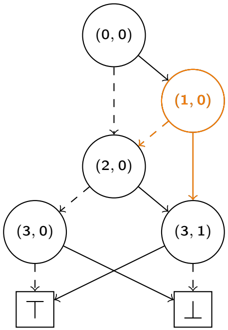
(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

Seek	Sum	Result
$(1,0)$	0	0

Priority Queue: Q_{count}:

$$[\ ((0,0) \xrightarrow{\top} (1,0),\ 1)\ ,$$
$$((0,0) \xrightarrow{\perp} (2,0),\ 1)\ ,\ ]$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	1	0

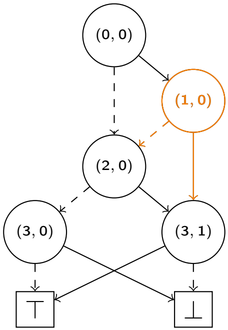
Priority Queue: Q_{count}:

[

$((0,0) \xrightarrow{\perp} (2,0), \quad 1)$,

]

Binary Decision Diagrams
Arge '96, Sølvsten '22



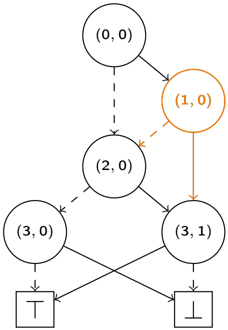
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(1, 0)	1	0

Priority Queue: Q_{count}:

$$[\; ((0,0) \xrightarrow{\perp} (2,0),\; 1),\; ((1,0) \xrightarrow{\perp} (2,0),\; 1),\; ((1,0) \xrightarrow{\top} (3,1),\; 1),\;]$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



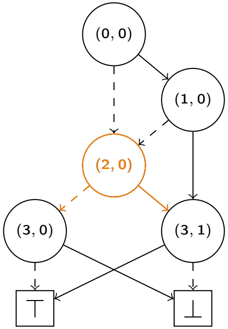
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(2, 0)$	0	0

Priority Queue: Q_{count}:

$$[\; ((0,0) \xrightarrow{\perp} (2,0),\ 1),\; ((1,0) \xrightarrow{\perp} (2,0),\ 1),\; ((1,0) \xrightarrow{\top} (3,1),\ 1),\;]$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

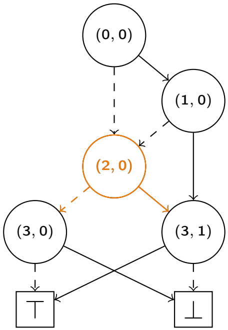
Seek	Sum	Result
$(2, 0)$	0	0

Priority Queue: Q_{count}:

[
$((0,0) \xrightarrow{\perp} (2,0),\quad 1)$,
$((1,0) \xrightarrow{\perp} (2,0),\quad 1)$,

$((1,0) \xrightarrow{\top} (3,1),\quad 1)$,
]

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

Seek	Sum	Result
$(2, 0)$	1	0

Priority Queue: Q_{count}:

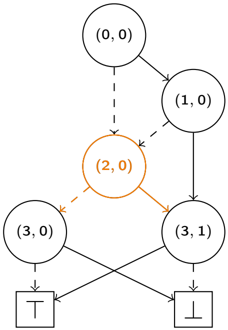
[

$((1, 0) \xrightarrow{\perp} (2, 0), \quad 1)$,

$((1, 0) \xrightarrow{\top} (3, 1), \quad 1)$,

]

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(2, 0)$	2	0

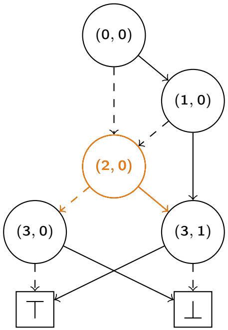
Priority Queue: Q_{count}:

[

$((1,0) \xrightarrow{\top} (3,1), \quad 1)$,

]

Binary Decision Diagrams
Arge '96, Sølvsten '22



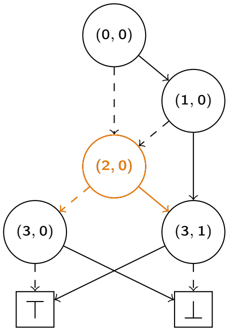
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(2, 0)$	2	0

Priority Queue: Q_{count}:

$$[\; ((2,0) \xrightarrow{\perp} (3,0),\ 2),\ ((1,0) \xrightarrow{\top} (3,1),\ 1),\ ((2,0) \xrightarrow{\top} (3,1),\ 2) \;]$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



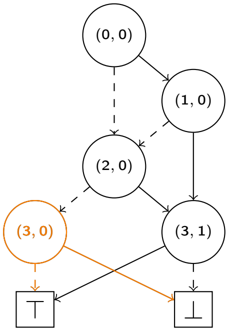
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(3, 0)$	0	0

Priority Queue: Q_{count}:

$$[\; ((2,0) \xrightarrow{\perp} (3,0),\quad 2),\; ((1,0) \xrightarrow{\top} (3,1),\quad 1),\; ((2,0) \xrightarrow{\top} (3,1),\quad 2) \;]$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



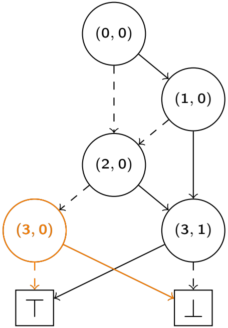
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(3, 0)$	0	0

Priority Queue: Q_{count}:

$$[\ ((2,0) \xrightarrow{\bot} (3,0),\quad 2)\ ,$$
$$((1,0) \xrightarrow{\top} (3,1),\quad 1)\ ,$$
$$((2,0) \xrightarrow{\top} (3,1),\quad 2)\ ]$$

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 0)	2	0

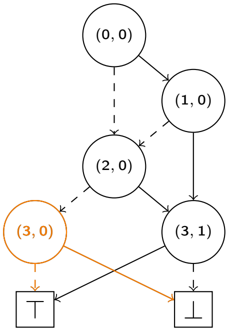
Priority Queue: Q_{count}:

[

$((1,0) \xrightarrow{\top} (3,1),\quad 1)$,

$((2,0) \xrightarrow{\top} (3,1),\quad 2)$]

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(3, 0)$	2	2

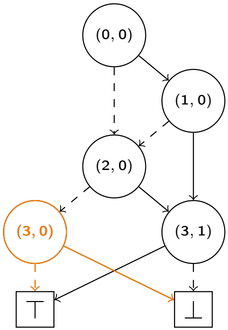
Priority Queue: Q_{count}:

[

$((1,0) \xrightarrow{\top} (3,1), \quad 1)$,

$((2,0) \xrightarrow{\top} (3,1), \quad 2)$]

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

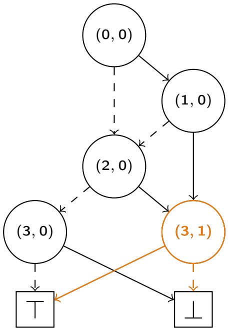
Seek	Sum	Result
$(3, 1)$	0	2

Priority Queue: Q_{count}:

[

$((1, 0) \xrightarrow{\top} (3, 1), \quad 1)$,
$((2, 0) \xrightarrow{\top} (3, 1), \quad 2)$]

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

Seek	Sum	Result
$(3, 1)$	0	2

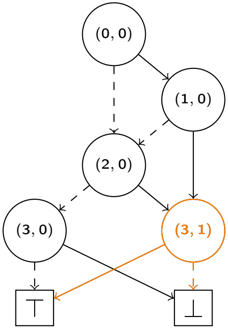
Priority Queue: Q_{count}:

$[$

$((1, 0) \xrightarrow{\top} (3, 1), \quad 1) \quad ,$

$((2, 0) \xrightarrow{\top} (3, 1), \quad 2) \quad]$

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

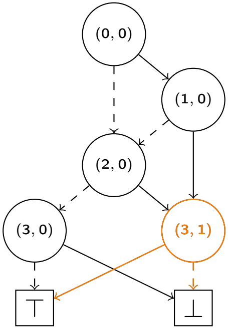
Seek	Sum	Result
$(3, 1)$	1	2

Priority Queue: Q_{count}:

[

$((2, 0) \xrightarrow{\top} (3, 1), \quad 2)$]

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

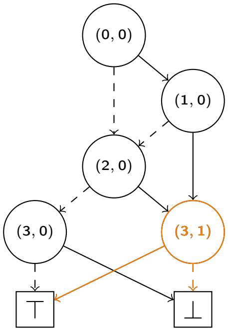
Seek	Sum	Result
$(3, 1)$	3	2

Priority Queue: Q_{count}:

[

]

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

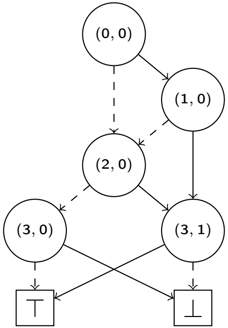
Seek	Sum	Result
$(3, 1)$	3	5

Priority Queue: Q_{count}:

[

]

Binary Decision Diagrams
Arge '96, Sølvsten '22



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Result

5

Priority Queue: Q_{count}:

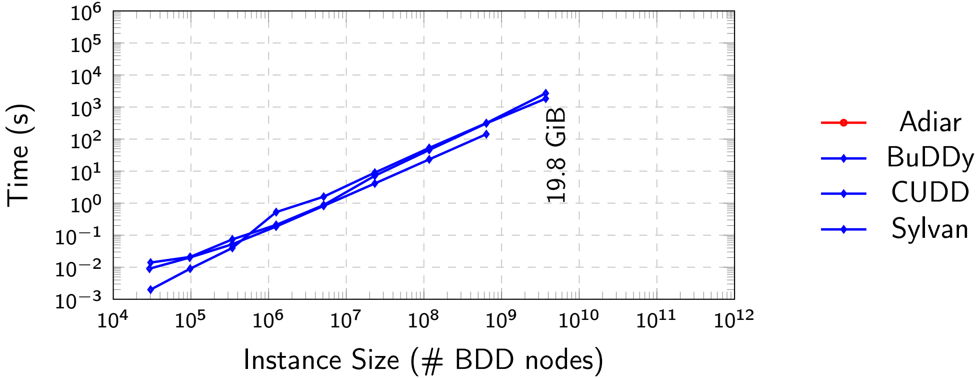
[

]

Adiar

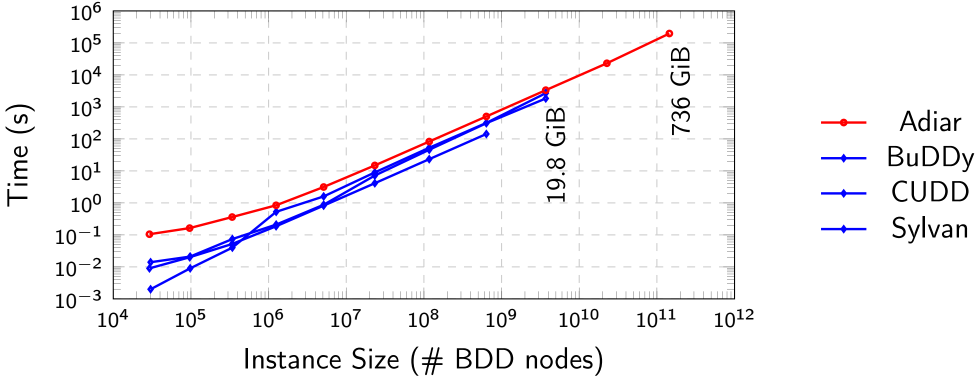
github.com/ssoelvsten/adiar

Binary Decision Diagrams
Arge '96, Sølvsten '22



Running time for the *N-Queens* problems.

Binary Decision Diagrams
Arge '96, Sølvsten '22



Running time for the *N-Queens* problems.

Further Reading : Foundations

- **Aggarwal and Vitter (1987)**
  "The Input/Output Complexity of Sorting and Related Problems"
  The I/O-model, Sorting, Permutation, FFT, and Matrix transposition.
- **Arge, Goodrich, Nelson, and Sitchinava (2008)**
  "Fundamental Parallel Algorithms for Private-cache Chip Multiprocessors."
  The I/O-model for Multi-Threading.

Further Reading : Data Structures

- **Arge (1995)**
  "The Buffer Tree: A new technique for Optimal I/O-algorithms"
  An I/O-efficient Tree, Priority Queue, and Range Tree.
- **Sanders (2002)**
  "Fast Priority Queues for Cached Memory"
  A much faster I/O-efficient Priority Queue.
- **Agarwal, Arge and Yi (2006)**
  "I/O-Efficient Batched Union-Find and Its Applications to Terrain Analysis"
  An I/O-efficient (Lazy) Union-Find.

Further Reading : Algorithms

- **Goodrich, Tsay, Vengroff, and Vitter (1993)**
  "External-Memory Computational Geometry"
  Distribution Sweeping and other algorithms.
- **Chiang, Goodrich, Grove, Tamassia, Vengroff, and Vitter (1995)**
  "External-memory Graph Algorithms"
  Time-forward Processing and other algorithms.
- **Arge, Toma, Vitter (2001)**
  "I/O-Efficient Algorithms for Problems on Grid-Based Terrains"
  The TerraFlow algorithm.

Further Reading : Libraries (C++)

- **TPIE : Templated Portable I/O Environment**
  `github.com/thomasmoelhave/tpie`
  Duke University and Aarhus University
- **STXXL : Standard Template library for XXL data sets**
  `github.com/stxxl/stxxl`
  University of Karlsruhe

Steffan Christ Sølvsten

soelvsten@cs.au.dk
@ssoelvsten

Adiar

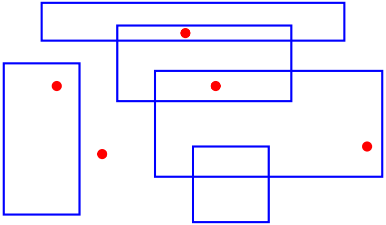
github.com/ssoelvsten/adiar
ssoelvsten.github.io/adiar

AARHUS UNIVERSITY

Distribution Sweeping
Goodrich, Tsay, Vengroff, and Vitter '93

Batched Range Searching

Given N axis-parallel rectangles and N points in the plane, compute for each point p all rectangles containing p.

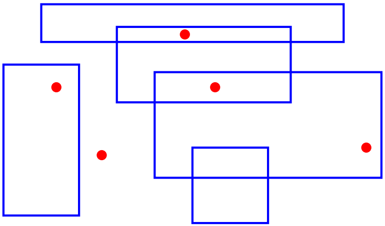


Theorem
Batched Range Searching can be solved in $O(sort(N) + scan(T))$ *I/Os.*

Distribution Sweeping
Goodrich, Tsay, Vengroff, and Vitter '93

Preprocessing:

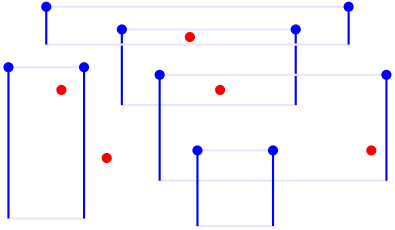
Algorithm:



Distribution Sweeping
Goodrich, Tsay, Vengroff, and Vitter '93

Preprocessing:

- Split each rectangle into two vertical lines.



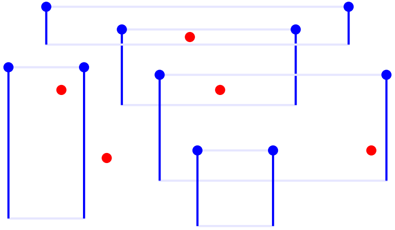
Algorithm:

Distribution Sweeping
Goodrich, Tsay, Vengroff, and Vitter ’93

Preprocessing:

- Split each rectangle into two vertical lines.
- Sort all lines and points by their x-value.

Algorithm:



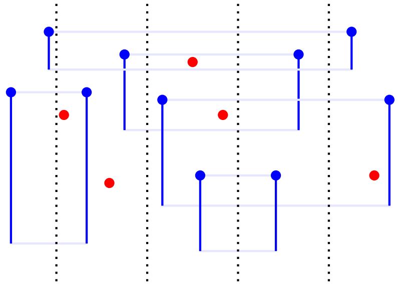
Distribution Sweeping
Goodrich, Tsay, Vengroff, and Vitter ’93

Preprocessing:

- Split each rectangle into two vertical lines.
- Sort all lines and points by their x-value.

Algorithm:

- Split all data into $\Theta(\sqrt{M/B})$ *slabs*. Solve these recursively; output is given sorted by y-value.



Distribution Sweeping
Goodrich, Tsay, Vengroff, and Vitter '93

Preprocessing:

- Split each rectangle into two vertical lines.
- Sort all lines and points by their x-value.

Algorithm:

- Split all data into $\Theta(\sqrt{M/B})$ *slabs*. Solve these recursively; output is given sorted by y-value.
- Merge slabs together, report points between line segments outside its slab.
  - Use $\Theta(\sqrt{M/B}^2) = \Theta(\sqrt{M/B})$ multi-slabs to maintain each *active* rectangle.
  - Output points and un-matched line segments.