

# I/O-Efficient Algorithms and Data Structures

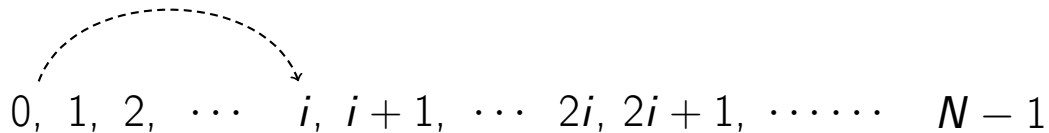
---

**Steffan Christ Sølvesten**

8<sup>th</sup> of September, 2023



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









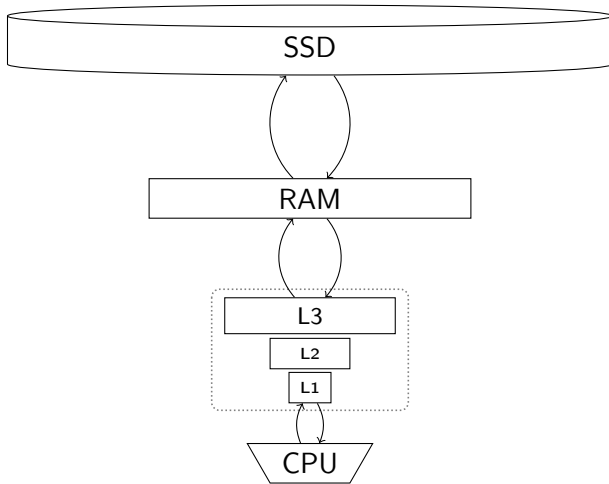










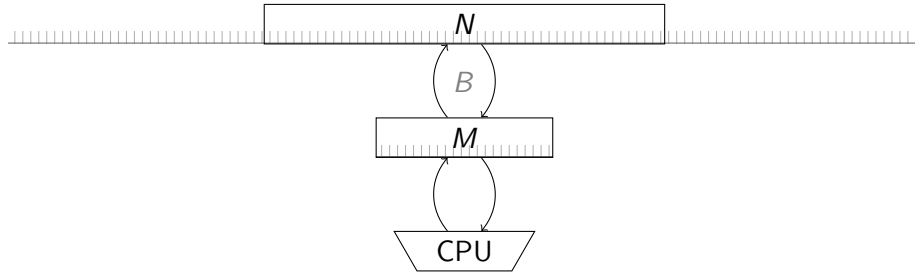






# I/O Model

Aggarwal and Vitter '87



# I/O Model : Sequential Access

## Aggarwal and Vitter '87

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



## 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
---------	---------	-----	---------



## 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
---------	---------	-----	---------



# I/O Model : Sequential Access

## Aggarwal and Vitter '87

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



# I/O Model : Sequential Access

## Aggarwal and Vitter '87

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

æ, ø

# I/O Model : Sequential Access

## Aggarwal and Vitter '87

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

æ, ø, å

## 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, ø, å
---------	---------	-----	---------	---------



Time :  $N$

I/O :  $N/B$

Memory :  $B$

## I/O Model : Stack



# I/O Model : Stack



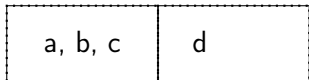
# I/O Model : Stack



# I/O Model : Stack



# I/O Model : Stack



# 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, h, i	j
---------	---

## I/O Model : Stack

a, b, c	d, e, f
---------	---------

g, h, i	j, k
---------	------

## I/O Model : Stack

a, b, c	d, e, f
---------	---------

g, h, i	j
---------	---



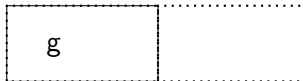
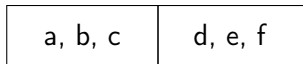
## I/O Model : Stack



## I/O Model : Stack



# I/O Model : Stack



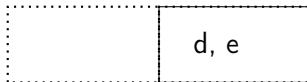
## I/O Model : Stack



## I/O Model : Stack



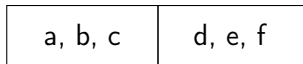
# I/O Model : Stack



## I/O Model : Stack



# I/O Model : Stack





# I/O Model : Stack

a, b, c	d, e, f
---------	---------

g', h'	d, e, f'
--------	----------

## 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

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

g', h', i'	j'
------------	----

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

# I/O Model : M/B-way Mergesort

## Aggarwal and Vitter '87



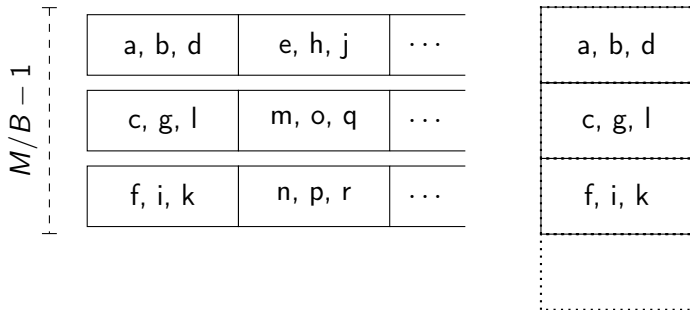
# I/O Model : M/B-way Mergesort

## Aggarwal and Vitter '87



# I/O Model : M/B-way Mergesort

## Aggarwal and Vitter '87





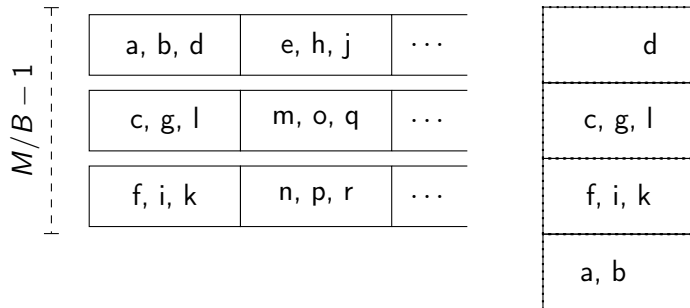
# I/O Model : M/B-way Mergesort

## 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





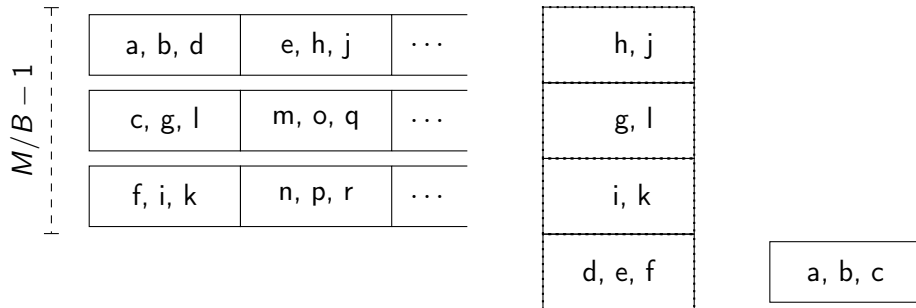
# I/O Model : M/B-way Mergesort

## 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



B

# 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



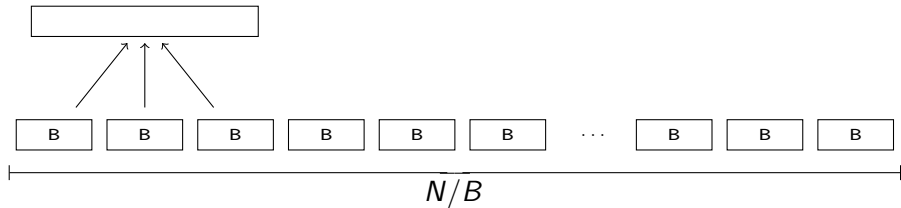
# I/O Model : M/B-way Mergesort

## Aggarwal and Vitter '87



# I/O Model : M/B-way Mergesort

## Aggarwal and Vitter '87



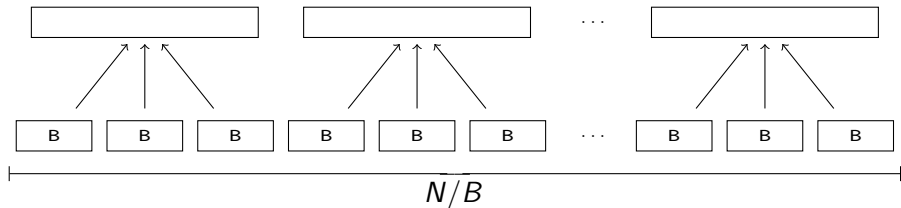
# I/O Model : M/B-way Mergesort

## 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  $p_i \in p_2, p_3, \dots, p_{N-1}$ :
  - 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  $p_i \in p_2, p_3, \dots, p_{N-1}$ :
  - 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  $p_i \in p_2, p_3, \dots, p_{N-1}$ :
  - 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  $p_i \in p_2, p_3, \dots, p_{N-1}$ :
  - 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  $p_i \in p_2, p_3, \dots, p_{N-1}$ :
  - 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  $p_i \in p_2, p_3, \dots, p_{N-1}$ :
  - 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  $p_i \in p_2, p_3, \dots, p_{N-1}$ :
  - 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  $S$

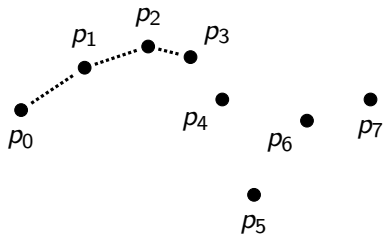


# Convex Hull : Graham Scan

## Graham '72

Upper Hull:

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





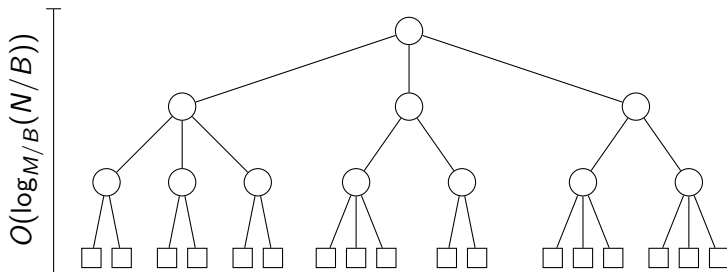
## a-b Tree

Huddleston and Mehlhorn '82



# Buffer Tree

Arge '95



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

# Buffer Tree

Arge '95



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

# Buffer Tree

Arge '95



# Buffer Tree

Arge '95





# Buffer Tree

Arge '95



# Buffer Tree

Arge '95



# Buffer Tree

Arge '95



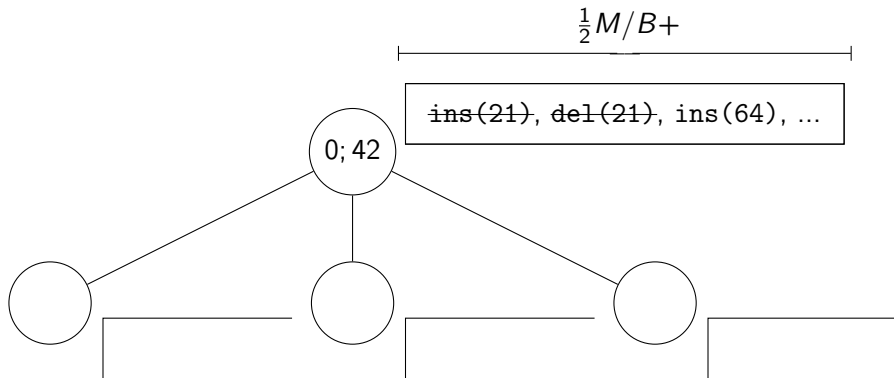
# Buffer Tree

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  $N$  inserts and 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  $N$  inserts and 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  $N$  push and delete min 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.  $\square$



# Binary Decision Diagrams

Arge '96, Sølvesten '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*

[ ((0, 0), (2, 0), (1, 0)) ,  
((1, 0), (2, 0), (3, 1)) ,  
((2, 0), (3, 0), (3, 1)) ,  
((3, 0),  $\top$ ,  $\perp$ ) ,  
((3, 1),  $\perp$ ,  $\top$ ) ]

*On-Disk Format*

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

# Binary Decision Diagrams

Arge '96, Sølvsten '22

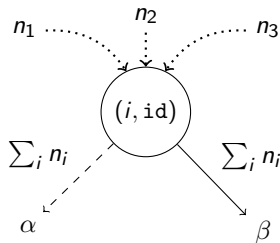


## Idea

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

# Binary Decision Diagrams

Arge '96, Sølvsten '22



## Time-Forward Processing

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

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

# Binary Decision Diagrams

Arge '96, Sølvsten '22



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

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

Priority Queue:  $Q_{count}$ :

[

]

# Binary Decision Diagrams

Arge '96, Sølvsten '22



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

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
$(1, 0)$	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, Sølvesten '22



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

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

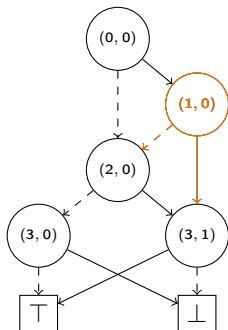
Priority Queue:  $Q_{count}$ :

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

]

# 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:  $Q_{count}$ :

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

# Binary Decision Diagrams

Arge '96, Sølvesten '22



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

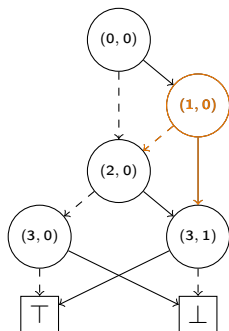
Seek	Sum	Result
(1, 0)	1	0

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, Sølvesten '22



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

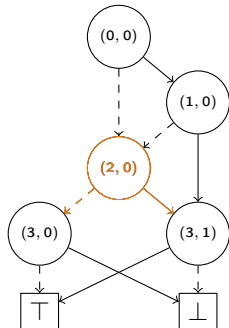
Seek	Sum	Result
(2, 0)	0	0

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

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

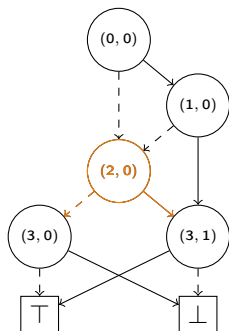
Seek	Sum	Result
(2, 0)	1	0

Priority Queue:  $Q_{count}$ :

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

# Binary Decision Diagrams

Arge '96, Sølvesten '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:  $Q_{count}$ :

[

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

# 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)	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:  $Q_{count}$ :

[

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

# 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

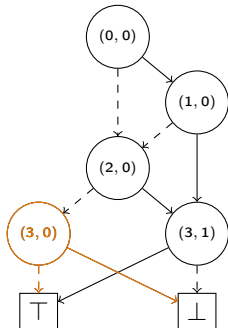
Priority Queue:  $Q_{count}$ :

[

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

# 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
<b>(3, 0)</b>	2	0

Priority Queue:  $Q_{count}$ :

[

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

# 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
<b>(3, 0)</b>	2	2

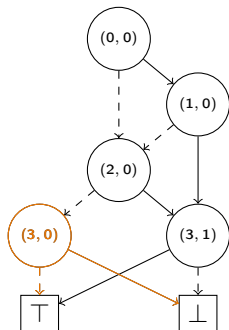
Priority Queue:  $Q_{count}$ :

[

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

# 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)	0	2

Priority Queue:  $Q_{count}$ :

[

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



# Binary Decision Diagrams

Arge '96, Sølvesten '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:  $Q_{count}$ :

[

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

# Binary Decision Diagrams

Arge '96, Sølvesten '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:  $Q_{count}$ :

[

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

# Binary Decision Diagrams

Arge '96, Sølvsten '22



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

Seek

(3, 1)

Sum

3

Result

2

Priority Queue:  $Q_{count}$ :

[

]

# Binary Decision Diagrams

Arge '96, Sølvesten '22



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

Seek

(3, 1)

Sum

3

Result

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:  $Q_{count}$ :

[

]

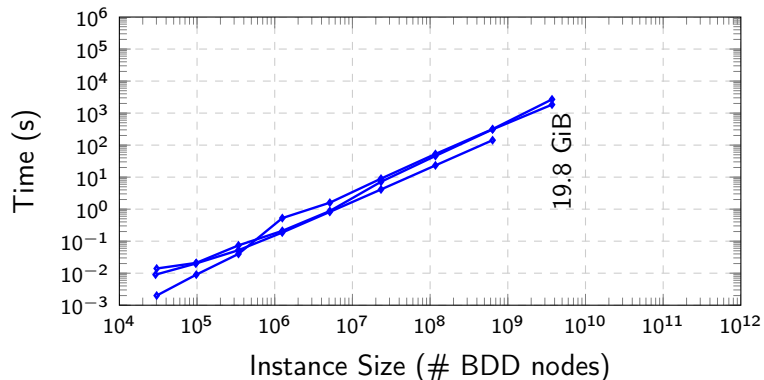


# Adiar

[github.com/ssoelvsten/adiar](https://github.com/ssoelvsten/adiar)

# Binary Decision Diagrams

Arge '96, Sølvsten '22



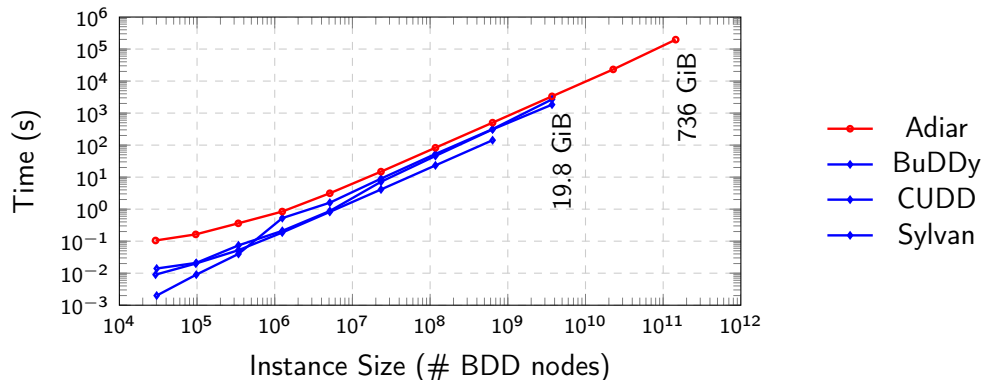
Adiar  
BuDDy  
CUDD  
Sylvan

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](https://github.com/thomasmoelhave/tpie)

Duke University and Aarhus University

- **STXXL : Standard Template library for XXL data sets**

[github.com/stxxl/stxxl](https://github.com/stxxl/stxxl)

University of Karlsruhe

# Steffan Christ Sølvesten

---

✉ [soelvsten@cs.au.dk](mailto:soelvsten@cs.au.dk)

🌐 [ssoelvsten.github.io](https://ssoelvsten.github.io)

## Adiar

---

🔗 [github.com/ssoelvsten/adiar](https://github.com/ssoelvsten/adiar)

📖 [ssoelvsten.github.io/adiar](https://ssoelvsten.github.io/adiar)





# 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  $O(\text{sort}(N) + \text{scan}(T))$  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

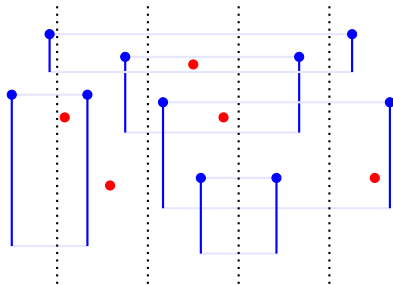
## Goodrich, Tsay, Vengroff, and Vitter '93

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.

