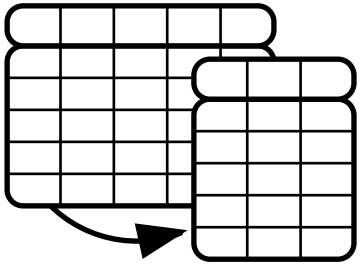
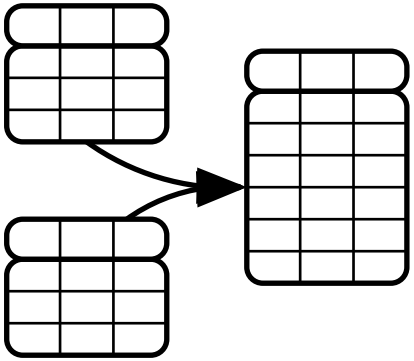
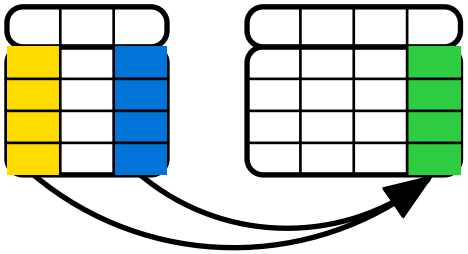
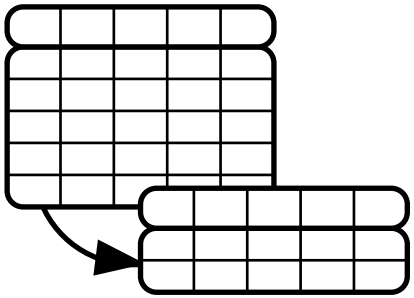
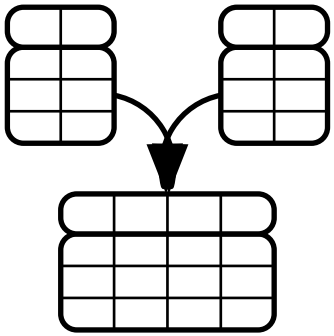
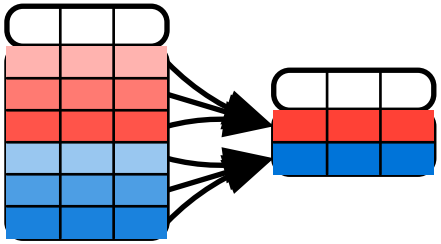
Extended Relational Algebra

CSE 4/562: Database Systems | Lecture 7

DB. Sys.: T.C.B.: Ch. 5.2, 6.3-6.4, 15.2, 15.4

Quiz

Relational Algebra

	Filter	Merge	Derive
Column			
Row			

Bags (or Sets)

- Encoding “Missing” Data (Null values)
- ‘At least once’ Joins (Outer Joins, $⟕$)

Sets

- Remove duplicate values (Distinct, δ)
- Remove elements from a set (Difference, $-$)

Lists

- Create a list (Sort, τ)
- Select a subset of the list (Limit, L)

Missing Data

Missing or Inapplicable Fields

- A tree with an unknown species
- The street of a tree in a park
- The ‘.’ on many people’s ID cards

Missing or Inapplicable Fields

- A tree with an unknown species
- The street of a tree in a park
- The ‘.’ on many people’s ID cards

SQL provides a special `NULL` value for these cases

NULL is horrible

1. “Not relevant” and “Not known” (should) have very different semantics.
2. The SQL spec takes shortcuts for performance reasons.

NULL is horrible

1. “Not relevant” and “Not known” (should) have very different semantics.
2. The SQL spec takes shortcuts for performance reasons.

> You need to take extra care around `NULL` values.

Arithmetic

`M.TimeA + M.TimeB`

What is the result for the row `[TimeA: NULL, TimeB: 15]`?

Arithmetic

`M.TimeA + M.TimeB`

What is the result for the row `[TimeA: NULL, TimeB: 15]`?

`NULL + X` and `X + NULL` are both `NULL`

Arithmetic

`M.TimeA + M.TimeB`

What is the result for the row `[TimeA: NULL, TimeB: 15]`?

`NULL + X` and `X + NULL` are both `NULL`

Comparisons

`M.TimeA < 20`

What is the result for the row `[TimeA: NULL, TimeB: 15]`?

Arithmetic

`M.TimeA + M.TimeB`

What is the result for the row `[TimeA: NULL, TimeB: 15]`?

`NULL + X` and `X + NULL` are both `NULL`

Comparisons

`M.TimeA < 20`

What is the result for the row `[TimeA: NULL, TimeB: 15]`?

Any comparison with a `NULL` is `UNKNOWN`

AND

A	B	$A \wedge B$
`unknown`	`unknown`	
`unknown`	`true`	
`unknown`	`false`	

Three-Valued Logic

AND

A	B	$A \wedge B$
unknown	unknown	unknown
unknown	`true`	
unknown	`false`	

Three-Valued Logic

AND

A	B	$A \wedge B$
unknown	unknown	unknown
unknown	`true`	unknown
unknown	`false`	

Three-Valued Logic

AND

A	B	$A \wedge B$
`unknown`	`unknown`	`unknown`
`unknown`	`true`	`unknown`
`unknown`	`false`	`false`

Three-Valued Logic

AND

A	B	$A \wedge B$
unknown	unknown	unknown
unknown	true	unknown
unknown	false	false

OR

A	B	$A \vee B$
unknown	unknown	unknown
unknown	true	true
unknown	false	unknown

NOT

A	$\neg A$
unknown	unknown

Filter (`WHERE`) eliminates all **non-true** rows.

The dangers of NULL


```
SELECT * FROM R
WHERE A = 2 OR NOT (A = 2)
```

The dangers of NULL


```
SELECT * FROM R
WHERE A = 2 OR NOT (A = 2)
```

$$(\texttt{NULL} = 2) \wedge \neg (\texttt{NULL} = 2)$$

The dangers of NULL


```
SELECT * FROM R
WHERE A = 2 OR NOT (A = 2)
```

$$(\texttt{NULL} = 2) \wedge \neg (\texttt{NULL} = 2)$$

$$\text{unknown} \wedge \neg \text{unknown}$$

The dangers of NULL


```
SELECT * FROM R
WHERE A = 2 OR NOT (A = 2)
```

$(\texttt{NULL} = 2) \wedge \neg (\texttt{NULL} = 2)$

$\text{unknown} \wedge \neg \text{unknown}$

$\text{unknown} \wedge \text{unknown}$

The dangers of NULL


```
SELECT * FROM R
WHERE A = 2 OR NOT (A = 2)
```

$$(\texttt{NULL} = 2) \wedge \neg (\texttt{NULL} = 2)$$

$$\text{unknown} \wedge \neg \text{unknown}$$

$$\text{unknown} \wedge \text{unknown}$$

$$\text{unknown}$$

At-least once

How many trains did each person take?

At-least once

How many trains did each person take?

```
SELECT name, count(*)
FROM Passenger R
GROUP BY name
```

At-least once

How many trains did each person take?

```
SELECT name, count(*)
FROM Passenger R
GROUP BY name
```

But what if there is a person who rode no trains?

At-least once

How many trains did each person take?

```
SELECT name, count(*)
FROM Passenger R
GROUP BY name
```

But what if there is a person who rode no trains?

```
SELECT P.name, count(*)
FROM Passenger R, Person P
WHERE P.name = R.name
GROUP BY P.name
```

At-least once

How many trains did each person take?

```
SELECT name, count(*)
FROM Passenger R
GROUP BY name
```

But what if there is a person who rode no trains?

```
SELECT P.name, count(*)
FROM Passenger R, Person P
WHERE P.name = R.name
GROUP BY P.name
```

At-least once

but this is still wrong...

Join only joins tuples that **are** present.

Set Difference

- Every **distinct** row that:
  - Is in the first input
  - Is not in the second input

We use − as the relational algebra operator for set difference.

Every person who is not a passenger:

```
SELECT name FROM Person
MINUS
SELECT name FROM Passenger
```

At-least once with set difference


```
SELECT name, count(*) FROM Passenger GROUP BY name
UNION ALL (
  SELECT name, 0 FROM Person
  MINUS
  SELECT name, 0 FROM Passenger
)
```

At-least once


```
SELECT name, train_no
FROM Person P LEFT OUTER JOIN Passenger R ON P.name = R.name
```

Join `Person` and `Passenger` but guarantee that each `Person` (the left relation) appears at least once.

`train_no` will be `NULL` for all rows where `Person` has no match.

Inner Join (⋈)

- Standard join operation

Left Outer Join (⟕)

- Every row in the left relation appears in the output

Right Outer Join (⟖)

- Every row in the right relation appears in the output

Full Outer Join (⟗)

- Every row in both relations appears in the output

Outer Join

Person	**Name**
	Argyle
	Beata
	Ceryl

Passenger	**Name**	**Train No**
	Argyle	48
	Argyle	7
	Beata	49
	Dwayne	6

Outer Join

Person ⋈ Passenger	Person.Name	Passenger.Name	Train No
	Argyle	Argyle	48
	Argyle	Argyle	7
	Beata	Beata	49
	Ceryl	NULL	NULL

Outer Join

Person ⋉ Passenger	Person.Name	Passenger.Name	Train No
	Argyle	Argyle	48
	Argyle	Argyle	7
	Beata	Beata	49
	NULL	Dwayne	6

Outer Join

Person ⟗ Passenger	Person.Name	Passenger.Name	Train No
	Argyle	Argyle	48
	Argyle	Argyle	7
	Beata	Beata	49
	Ceryl	NULL	NULL
	NULL	Dwayne	6

NULLs are WEIRD


```
SELECT p.name, count(train_no)
FROM Person p LEFT OUTER JOIN Passenger r ON p.name = r.name
```

`NULL` values passed to an aggregate are discarded.

NULLs are WEIRD

R	A
	1
	NULL

```
SELECT SUM(A) FROM R
```

S	A	B
	1	NULL

```
SELECT A + B FROM S
```

These queries produce *different* results

More Sets

Distinct


```
SELECT DISTINCT name FROM Passenger
```

Distinct


```
SELECT DISTINCT name FROM Passenger
```

We use δ as the relational algebra operator for distinct.

Distinct


```
SELECT DISTINCT name FROM Passenger
```

We use δ as the relational algebra operator for distinct.

This is the same as (and evaluated similarly to)

```
SELECT name FROM Passenger
GROUP BY name
```

Lists

Sort

```
SELECT * FROM Train
ORDER BY route ASC, train_no DESC
```

We use τ as the relational operator for sort.

Sort

```
SELECT * FROM Train
ORDER BY route ASC, train_no DESC
```

We use τ as the relational operator for sort.

Limit

```
SELECT * FROM Train
ORDER BY route ASC, train_no DESC
LIMIT 20
```

We use L as the relational operator for sort.

Without an order-by clause, the DB can return any N rows.

```
class LimitIterator:
  def __init__(self, count)
    self.count = count

  def next(self):
    if self.count > 0:
      self.count -= 1;
      return input.next()
    else:
      return None
```

(Even if the input is not sorted)

Sort


```
def __init__():
  this.rows = []
  while row := input.next() is not None:
    this.rows += [row]
  sort(this.rows)

def next():
  # return the next row of rows
```

… but $O(N)$ memory 😕

2-Phase Sort

Idea: Merging 2 sorted lists requires $O(1)$ memory.

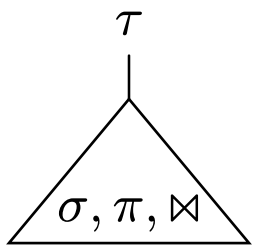
Phase 1

Create lots of sorted lists using available memory.

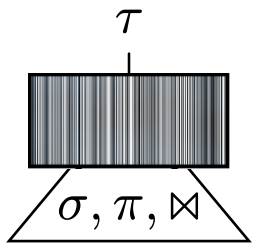
Phase 2

Repeatedly merge sorted lists of size N into sorted lists of size $2N$

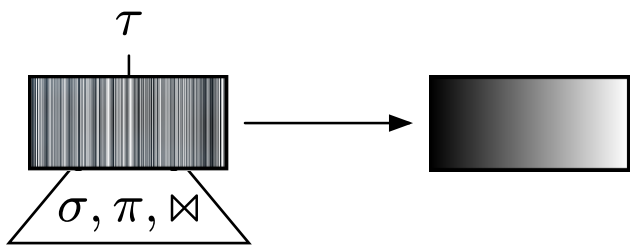
2-Phase Sort: Phase 1



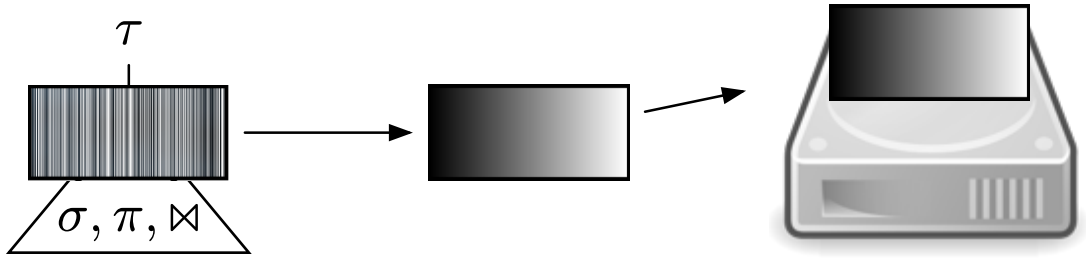
2-Phase Sort: Phase 1



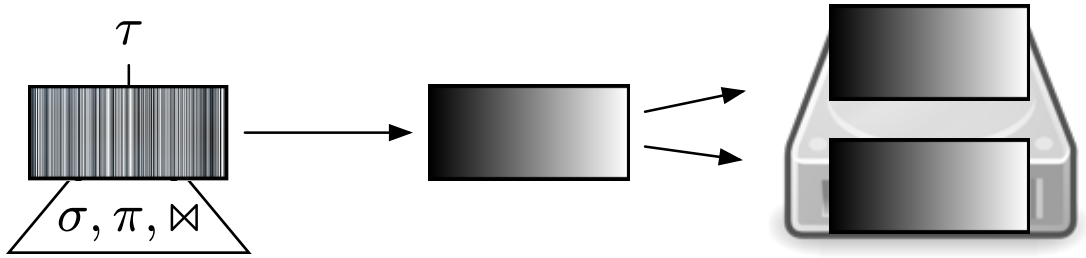
2-Phase Sort: Phase 1



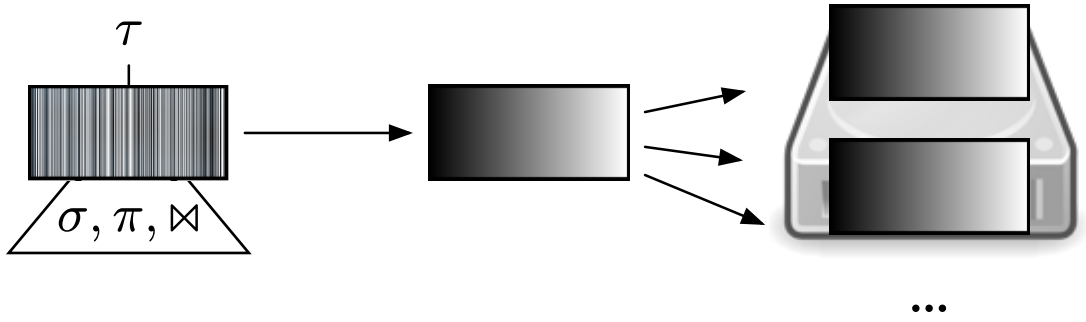
2-Phase Sort: Phase 1



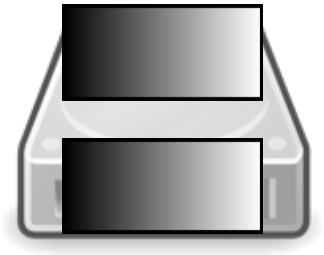
2-Phase Sort: Phase 1



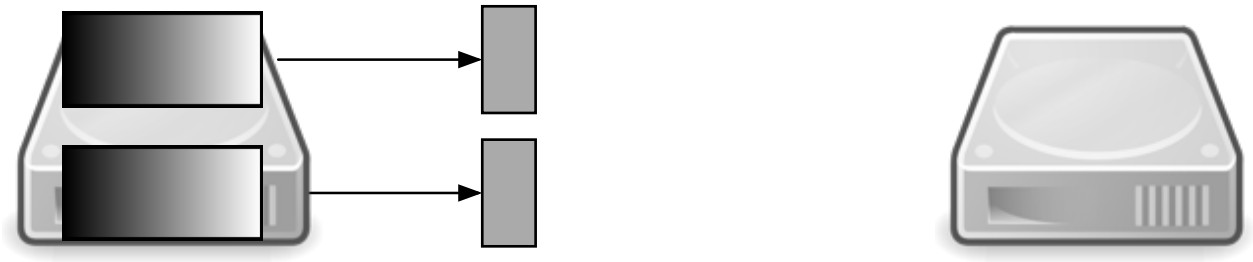
2-Phase Sort: Phase 1



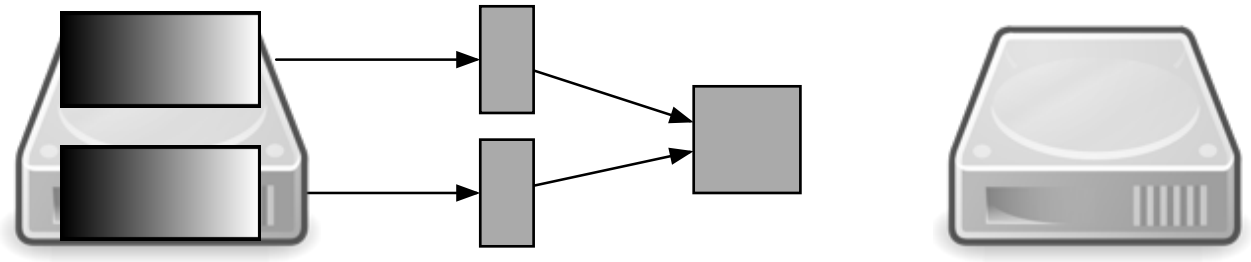
2-Phase Sort: Phase 2



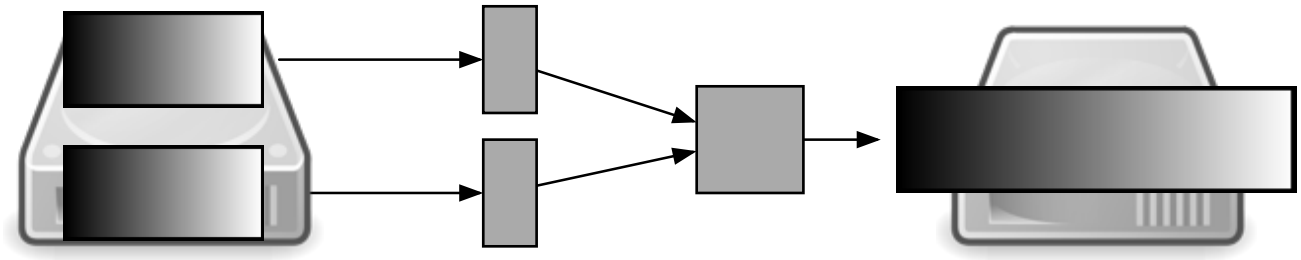
2-Phase Sort: Phase 2



2-Phase Sort: Phase 2



2-Phase Sort: Phase 2

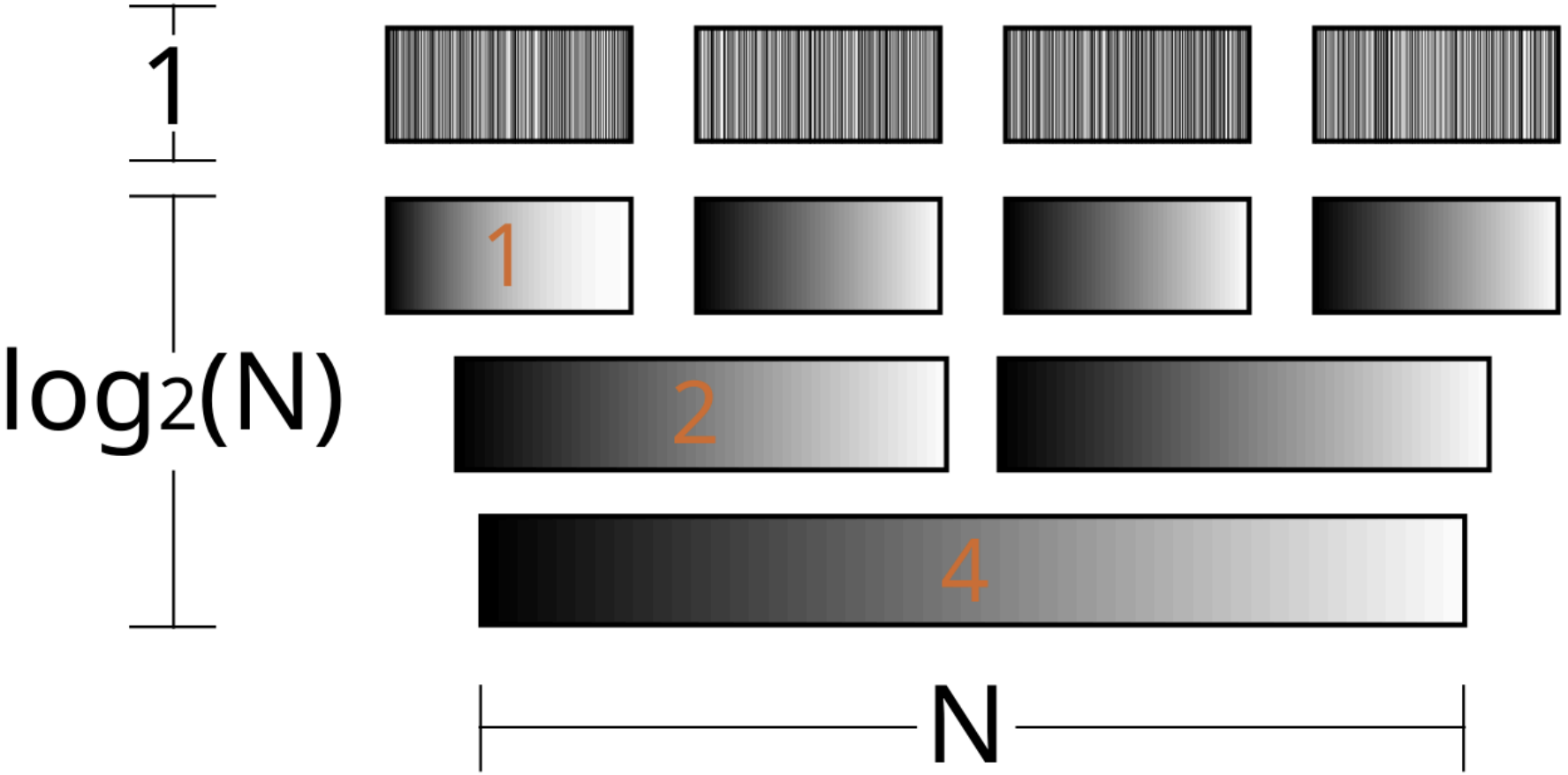


2-Phase Sort: Phase 2

Repeat as needed...

If we use B memory, how much IO is required?

2 Phase Sort



Using more memory

Pass 1

- Sort Bigger Buffers

Pass 2

- Merge K runs simultaneously: $O\left(N \cdot \left\lceil \log_{K(N)} \right\rceil\right)$ IO

Using more memory

Pass 1

- Sort Bigger Buffers
- Re-use Memory When Done

Pass 2

- Merge K runs simultaneously: $O\left(N \cdot \left\lceil \log_{K(N)} \right\rceil\right)$ IO

Replacement Sort

Replacement Sort

Input	Working Set	Output
12	2	3
4	8	5
3	10	
...		

Replacement Sort

Input	Working Set	Output
12	2	3
4		5
3	10	8
...		

Replacement Sort

Input	Working Set	Output
4	2	3
3	12	5
...	10	8

Replacement Sort

Input	Working Set	Output
4	2	3
3	10	5
...	12	8

Replacement Sort

Input	Working Set	Output
4	2	3
3		5
...	12	8
		10

Replacement Sort

Input	Working Set	Output
3	2	3
...	4	5
	12	8
		10

Replacement Sort

Input	Working Set	Output
3	2	3
...	4	5
		8
		10
		12

Replacement Sort

Input	Working Set	Output
...	2	3
	3	5
	4	8
		10
		12

Replacement Sort

Input	Working Set	Output
...	2	3
	3	5
	4	8
		10
		12

Replacement Sort

Input	Working Set	Output
...	2	
	3	
	4	

Replacement Sort

Input	Working Set	Output
...		2
	3	
	4	

Replacement Sort

For random data, runs will be of average size $2 \cdot B$

For sorted data, runs will be of size N

Even in the worst case, runs will be at least size B

Reminders

- Checkpoint 2 posted; Autolab up soon.
- Checkpoint 1 solutions to be posted soon.
- Schedule code review meetings.