Recap

problem: Retrieving data efficiently

solution: Layout Strategies
- delimited
- fixed size
- directory

X
- For fields in a record
- For ~~records~~
- For records on a page

problem: Retrieving specific data efficiently

solution: 1) Sort data
2) Sort + Tree index
3) Secondary Index
4) Hash table

problem: Sorting data

solution: ?

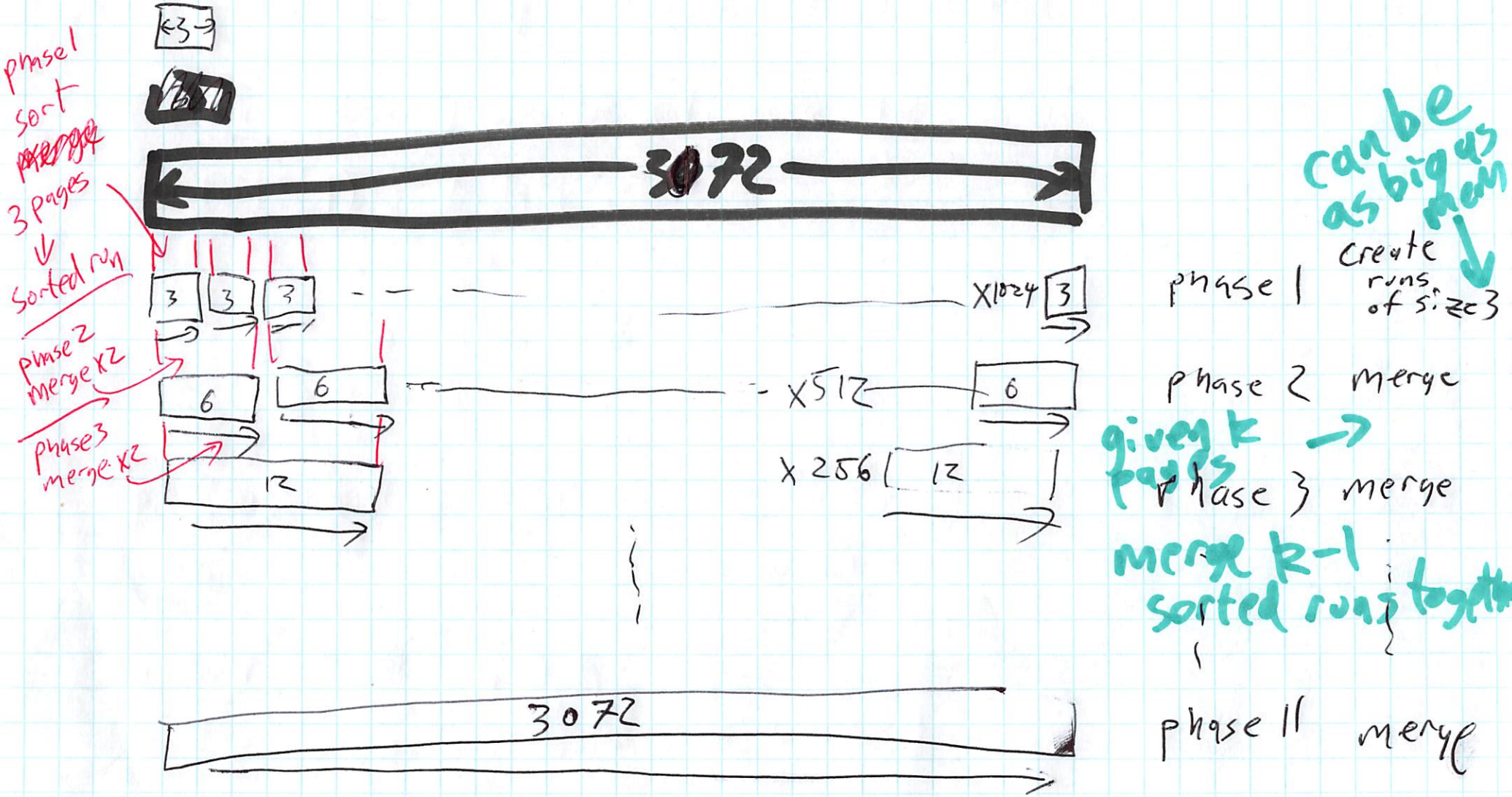
problem: Summarizing Data

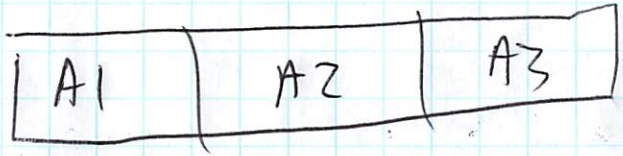
solution: ?

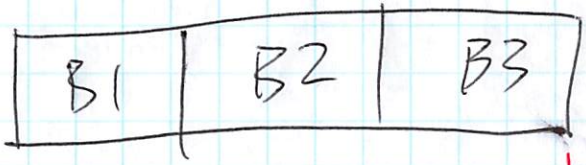
} with limited memory

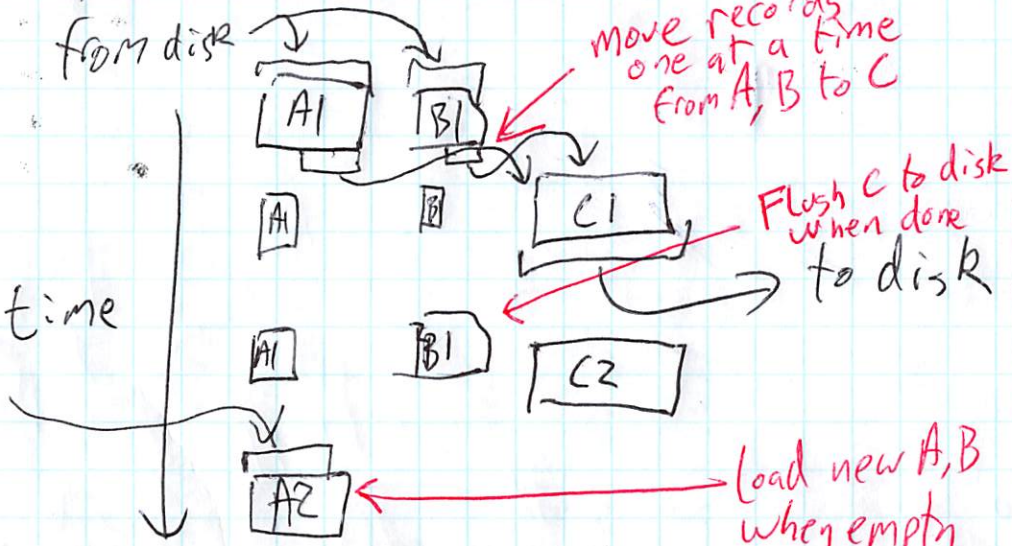
3072 pages of data

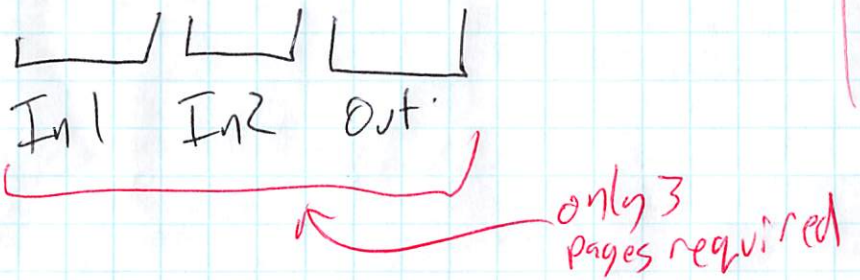
3 pages of memory

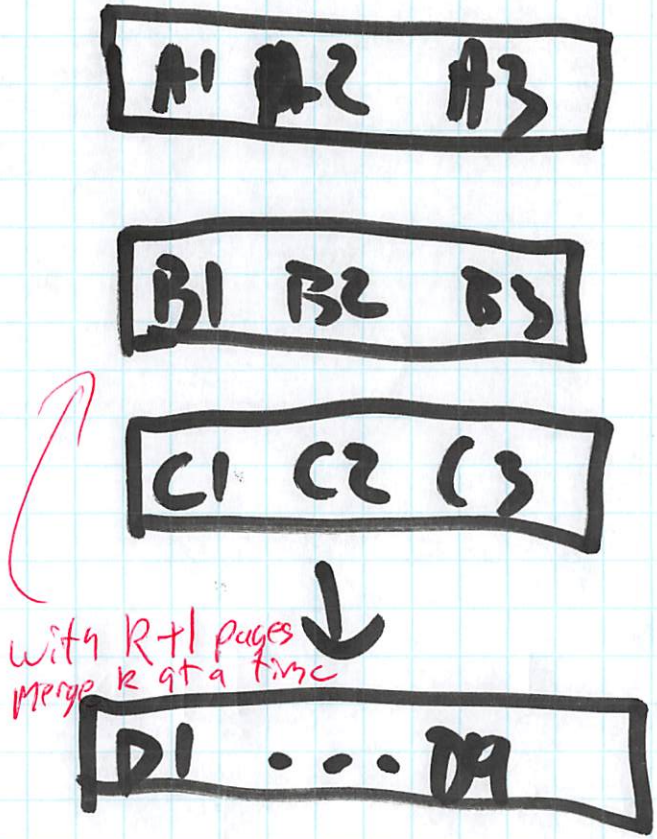



A1
A2
A3


B1
B2
B3


from disk
A1
B1
move records one at a time from A, B to C
A1
B1
C1
Flush C to disk when done
to disk
time
A1
B1
C2
A2
load new A, B when empty


In1
In2
Out
only 3 pages required


A1 A2 A3
B1 B2 B3
C1 C2 C3
with R+1 pages merge k at a time
D1 ... Dn

Cost analysis

Phase 1 : $2 \cdot \#pages$ IOs

phases 2-11: $2 \cdot \#pages$ IOs

After phase	this big sorted runs
1	3
2	6
3	12
N	$3 \cdot 2^{N-1}$

$\#pages \leq 3 \cdot 2^{N-1}$

$\frac{\#pages}{3} \leq 2^{N-1}$

$\log_2\left(\frac{\#pages}{3}\right) \leq N-1$

$N \geq 1 + \log\left(\frac{\#pages}{3}\right)$

Total cost

$\left(1+\log\left(\frac{\#pages}{3}\right)\right) \cdot \#pages \cdot 2$ IOs

have k pages of mem

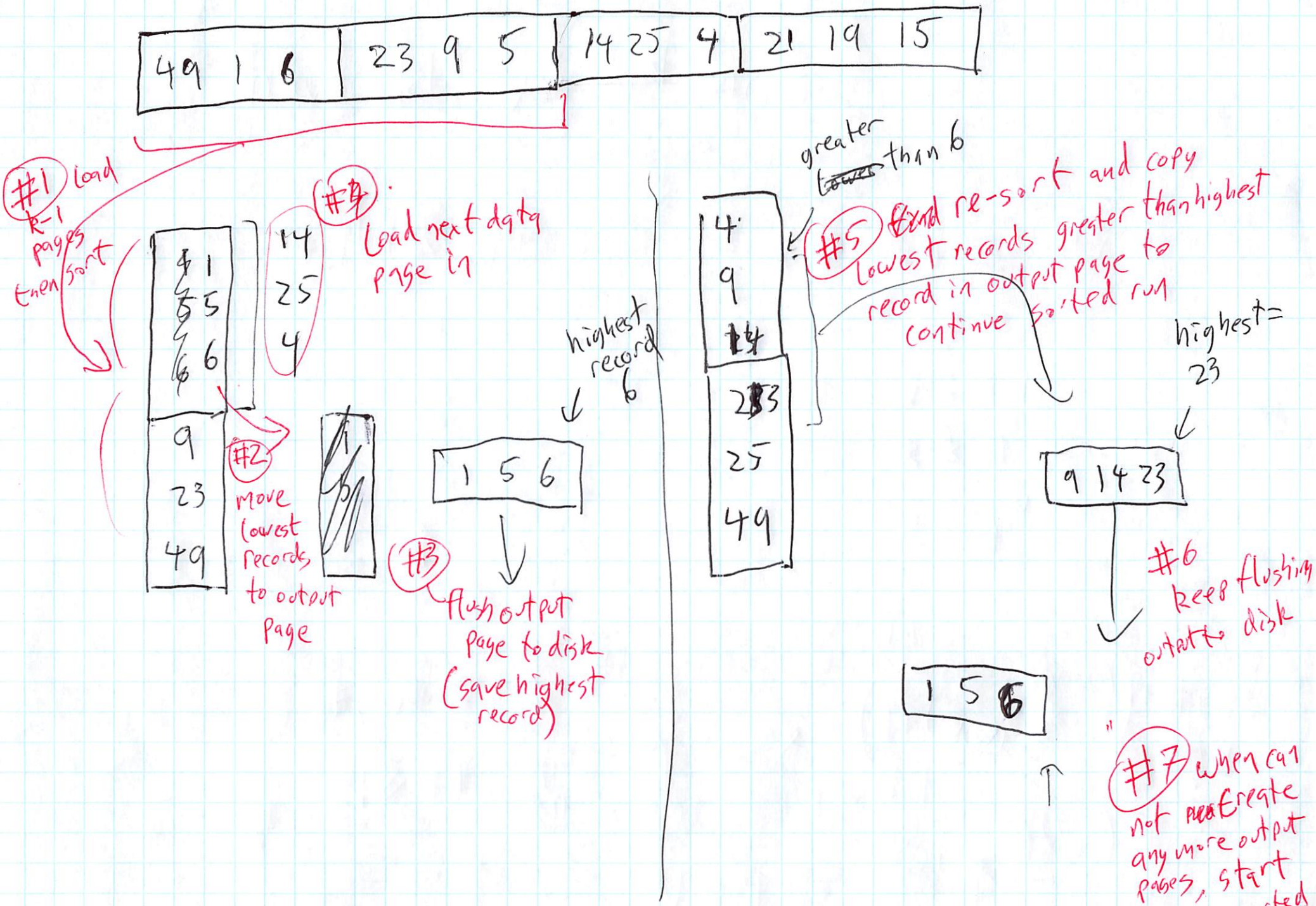
After phase	this big sorted runs
1	k
2	$k \cdot (k-1)$
3	$k \cdot (k-1)^2$
N	$k \cdot (k-1)^{N-1}$

$$\#pages \leq k \cdot (k-1)^{N-1}$$

$$N \geq 1 + \left(\log_{(k-1)}\left(\frac{\#pages}{k}\right)\right)$$

$$\left(1 + \log_{(k-1)}\left(\frac{\#pages}{k}\right)\right) \cdot \#pages \cdot 2 \text{ IOs}$$

$$\approx \log_k(\#pages) \cdot \#pages \cdot 2$$


49 1 6
23 9 5
14 25 4
21 19 15
#1 Load k-1 pages then sort
#2 move lowest records to output page
#3 flush output page to disk (save highest record)
#4 Load next data page in
14
25
4
highest record 6
1 5 6
greater than 6
#5 find lowest records greater than highest record in output page to continue sorted run
highest= 23
9 14 23
#6 keep flushing output to disk
1 5 6
#7 when can not create any more output pages, start a new sorted run

Fold (default) (merge (current value, record value))

Sum= fold (0) (current + record)

record		current (sum)	current (count)
1	+	0 +1	0 +1
10	+	1 +10	1 +1
19	+	11 +19	2 +1
13	+	30 +13	3 +1
4	+	43 +4	4 +1
2	+	47 +2	5 +1
		49	6

Count = fold (0) (current + 1)

MIN = fold ($+\infty$) (min(current, record))

Max = " $-\infty$ max " "

Average = fold (<count: 0, sum: 0>) (<count: current.count + 1
sum: current.sum + record>)

with $\frac{\text{current.sum}}{\text{current.count}}$

1 10 19 13 4 2 9

~~Stddev~~
~~Variance~~
Stddev

fold (<sum: 0, sumsq: 0>) (sum: current.sum + record
sumsq: current.sumsq + record^2
count:

with $\sqrt{(\text{current.sum})^2 - (\text{current.sumsq})}$

$R \to \sqrt{\frac{\left(\sum_{x \in R} x\right)^2 - \sum_{x \in R} x^2}{N^2}}$

Aggregate	Alg. vs Hol.	Intermediate
SUM	Algebraic	
COUNT	"	
AVERAGE	"	
STDDEV	"	
MIN	"	
MAX	"	
MEDIAN	Holistic	
COUNT DISTINCT	"	

Grades (Student, Grade, Class)

```
SELECT Class, Average(grade)
FROM Grades
GROUP BY Class
```

```
# Classes = { }
for record in grades:
    classes[record.class] += [record.grade]
```