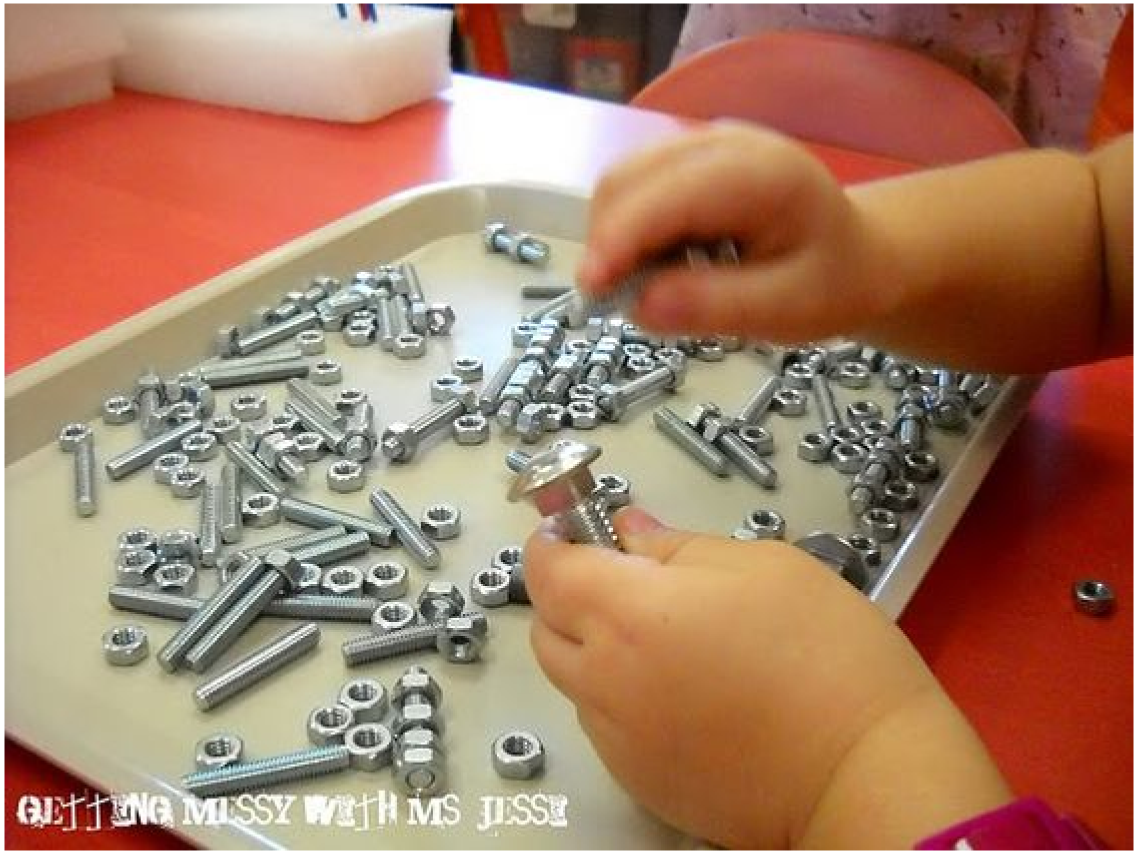
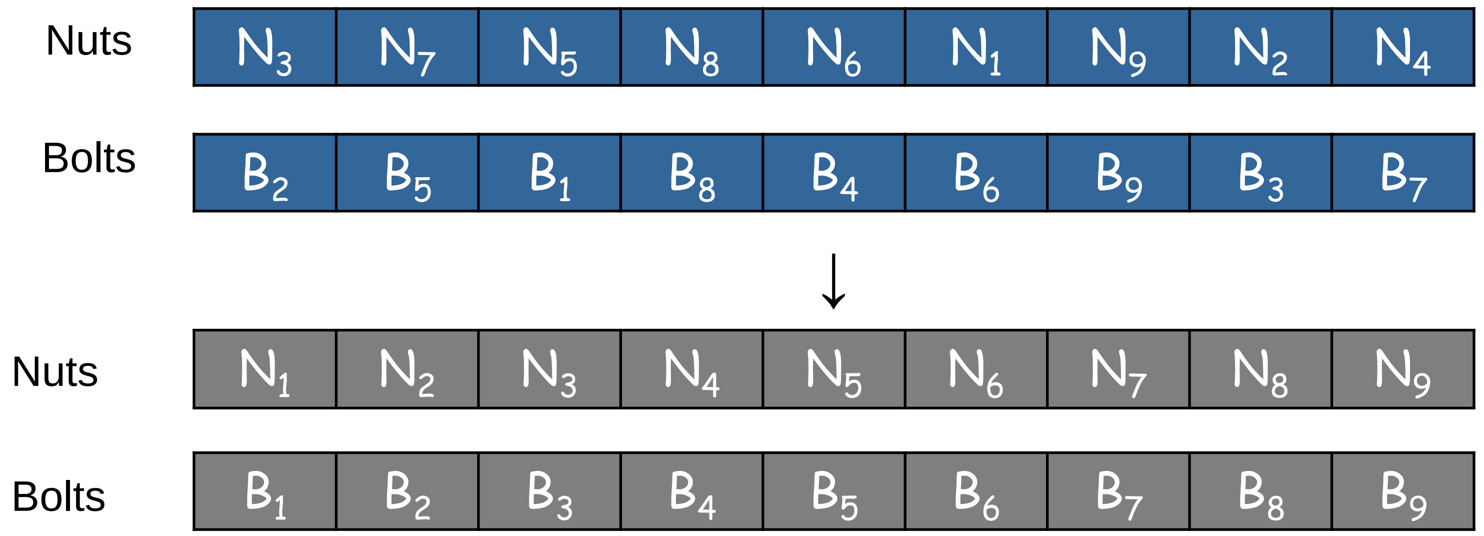
Matching Nuts & Bolts



Matching Nuts & Bolts


Nuts
N_3 N_7 N_5 N_8 N_6 N_1 N_9 N_2 N_4
Bolts
B_2 B_5 B_1 B_8 B_4 B_6 B_9 B_3 B_7
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9

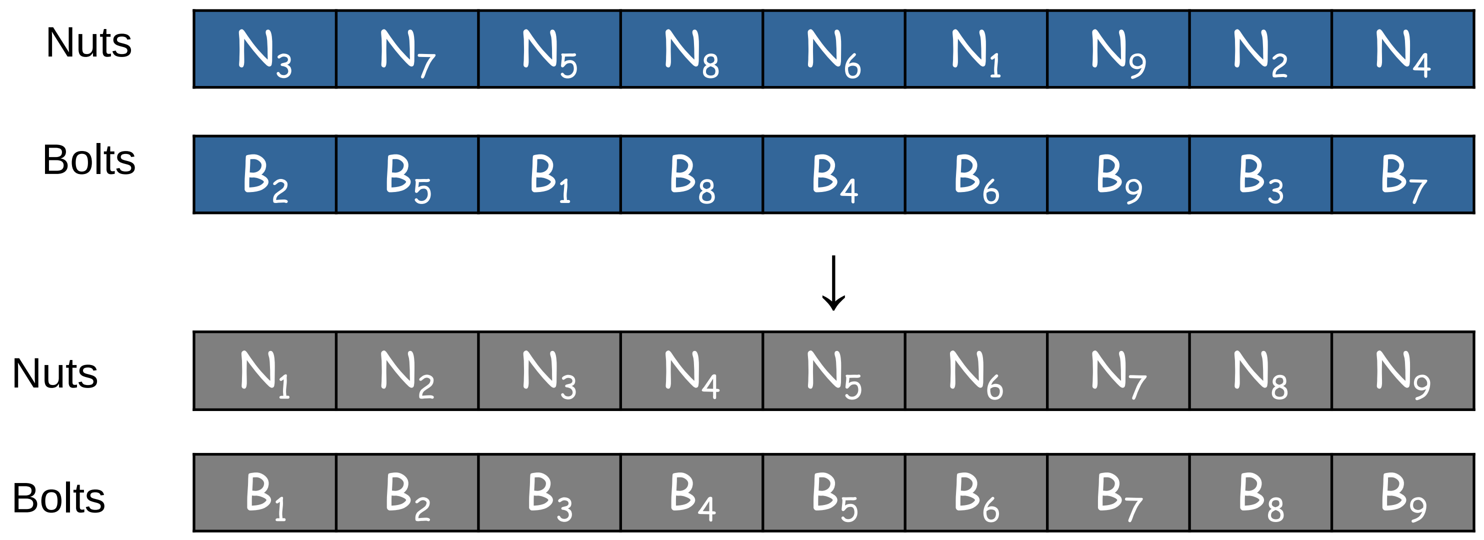
You are given n pairs of nuts and bolts

$$(N_1, B_1), (N_2, B_2), \ldots.. (N_n, B_n)$$

Each pair is a different size than the others. Someone has unscrewed all of the nuts out of the bolts and mixed them up.

Problem: Match all nuts up with their corresponding bolts.

Matching Nuts & Bolts

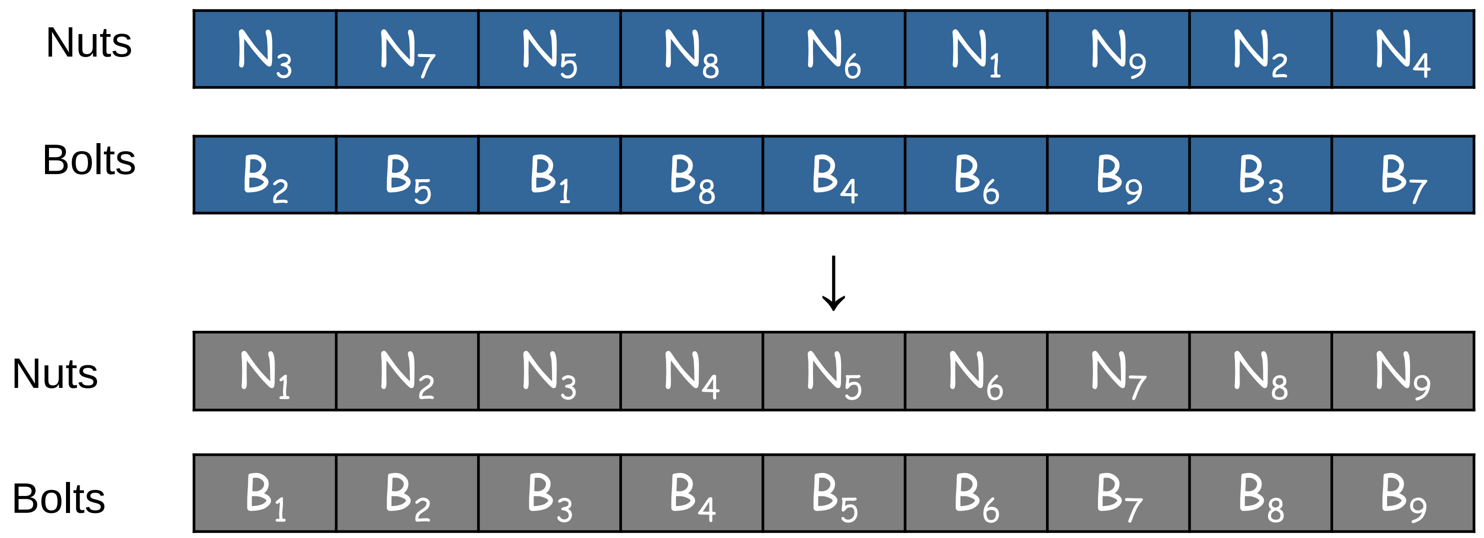

Nuts
N_3 N_7 N_5 N_8 N_6 N_1 N_9 N_2 N_4
Bolts
B_2 B_5 B_1 B_8 B_4 B_6 B_9 B_3 B_7
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9

If we could separately
(i) sort all the bolts by increasing size and then
(ii) sort all the nuts by increasing thread size
=> problem would be easily solvable in O (n log n) time.

After sorting, just match them up in order from smallest to largest.

We can't do this because we can't compare the sizes of two nuts directly or the sizes of two bolts **directly.**

Matching Nuts & Bolts


Nuts
N_3 N_7 N_5 N_8 N_6 N_1 N_9 N_2 N_4
Bolts
B_2 B_5 B_1 B_8 B_4 B_6 B_9 B_3 B_7
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9

Only operation available is to try to screw a bolt B into a nut N
and then, by seeing whether the bolt

(a) goes loosely in, (b) perfectly fits or (c) can't go in at all,

decide whether their thread sizes satisfy

(a) B<N, (b) B = N or (c) B>n.

Observation:

Given bolt B, above permits finding all NUTS bigger/=/less than B in O(N) time.

Given nut N, above permits finding all BOLTS bigger/=/less than N in O(N) time

Hint: Try to use a modified Quicksort

Let $X = \{B_1, B_2, \dots, B_n\}$; $Y = \{N_1, N_2, \dots, N_n\}$;

<u>MATCH(X;Y)</u>

```
1. If |X| = |Y| = 1 match the unique nut with the unique bolt.
      Otherwise
2.    Choose a nut N at random from Y

3.          Compare ALL bolts in X with N; After doing this
              Let B be unique bolt that fits N
              Split the remaining bolts into two sets
              B_s = Bolts smaller than B;  B_L = Bolts larger than N.
4.          Compare ALL nuts in Y except for N with B; Set
              N_s = Nuts smaller than B ;  N_L = Nuts larger than B.

5.          Match  B ↔ N.
6.          Recursively call MATCH(B_s, N_s),  MATCH(B_L, N_L).
```

```
MATCH(X;Y )
1. If $|X| = |Y| = 1$ match the unique nut with the unique bolt.
     Otherwise
2.     Choose a nut N at random from Y
3.           Compare ALL bolts in $X$ with N;  After doing this, Let $B$ be unique bolt that fits $N$
               Split the remaining bolts into two sets
               $B_s$ = Bolts smaller than $B$;  $B_L$ = Bolts larger than $N$.
4.         Compare ALL nuts in $Y$ except for $N$ with $B$; Set
               $N_s$ = Nuts smaller than $B$ ;   $N_L$ = Nuts larger than $B$.
5.        Match   $B \leftrightarrow N$  and recursively call
6.             MATCH$(B_s, N_s)$,    MATCH$(B_L, N_L)$,
```

Visualize X and Y as being stored in two separate ARRAYS.

Choosing the random nut in 2 is like choosing a random pivot upon which to partition X array (the bolts) in Quicksort.

Steps 3-4 can be implemented in O (n) time using almost exactly the same code as partition in Quicksort.

After Steps 3,4,5 the X and Y arrays have been partitioned around their pivots and the recursive calls are done into the two pairs of subarrays.

The analysis is then EXACTLY like Quicksort and this algorithm takes O (n log n) Expected time to match the nuts and bolts.

```
MATCH(X;Y )
1. If $|X| = |Y| = 1$ match the unique nut with the unique bolt.
      Otherwise
2.     Choose a nut N at random from Y
3.           Compare ALL bolts in $X$ with N;  After  doing this, Let $B$ be unique bolt that fits $N$
               Split the remaining bolts into two sets
               $B_s$ = Bolts smaller than $B$;  $B_L$ = Bolts larger than $N$.
4.         Compare ALL nuts in $Y$ except for $N$ with $B$; Set
               $N_s$ = Nuts smaller than $B$ ;   $N_L$ = Nuts larger than $B$.
5.        Match   $B \leftrightarrow$ N  and recursively call
6.              MATCH$(B_s, N_s)$,   MATCH$(B_L, N_L)$,
```

Visualize X and Y as being stored in two separate ARRAYS.
The analysis is then EXACTLY like Quicksort and this algorithm takes
O (n log n) Expected time to match the nuts and bolts.

To see why this analysis is the same as Quicksort's,
forget about the nuts and bolts and
consider the algorithm from the point of view of sorting Y.

Step 2 chooses a random nut N as a pivot.
Steps 3-4 does O(n) work and partitions Y around N.
Step 6 then recursively calls the algorithm on the subarrays to left and right of N.
This exactly mimics Quicksort!

Worked Example

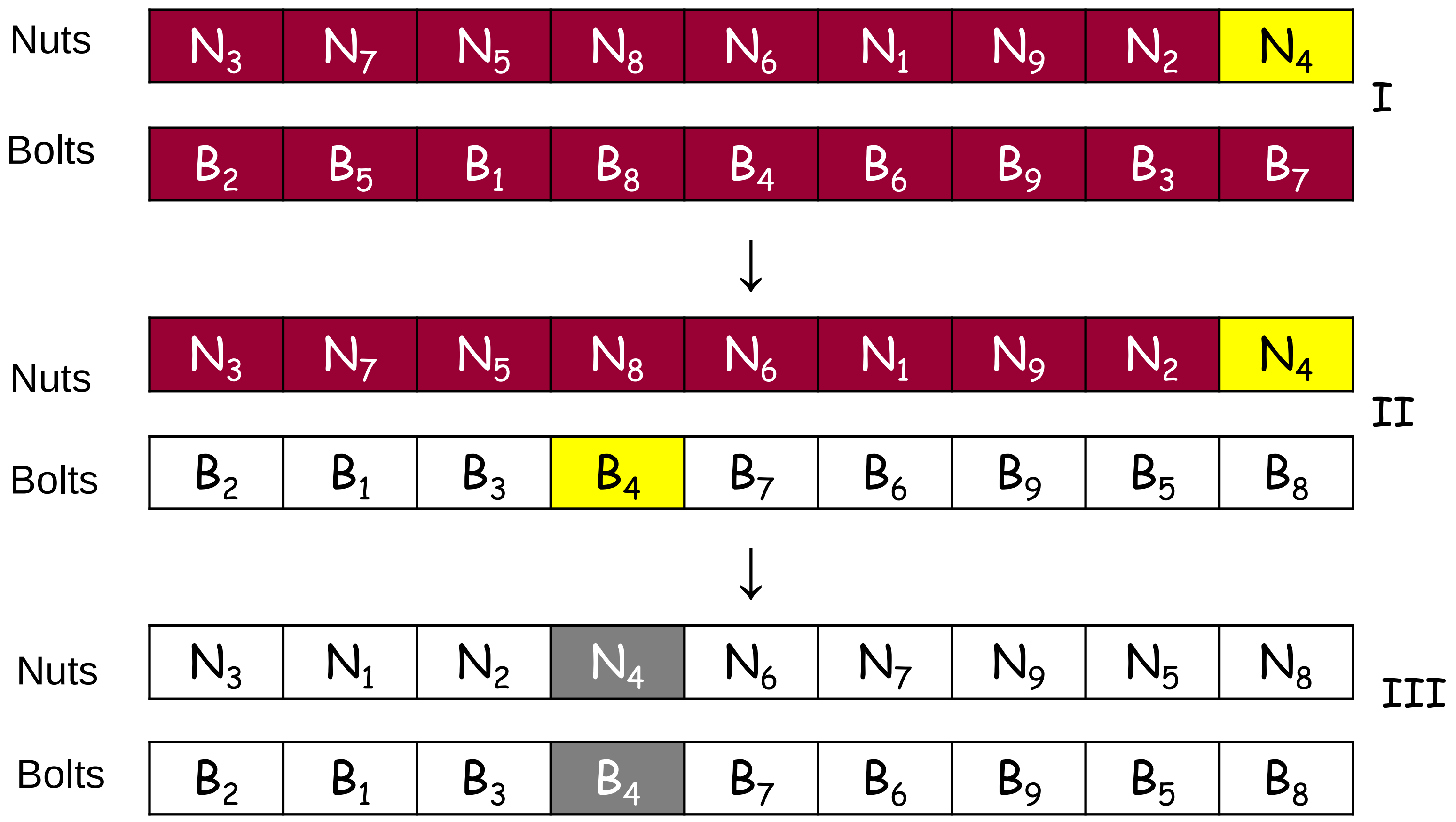
We will see how to transform the top into the bottom

Nuts	N_3	N_7	N_5	N_8	N_6	N_1	N_9	N_2	N_4
Bolts	B_2	B_5	B_1	B_8	B_4	B_6	B_9	B_3	B_7

$$\downarrow$$

Nuts	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9
Bolts	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9

1st Call


Nuts
N3 N7 N5 N8 N6 N1 N9 N2 N4
I
Bolts
B2 B5 B1 B8 B4 B6 B9 B3 B7
Nuts
N3 N7 N5 N8 N6 N1 N9 N2 N4
II
Bolts
B2 B1 B3 B4 B7 B6 B9 B5 B8
Nuts
N3 N1 N2 N4 N6 N7 N9 N5 N8
III
Bolts
B2 B1 B3 B4 B7 B6 B9 B5 B8

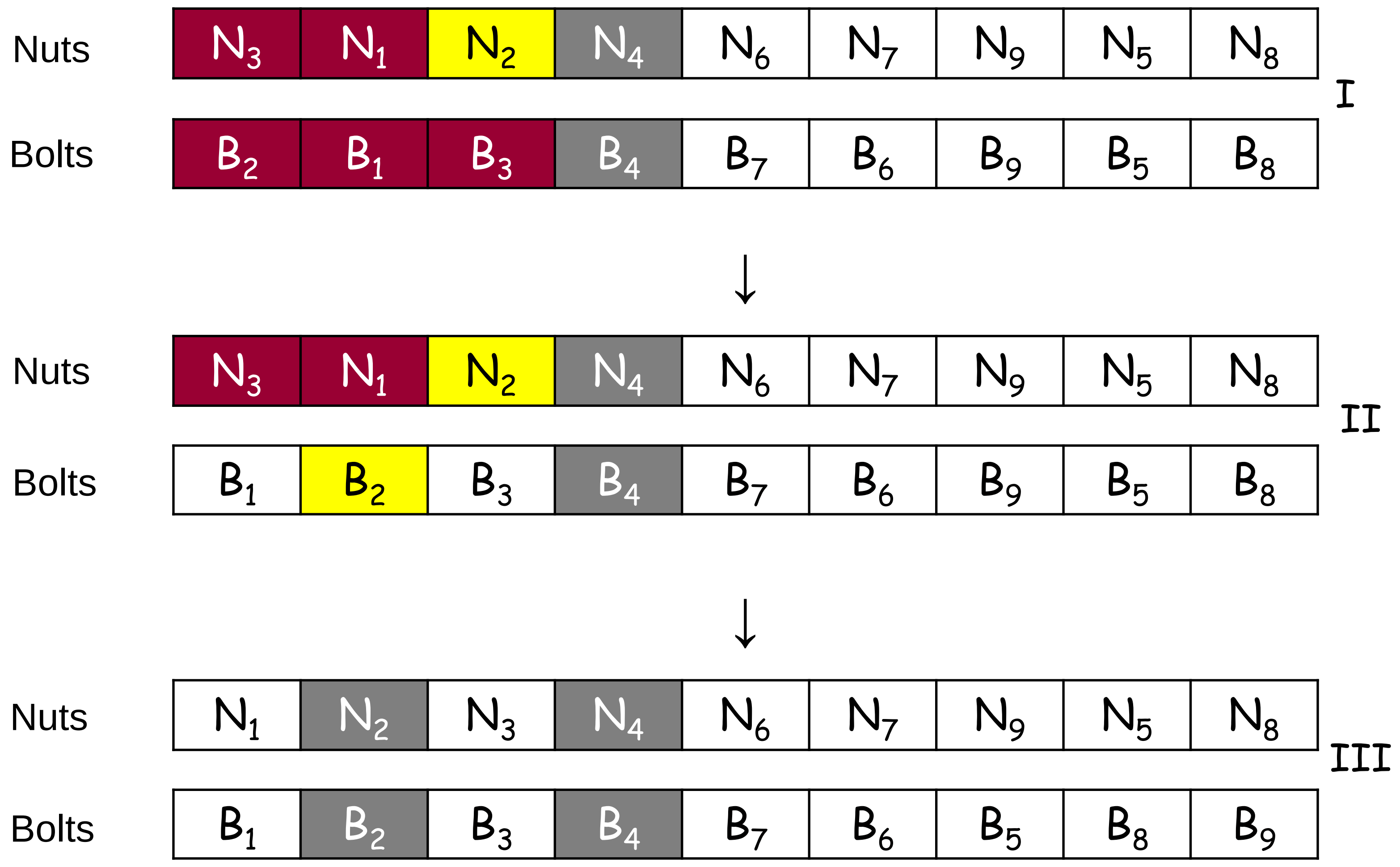
Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)

and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

2^{nd} Call



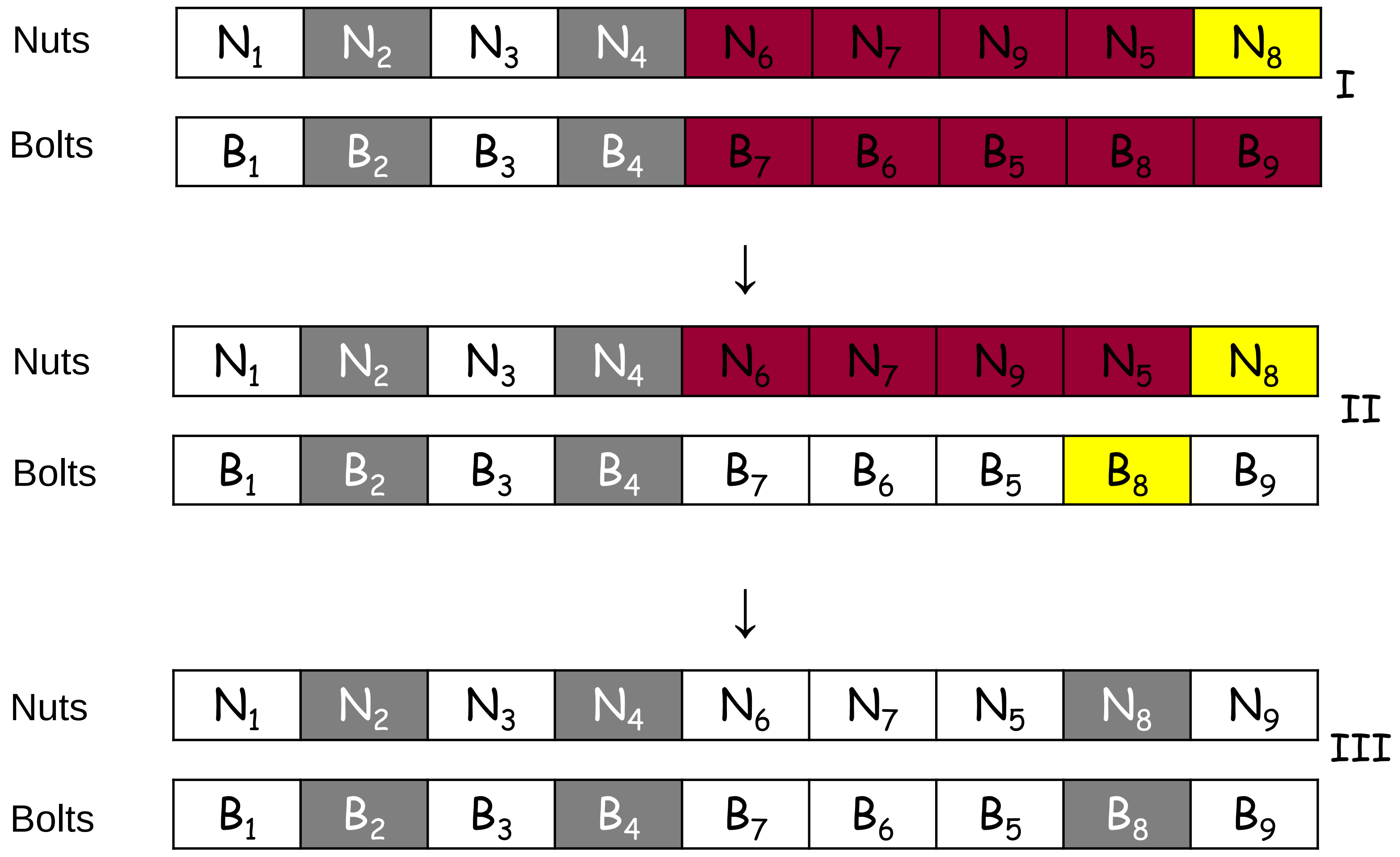
Nuts N_3 N_1 N_2 N_4 N_6 N_7 N_9 N_5 N_8
Bolts B_2 B_1 B_3 B_4 B_7 B_6 B_9 B_5 B_8
I
Nuts N_3 N_1 N_2 N_4 N_6 N_7 N_9 N_5 N_8
Bolts B_1 B_2 B_3 B_4 B_7 B_6 B_9 B_5 B_8
II
Nuts N_1 N_2 N_3 N_4 N_6 N_7 N_9 N_5 N_8
Bolts B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
III

Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)
and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

3rd Call

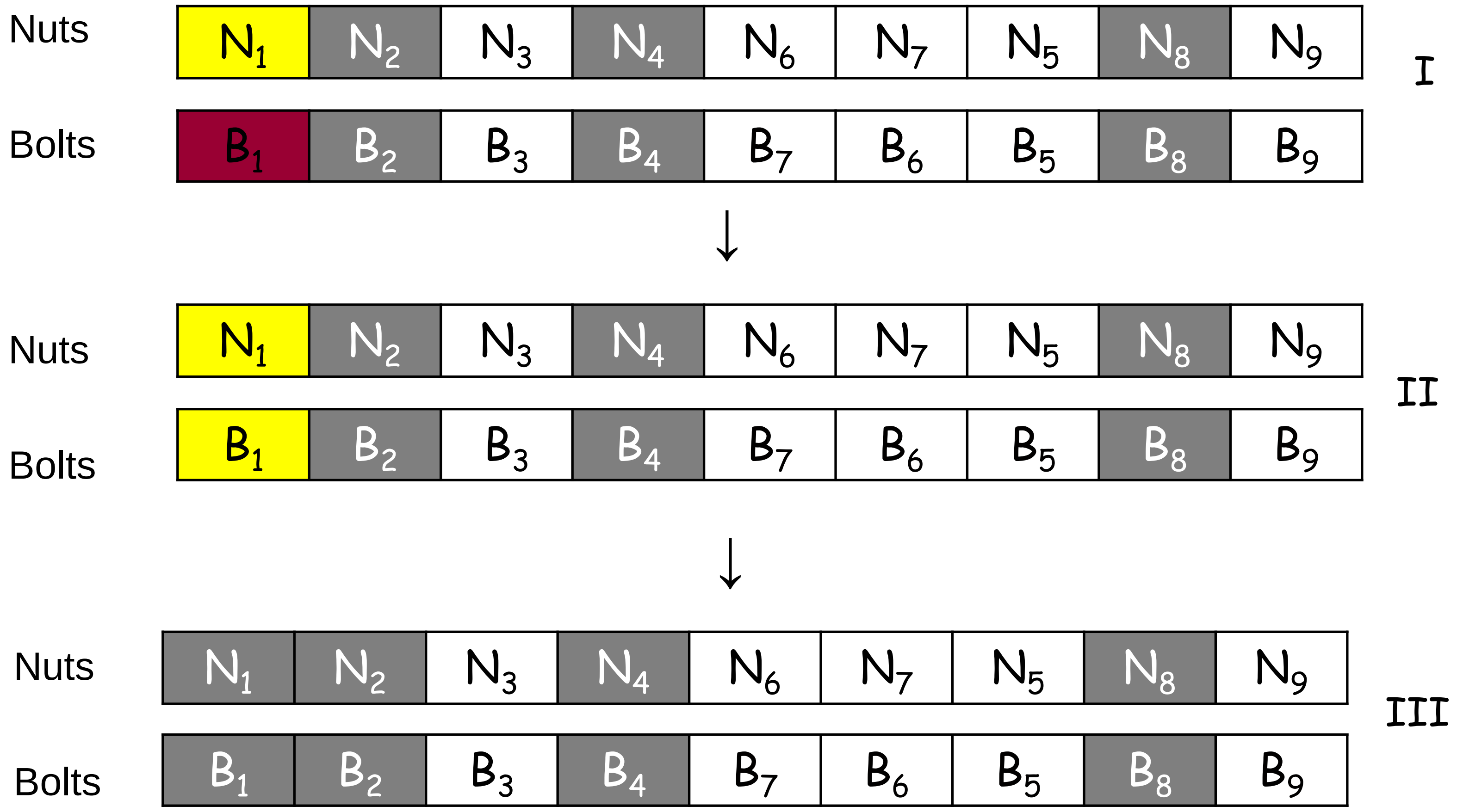

Nuts
N_1 N_2 N_3 N_4 N_6 N_7 N_9 N_5 N_8
Bolts
B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
I
Nuts
N_1 N_2 N_3 N_4 N_6 N_7 N_9 N_5 N_8
Bolts
B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
II
Nuts
N_1 N_2 N_3 N_4 N_6 N_7 N_5 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
III

Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)
and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

4th Call


Nuts
N_1 N_2 N_3 N_4 N_6 N_7 N_5 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
I
Nuts
N_1 N_2 N_3 N_4 N_6 N_7 N_5 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
II
Nuts
N_1 N_2 N_3 N_4 N_6 N_7 N_5 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
III

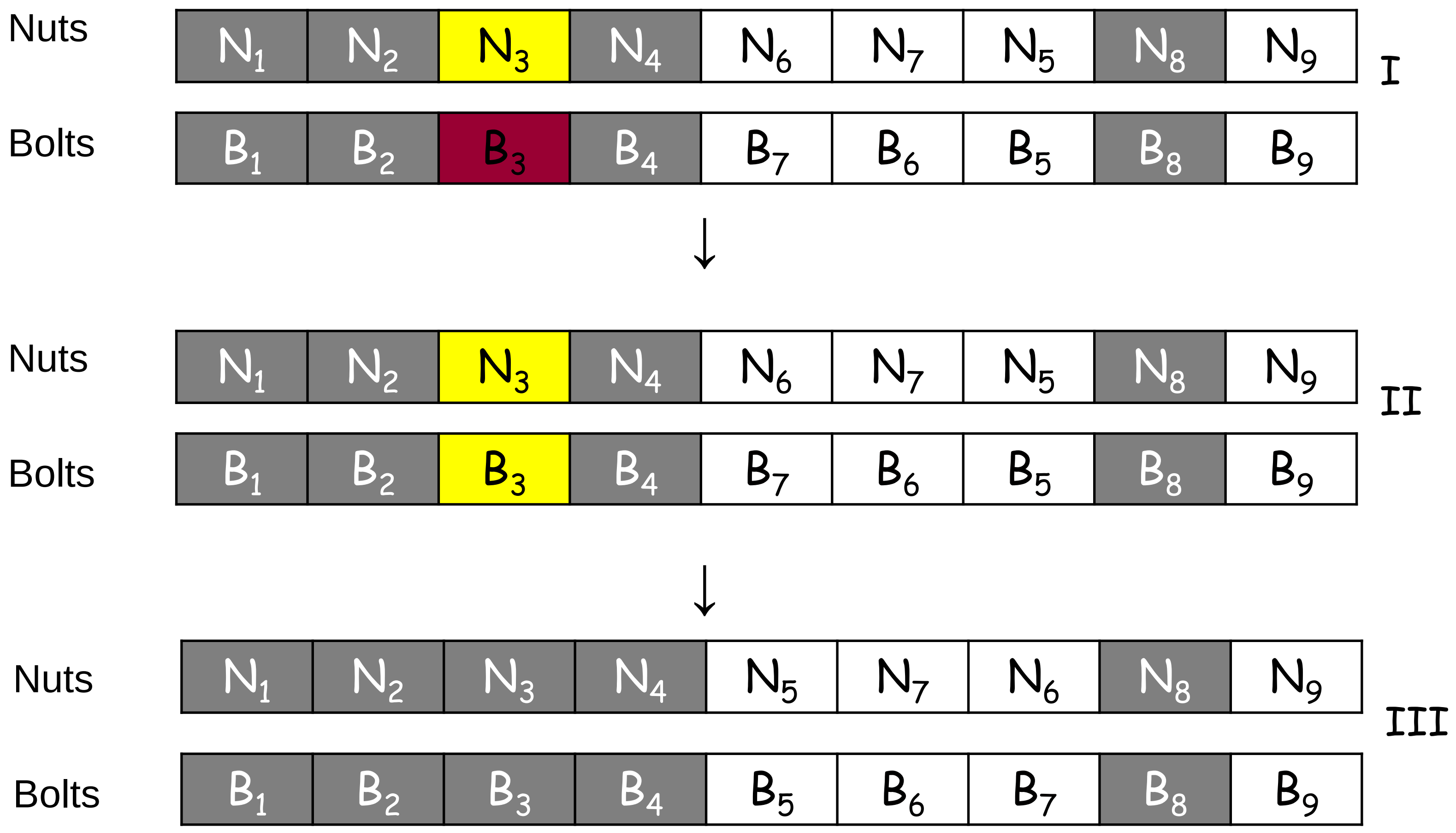
Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)

and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

5th Call


Nuts
N_1 N_2 N_3 N_4 N_6 N_7 N_5 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
I
Nuts
N_1 N_2 N_3 N_4 N_6 N_7 N_5 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_7 B_6 B_5 B_8 B_9
II
Nuts
N_1 N_2 N_3 N_4 N_5 N_7 N_6 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
III

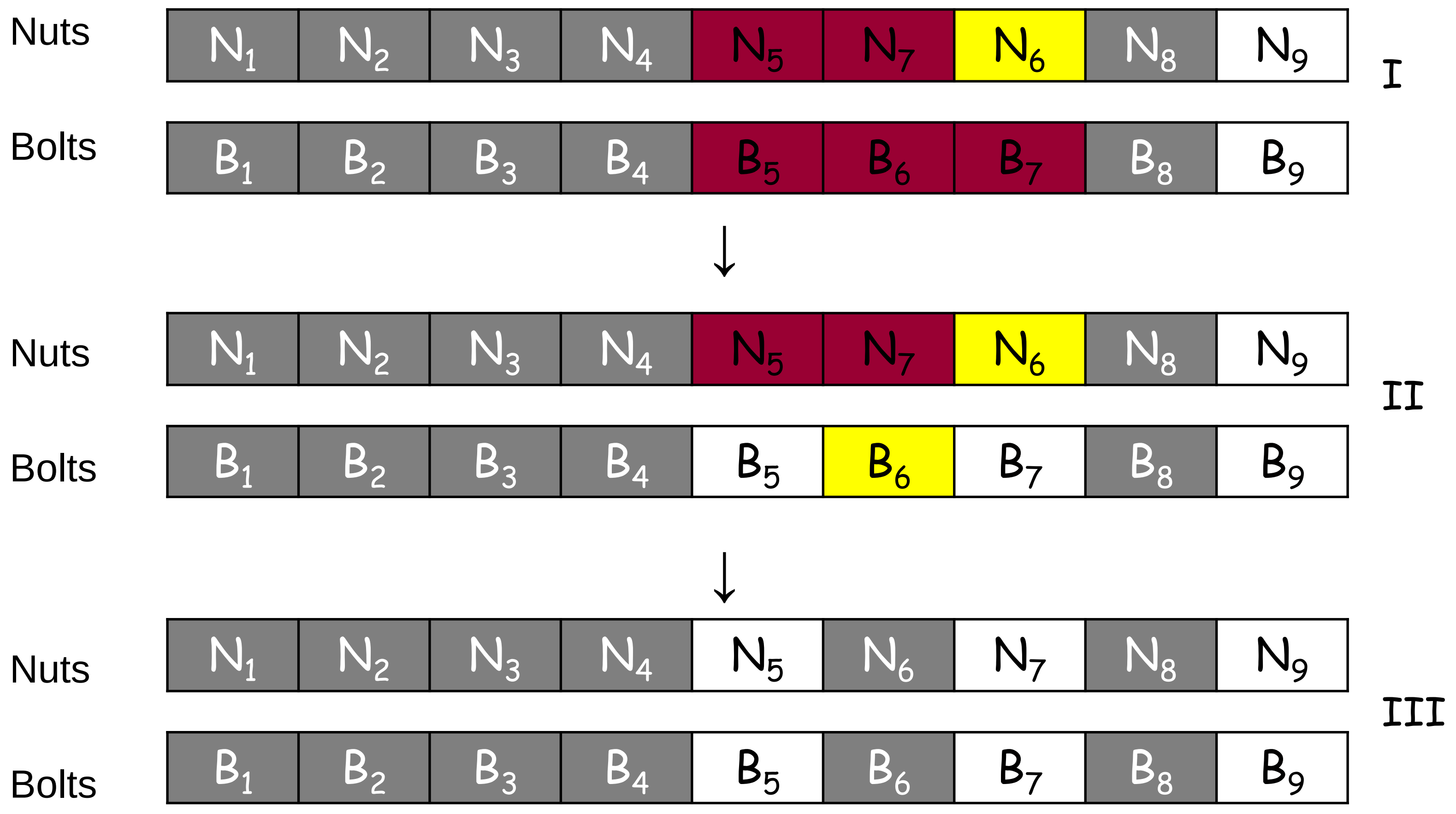
Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)

and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

6^{th} Call


Nuts
N_1 N_2 N_3 N_4 N_5 N_7 N_6 N_8 N_9
I
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
Nuts
N_1 N_2 N_3 N_4 N_5 N_7 N_6 N_8 N_9
II
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
III
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9

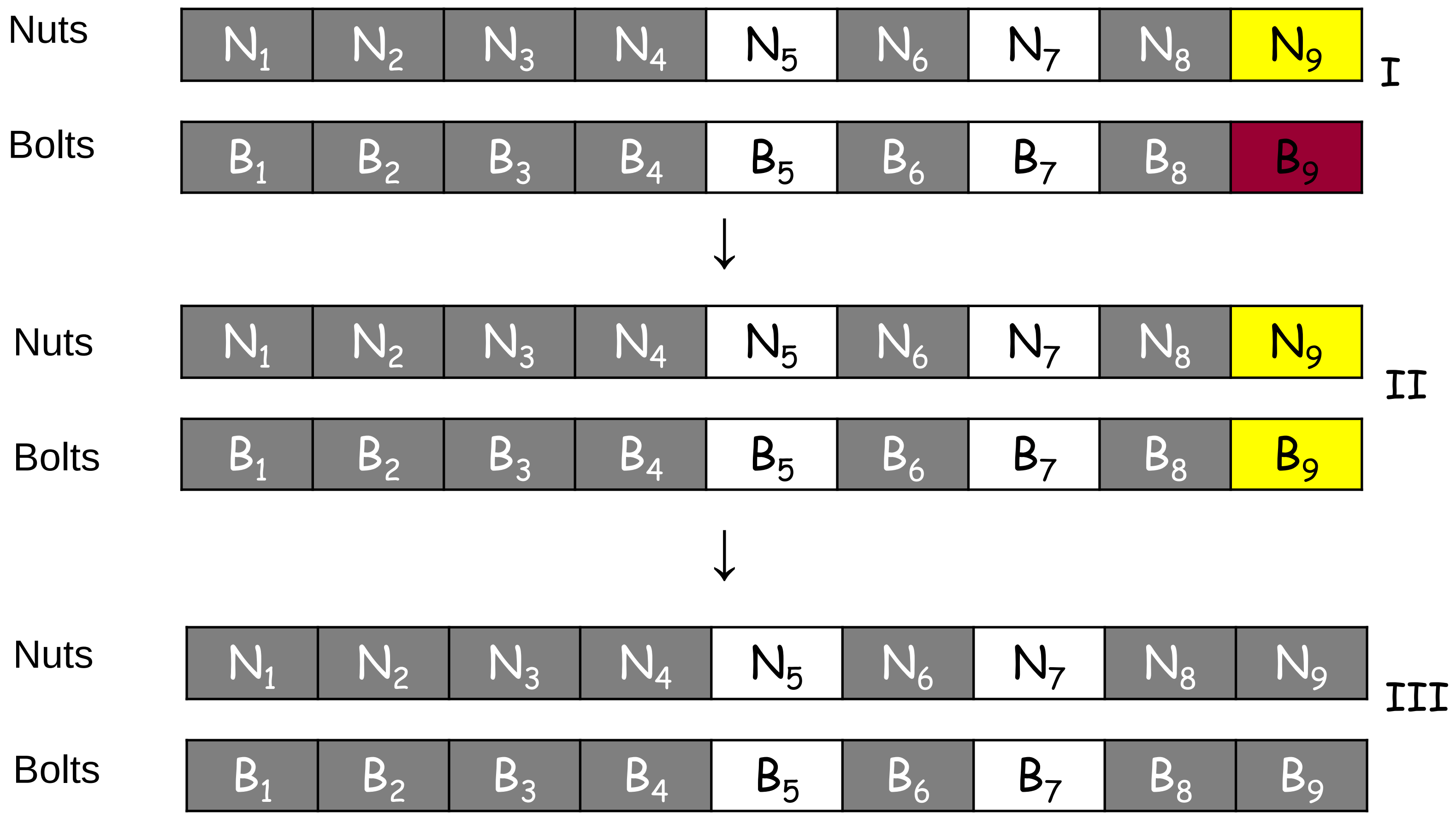
Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)

and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

7th Call


Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
I
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
II
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
III

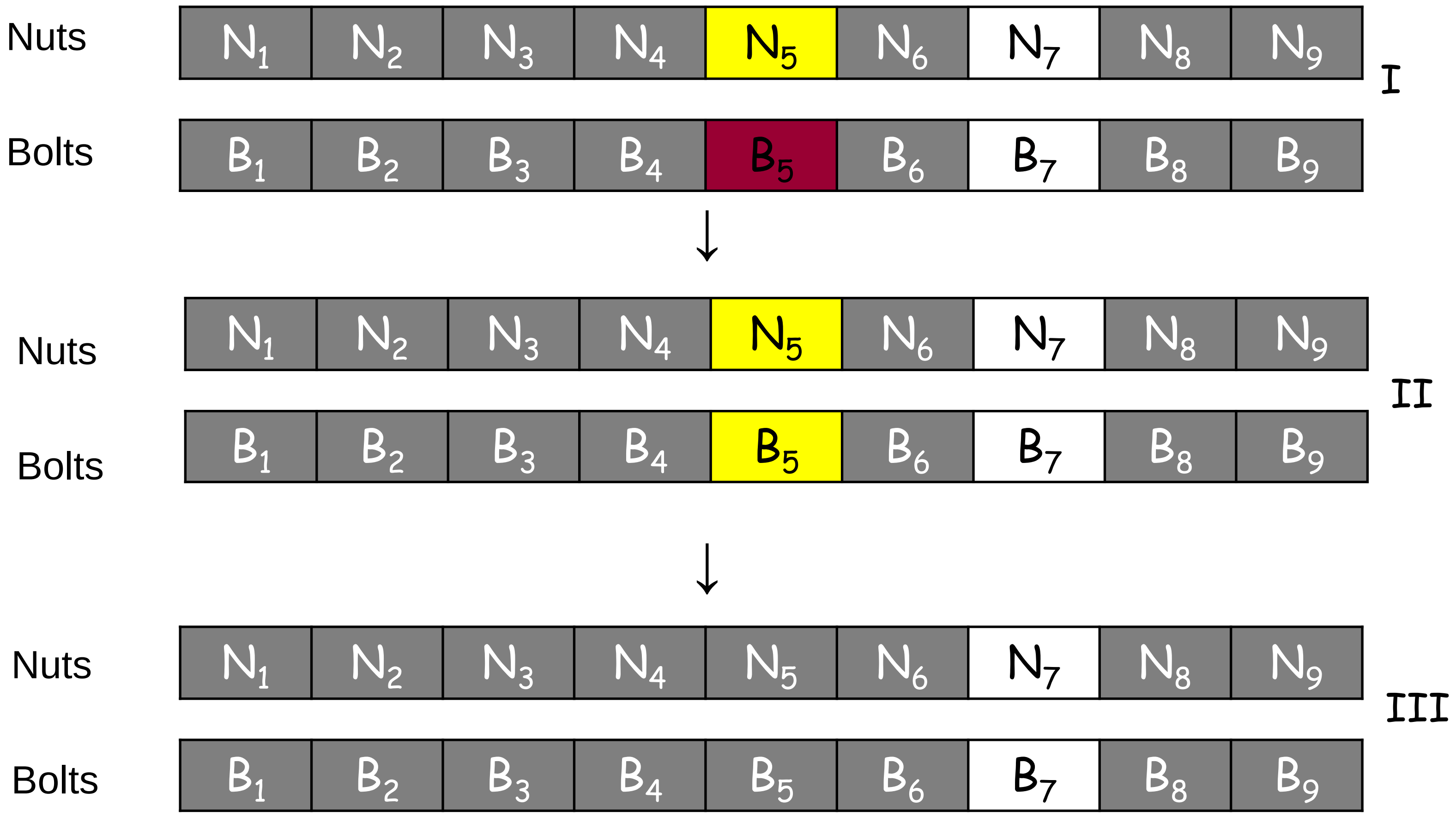
Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)

and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

8th Call


Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
I
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
II
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
III

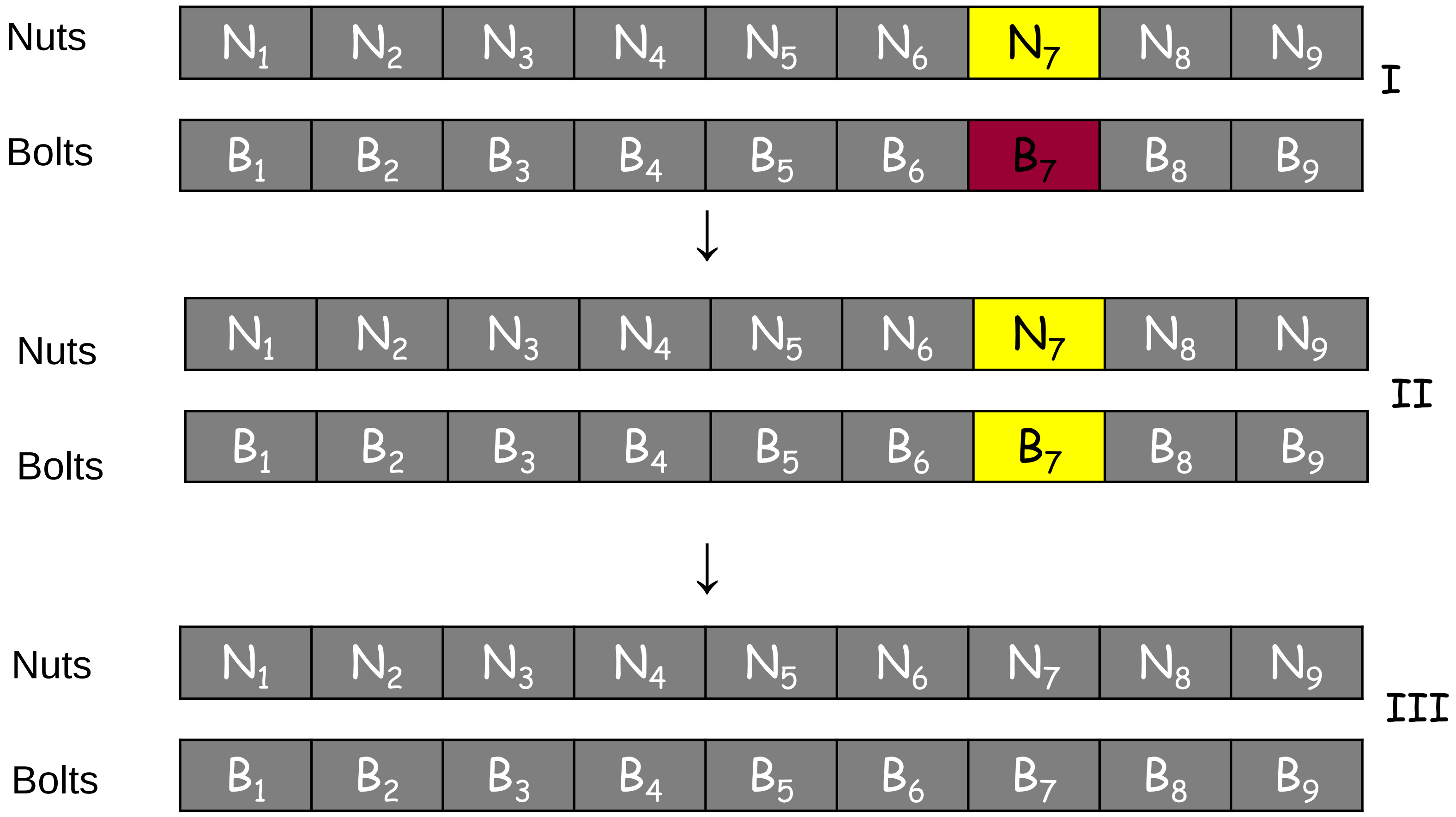
Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)

and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

8th Call



Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
I
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
II
Nuts
N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8 N_9
Bolts
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
III

Step I: Starting Position. Pivot Nut is Yellow. Nuts & Bolts to be partitioned are Red

Step II: Compare Red bolts to Pivot Nut, finding Matching Bolt (Yellow)

and Partitioning remaining Red Bolts

Step III: Partition Red Nuts around Matching Yellow Bolt. Set matching Nut/Bolt as Gray

More

The randomized algorithm from the previous pages was discovered quite early but it took a long time for researchers to find a good deterministic (non-randomized) one.

For a while, the best known deterministic algorithm ran in $O(n(\log n)^4)$ time.

O(n log n) worst case time algorithms are now known to exist but, unlike in the case of sorting, these deterministic algorithms are MUCH more complicated than the randomized one.

For more information about this problem, Google the phrase **Matching Nuts and Bolts Algorithms.**