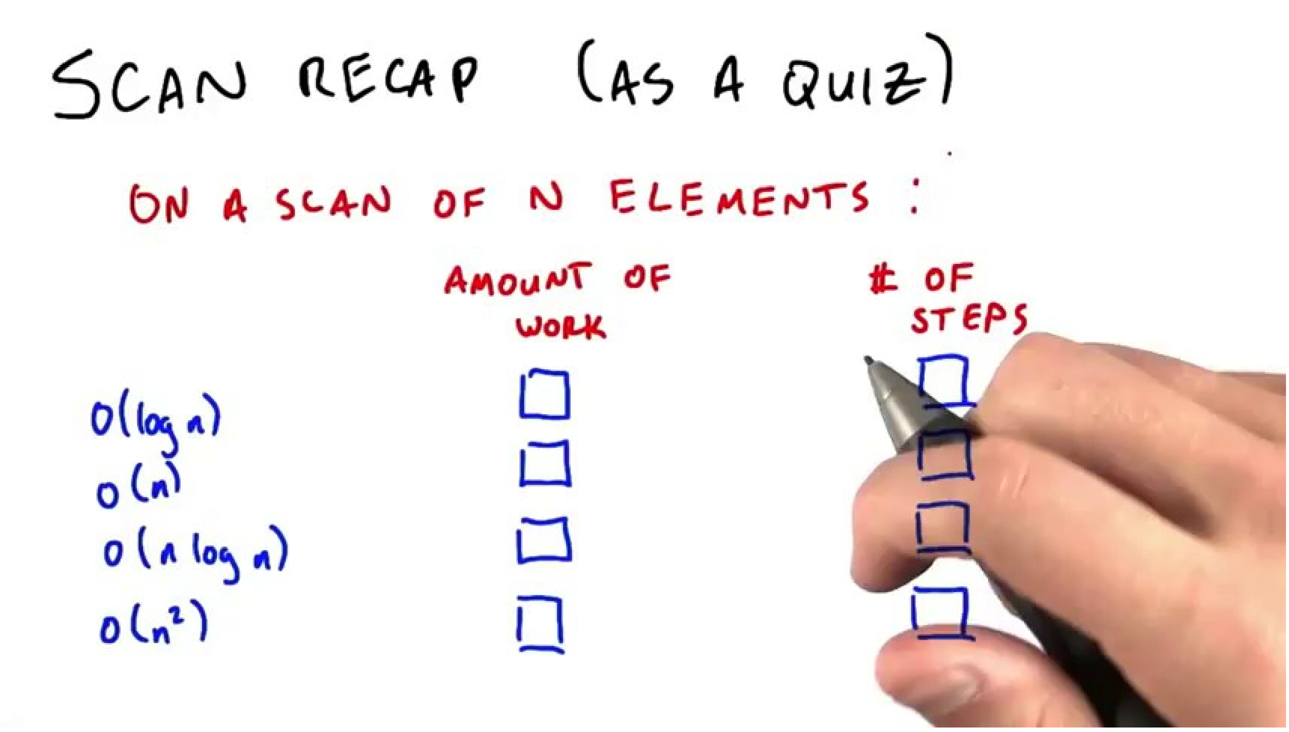
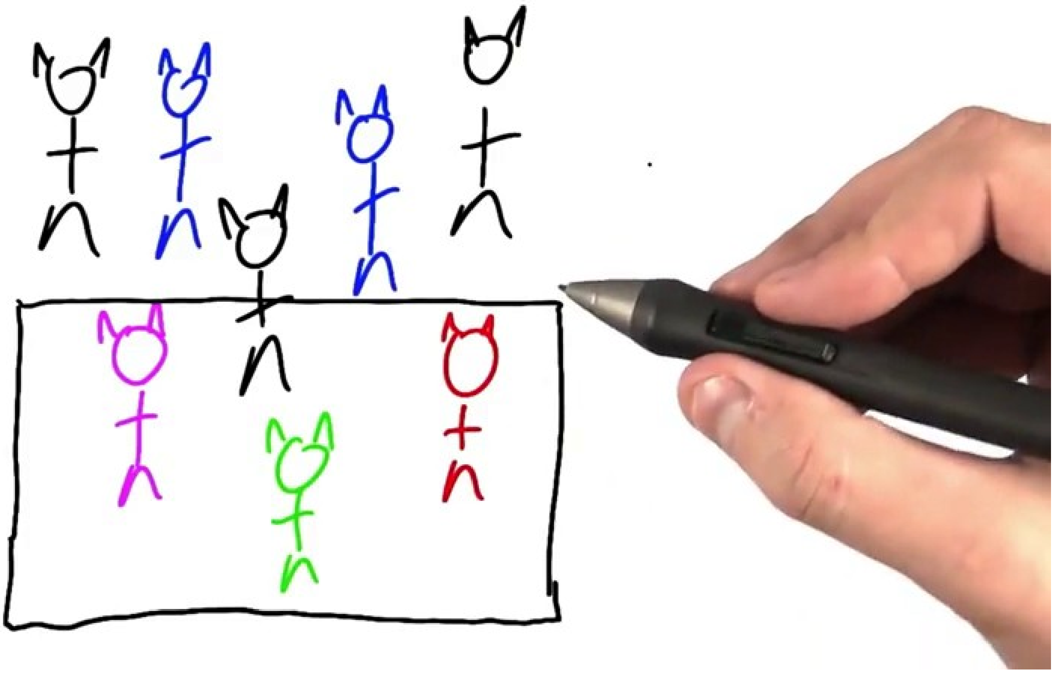
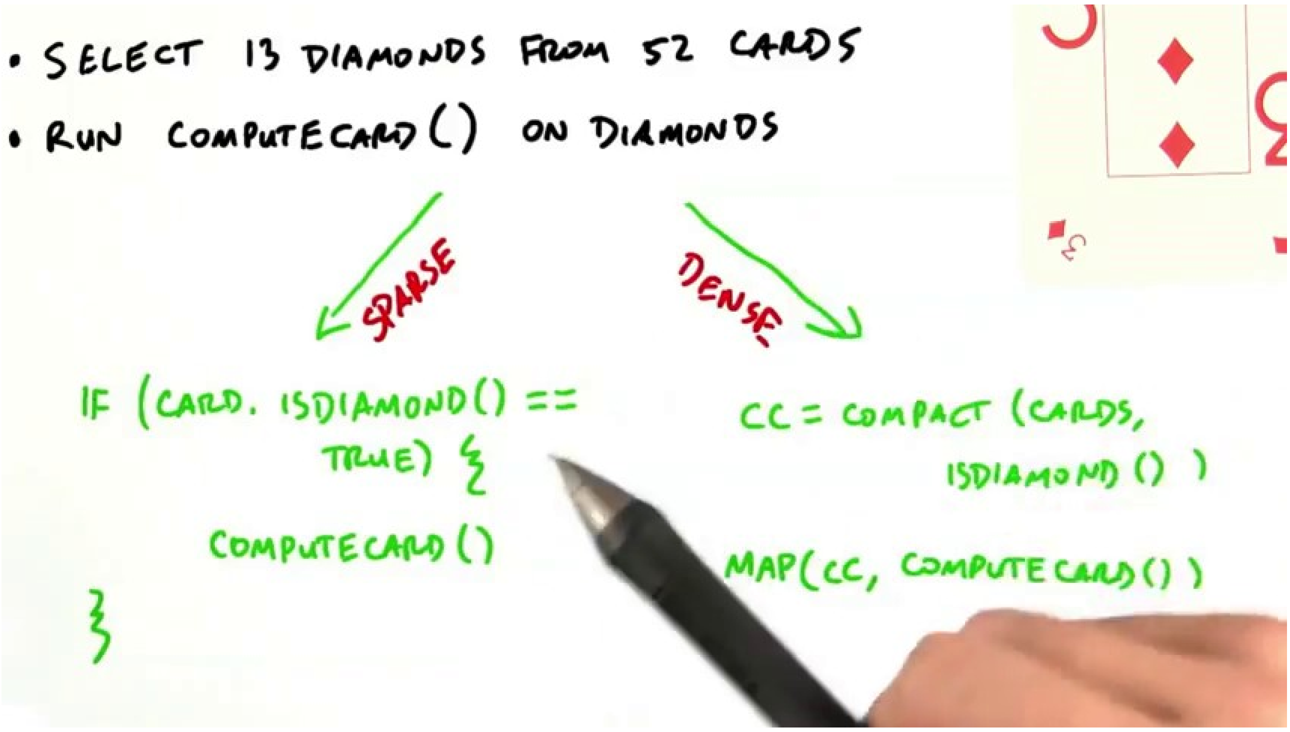

SCAN RECAP (AS A QUIZ)
ON A SCAN OF N ELEMENTS:
AMOUNT OF WORK
# OF STEPS
O(log n)
O(n)
O(n log n)
O(n²)





• SELECT 13 DIAMONDS FROM 52 CARDS
• RUN COMPUTECARD() ON DIAMONDS
SPARSE
DENSE
IF (CARD. ISDIAMOND() ==
TRUE) {
COMPUTECARD()
}
CC = COMPACT (CARDS,
ISDIAMOND())
MAP(CC, COMPUTECARD())

QUIZ: WHEN TO USE COMPACT?

COMPACT IS MOST USEFUL WHEN WE COMPACT AWAY A

- [ ] SMALL
- [ ] LARGE

NUMBER OF ELEMENTS

AND THE COMPUTATION ON EACH SURVIVING ELEMENT IS

- [ ] CHEAP
- [ ] EXPENSIVE

?

CORE ALGORITHM FOR COMPACT

PRED	T	F	F	T	T	F	T	F
ADDRESSES								

CORE ALGORITHM FOR COMPACT

PRED	T	F	F	T	T	F	T	F
ADDRESSES	0	–	–	1	2	–	3	–

PRED	1	0	0	1	1	0	1	0
ADDRESSES	0	1	1	1	2	3	3	4

STEPS TO COMPACT

1) PREDICATE
2) SCAN-IN ARRAY: TRUE 1, FALSE 0
3) EXCLUSIVE-SUM-SCAN (SCAN-IN)

   OUTPUT IS SCATTER ADDRESSES FOR COMPACTED ARRAY

4) SCATTER INPUT INTO OUTPUT USING ADDRESSES

QUIZ

COMPACT 1M ELEMENTS (1 → 1M)

A: ISDIVISIBLE BY 17 [KEEPS FEW ITEMS]

B: ISNOT DIVISIBLE BY 31 [KEEPS MANY ITEMS]

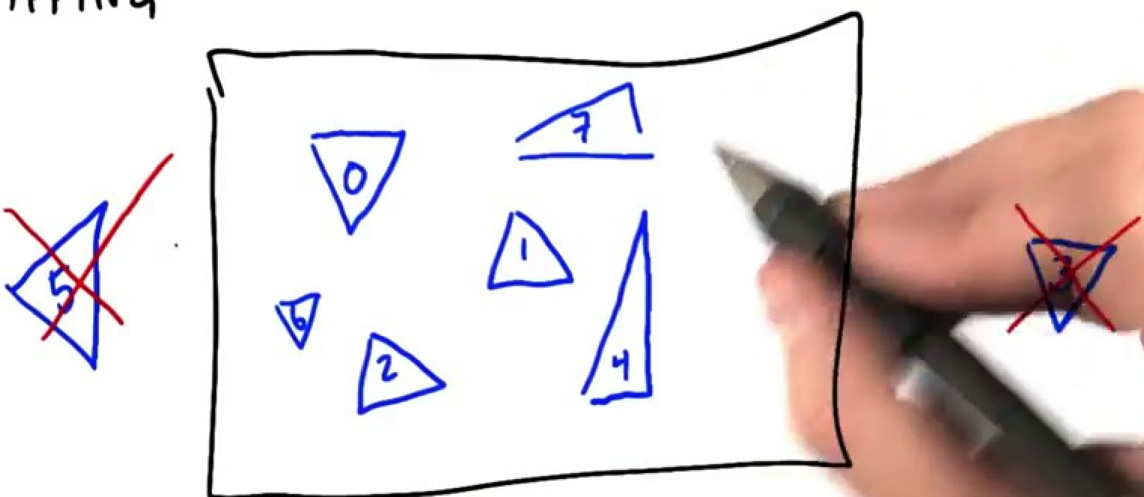
	A RUNS FASTER	SAME	B RUNS FASTER
PREDICATE	☐	☐	☐
SCAN	☐	☐	☐
SCATTER	☐	☐	☐

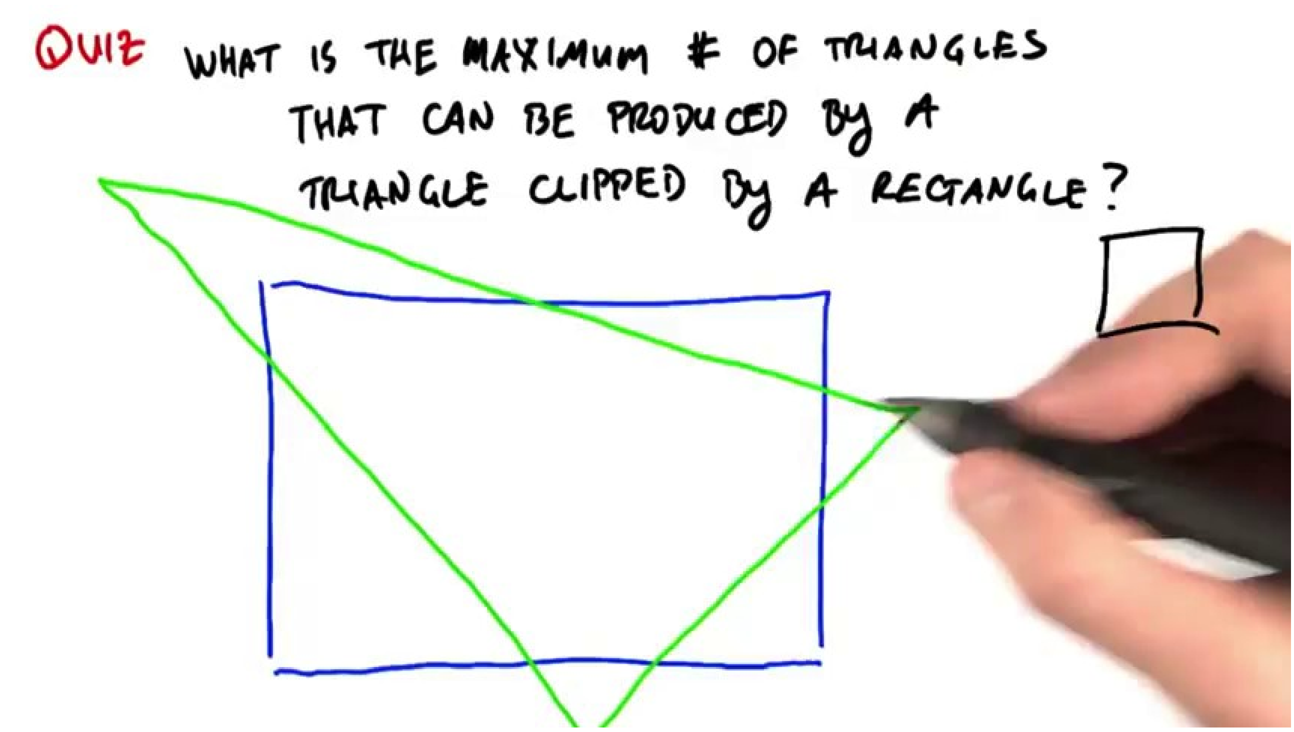
ALLOCATE

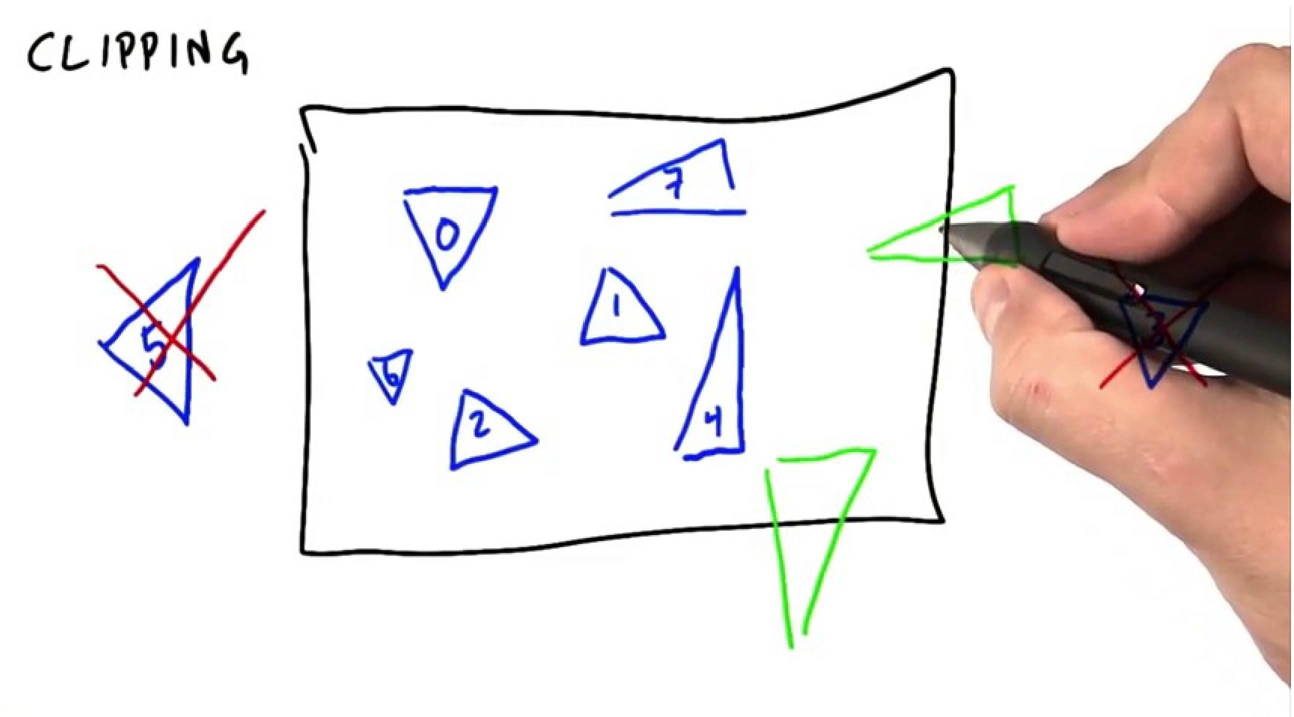
COMPACT GENERATES ↗ 1 OUTPUT FOR TRUE INPUTS
Ø FOR FALSE

CAN WE GENERALIZE?

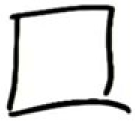
CLIPPING




QUIZ
WHAT IS THE MAXIMUM # OF TRIANGLES
THAT CAN BE PRODUCED BY A
TRIANGLE CLIPPED BY A RECTANGLE?


CLIPPING
0
7
1
5
6
2
4
3

QUIZ WHAT IS THE MAXIMUM # OF TRIANGLES THAT CAN BE PRODUCED BY A TRIANGLE CLIPPED BY A RECTANGLE?



POSSIBLE ALLOCATE STRATEGY

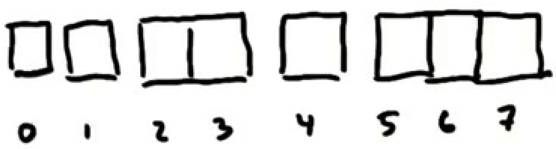
- ALLOCATE MAXIMUM SPACE IN INTERMEDIATE ARRAY
- COMPACT RESULT

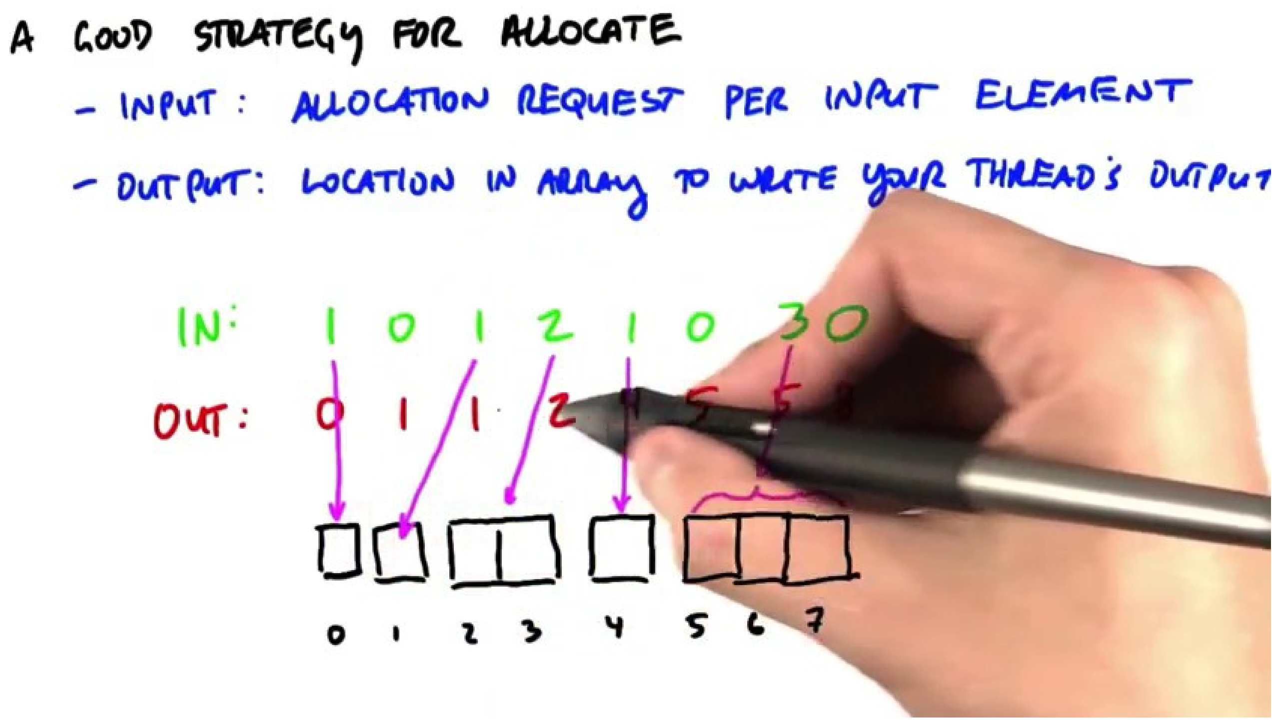
A GOOD STRATEGY FOR ALLOCATE

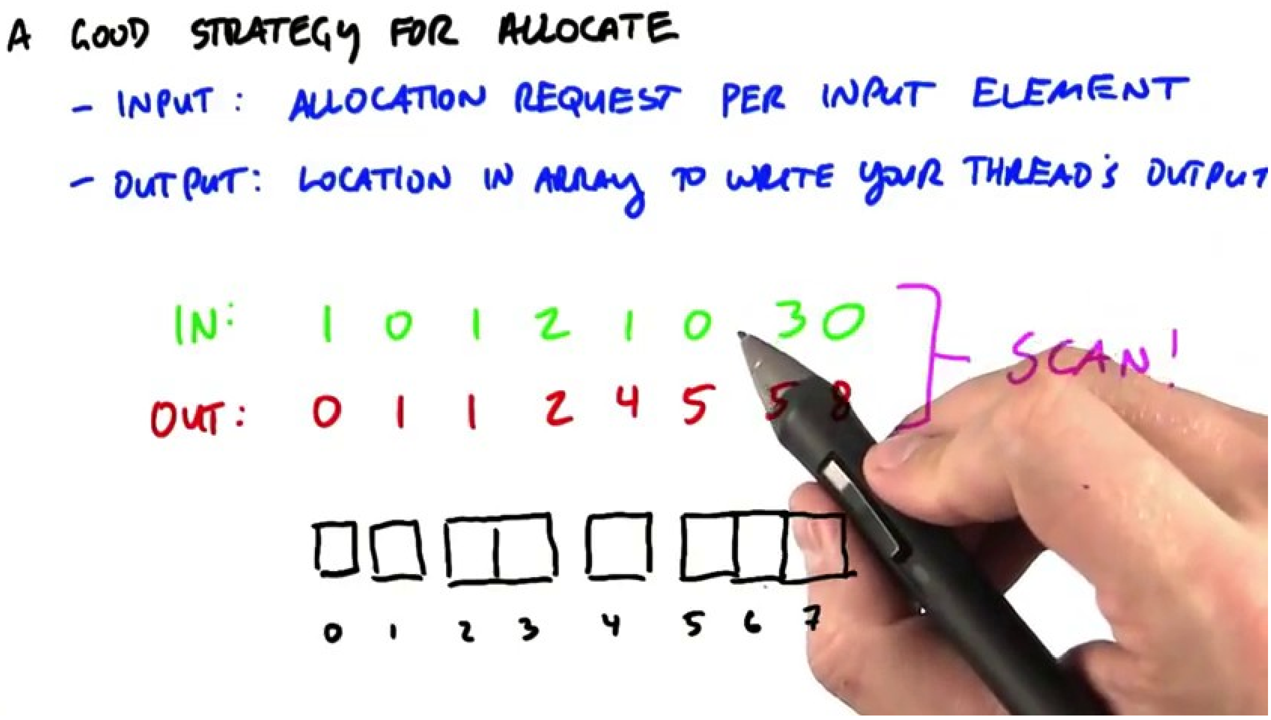
- INPUT : ALLOCATION REQUEST PER INPUT ELEMENT
- OUTPUT: LOCATION IN ARRAY TO WRITE YOUR THREAD'S OUTPUT

IN: 1 0 1 2 1 0 3 0

OUT: 0 1 1 2 4 5 5 8




A GOOD STRATEGY FOR ALLOCATE
- INPUT: ALLOCATION REQUEST PER INPUT ELEMENT
- OUTPUT: LOCATION IN ARRAY TO WRITE YOUR THREAD'S OUTPUT
IN: 1 0 1 2 1 0 3 0
OUT: 0 1 1 2 4 5 5 8
0 1 2 3 4 5 6 7


A GOOD STRATEGY FOR ALLOCATE
- INPUT: ALLOCATION REQUEST PER INPUT ELEMENT
- OUTPUT: LOCATION IN ARRAY TO WRITE YOUR THREAD'S OUTPUT
IN: 1 0 1 2 1 0 3 0
OUT: 0 1 1 2 4 5 5 8
SCAN!
0 1 2 3 4 5 6 7

SEGMENTED SCAN

- MANY SMALL SCANS?
  - LAUNCH EACH INDEPENDENTLY
  - COMBINE AS SEGMENTS ⇒ SEGMENTED SCAN

EXCLUSIVE SUM SCAN:

(1 2 3 4 5 6 7 8) ⟶ (0 1 3 6 10 15 21 28)

(1 2 | 3 4 5 | 6 7 8) ⟶ (0 1 | 0 3 7 | 0 6 13)

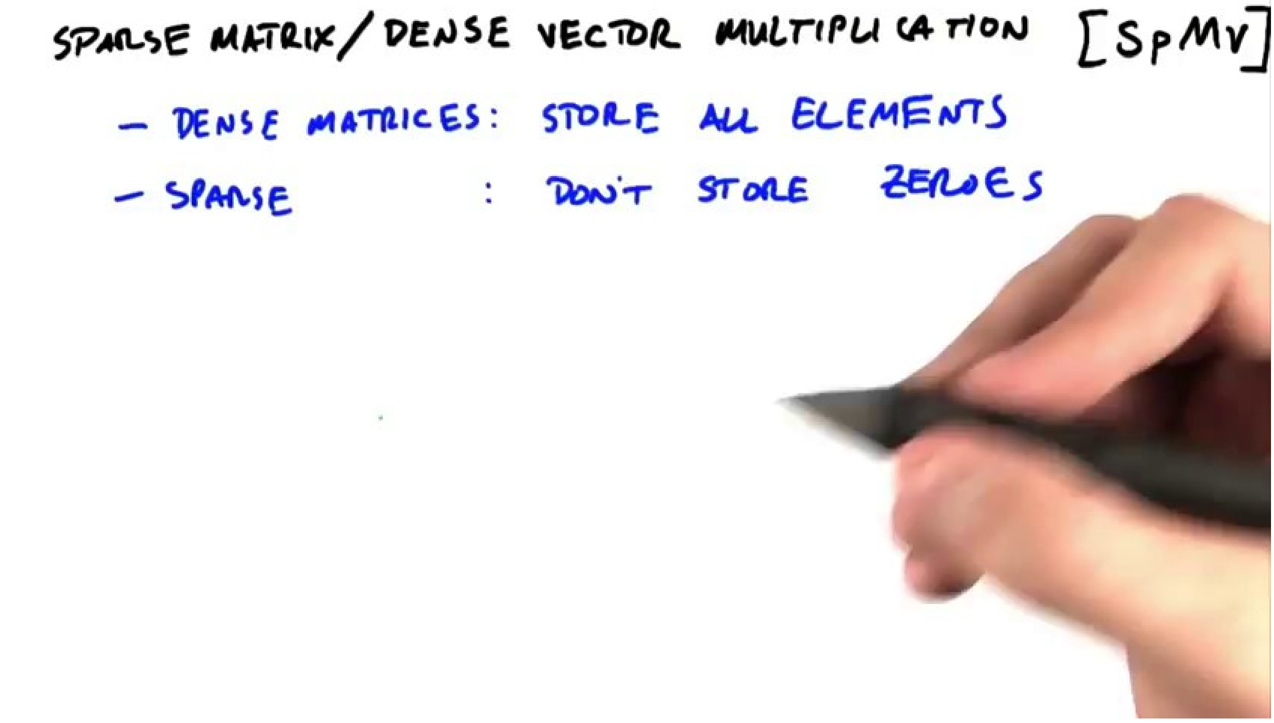
(1 0 1 0 0 1 0 0): SEGMENT HEADS

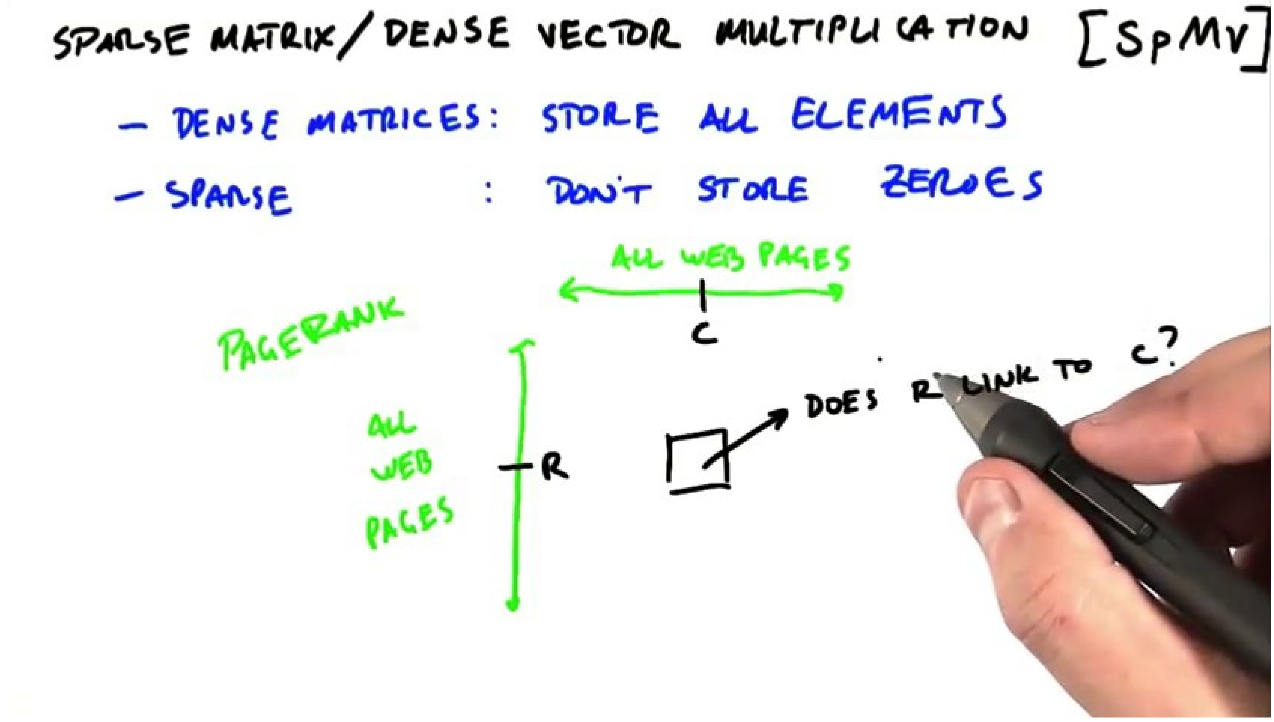
QUIZ

INCLUSIVE SEGMENTED SUM SCAN ON SAME ARRAY:

(1 2 | 3 4 5 | 6 7 8)

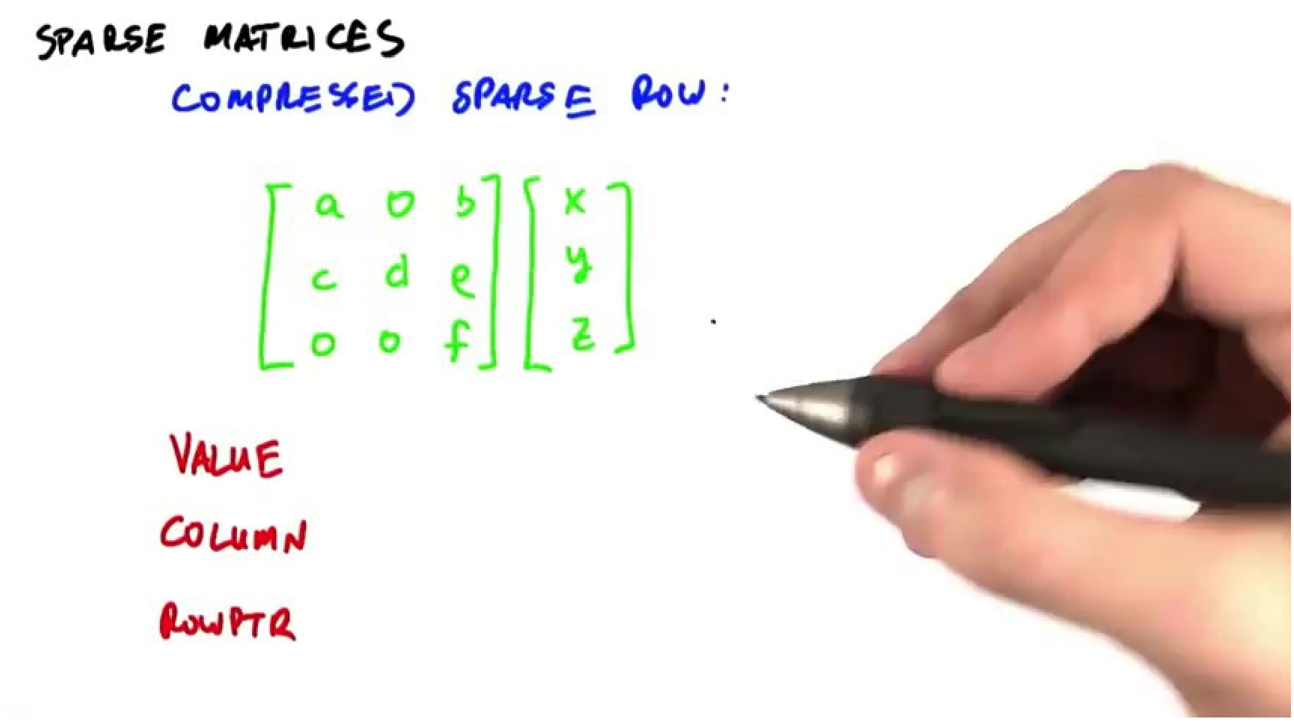
$\boxed{1}$ $\boxed{3}$ $\boxed{3}$ $\boxed{7}$ $\boxed{12}$ $\boxed{6}$ $\boxed{13}$ $\boxed{21}$

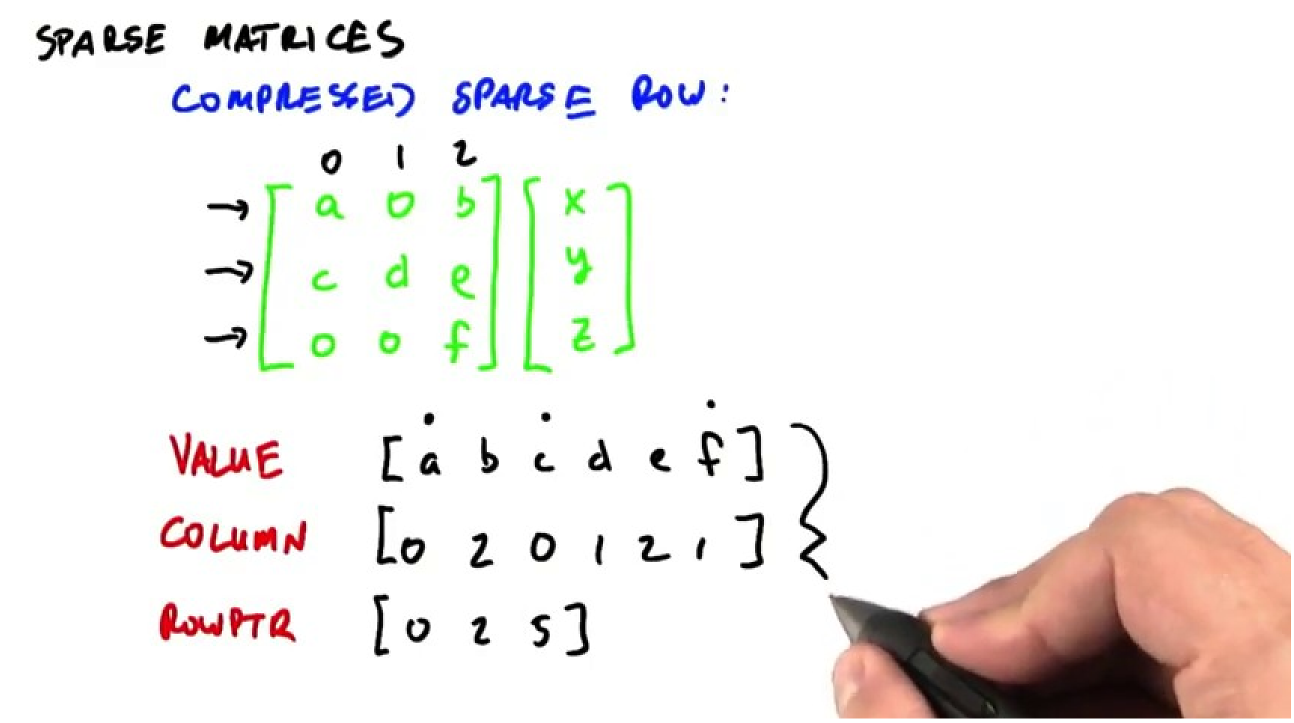

SPARSE MATRIX / DENSE VECTOR MULTIPLICATION [SpMV]
- DENSE MATRICES: STORE ALL ELEMENTS
- SPARSE : DON'T STORE ZEROES


SPARSE MATRIX / DENSE VECTOR MULTIPLICATION [SpMV]
- DENSE MATRICES: STORE ALL ELEMENTS
- SPARSE : DON'T STORE ZEROES
PAGERANK
ALL WEB PAGES
C
ALL WEB PAGES
R
DOES R LINK TO C?

MATRIX × VECTOR

$$\underbrace{\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}}_{3\times3} \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{3\times1} = \underbrace{\begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}}_{3\times1}$$


SPARSE MATRICES
COMPRESSED SPARSE ROW:
a 0 b
c d e
0 0 f
x
y
z
VALUE
COLUMN
ROWPTR


SPARSE MATRICES
COMPRESSED SPARSE ROW:
0 1 2
a 0 b
c d e
0 0 f
x
y
z
VALUE [a b c d e f]
COLUMN [0 2 0 1 2 1]
ROWPTR [0 2 5]

QUIZ: GENERATE CSR FOR

$$\begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 1 & 3 \end{bmatrix}$$

VALUE

INDEX

ROWPTR

VALUE $[\dot{a}\ b\ \dot{c}\ d\ e\ \dot{f}]$

COLUMN $[0\ 2\ 0\ 1\ 2\ 1]$

ROWPTR $[0\ 2\ 5]$

VECTOR

$$\begin{array}{cc} \begin{bmatrix} x \\ y \\ z \end{bmatrix} & \begin{array}{c} 0 \\ 1 \\ 2 \end{array} \end{array}$$

1 CREATE SEGMENTED REP'N FROM VALUE + ROWPTR

2 GATHER VECTOR VALUES USING COLUMN

3 PAIRWISE MULTIPLY 1·2

1 CREATE SEGMENTED REP'N FROM VALUE + ROWPTR

2 GATHER VECTOR VALUES USING COLUMN

3 PAIRWISE MULTIPLY 1·2

(BACKWARDS)

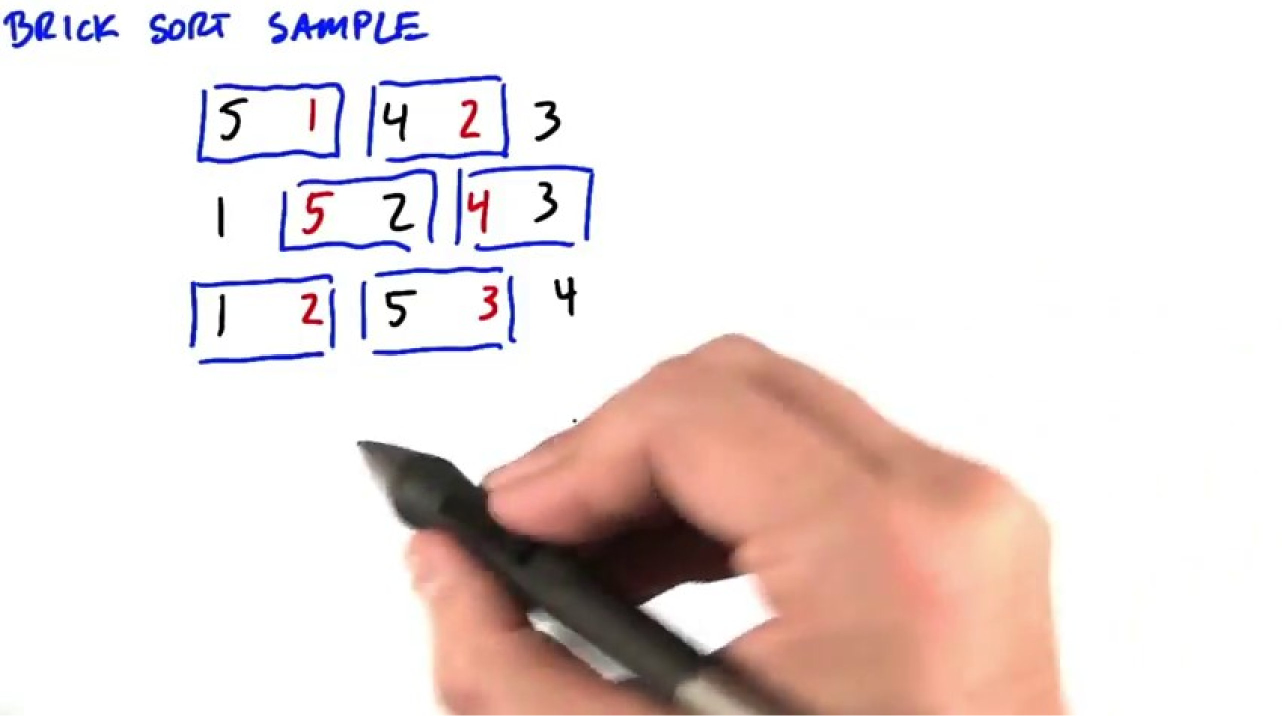
4 EXCLUSIVE SEGMENTED SUM SCAN

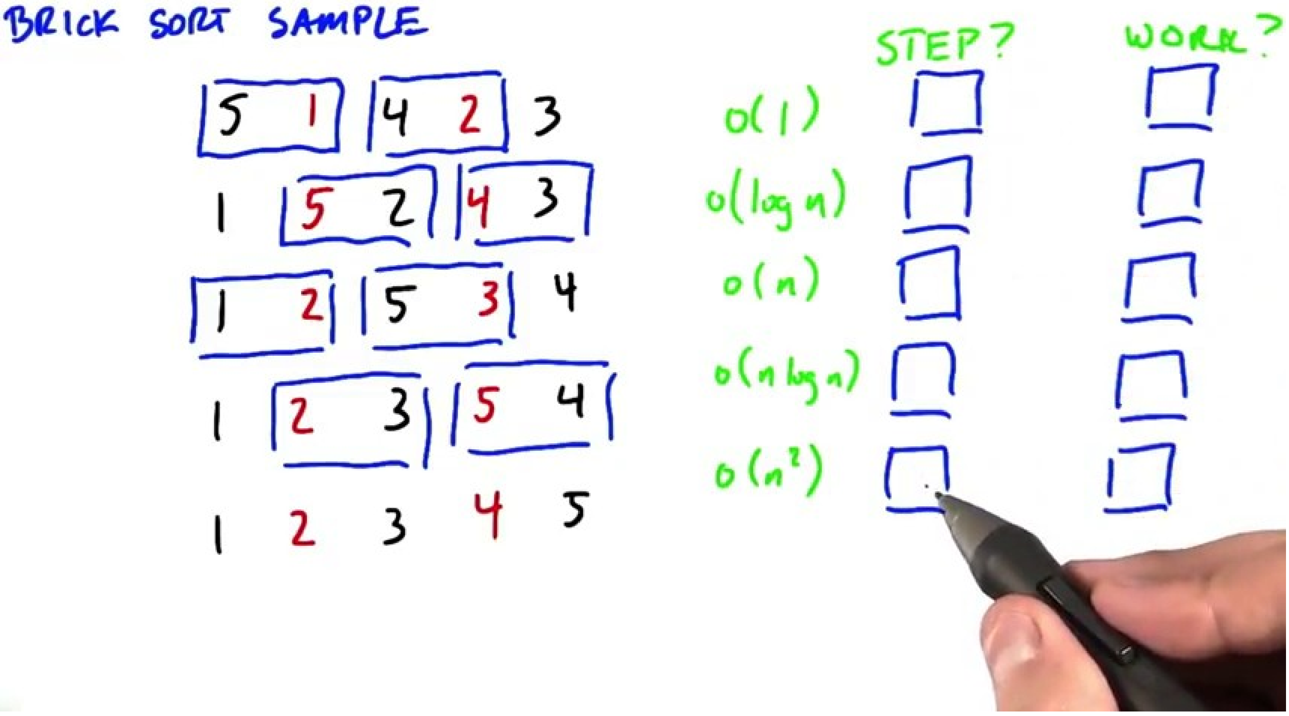
$[a\ b \mid c\ d\ e \mid f]$

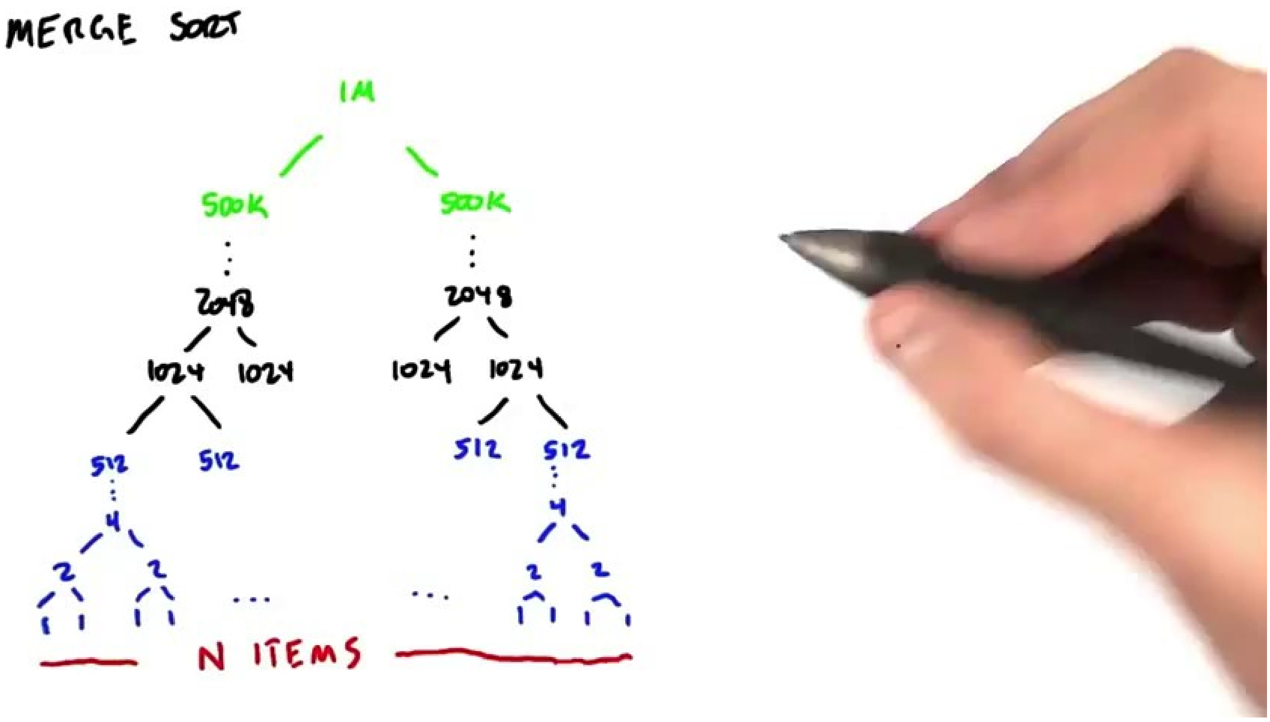
$[x\ z\ x\ y\ z\ y]$

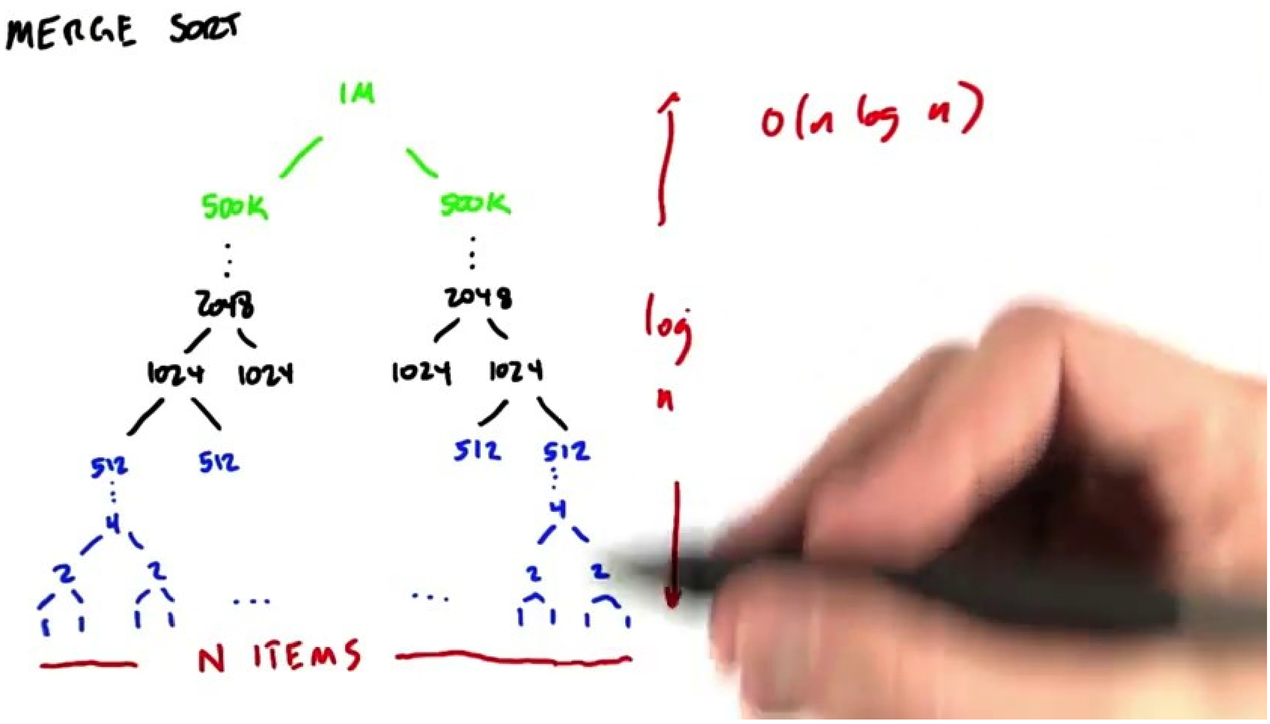
$[\underbrace{a\cdot x + b\cdot z}_{\text{OUT}(0)} \mid \underbrace{c\cdot x\ \ d\cdot y\ \ e\cdot z}_{\text{OUT}(1)} \mid \underbrace{f\cdot y}_{\text{OUT}(2)}]$

$$\begin{bmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + \cancel{0y} + bz \\ cx + dy + ez \\ \cancel{0x} + \cancel{0y} + fz \end{bmatrix}$$

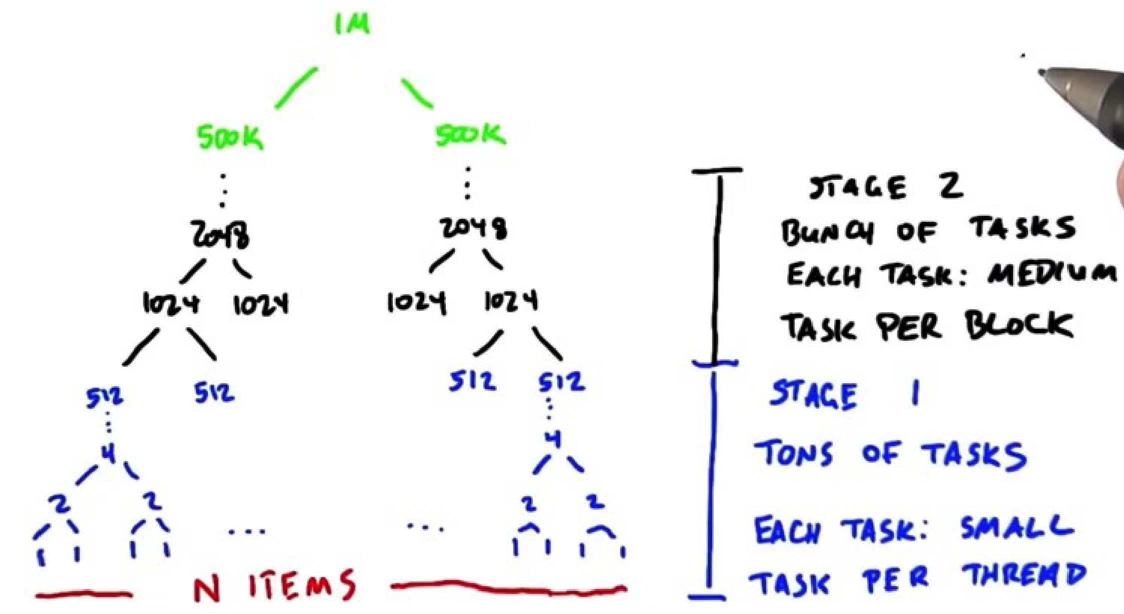

BRICK SORT SAMPLE
5 1 4 2 3
1 5 2 4 3
1 2 5 3 4


BRICK SORT SAMPLE
5 1 4 2 3
1 5 2 4 3
1 2 5 3 4
1 2 3 5 4
1 2 3 4 5
STEP?
WORK?
O(1)
O(log n)
O(n)
O(n log n)
O(n²)

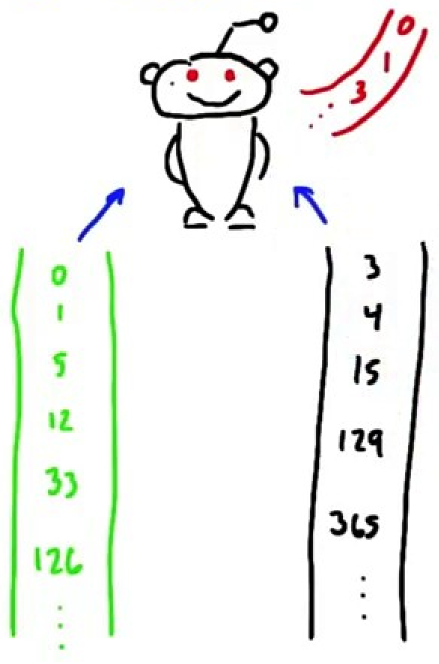

MERGE SORT
1M
500K
500K
2048
2048
1024 1024
1024 1024
512 512
512 512
4
4
2 2
2 2
1 1 1 1
1 1 1 1
N ITEMS


MERGE SORT
1M
500K
500K
O(n log n)
2048
2048
log n
1024 1024
1024 1024
512 512
512 512
4
4
2 2
2 2
1 1 1 1
1 1 1 1
N ITEMS

MERGE SORT



INTERMEDIATE MERGE: SERIAL ALGORITHM

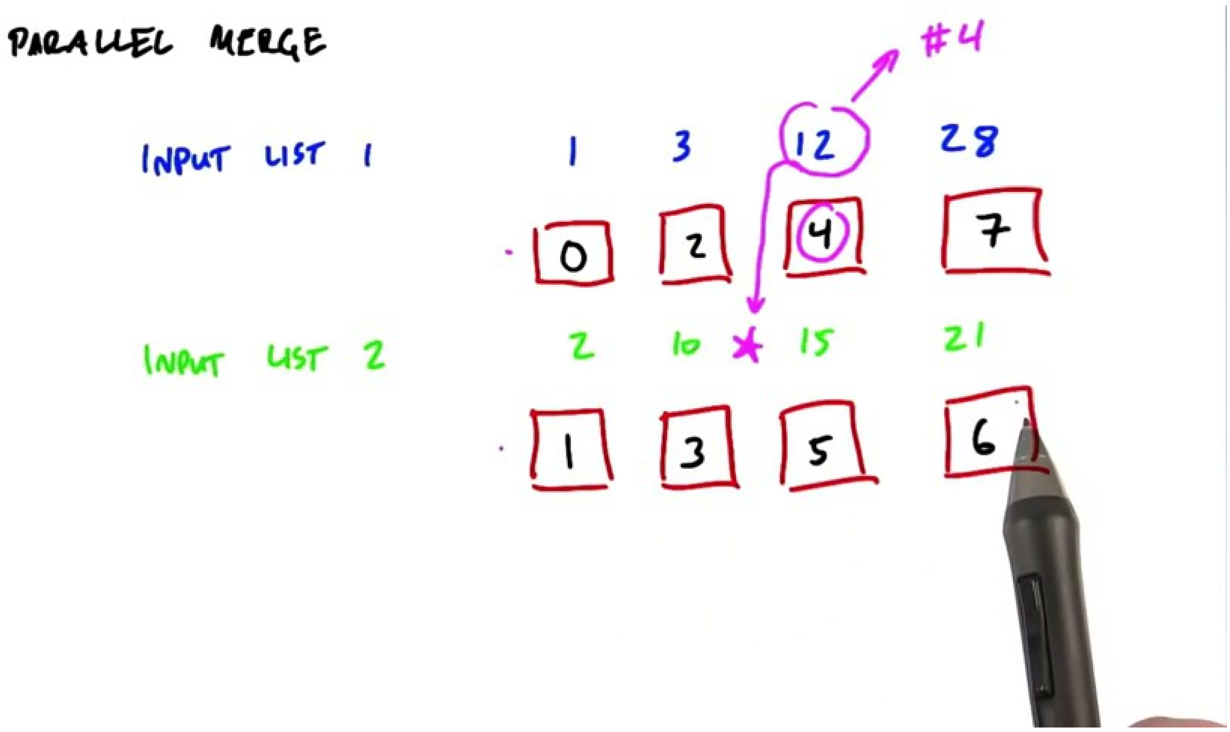


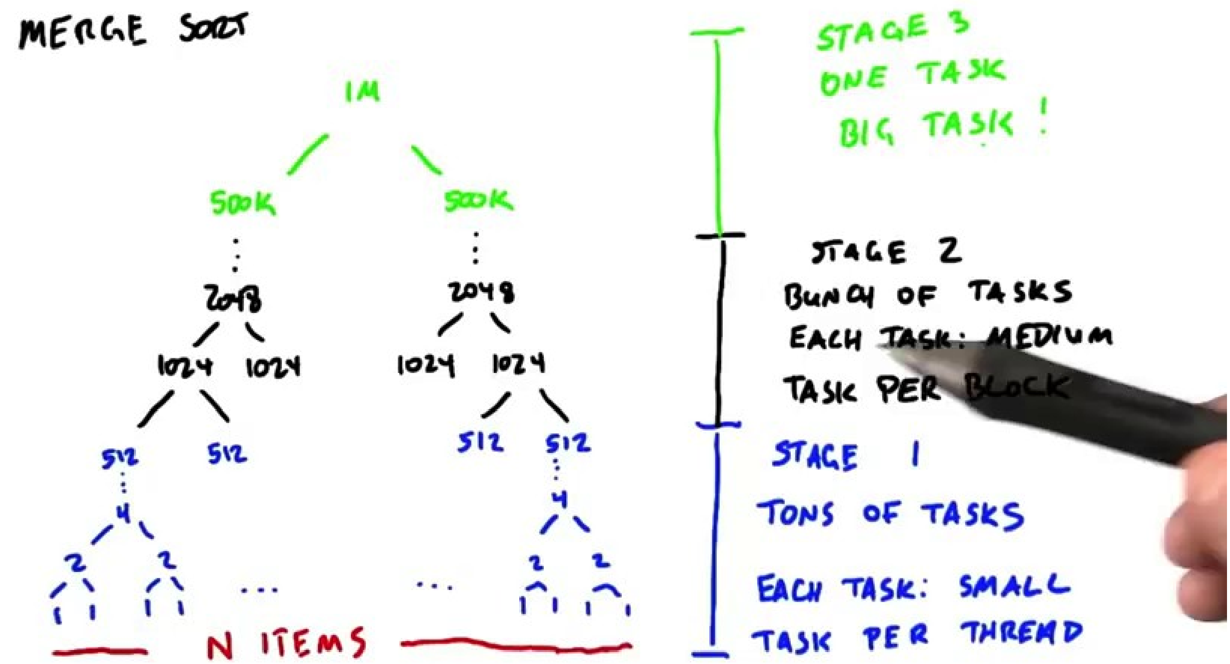
REVIEW: COMPACT

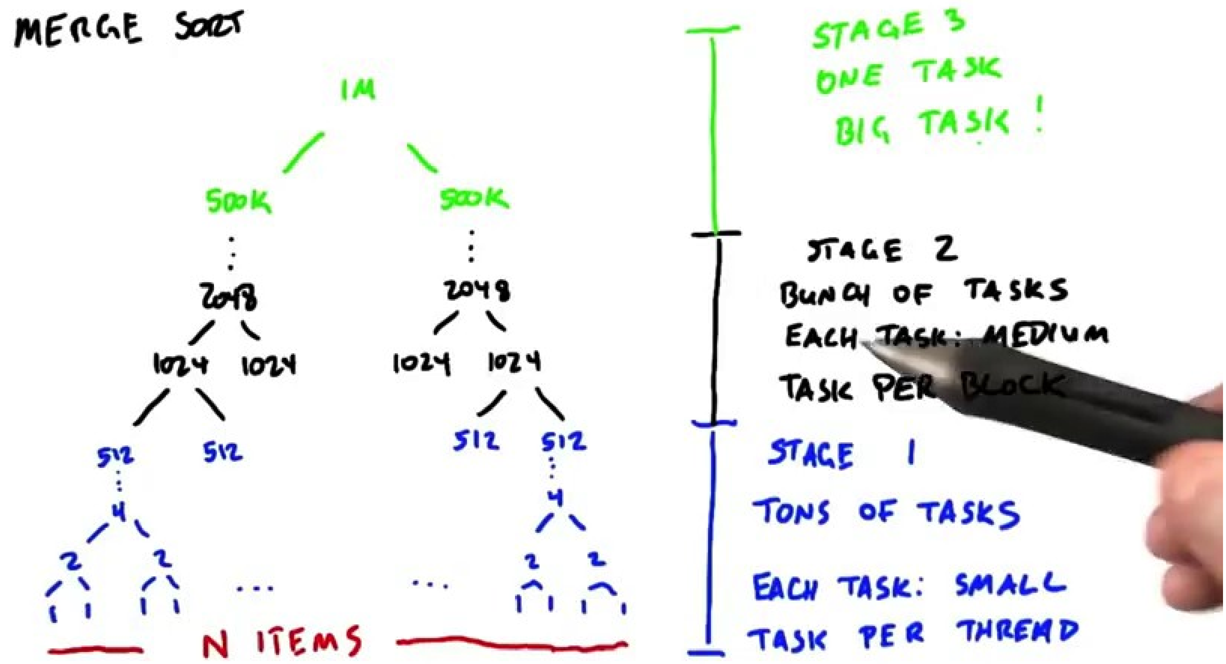
IN	1	1	2	3	5	8	13	21
PRED	T	T	F	T	T	F	T	T
SCATTER ADDRESS	0	1		2	3		4	5
OUT	1	1	3	5	13	21		

PARALLEL MERGE

INPUT LIST 1	1	3	12	28
INPUT LIST 2	2	10	15	21

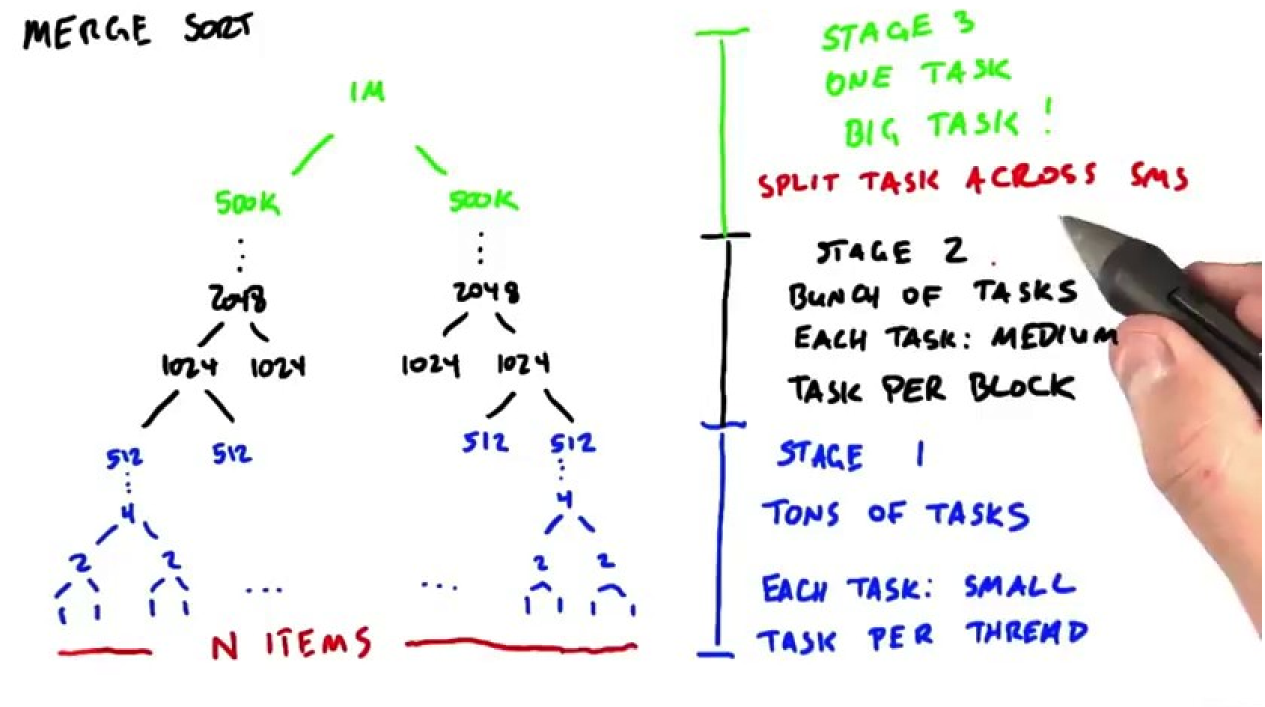

PARALLEL MERGE
#4
INPUT LIST 1
1 3 12 28
0 2 4 7
INPUT LIST 2
2 10 15 21
1 3 5 6

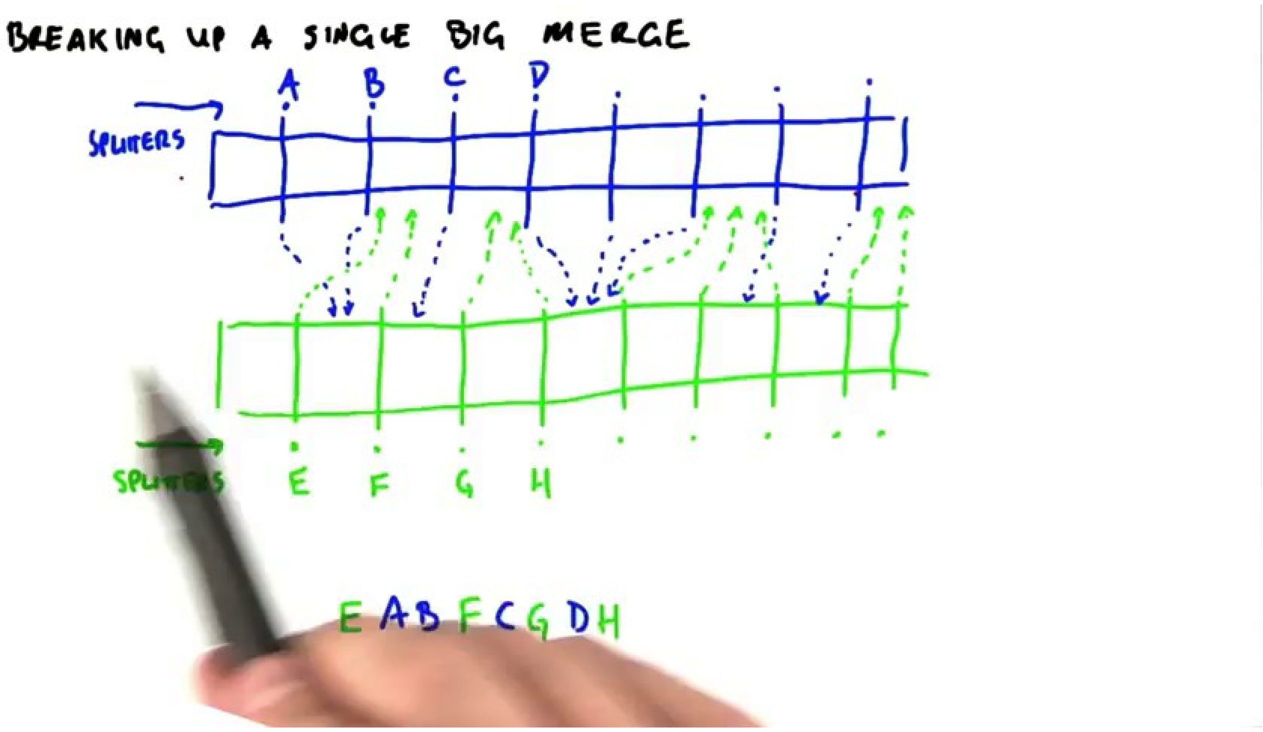

MERGE SORT
1M
500K 500K
2048 2048
1024 1024 1024 1024
512 512 512 512
4 4
2 2 2 2
N ITEMS
STAGE 3
ONE TASK
BIG TASK !
STAGE 2
BUNCH OF TASKS
EACH TASK: MEDIUM
TASK PER BLOCK
STAGE 1
TONS OF TASKS
EACH TASK: SMALL
TASK PER THREAD

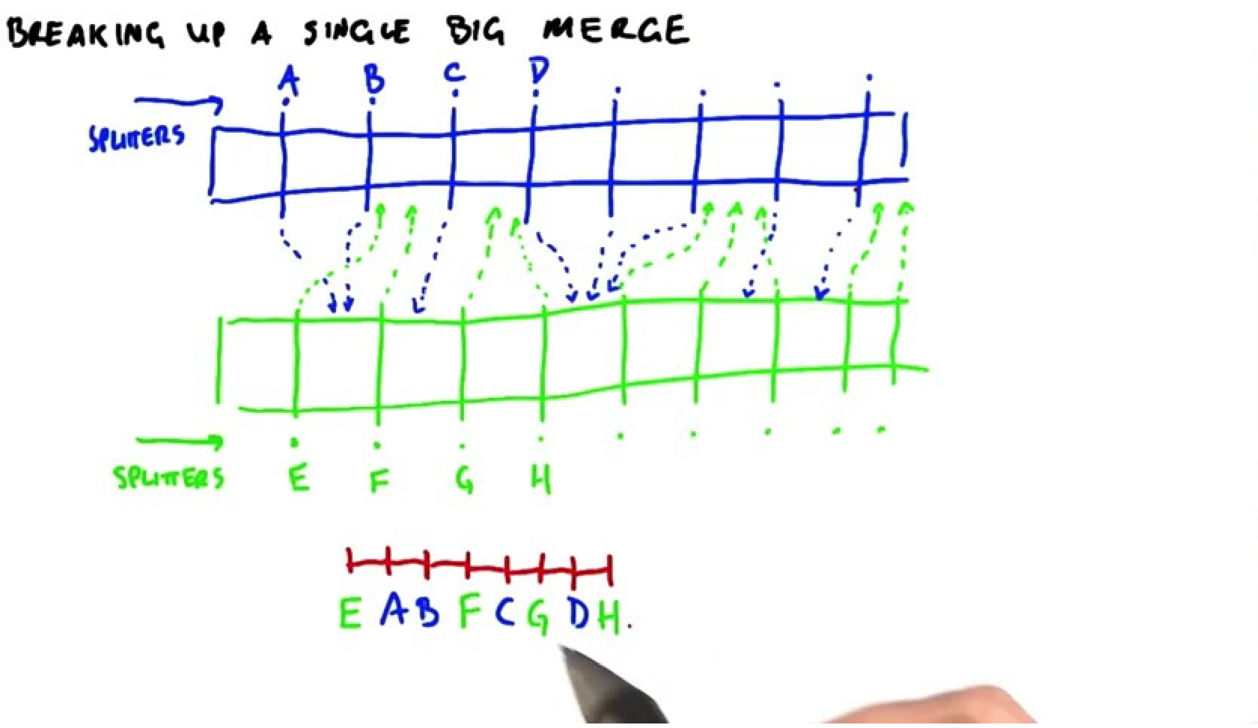


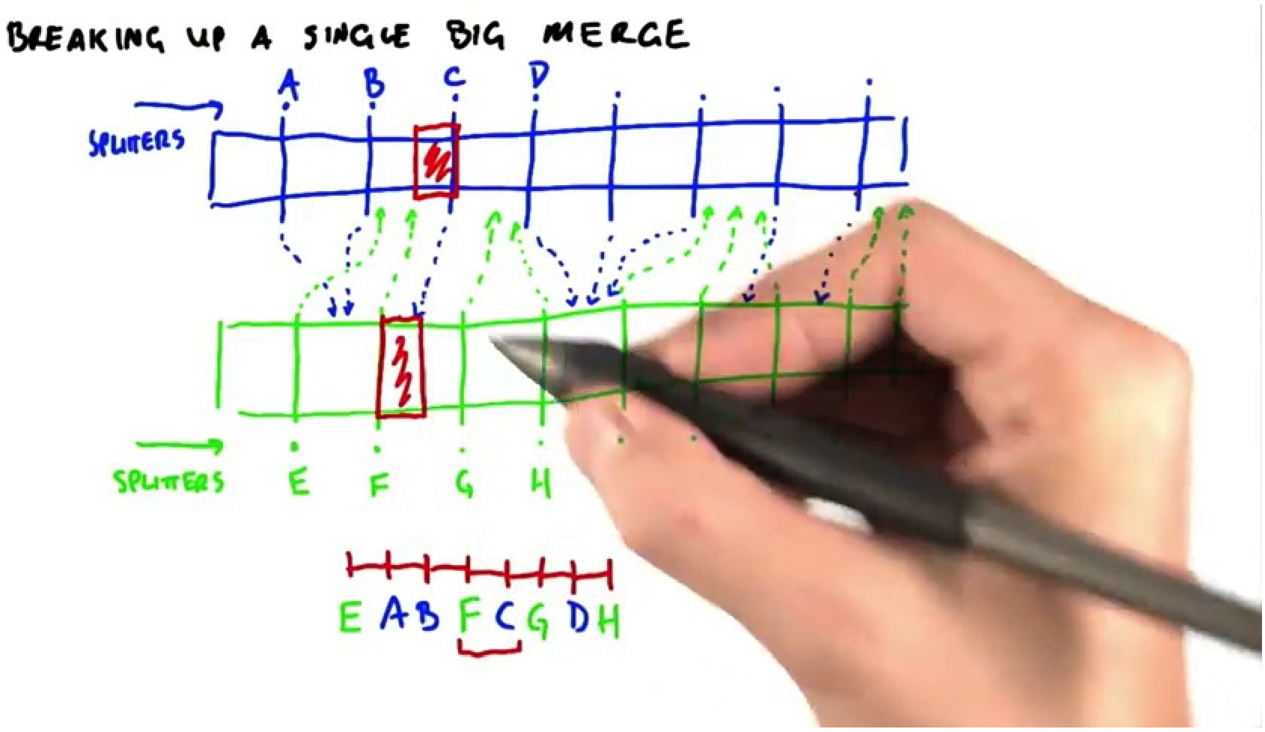
QUIZ: WHY IS IT BAD TO HAVE ONLY ONE MERGE TASK REMAINING?

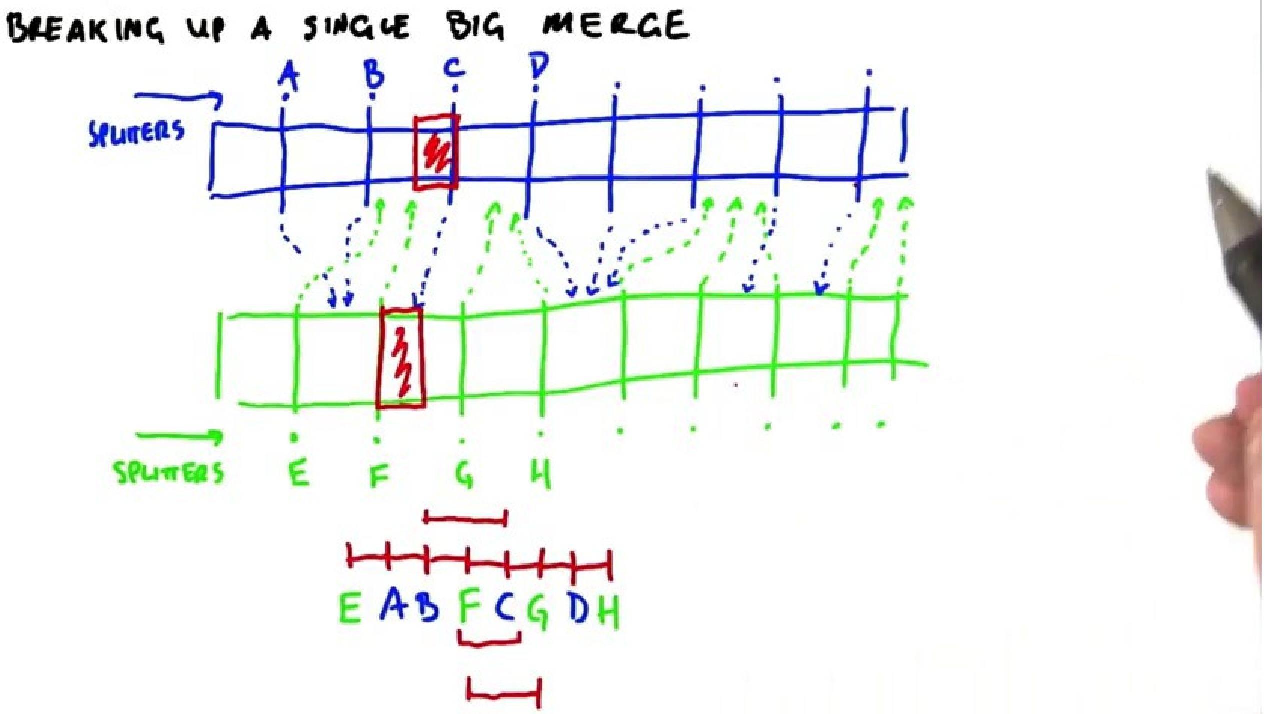
- [ ] LOTS OF THREADS IDLE PER SM
- [ ] LOTS OF SMS IDLE
- [ ] VERY HIGH BRANCH DIVERGENCE

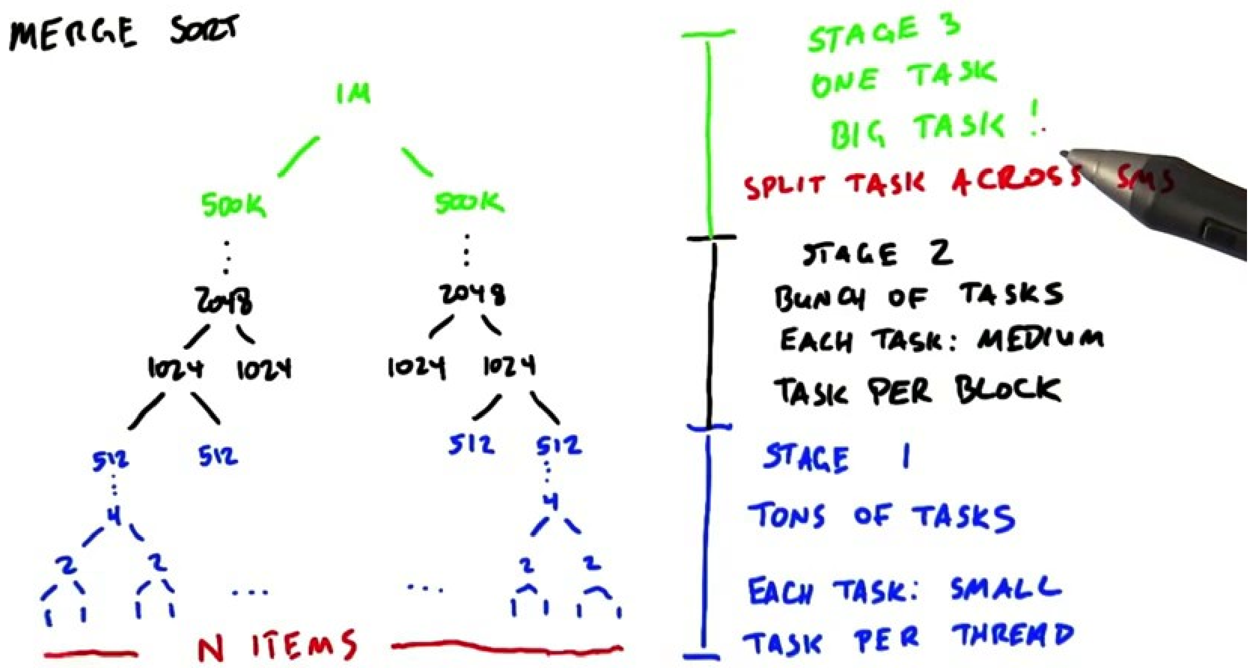

MERGE SORT
1M
500K
500K
2048
2048
1024
1024
1024
1024
512
512
512
512
4
4
2
2
2
2
N ITEMS
STAGE 3
ONE TASK
BIG TASK!
SPLIT TASK ACROSS SMS
STAGE 2
BUNCH OF TASKS
EACH TASK: MEDIUM
TASK PER BLOCK
STAGE 1
TONS OF TASKS
EACH TASK: SMALL
TASK PER THREAD

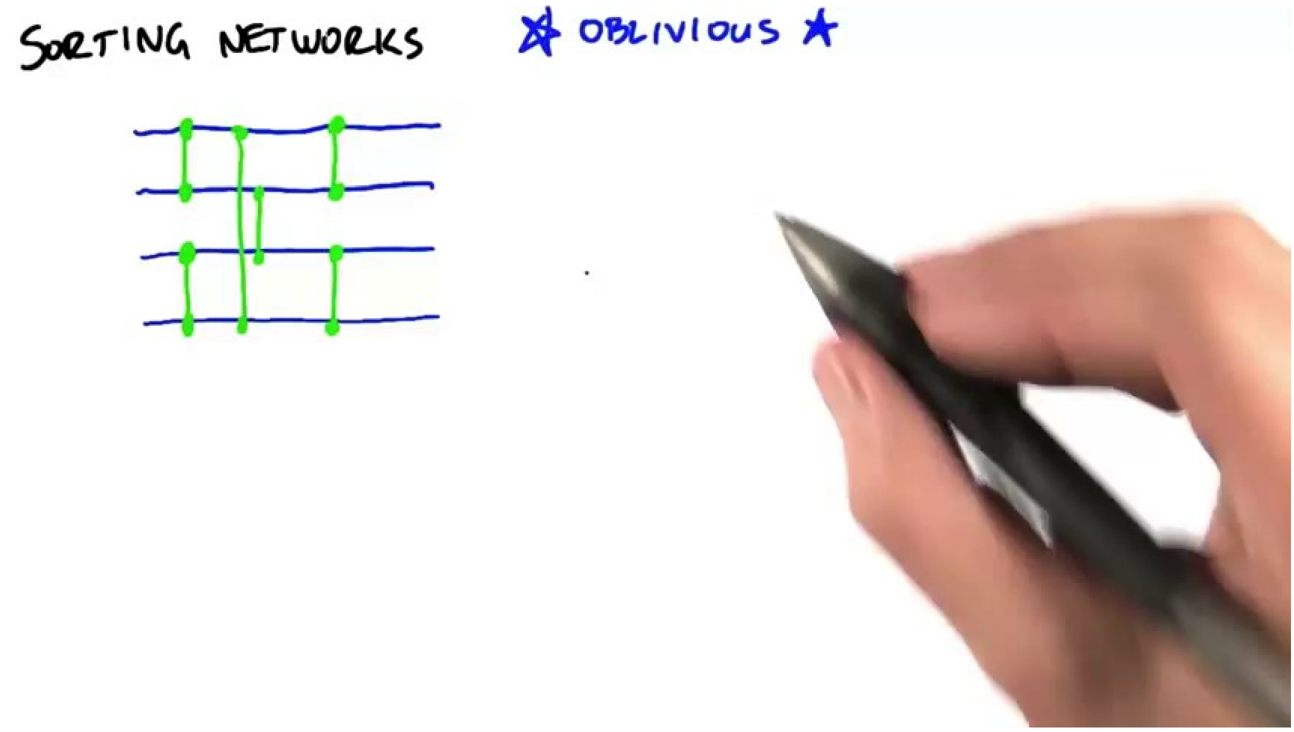

BREAKING UP A SINGLE BIG MERGE
A
B
C
D
SPLITTERS
SPLITTERS
E
F
G
H
E A B F C G D H

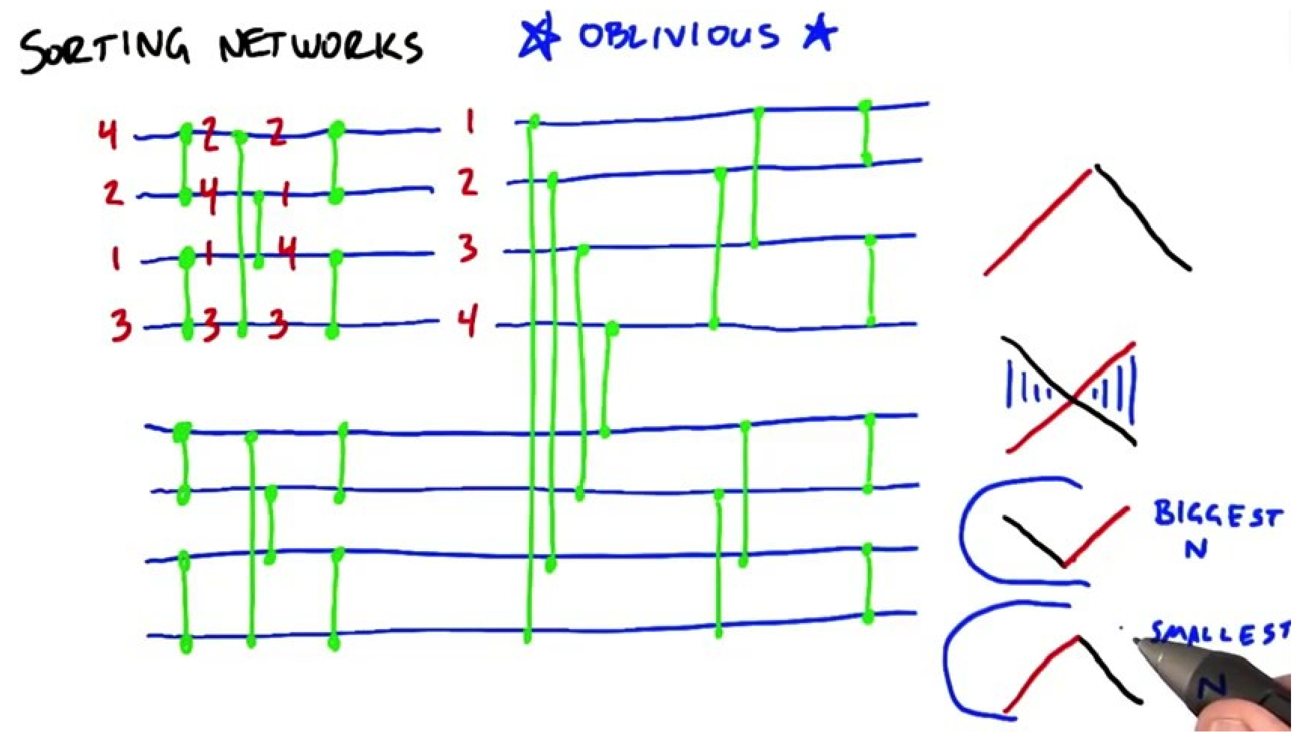

BREAKING UP A SINGLE BIG MERGE
A B C D
SPLITTERS
SPLITTERS
E F G H
E A B F C G D H

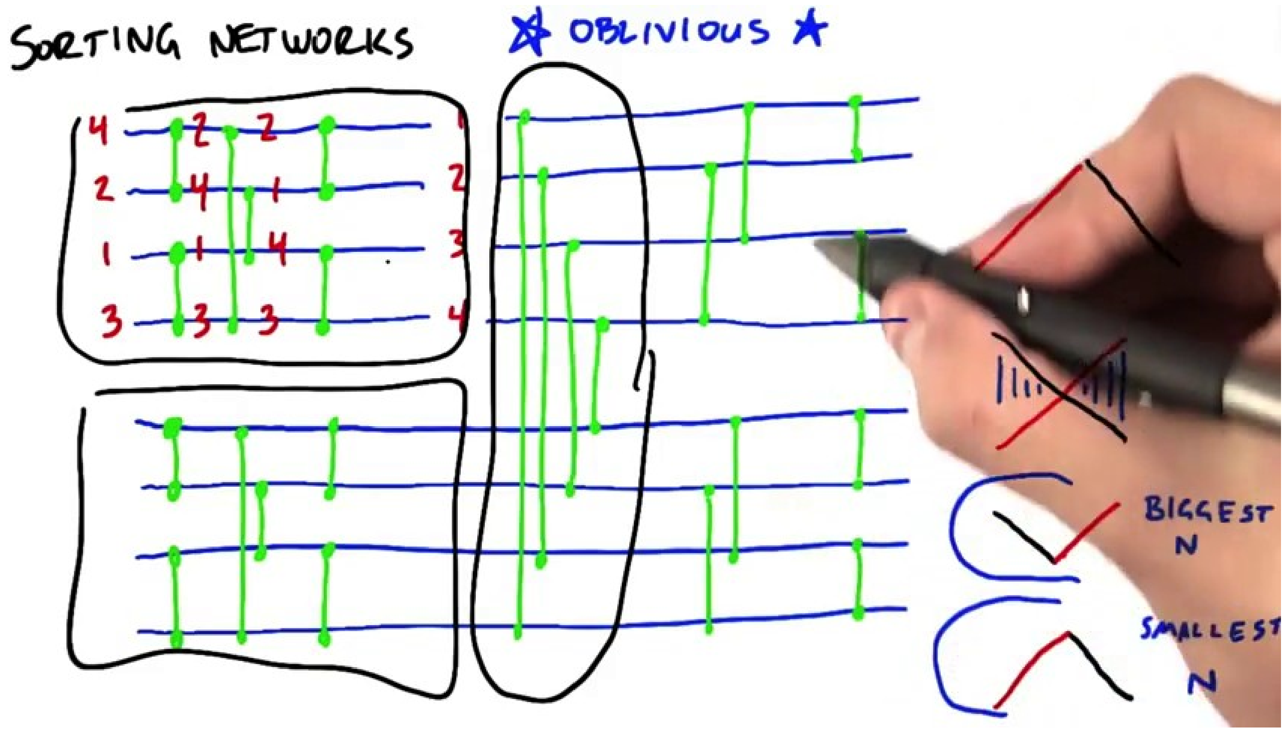

BREAKING UP A SINGLE BIG MERGE
A B C D
SPLITTERS
SPLITTERS
E F G H
E A B F C G D H

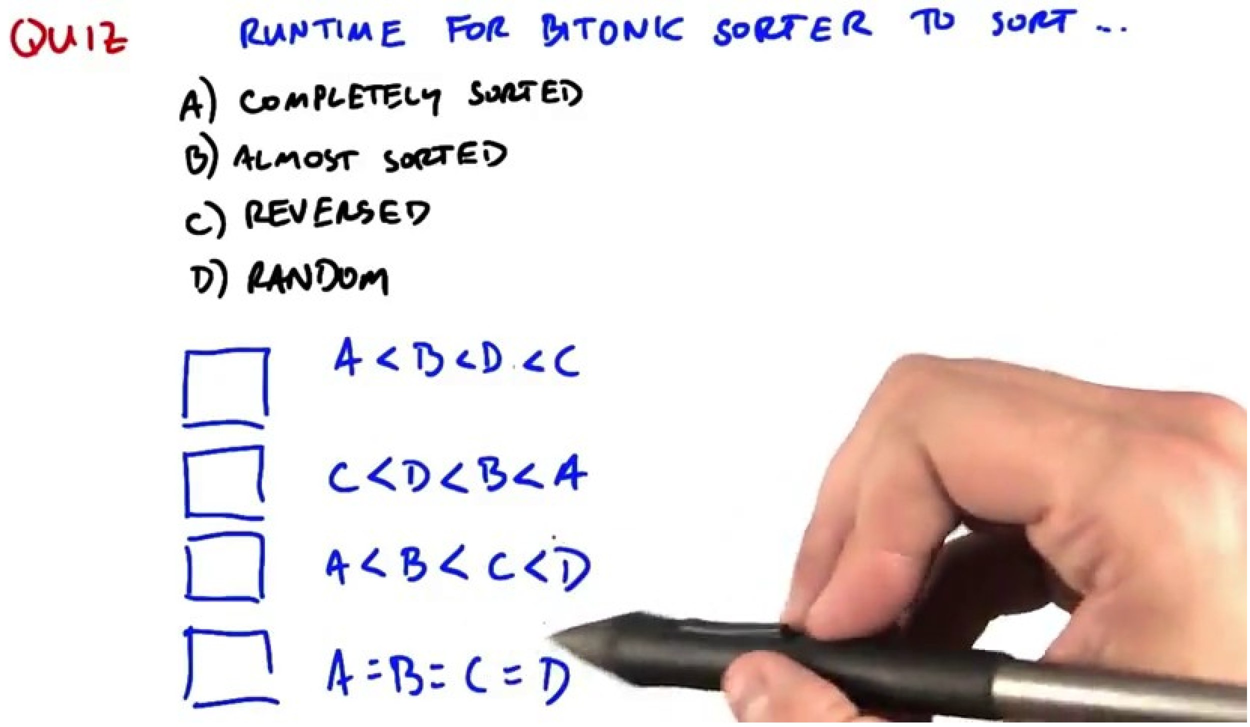

BREAKING UP A SINGLE BIG MERGE
A
B
C
D
SPLITTERS
SPLITTERS
E
F
G
H
E A B F C G D H

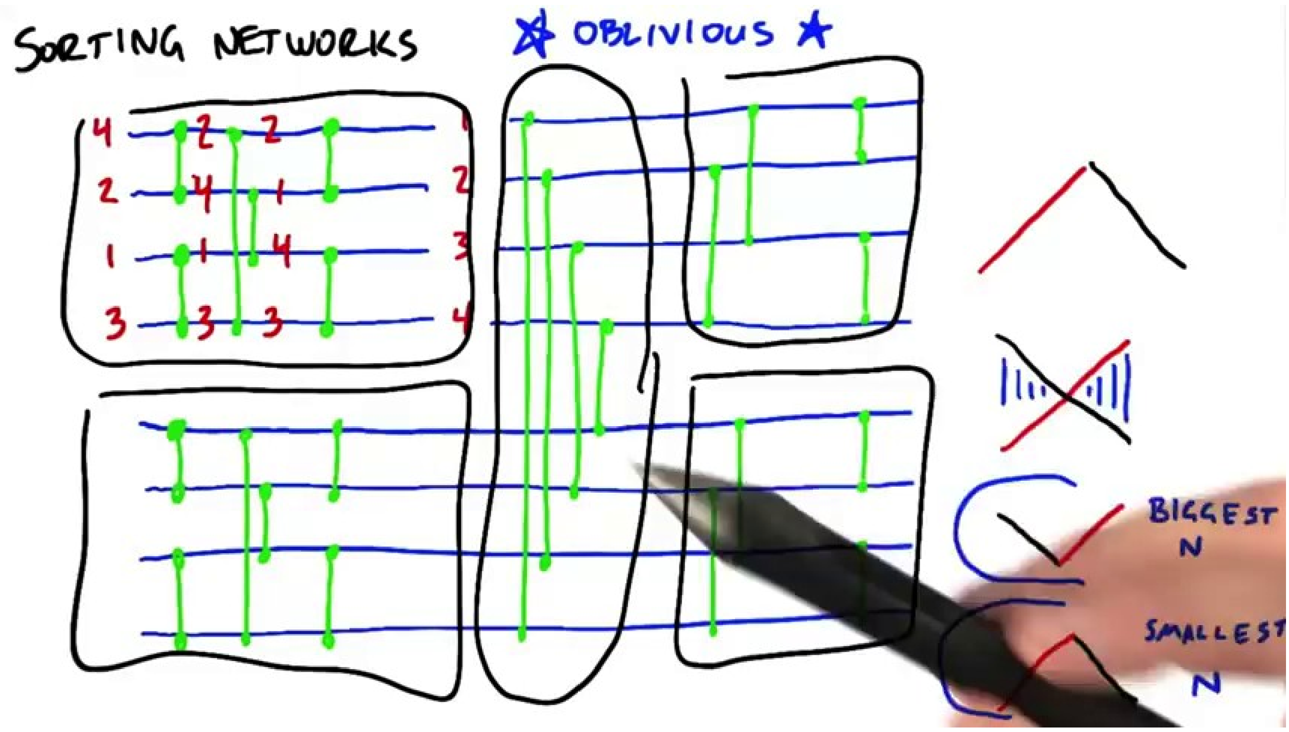

MERGE SORT
1M
500K
500K
2048
2048
1024
1024
1024
1024
512
512
512
512
4
4
2
2
2
2
N ITEMS
STAGE 3
ONE TASK
BIG TASK !
SPLIT TASK ACROSS SMS
STAGE 2
BUNCH OF TASKS
EACH TASK: MEDIUM
TASK PER BLOCK
STAGE 1
TONS OF TASKS
EACH TASK: SMALL
TASK PER THREAD

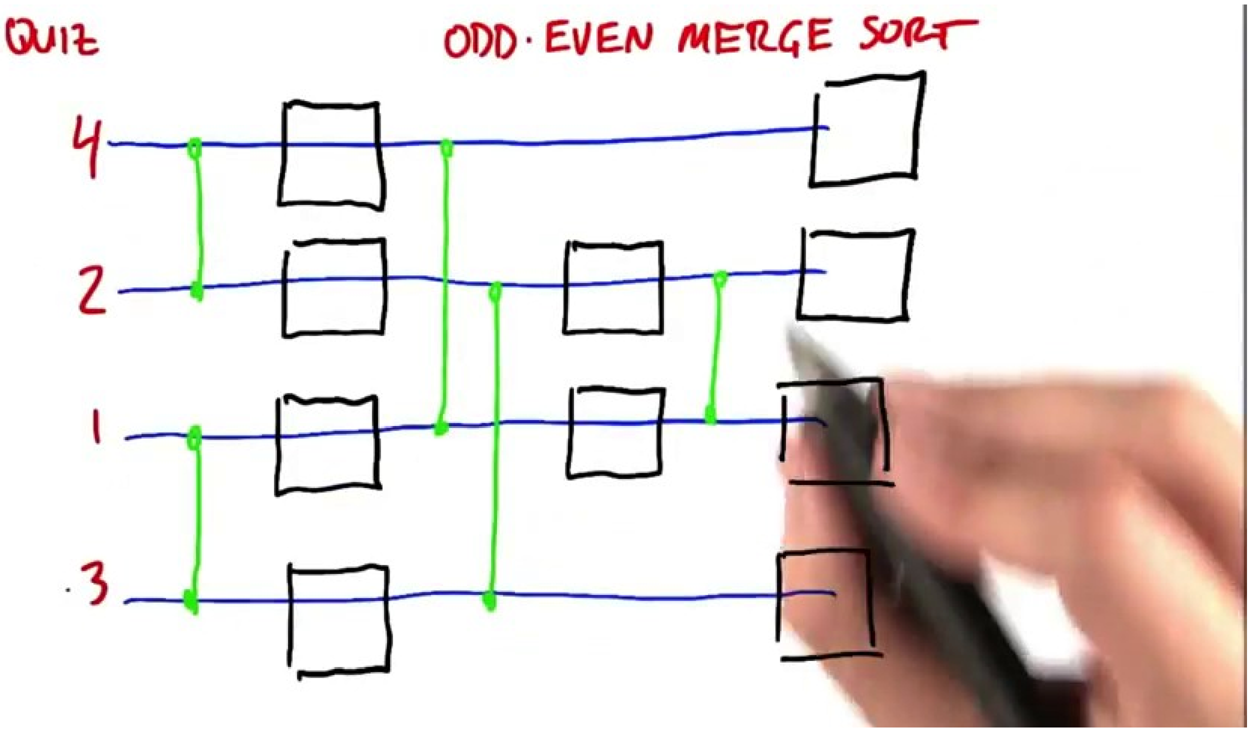

SORTING NETWORKS
OBLIVIOUS


SORTING NETWORKS
OBLIVIOUS
4 2 2
2 4 1
1 1 4
3 3 3
1
2
3
4
BIGGEST N
SMALLEST N


SORTING NETWORKS
OBLIVIOUS
4 2 1 3
2 2
4 1
1 4
3 3
1 2 3 4
BIGGEST N
SMALLEST N


QUIZ
RUNTIME FOR BITONIC SORTER TO SORT ..
A) COMPLETELY SORTED
B) ALMOST SORTED
C) REVERSED
D) RANDOM
A < B < D < C
C < D < B < A
A < B < C < D
A = B = C = D


SORTING NETWORKS
OBLIVIOUS
BIGGEST N
SMALLEST N


QUIZ
ODD·EVEN MERGE SORT

RADIX SORT

1 START WITH LSB

2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.

3 MOVE TO NEXT MSB, REPEAT

RADIX SORT

1 START WITH LSB

2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.

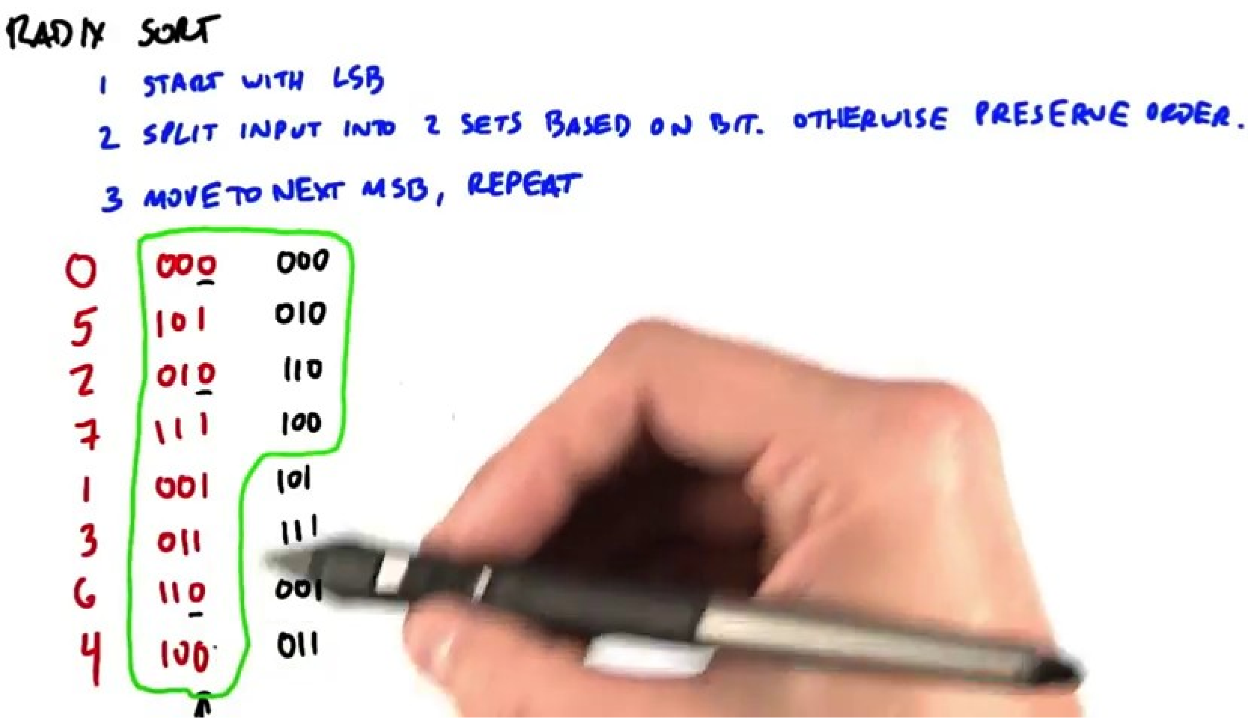
3 MOVE TO NEXT MSB, REPEAT

0	000	000	000	000
5	101	010	100	001
2	010	110	101	010
7	111	100	001	011
1	001	101	010	100
3	011	111	110	101
6	110	001	111	110
4	100	011	011	111

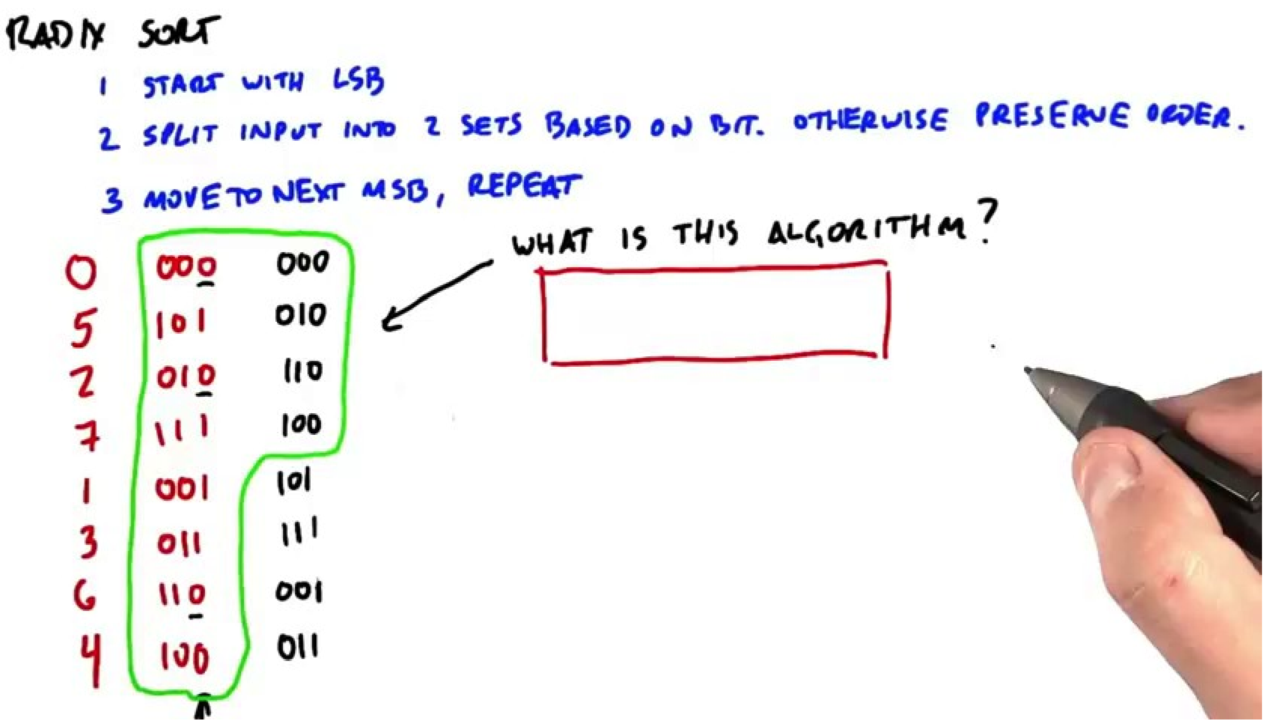
$O(kn)$

k: bits in representation

n: items to sort



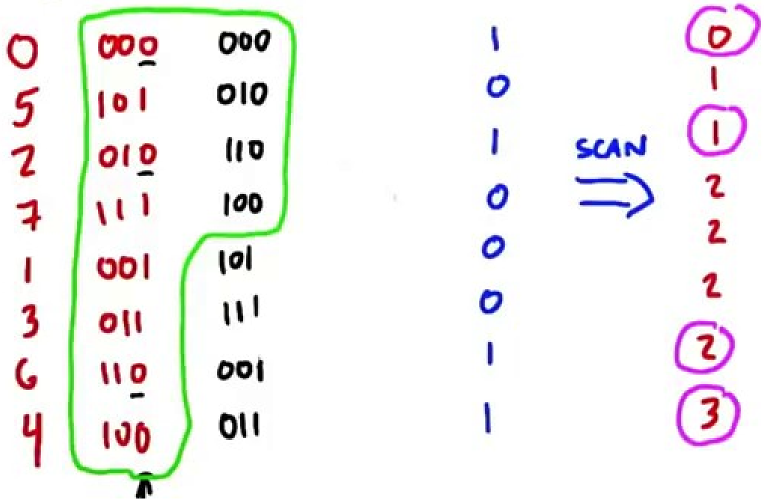
RADIX SORT
1 START WITH LSB
2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
3 MOVE TO NEXT MSB, REPEAT
0 000 000
5 101 010
2 010 110
7 111 100
1 001 101
3 011 111
6 110 001
4 100 011



RADIX SORT
1 START WITH LSB
2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
3 MOVE TO NEXT MSB, REPEAT
0 000 000
5 101 010
2 010 110
7 111 100
1 001 101
3 011 111
6 110 001
4 100 011
WHAT IS THIS ALGORITHM?

RADIX SORT

1 START WITH LSB
2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
3 MOVE TO NEXT MSB, REPEAT



QUICKSORT

1 CHOOSE PIVOT ELEMENT
2 COMPARE ALL ELEMENTS VS PIVOT
3 SPLIT INTO 3 ARRAYS: <P =P >P
4 RECURSE ON EACH ARRAY

QUICKSORT

1 CHOOSE PIVOT ELEMENT
2 COMPARE ALL ELEMENTS VS PIVOT
3 SPLIT INTO 3 ARRAYS: <P =P >P
4 RECURSE ON EACH ARRAY

3	5	2	4	1	P = 3
=	>	<	>	<	

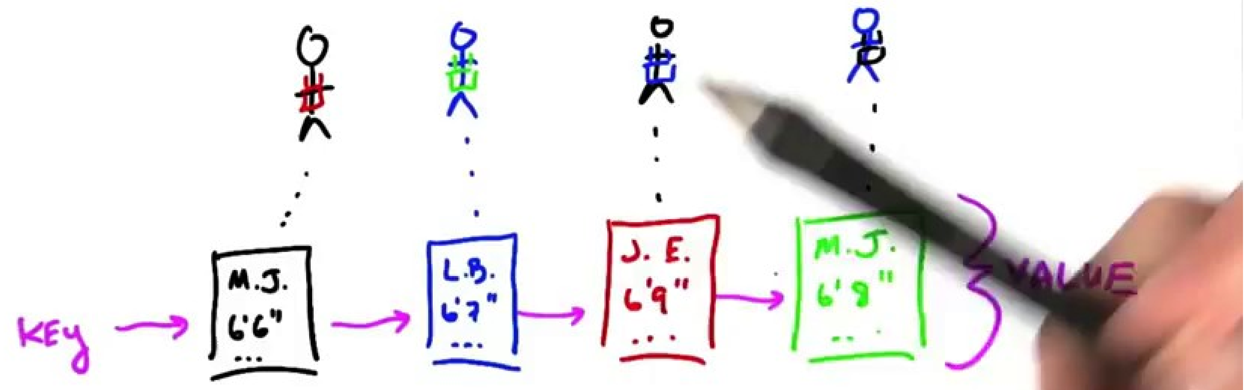
2 1	3	5 4
1 2	3	

QUICKSORT + SEGMENTS

	[3	5	2	4	1]
DISTRIBUTE (SEG.)	3	3	3	3	3
MAP	=	>	<	>	<
COMPACT <	2	1			
COMPACT =			3		
COMPACT >				5	4

[2 1 | 3 | 5 4]

KEY VALUE SORTS



SUMMARY : WHAT WE LEARNED

- COMPACT
- ALLOCATE
- SEGMENTED SCAN
- ODD EVEN SORT
- MERGE SORT
- SORTING NETWORKS
- QUICK SORT
- RADIX SORT