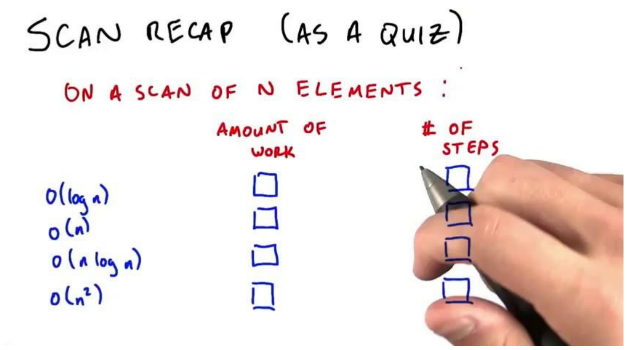
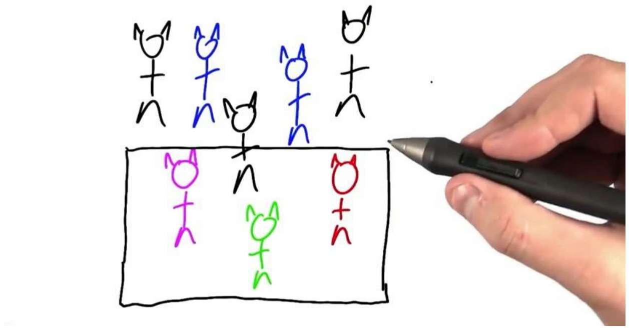
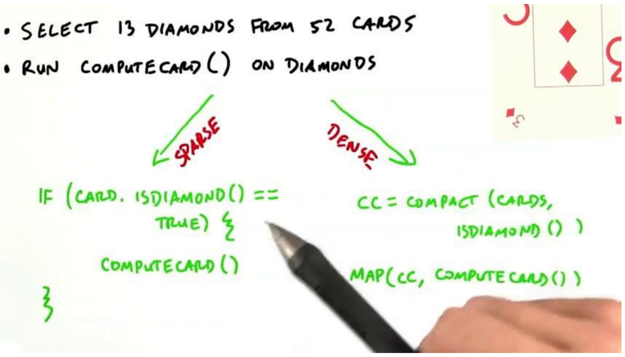


SCAN RECAP (AS A QUIZ)

ON A SCAN OF N ELEMENTS:

	AMOUNT OF WORK	# OF STEPS
O(log n)	□	□
O(n)	□	□
O(n log n)	□	□
O(n²)	□	□

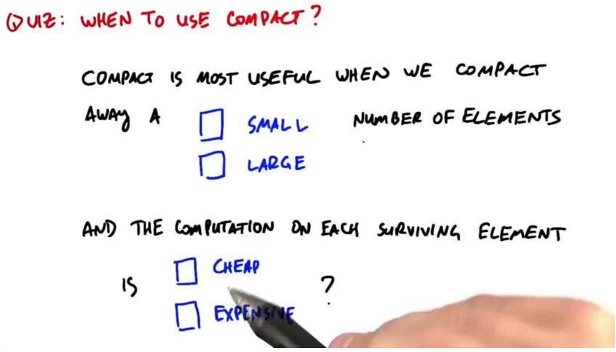




- SELECT 13 DIAMONDS FROM 52 CARDS
- RUN COMPUTECARD() ON DIAMONDS

```
IF (CARD.ISDIAMOND() ==
        TRUE) {
    COMPUTECARD()
}
```

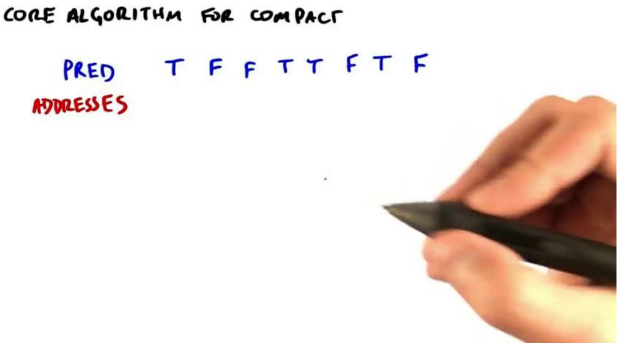
```
CC = COMPACT(CARDS,
        ISDIAMOND() )
MAP(CC, COMPUTECARD() )
```



QUIZ: WHEN TO USE COMPACT?

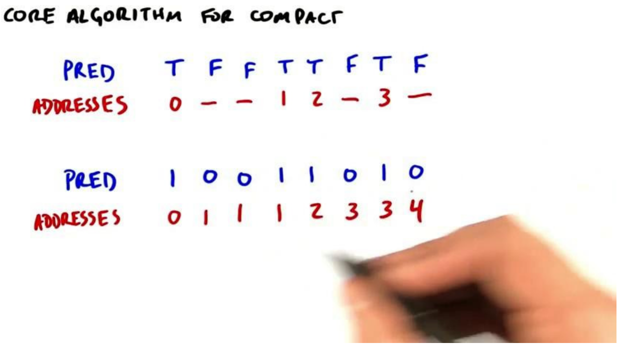
COMPACT IS MOST USEFUL WHEN WE COMPACT AWAY A □ SMALL □ LARGE NUMBER OF ELEMENTS

AND THE COMPUTATION ON EACH SURVIVING ELEMENT IS □ CHEAP □ EXPENSIVE ?



CORE ALGORITHM FOR COMPACT

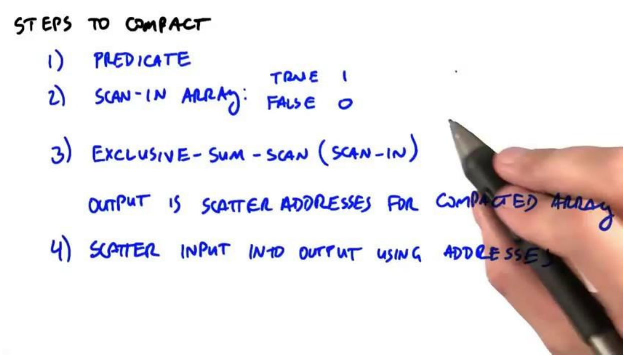
PRED	T	F	F	T	T	F	T	F
ADDRESSES								



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



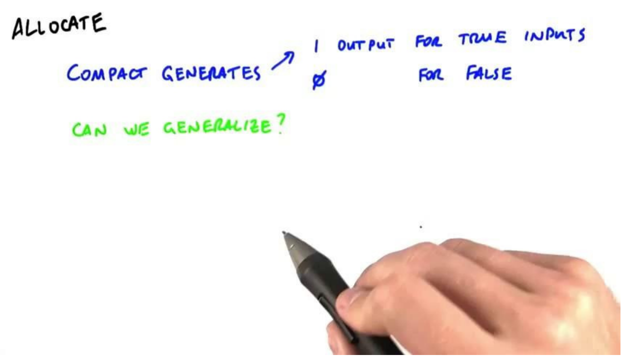
STEPS TO COMPACT

1) PREDICATE
2) SCAN-IN ARRAY: TRUE 1 / FALSE 0

3) EXCLUSIVE-SUM-SCAN (SCAN-IN)

OUTPUT IS SCATTER ADDRESSES FOR COMPACTED ARRAY

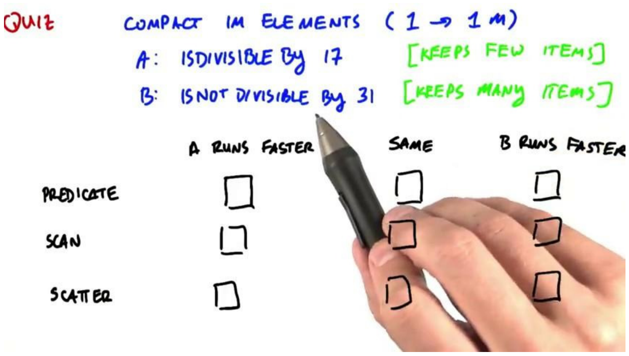
4) SCATTER INPUT INTO OUTPUT USING ADDRESSES



ALLOCATE

COMPACT GENERATES ↗ 1 OUTPUT FOR TRUE INPUTS
↘ ∅ FOR FALSE

CAN WE GENERALIZE?

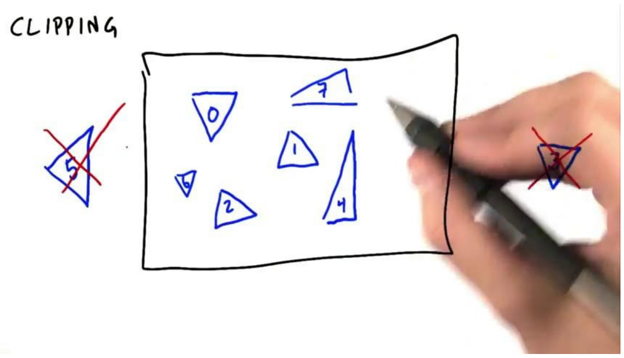


QUIZ COMPACT 1M ELEMENTS (1 → 1 M)

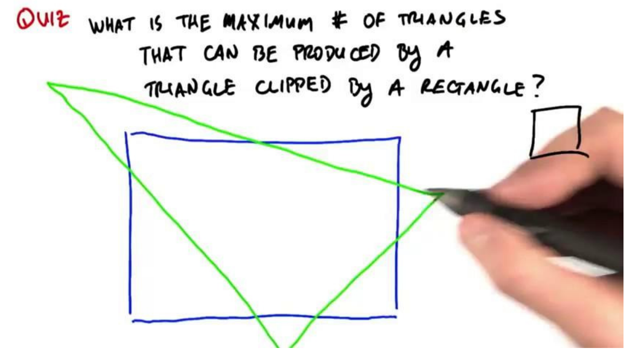
A: ISDIVISIBLE BY 17 [KEEPS FEW ITEMS]

B: ISNOT DIVISIBLE BY 31 [KEEPS MANY ITEMS]

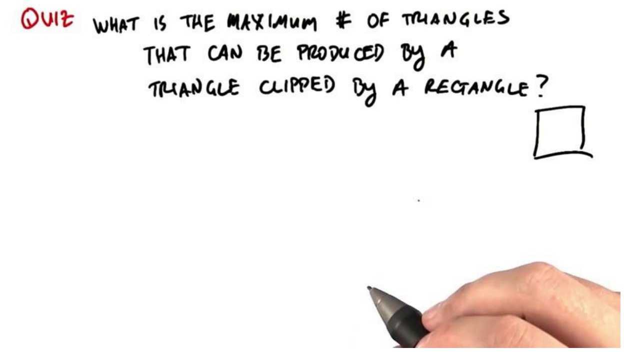
	A RUNS FASTER	SAME	B RUNS FASTER
PREDICATE	□	□	□
SCAN	□	□	□
SCATTER	□	□	□



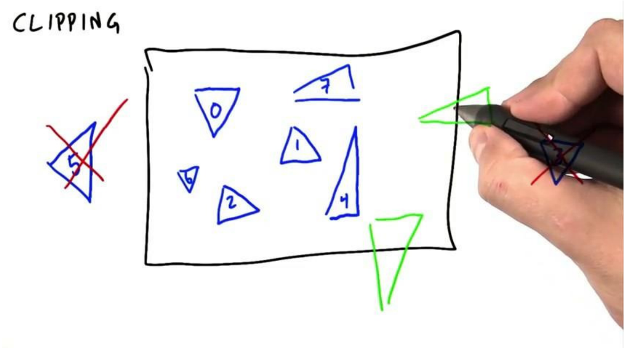
CLIPPING



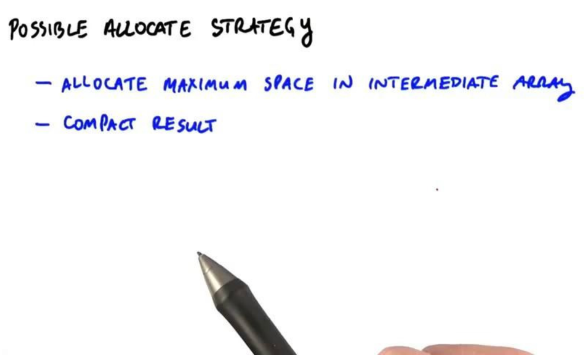
QUIZ WHAT IS THE MAXIMUM # OF TRIANGLES THAT CAN BE PRODUCED BY A TRIANGLE CLIPPED BY A RECTANGLE? □



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

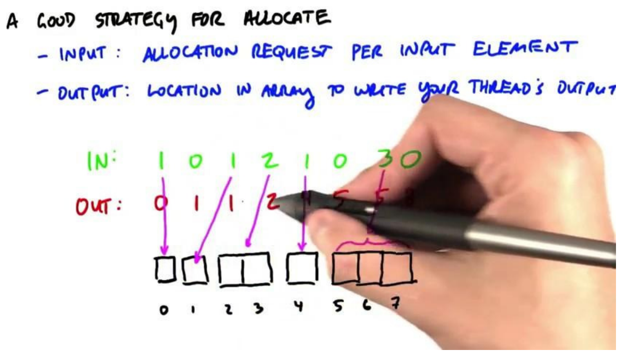


CLIPPING



POSSIBLE ALLOCATE STRATEGY

- ALLOCATE MAXIMUM SPACE IN INTERMEDIATE ARRAY
- COMPACT RESULT

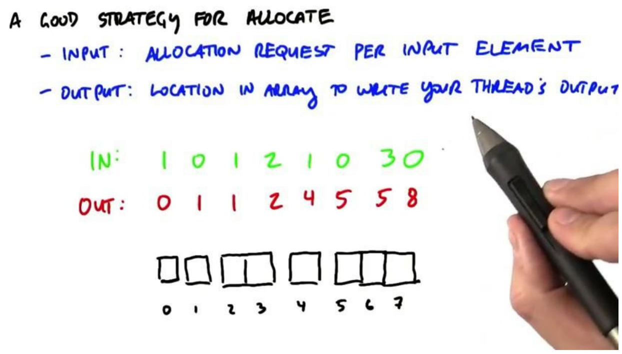


A GOOD STRATEGY FOR ALLOCATE

- 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

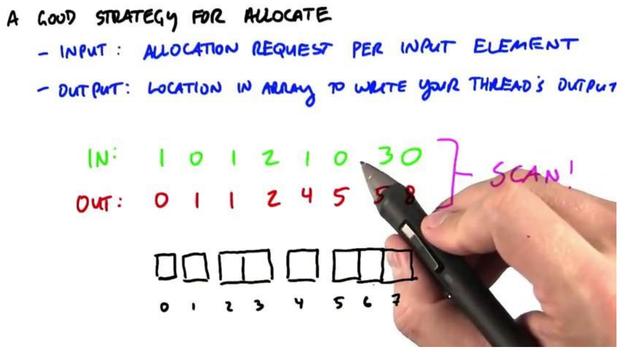


A GOOD STRATEGY FOR ALLOCATE

- 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



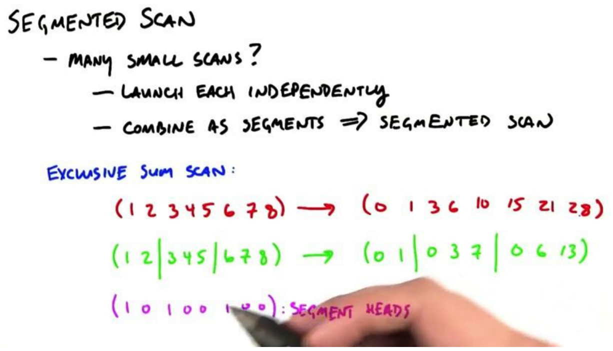
A GOOD STRATEGY FOR ALLOCATE

- 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!



SEGMENTED SCAN

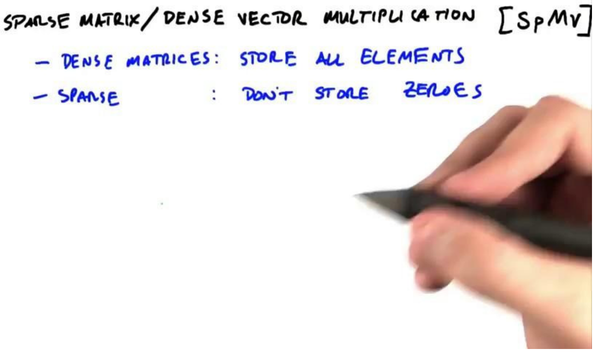
- MANY SMALL SCANS?
  - LAUNCH EACH INDEPENDENTLY
  - COMBINE AS SEGMENTS ⇒ SEGMENTED SCAN

EXCLUSIVE SUM SCAN:

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

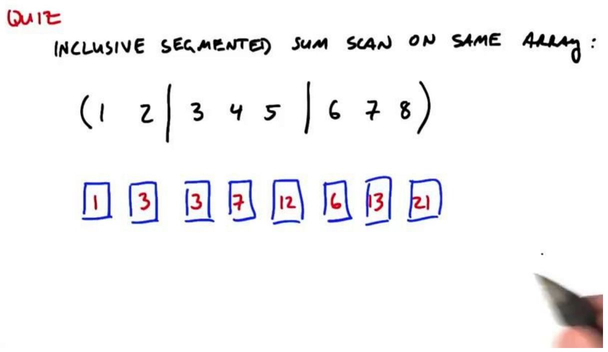
(1 2 | 3 4 5 | 6 7 8) → (0 1 | 0 3 7 | 0 6 13)

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



SPARSE MATRIX/DENSE VECTOR MULTIPLICATION [SpMV]

- DENSE MATRICES: STORE ALL ELEMENTS
- SPARSE : DON'T STORE ZEROES

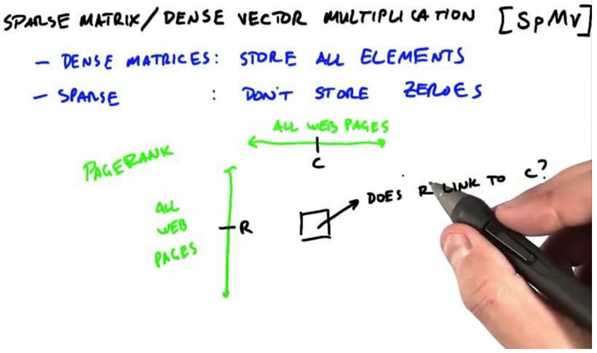


QUIZ

INCLUSIVE SEGMENTED SUM SCAN ON SAME ARRAY:

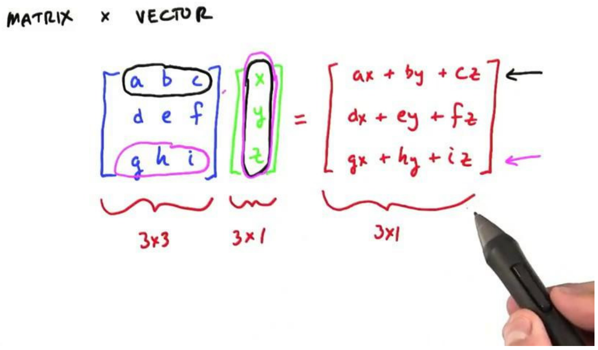
(1 2 | 3 4 5 | 6 7 8)

1 3 3 7 12 6 13 21



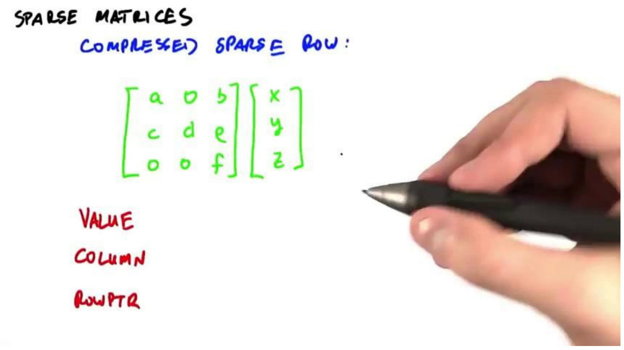
SPARSE MATRIX/DENSE VECTOR MULTIPLICATION [SpMV]

- DENSE MATRICES: STORE ALL ELEMENTS
- SPARSE : DON'T STORE ZEROES



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

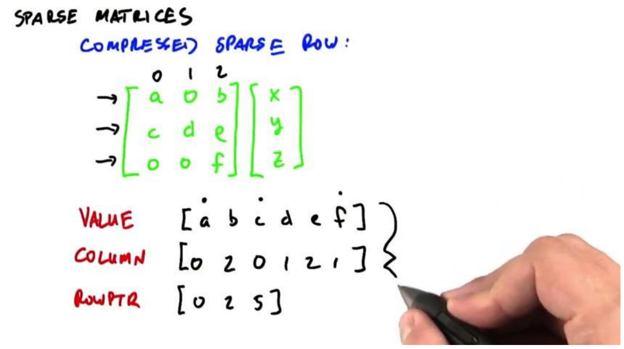
COMPRESSED SPARSE ROW:

$$\begin{bmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

VALUE

COLUMN

ROWPTR



SPARSE MATRICES

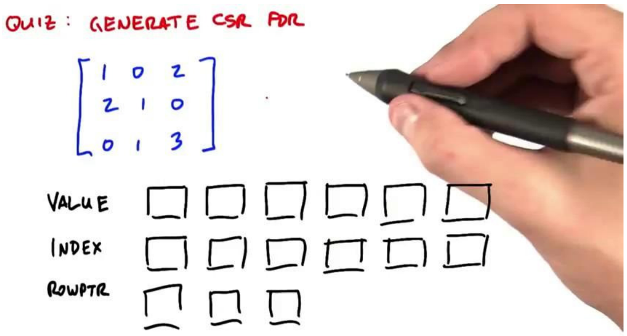
COMPRESSED SPARSE ROW:

$$\begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} \end{array}$$

VALUE [a b c d e f]

COLUMN [0 2 0 1 2 1]

ROWPTR [0 2 5]



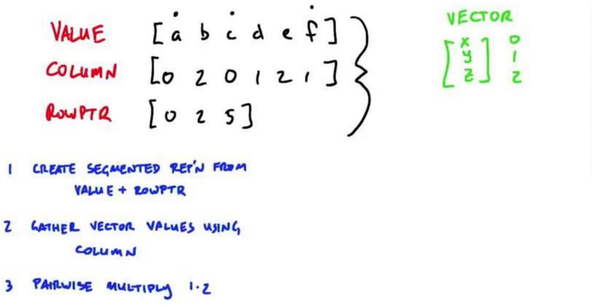
QUIZ: GENERATE CSR FOR

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

VALUE □ □ □ □ □ □

INDEX □ □ □ □ □ □

ROWPTR □ □ □



VALUE [a b c d e f]

COLUMN [0 2 0 1 2 1]

ROWPTR [0 2 5]

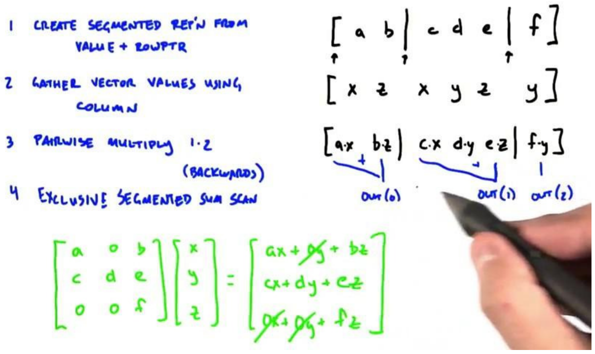
VECTOR

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

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 | c d e | f]

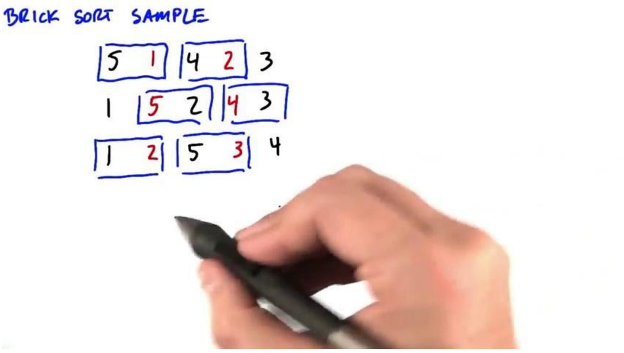
[x z x y z y]

[a·x b·z | c·x d·y e·z | f·y]

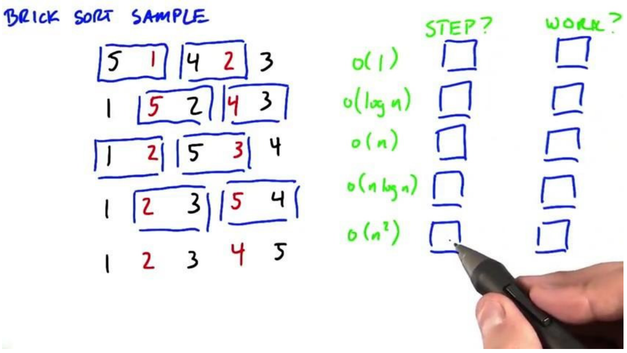
out(0) out(1) out(2)

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

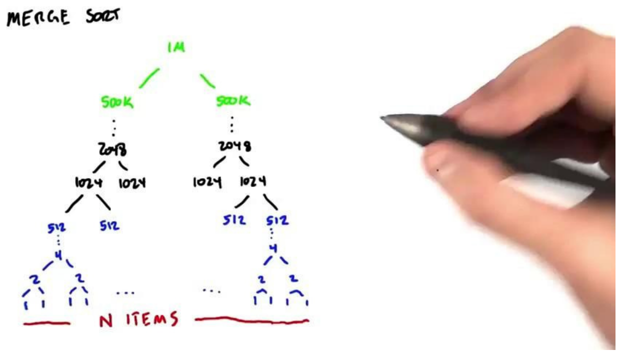
BRICK SORT SAMPLE



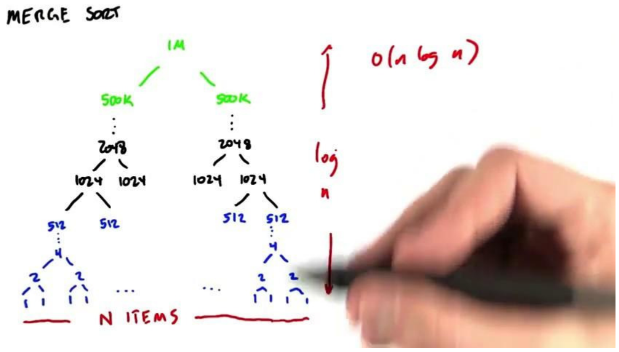
BRICK SORT SAMPLE



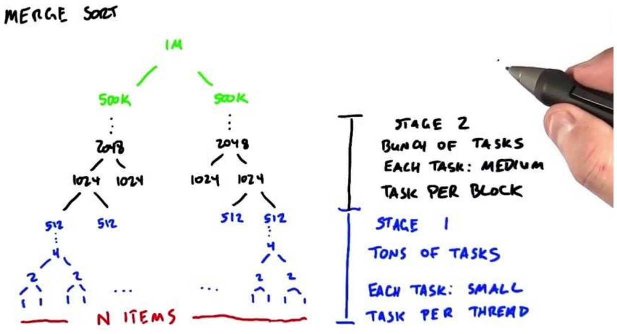
MERGE SORT



MERGE SORT



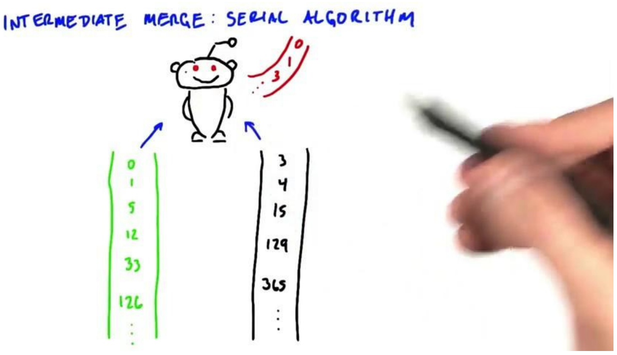
MERGE SORT



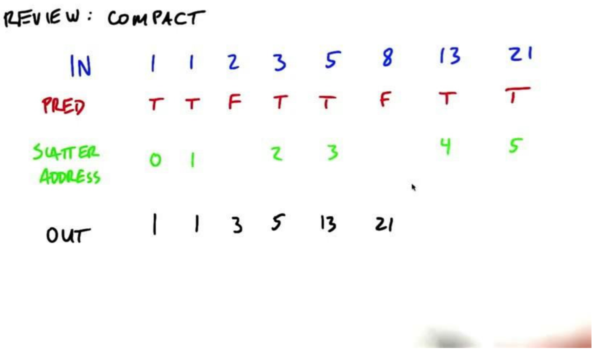
STAGE 2
BUNCH OF TASKS
EACH TASK: MEDIUM
TASK PER BLOCK

STAGE 1
TONS OF TASKS
EACH TASK: SMALL
TASK PER THREAD

INTERMEDIATE MERGE: SERIAL ALGORITHM

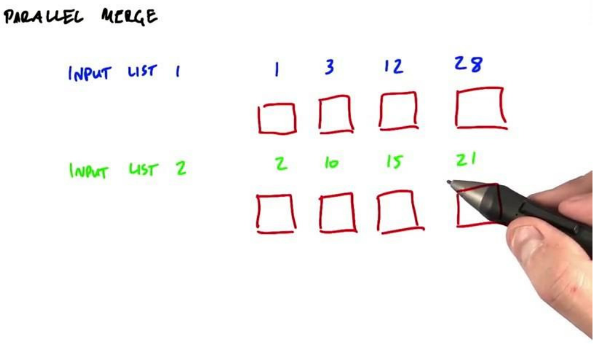


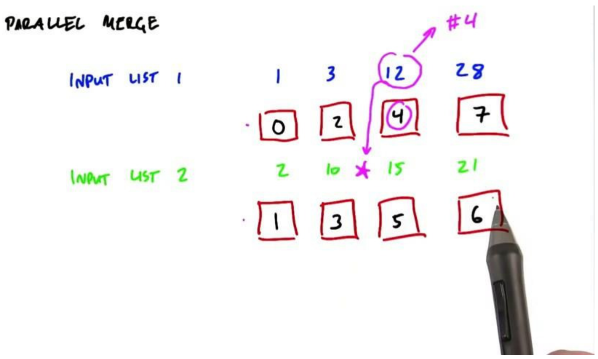
REVIEW: COMPACT

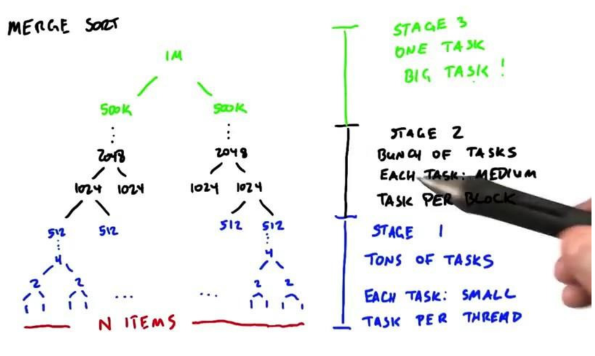


IN	1	1	2	3	5	8	13	21
PRED	T	T	F	T	T	F	T	T
SCATTER ADDRESS	0	1		2	3		4	5
OUT	1	1	3	5	13	21		

PARALLEL MERGE



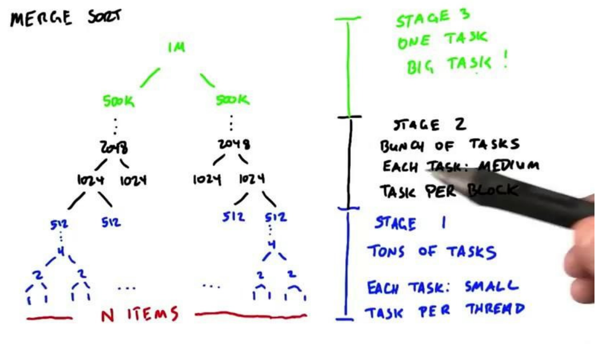

PARALLEL MERGE


MERGE SORT

STAGE 3
ONE TASK
BIG TASK!

STAGE 2
BUNCH OF TASKS
EACH TASK: MEDIUM
TASK PER BLOCK

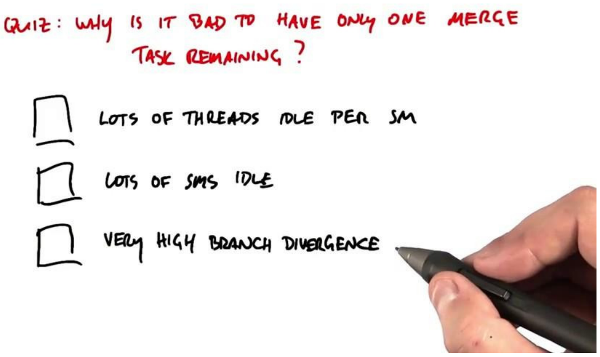
STAGE 1
TONS OF TASKS
EACH TASK: SMALL
TASK PER THREAD


MERGE SORT

STAGE 3
ONE TASK
BIG TASK!

STAGE 2
BUNCH OF TASKS
EACH TASK: MEDIUM
TASK PER BLOCK

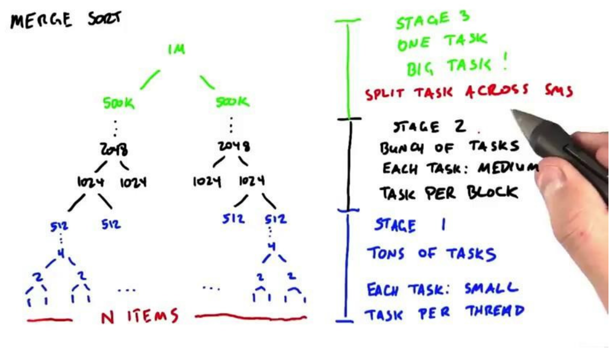
STAGE 1
TONS OF TASKS
EACH TASK: SMALL
TASK PER THREAD


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

□ LOTS OF THREADS IDLE PER SM

□ LOTS OF SMS IDLE

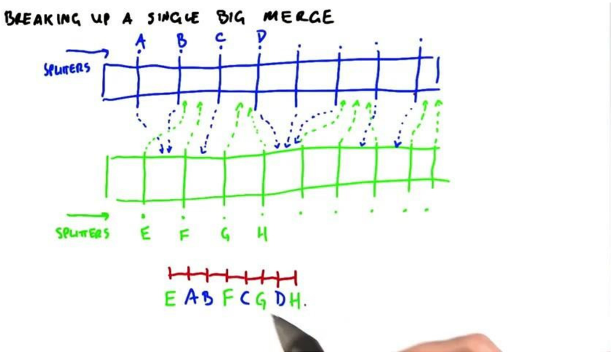
□ VERY HIGH BRANCH DIVERGENCE

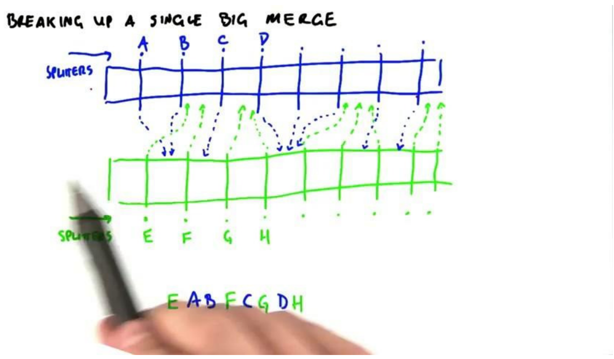

MERGE SORT

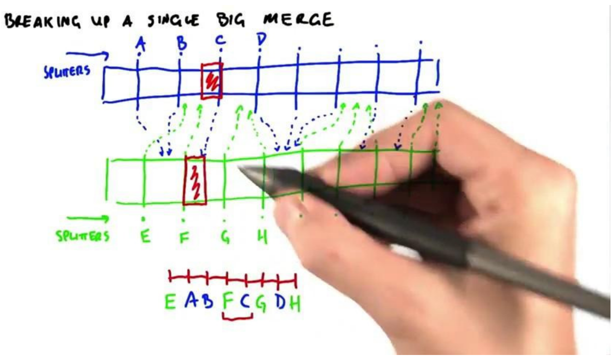
STAGE 3
ONE TASK
BIG TASK!
SPLIT TASK ACROSS SMS

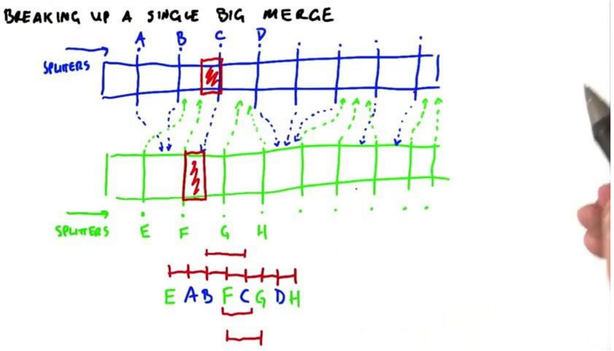
STAGE 2
BUNCH OF TASKS
EACH TASK: MEDIUM
TASK PER BLOCK

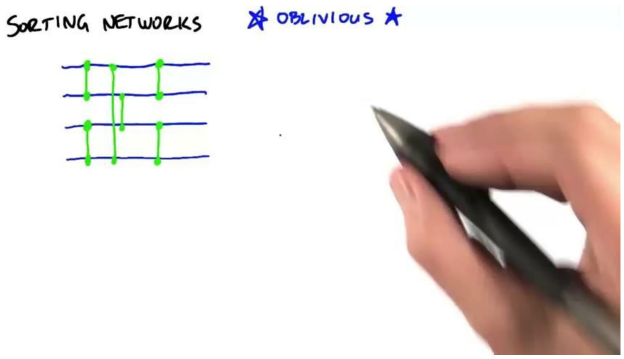
STAGE 1
TONS OF TASKS
EACH TASK: SMALL
TASK PER THREAD

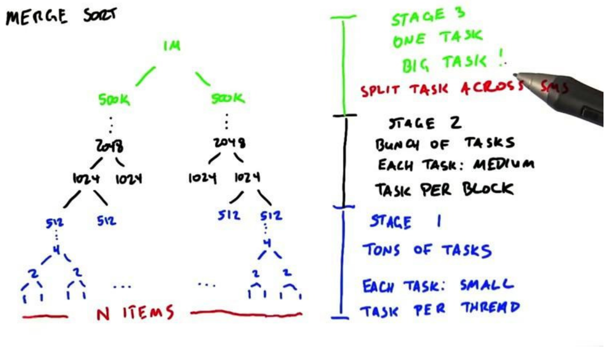

BREAKING UP A SINGLE BIG MERGE

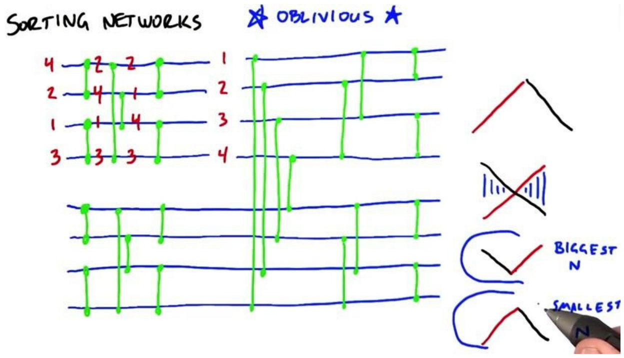

BREAKING UP A SINGLE BIG MERGE

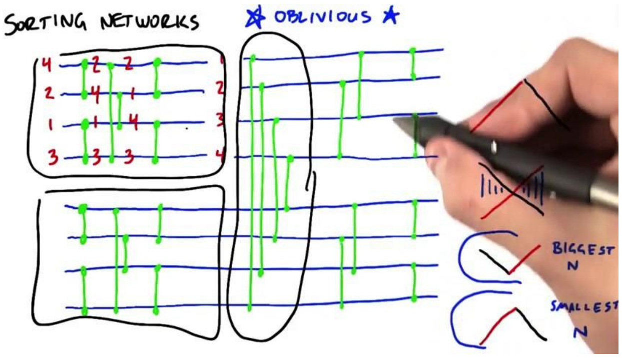

BREAKING UP A SINGLE BIG MERGE

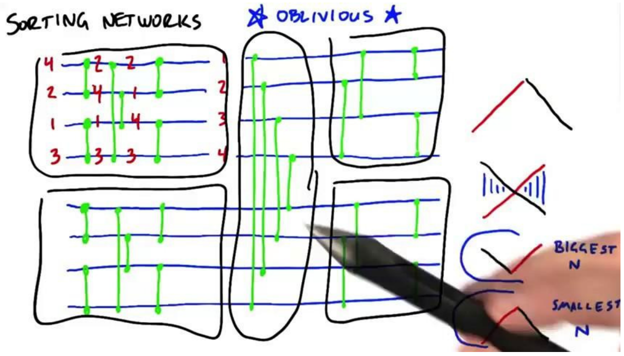


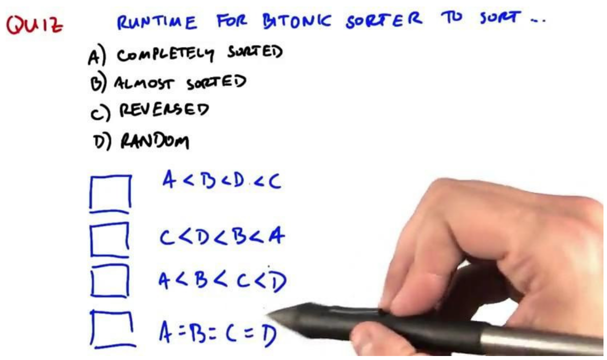












QUIZ RUNTIME FOR BITONIC SORTER TO SORT...

A) COMPLETELY SORTED

B) ALMOST SORTED

C) REVERSED

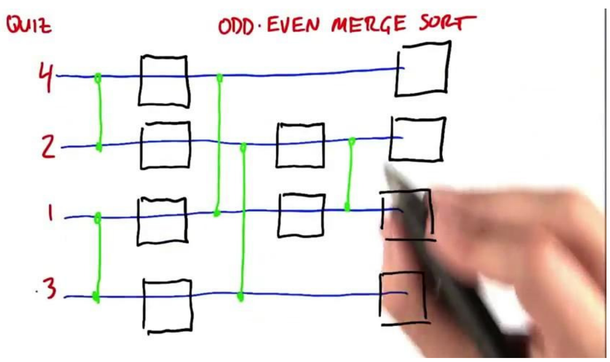
D) RANDOM

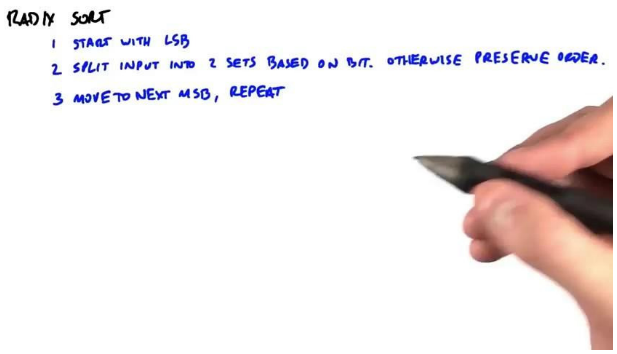
□ A<B<D<C

□ C<D<B<A

□ A<B<C<D

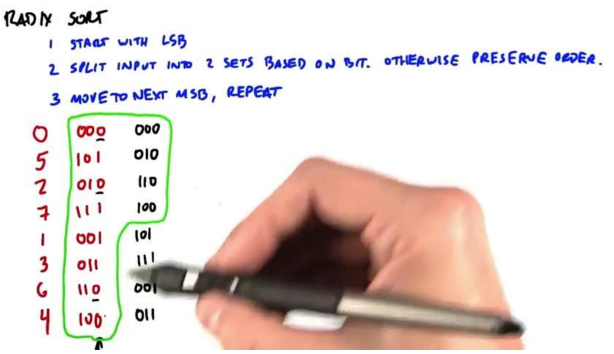
□ A=B=C=D





RADIX SORT

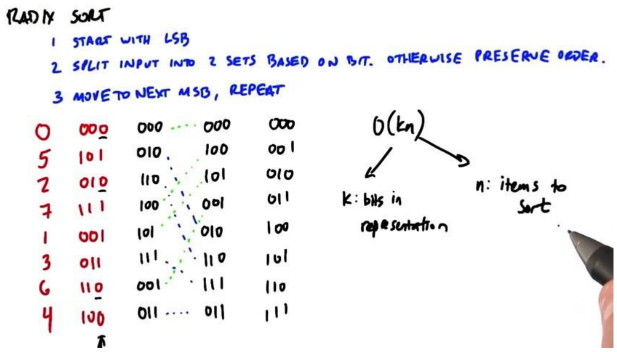
1 START WITH LSB
2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
3 MOVE TO NEXT MSB, REPEAT



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



RADIX SORT

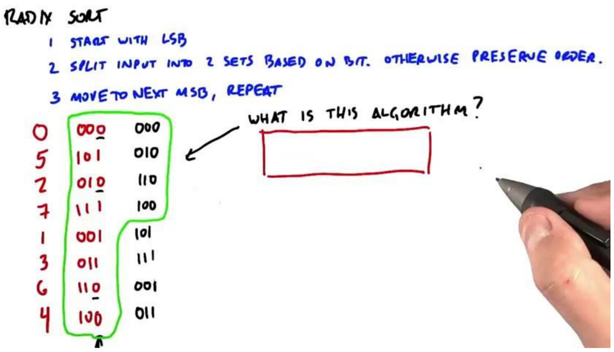
1 START WITH LSB
2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
3 MOVE TO NEXT MSB, REPEAT

0	000	000	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

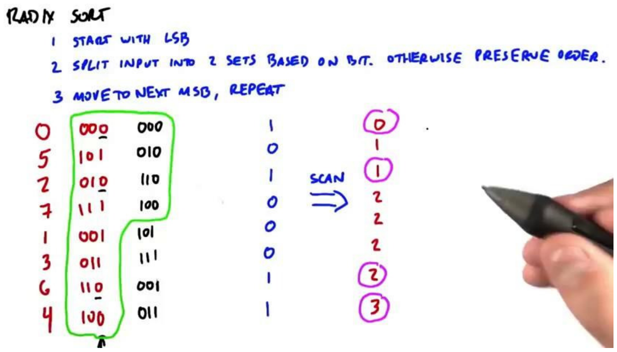


RADIX SORT

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

WHAT IS THIS ALGORITHM?

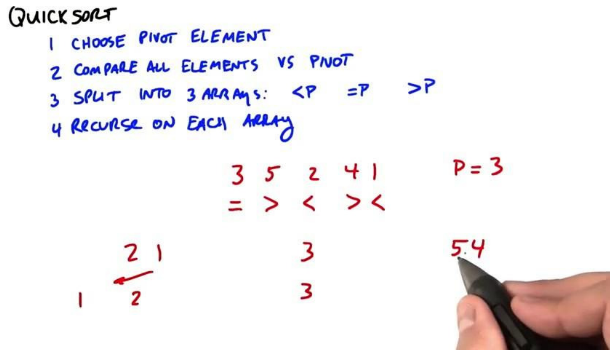


RADIX SORT

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

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

SCAN ⇒



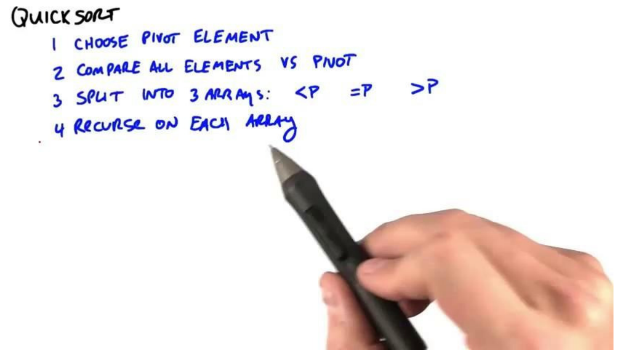
QUICKSORT

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
= > < > <

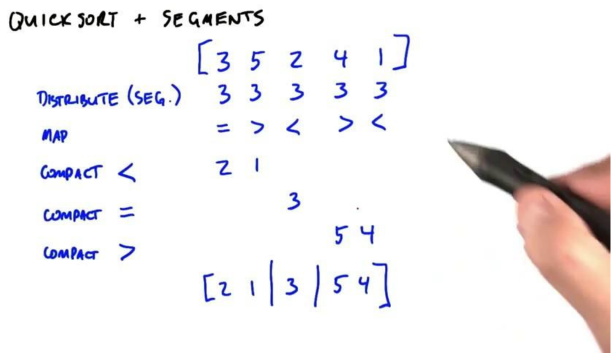
2 1 3 5 4

1 2 3



QUICKSORT

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



QUICKSORT + SEGMENTS

[3 5 2 4 1]

DISTRIBUTE (SEG.) 3 3 3 3 3

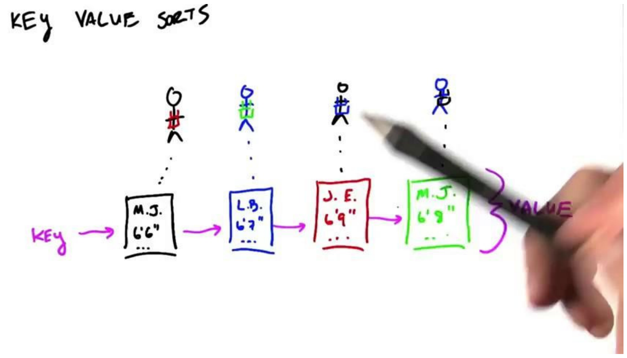
MAP = > < > <

COMPACT < 2 1

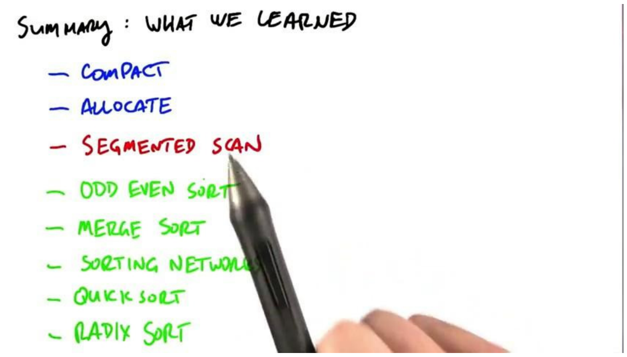
COMPACT = 3

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
- QUICKSORT
- RADIX SORT