Sort

郭至軒（KuoE0）

KuoE0.tw@gmail.com
KuoE0.ch

Latest update: Feb 27, 2013

Bubble Sort

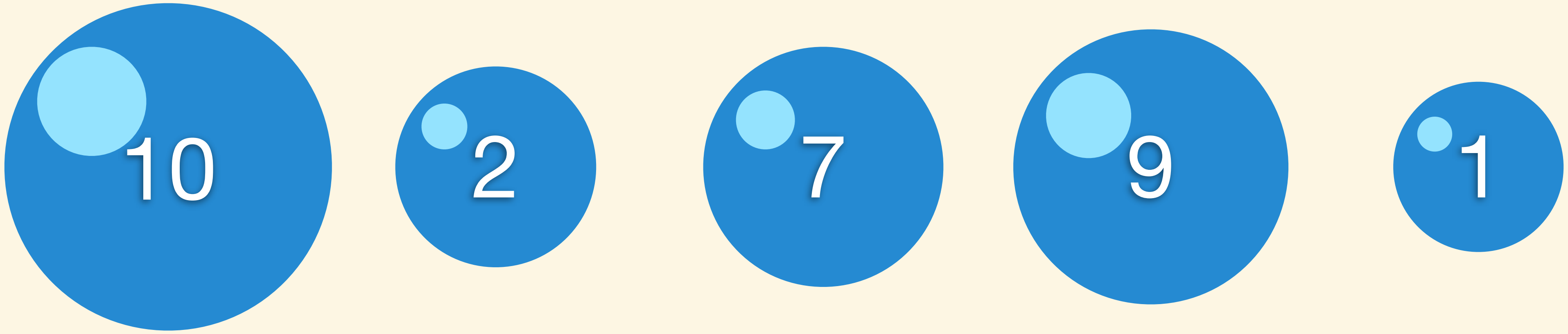
氣泡排序法

Algorithm

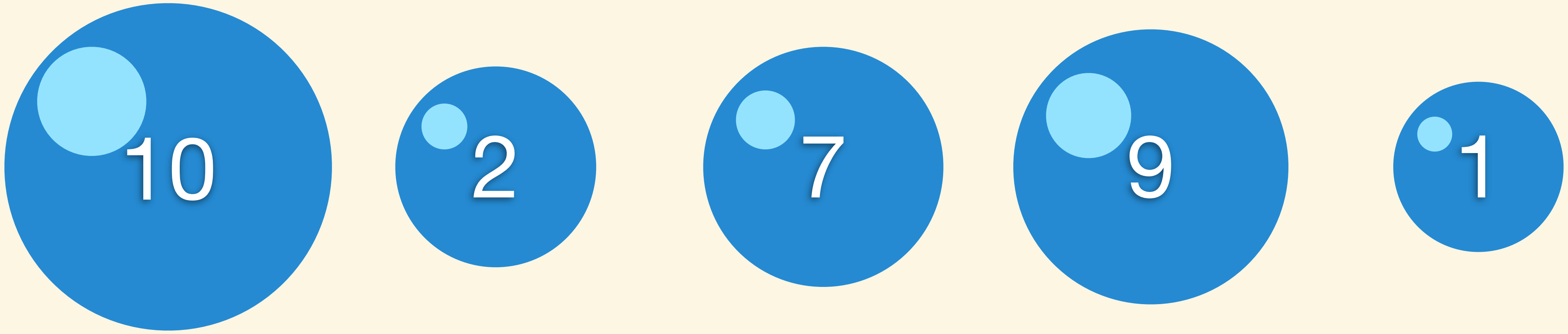
1. 從第一個元素開始比較每一對相鄰元素
2. 若第一個元素大於第二個元素則交換
3. 每次遞減需要比較的元素對直到不需比較

Origin

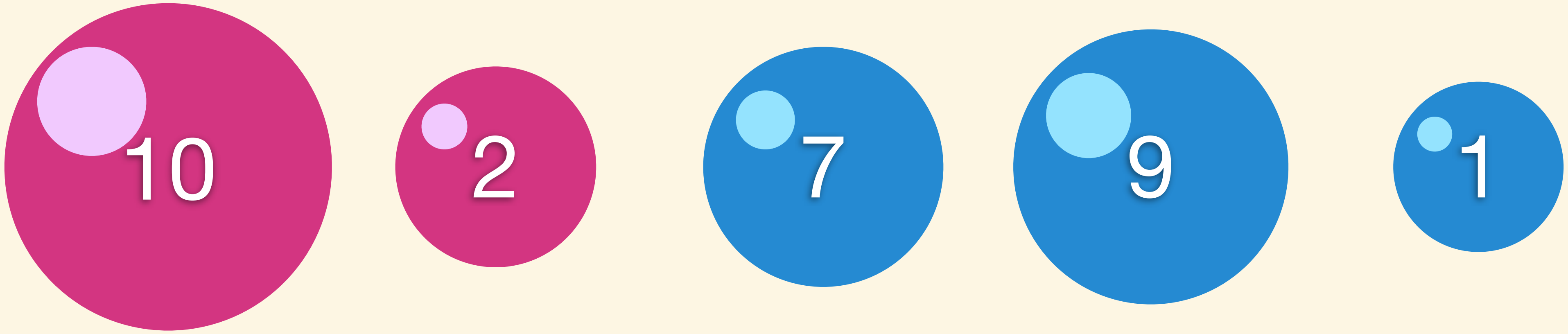
10	2	7	9	1



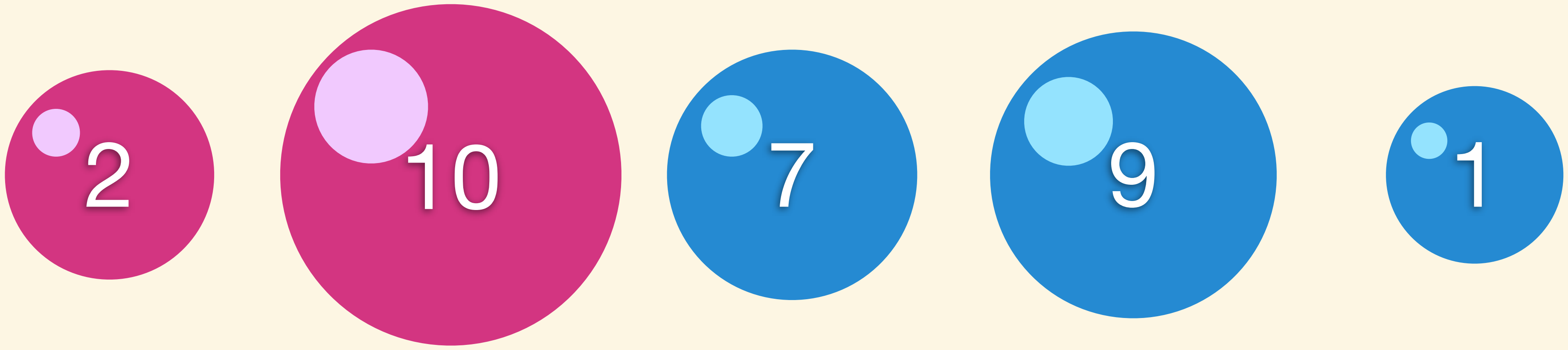
1st Iteration



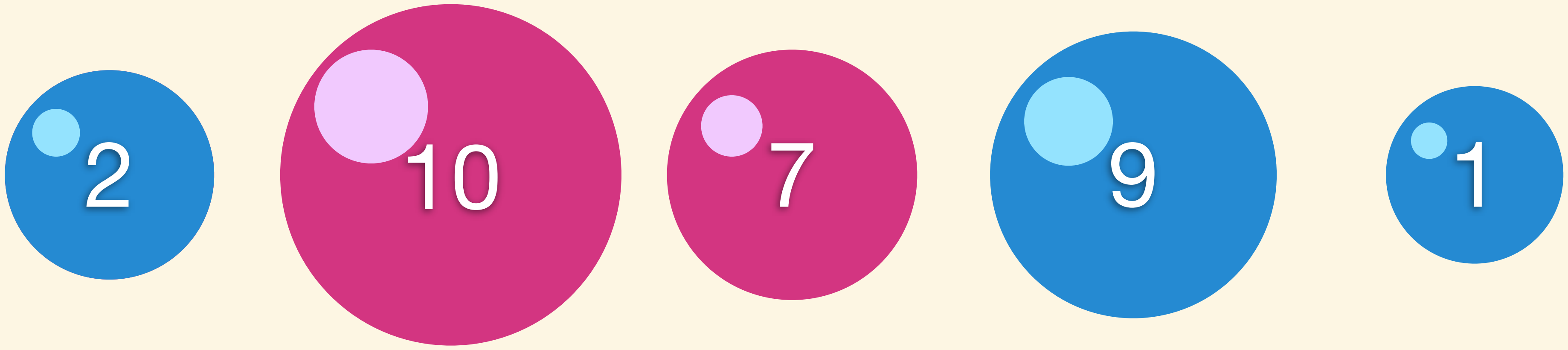
1st Iteration



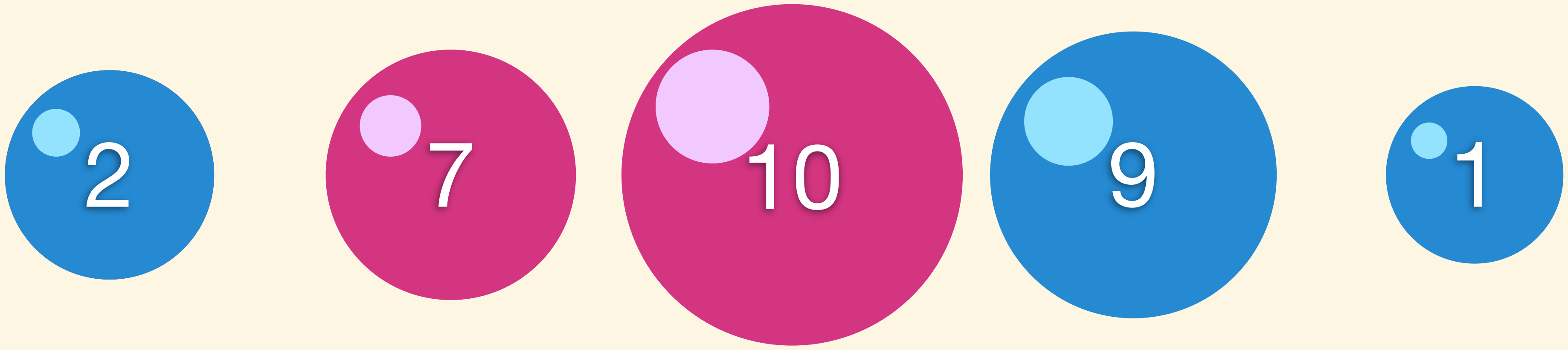
1st Iteration



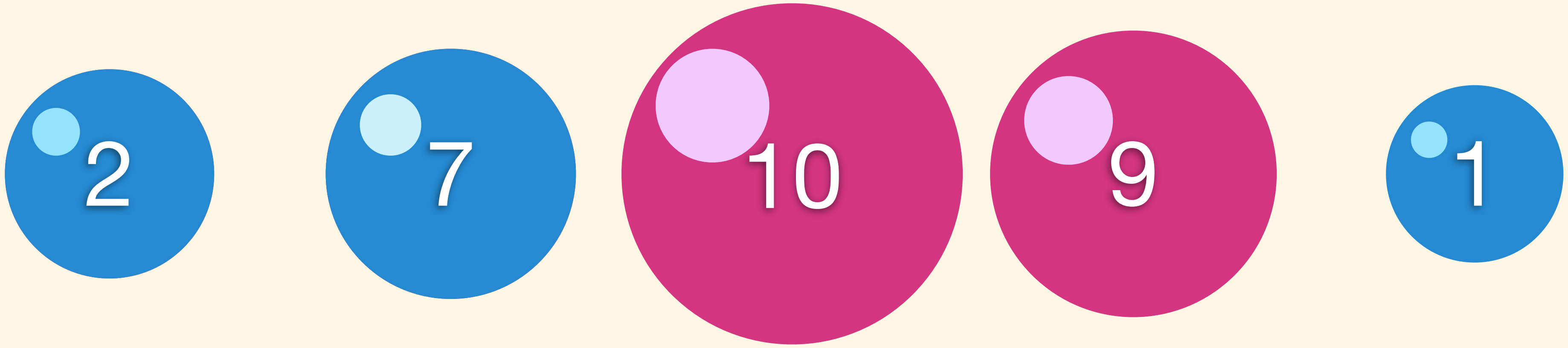
1st Iteration



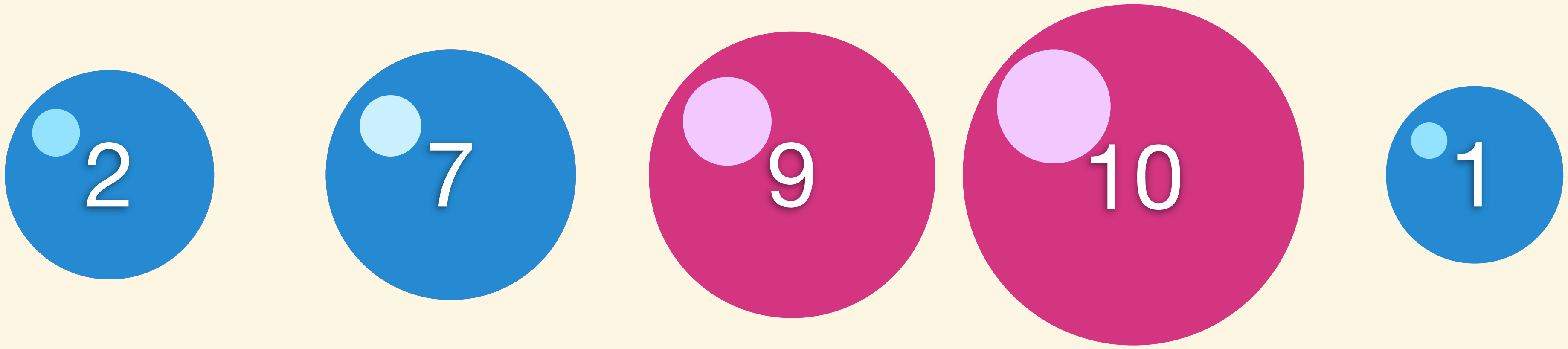
1st Iteration



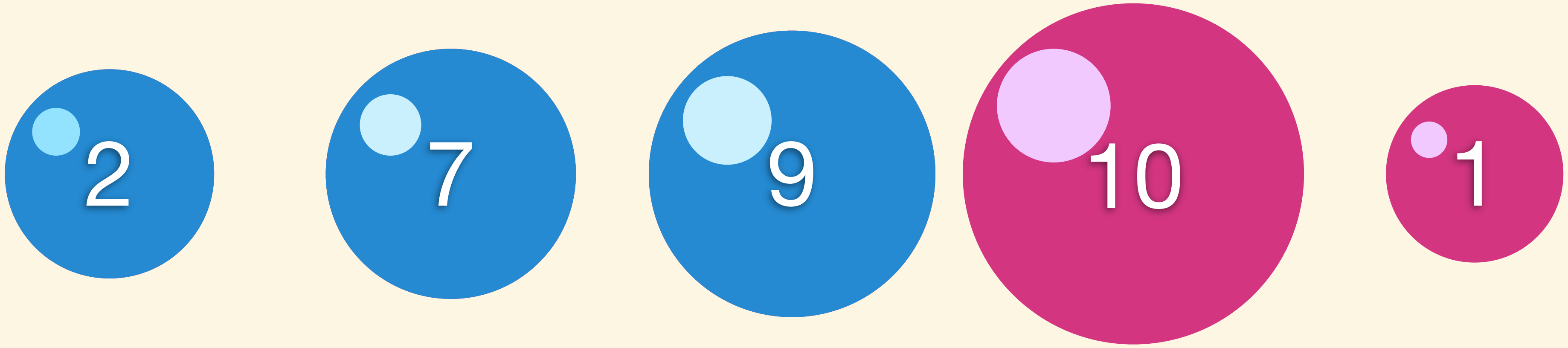
1st Iteration



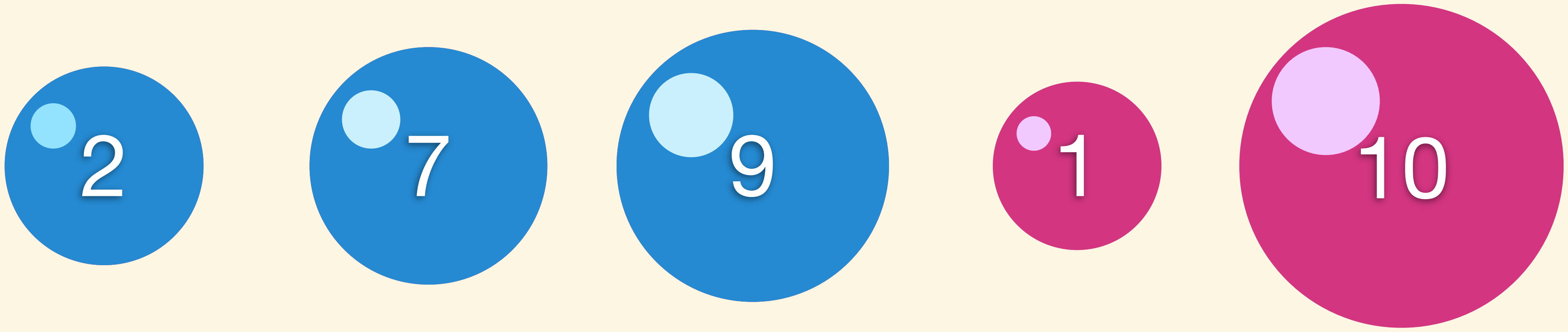
1st Iteration



1st Iteration

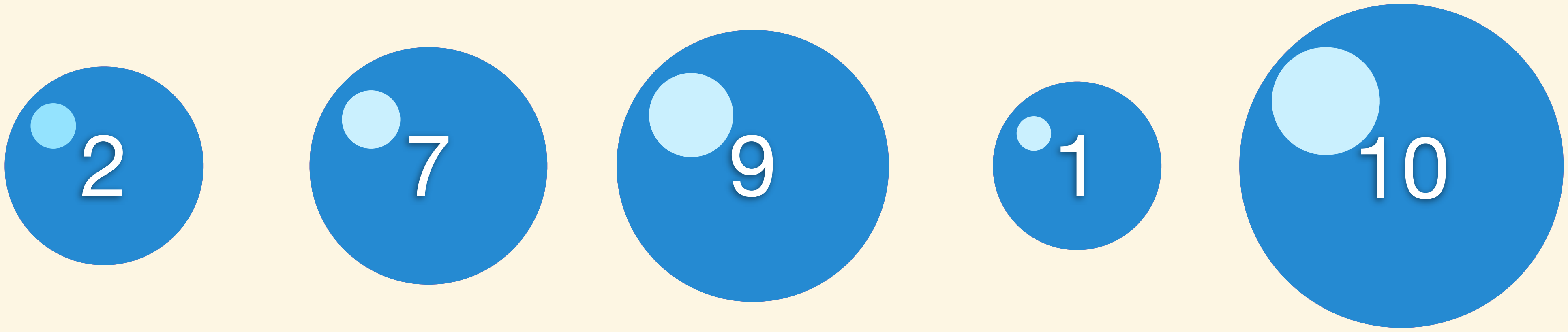


1st Iteration

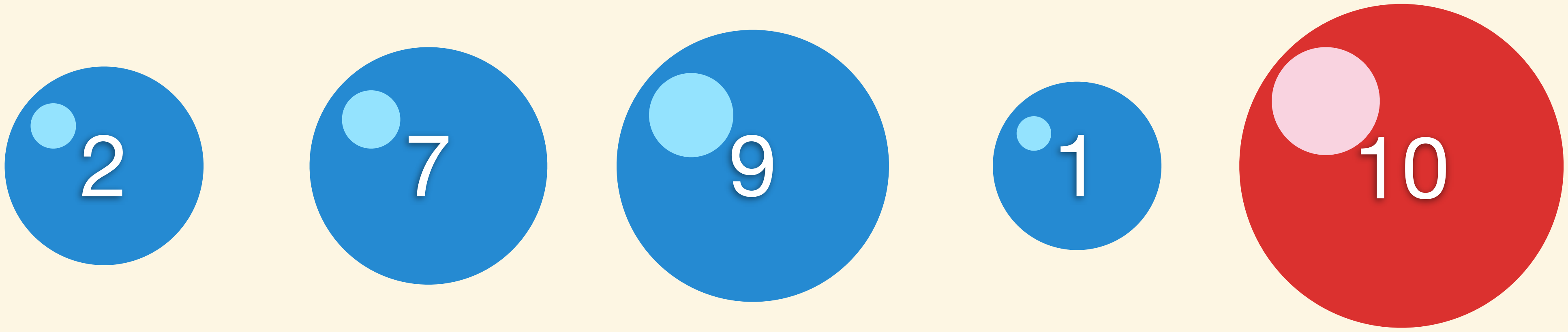


1st Iteration

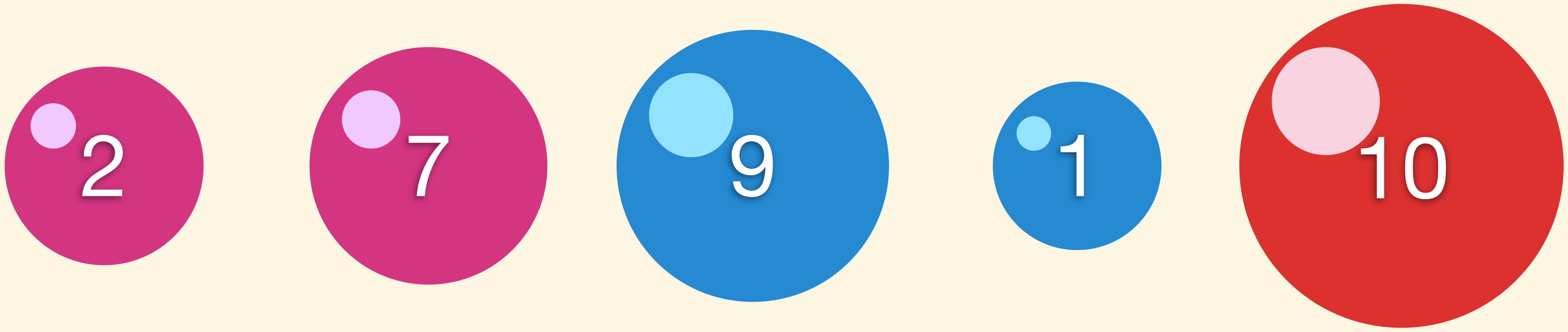
2	7	9	1	10



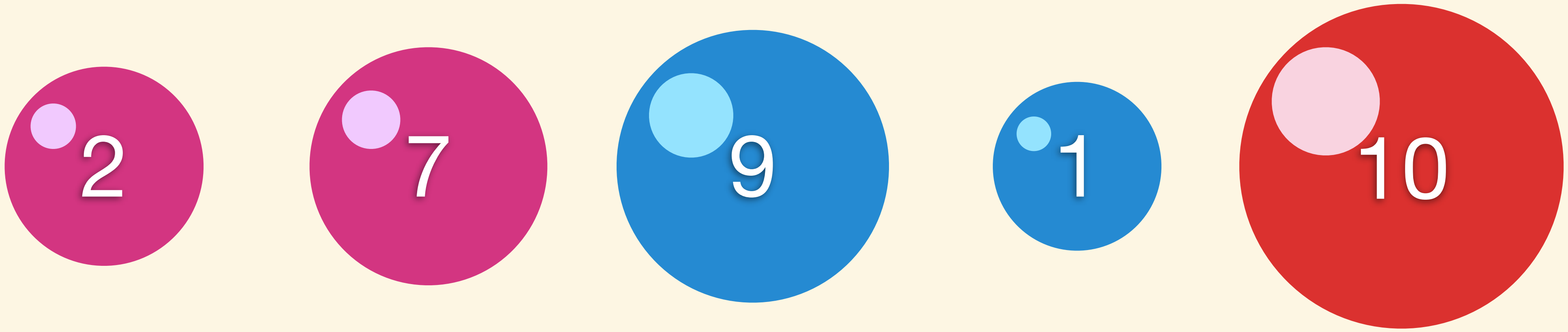
2nd Iteration



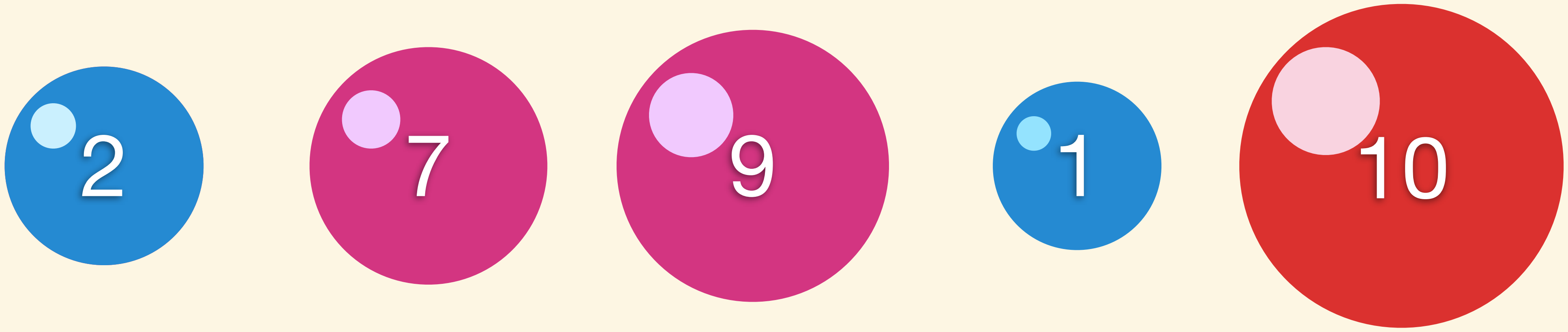
2nd Iteration



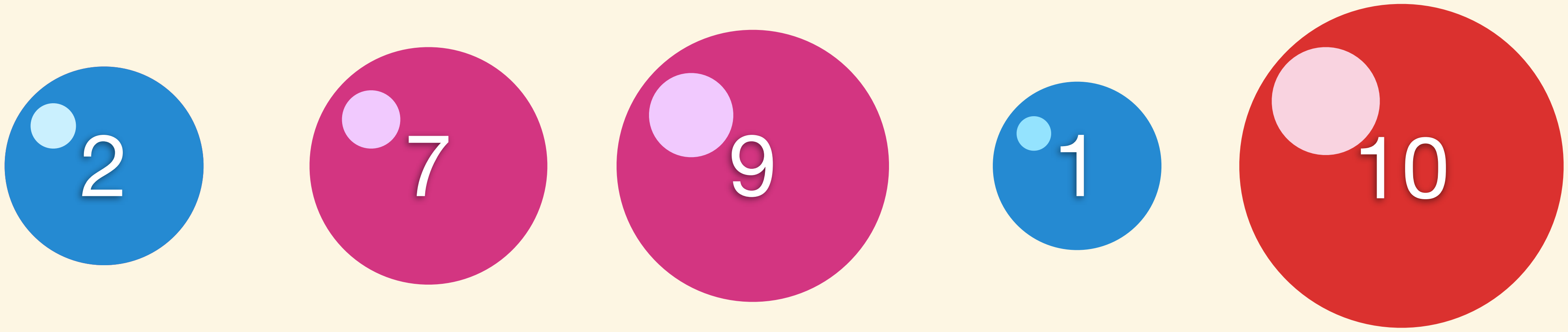
2nd Iteration



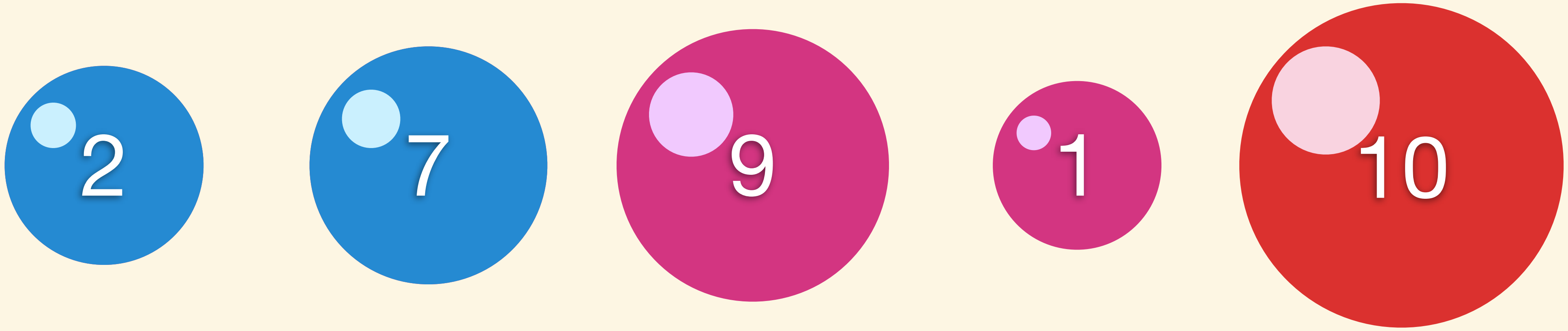
2nd Iteration



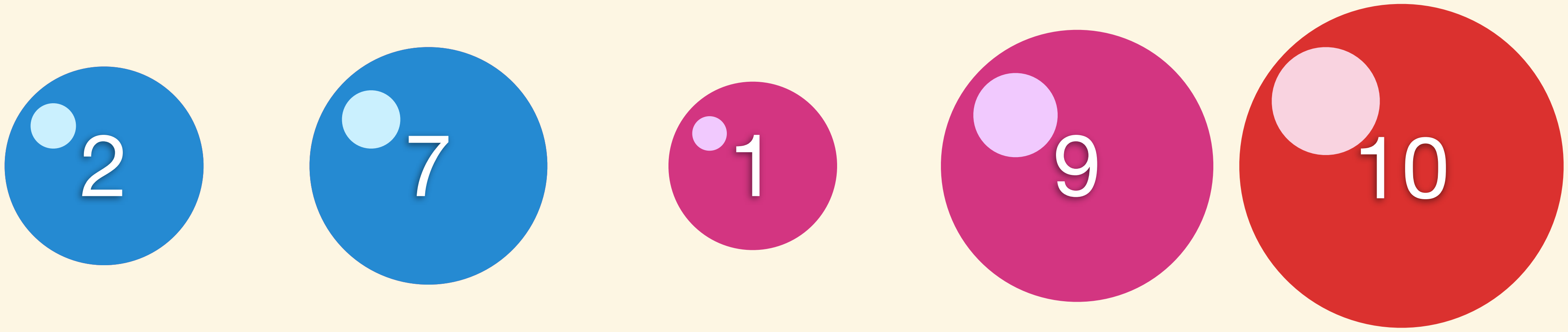
2nd Iteration



2nd Iteration

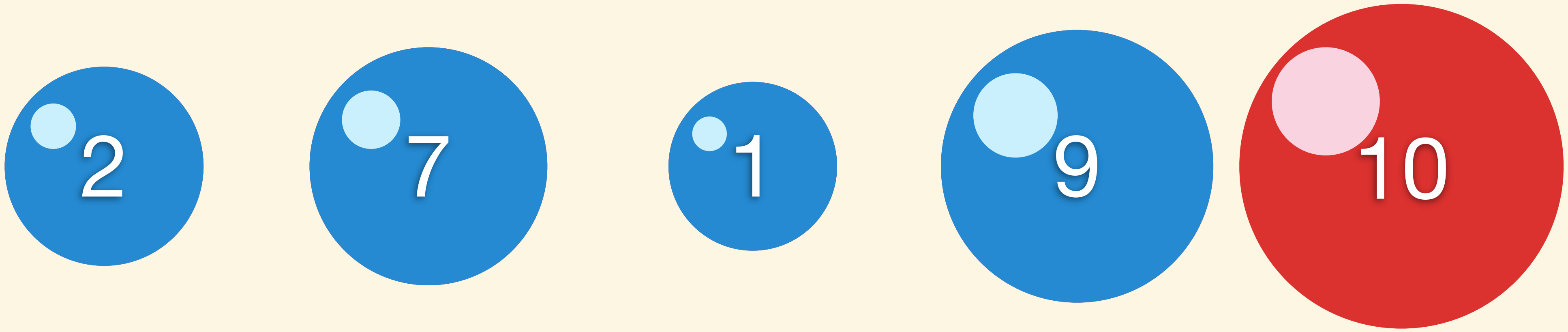


2nd Iteration

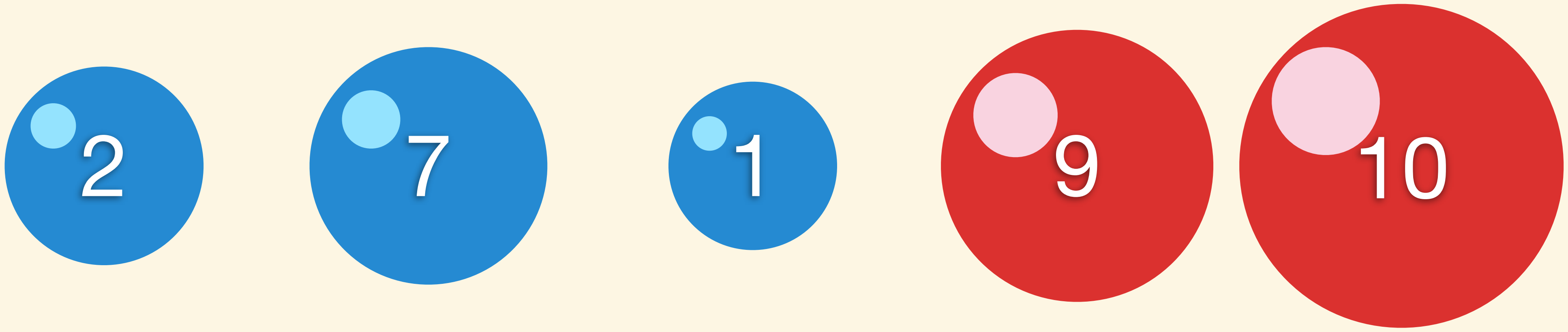


2nd Iteration

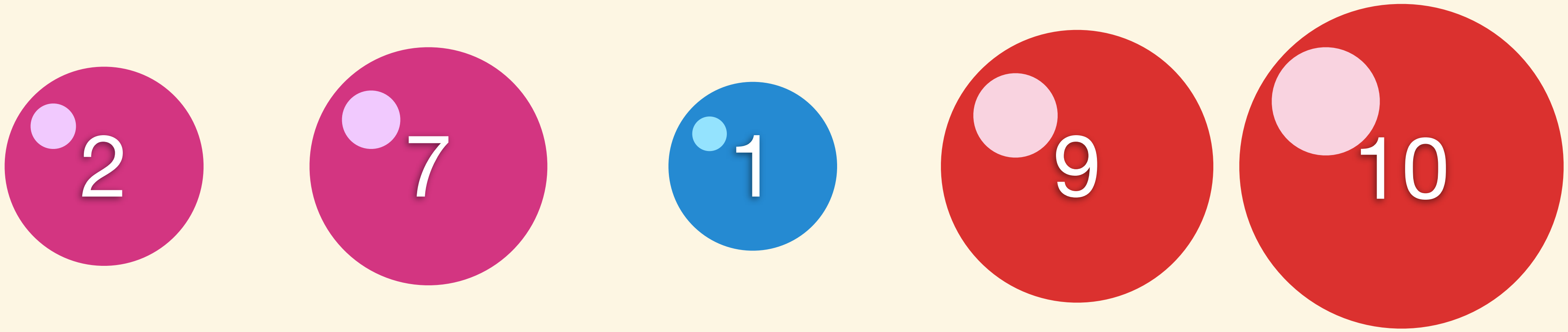
2	7	1	9	10



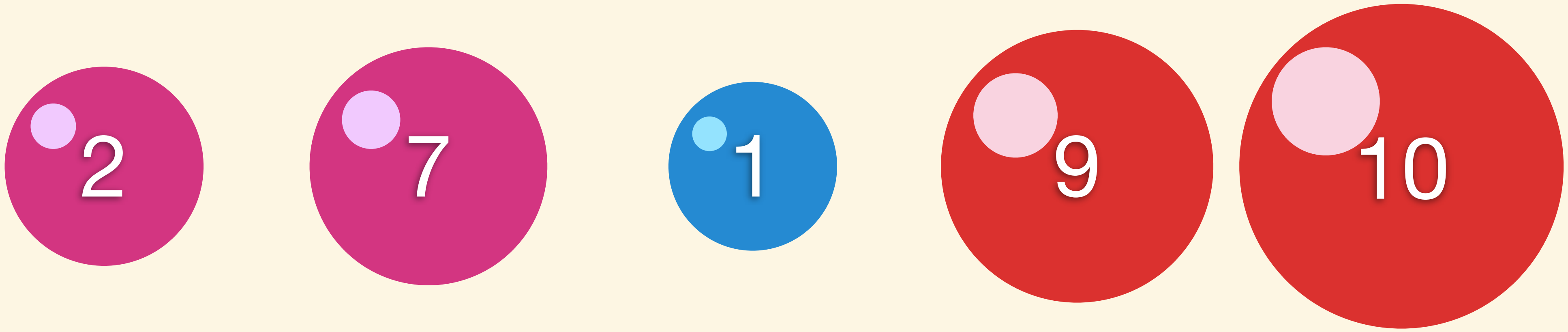
3rd Iteration



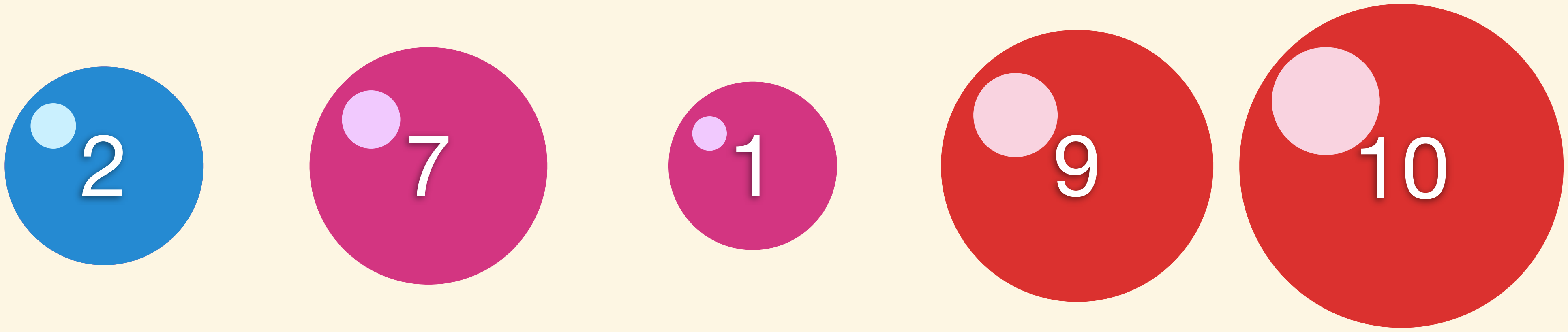
3rd Iteration



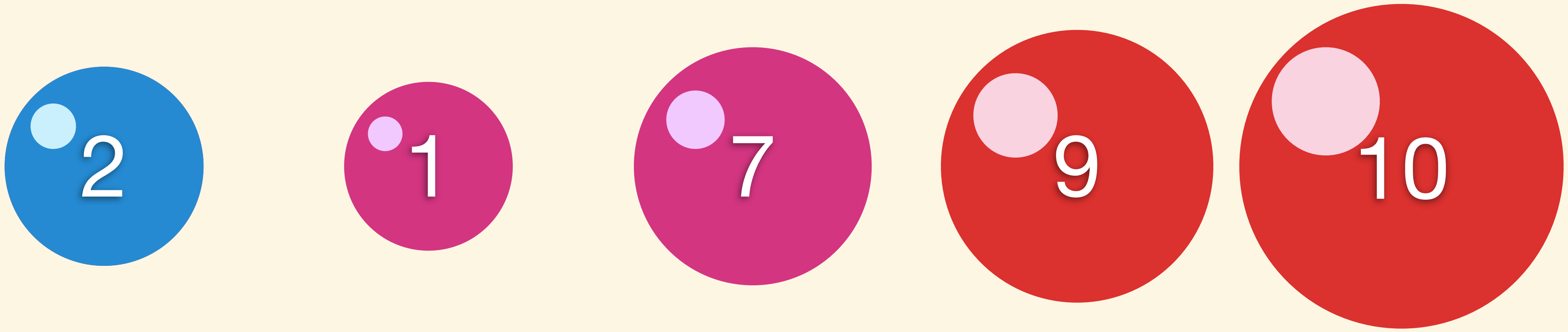
3rd Iteration



3rd Iteration

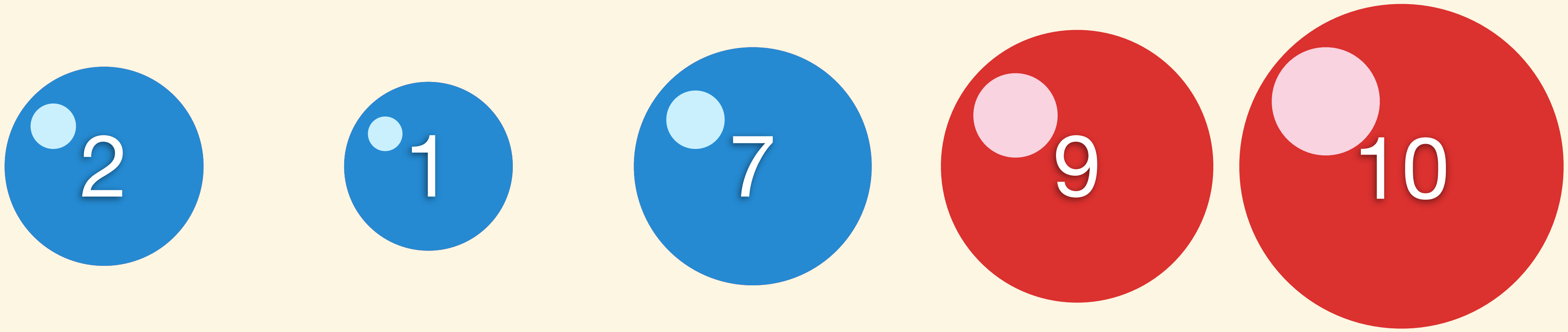


3rd Iteration

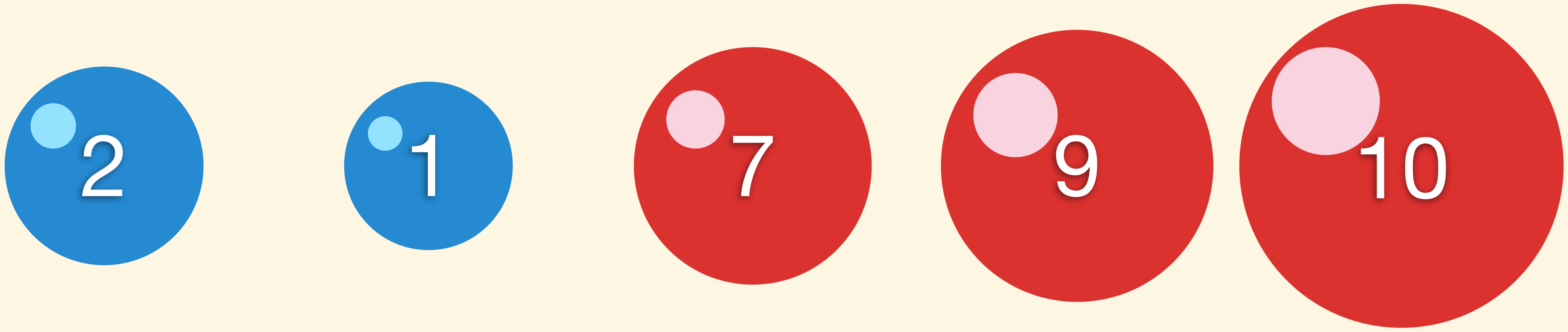


3rd Iteration

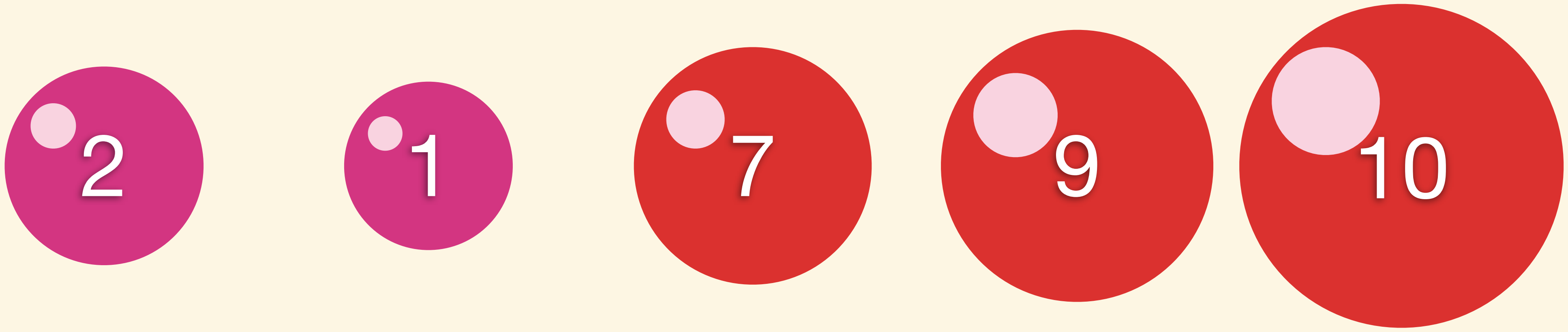
2	1	7	9	10



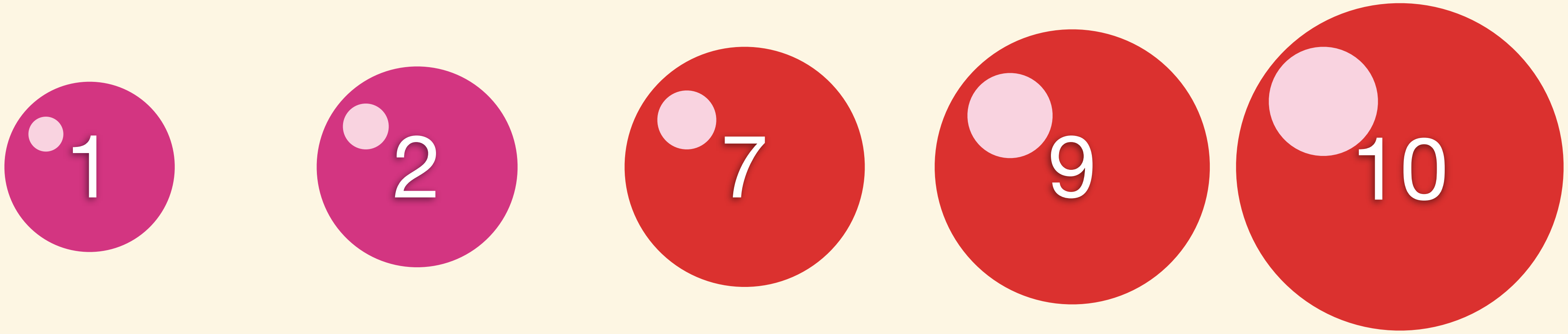
4th Iteration



4th Iteration

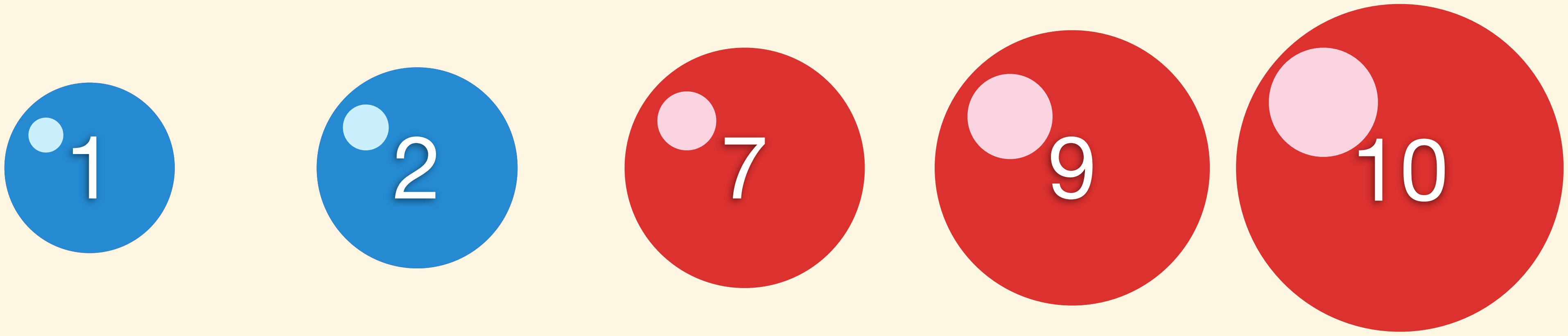


4th Iteration



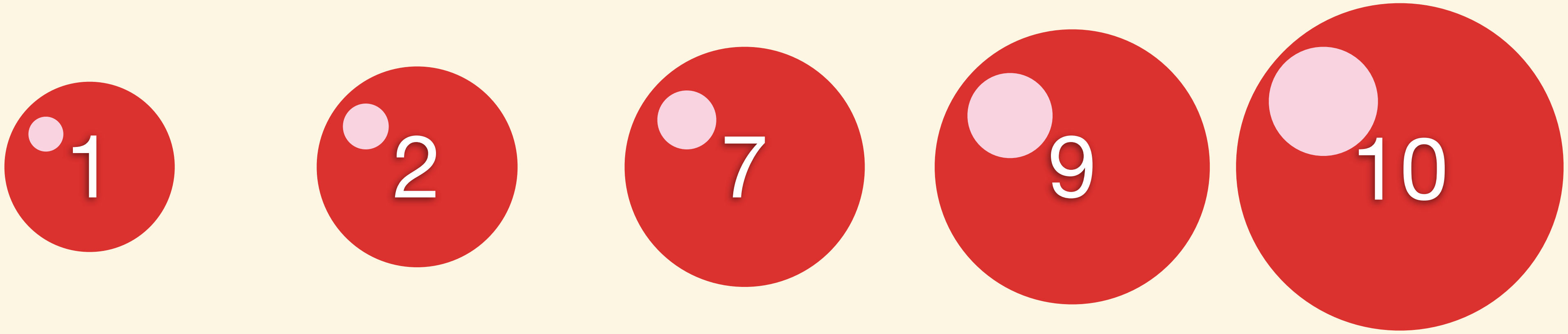
4th Iteration

1	2	7	9	10



Result

1	2	7	9	10



Source Code

```
for ( int i = 0; i < n; ++i )
 for ( int j = 0; j < n - 1 - i; ++j )
  if ( A[ j ] > A[ j + 1 ] )
   swap( A[ j ], A[ j + 1 ] );
```

Time Complexity

```
for ( int i = 0; i < n; ++i )
  for ( int j = 0; j < n - 1 - i; ++j )
    if ( A[ j ] > A[ j + 1 ] )
      swap( A[ j ], A[ j + 1 ] );
```

	最大循環次數略估
第 1 層迴圈	n
第 2 層迴圈	n

$$n \times n = n^2$$

Time Complexity: O(n^2)

從範例中可發現...

耗費許多時間檢查**有序數對**

Merge Sort

合併排序法

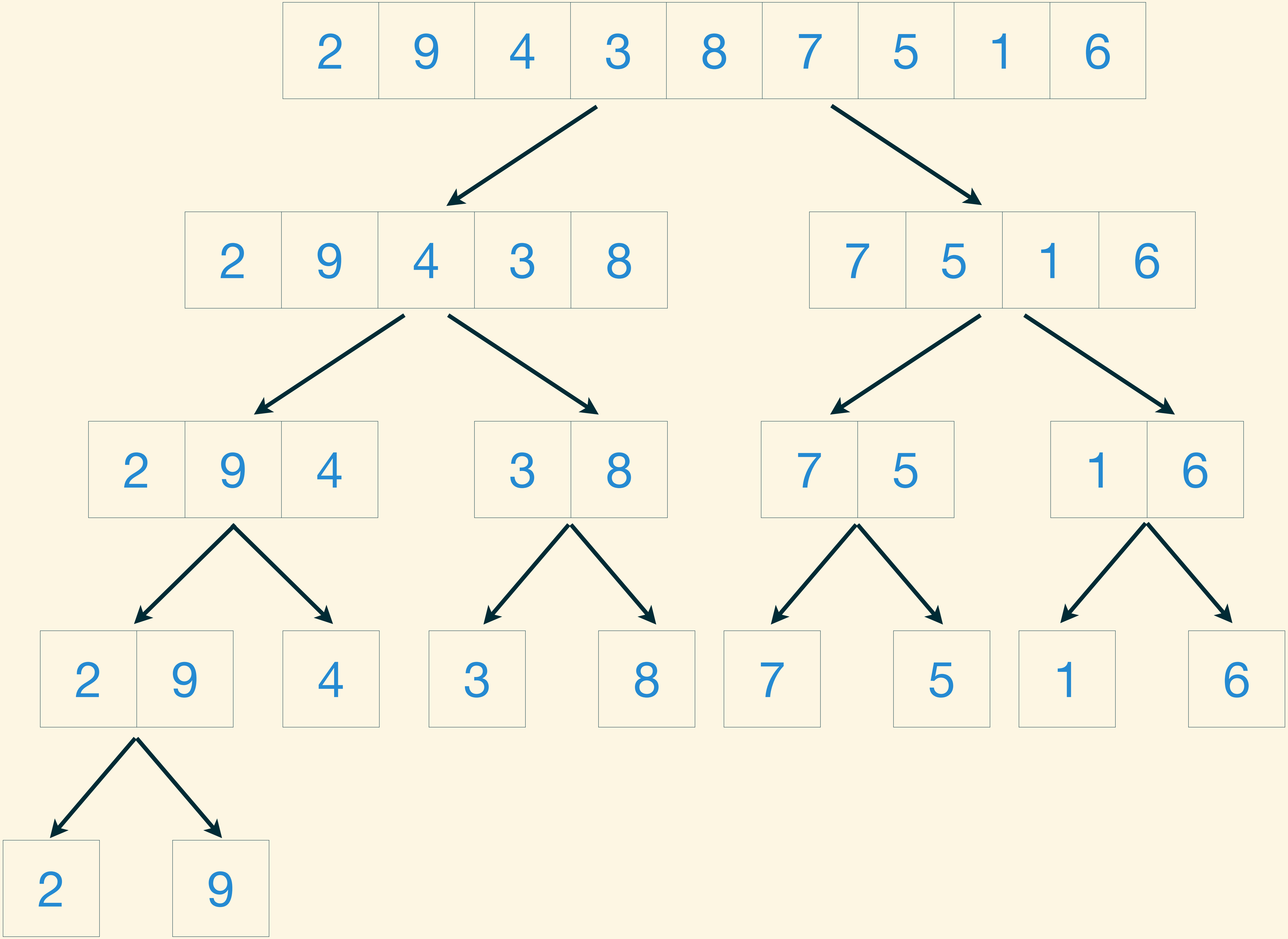
Algorithm

採用 divide & conquer 策略

Divide

將當前數列對半切割遞迴進行

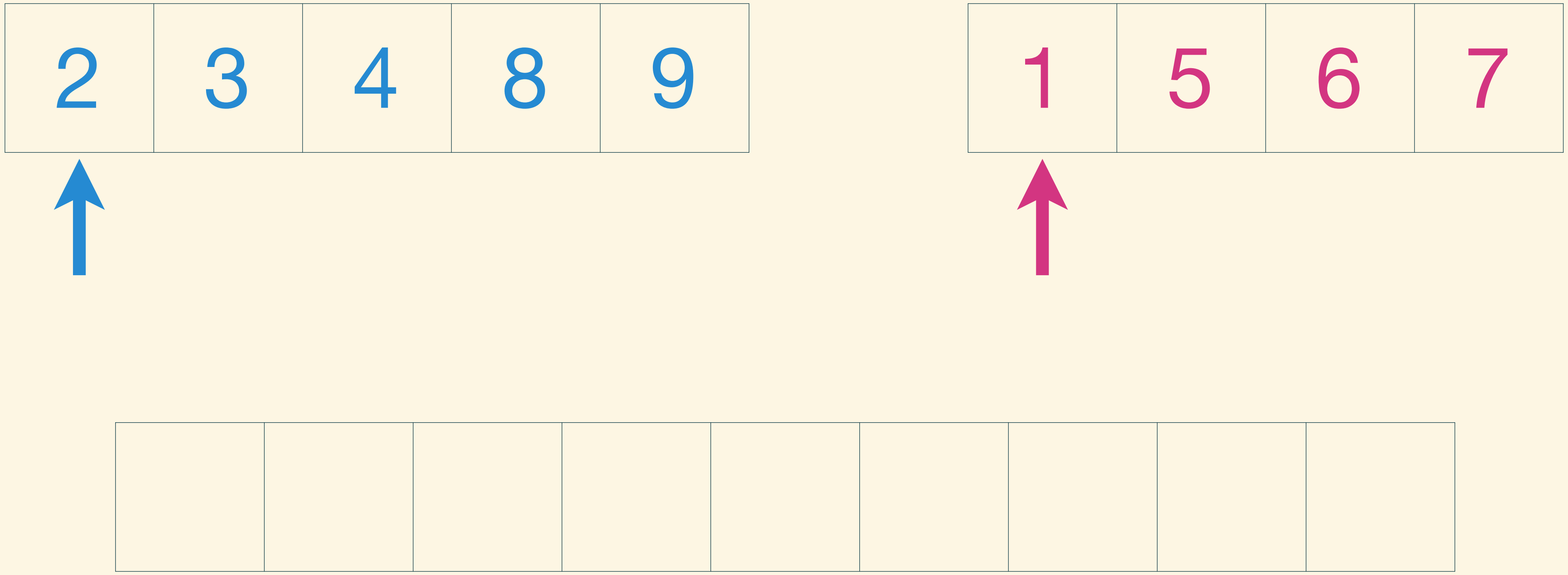
直到僅剩一個元素


2 9 4 3 8 7 5 1 6
2 9 4 3 8
7 5 1 6
2 9 4
3 8
7 5
1 6
2 9
4
3
8
7
5
1
6
2
9

Conquer

1. 利用兩個指標指向兩個有序數列 A 與 B
2. 比較指標指向的數值
3. 將較小的數值放入新的數列 C，並將該指標指向下一數值
4. 直到某一指標指向數列結尾，將另一數列剩餘的數值放都入數列 C

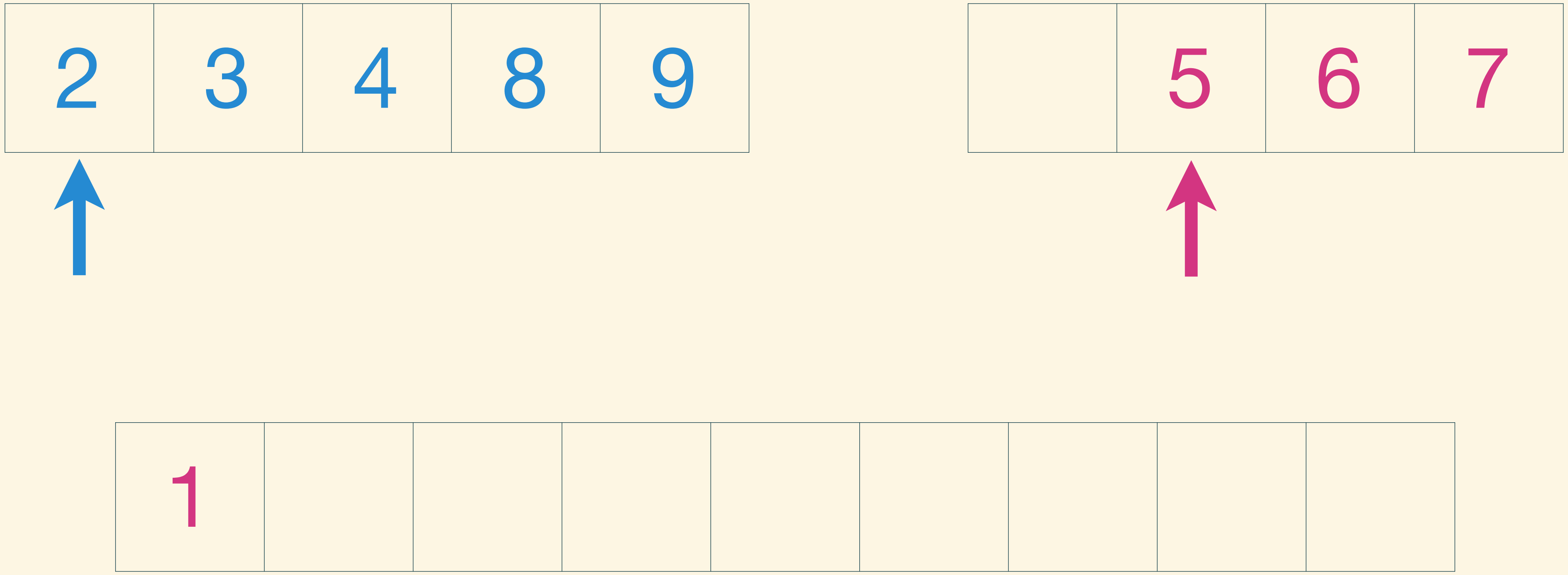
Merge Two Sequence



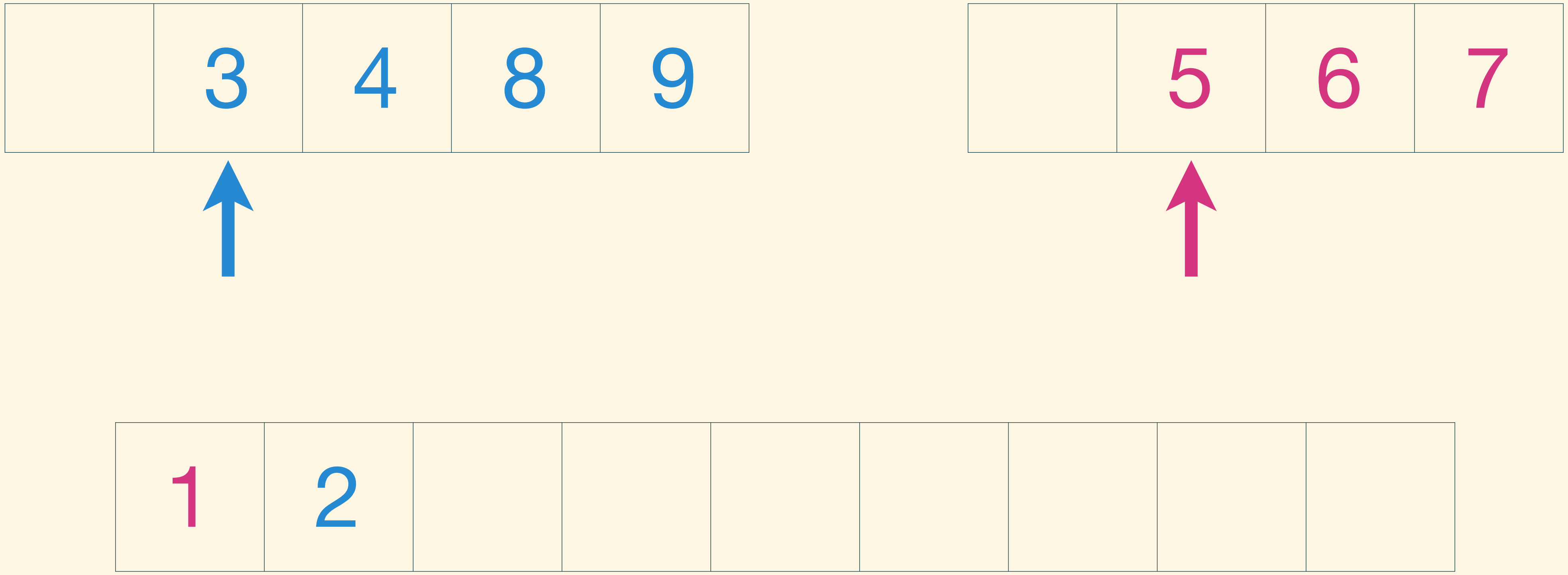
2	3	4	8	9

1	5	6	7

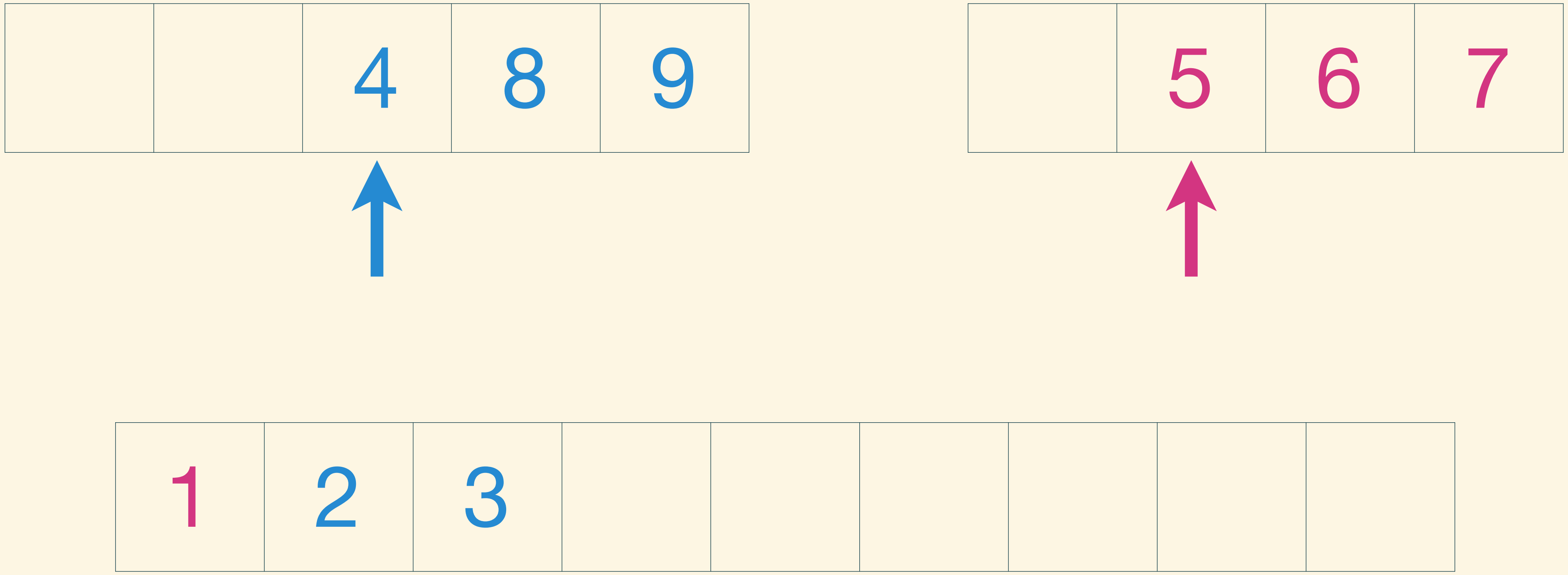
Merge Two Sequence



Merge Two Sequence



Merge Two Sequence

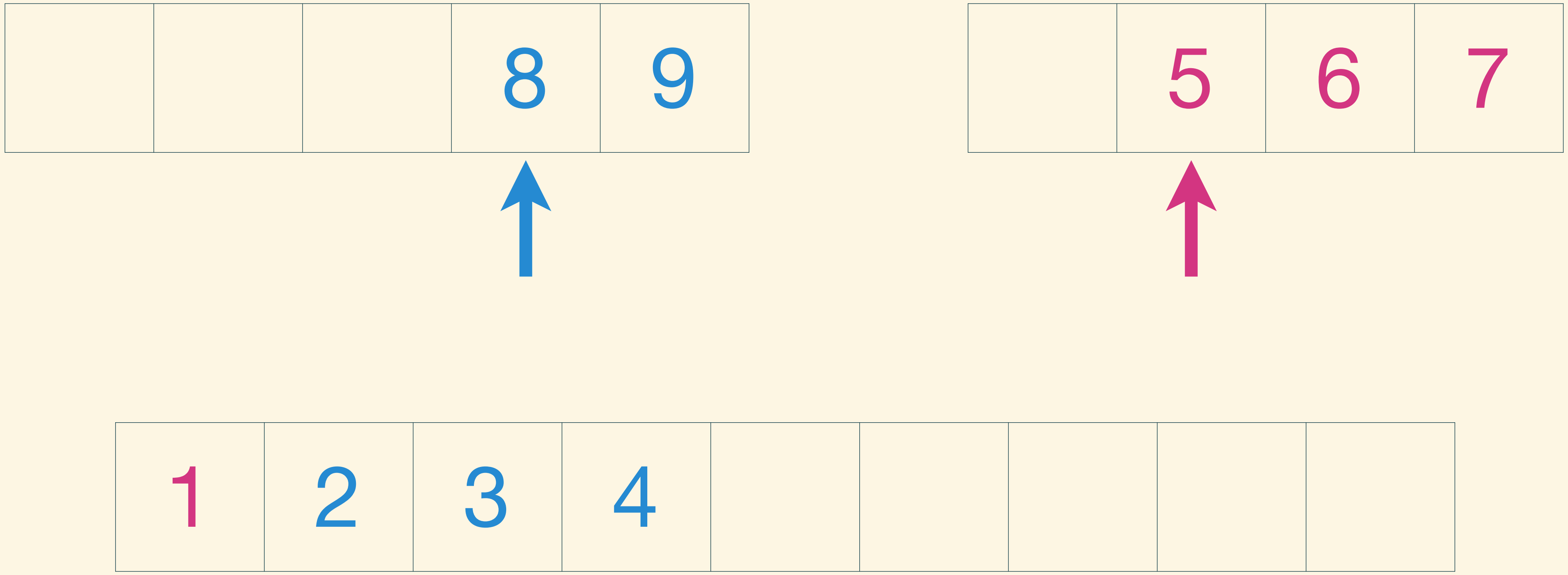


		4	8	9

	5	6	7

1	2	3						

Merge Two Sequence

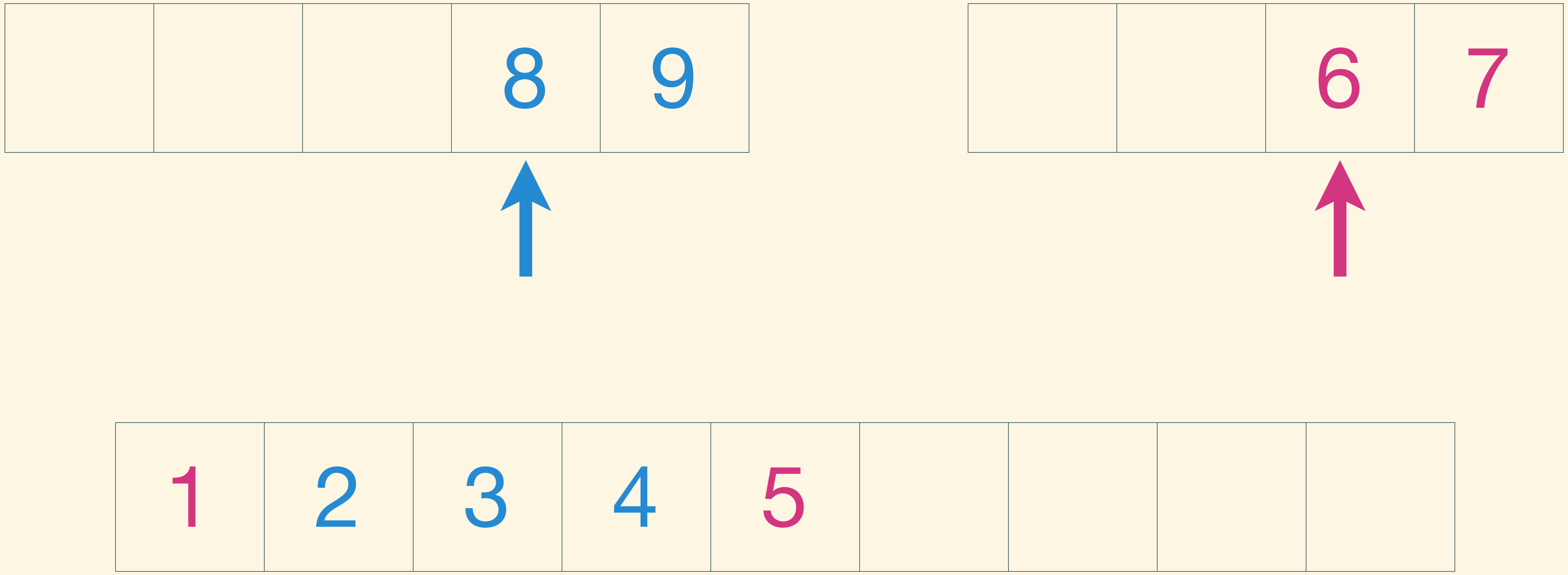


8 9

5 6 7

1 2 3 4

Merge Two Sequence

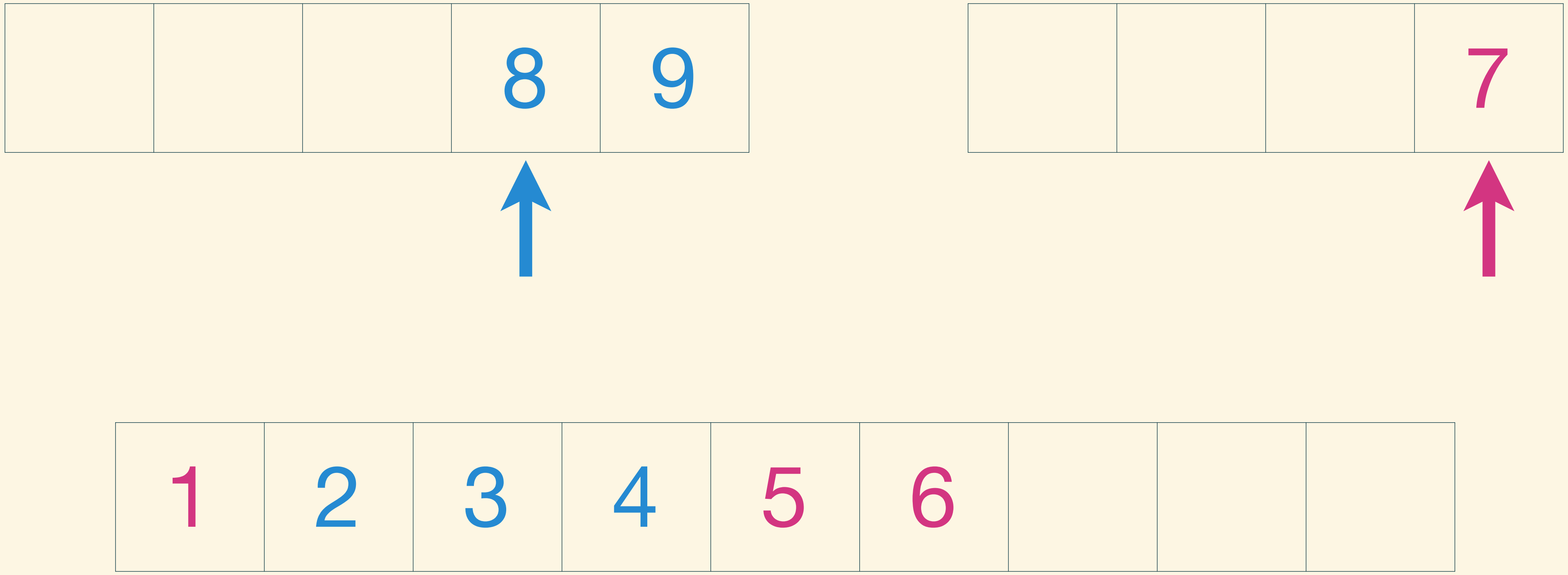


			8	9

		6	7

1	2	3	4	5				

Merge Two Sequence

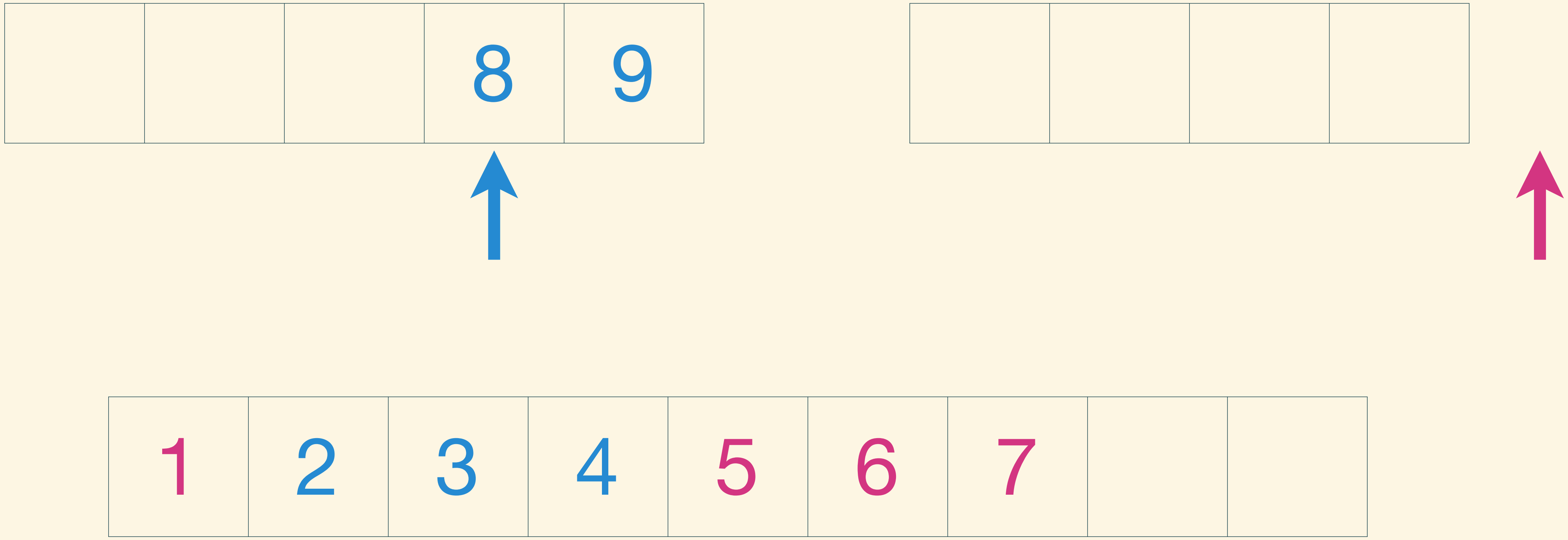


			8	9

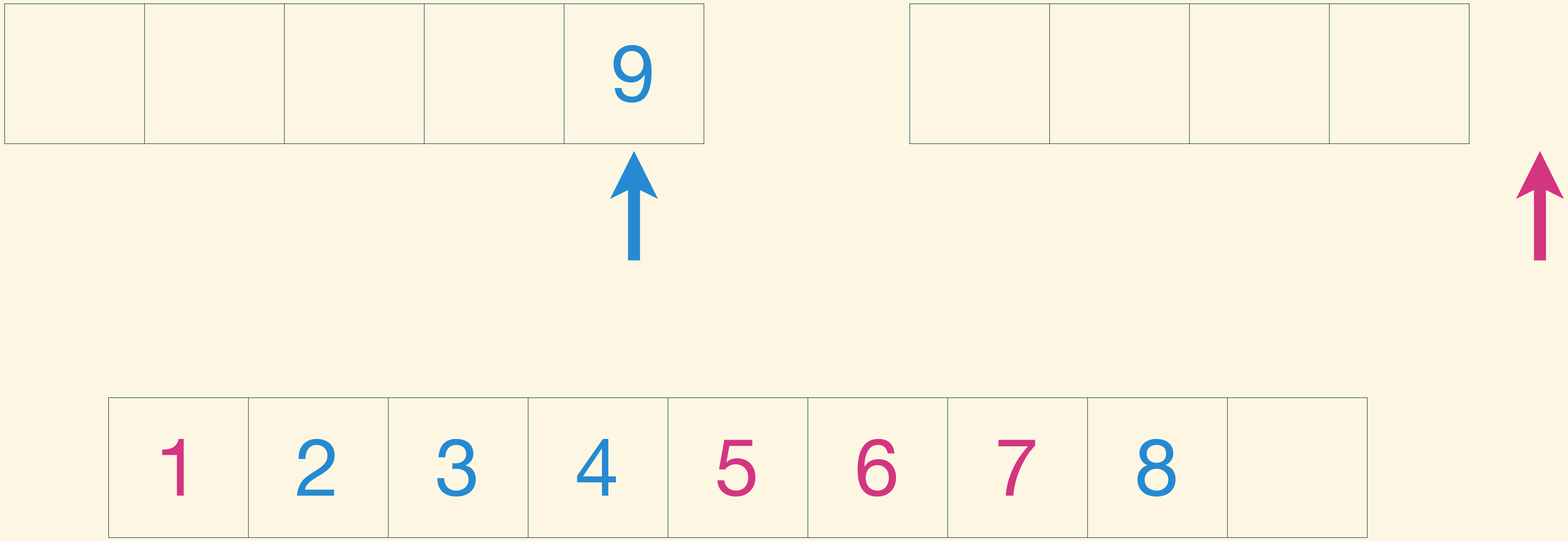
			7

1	2	3	4	5	6			

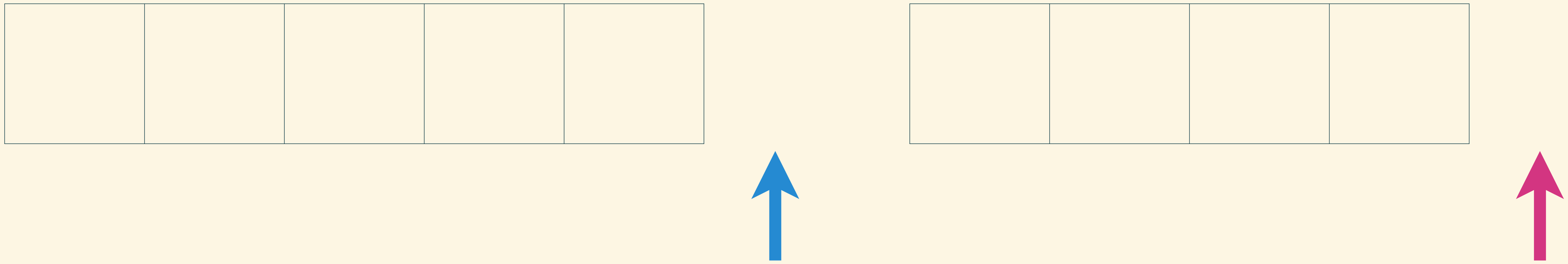
Merge Two Sequence



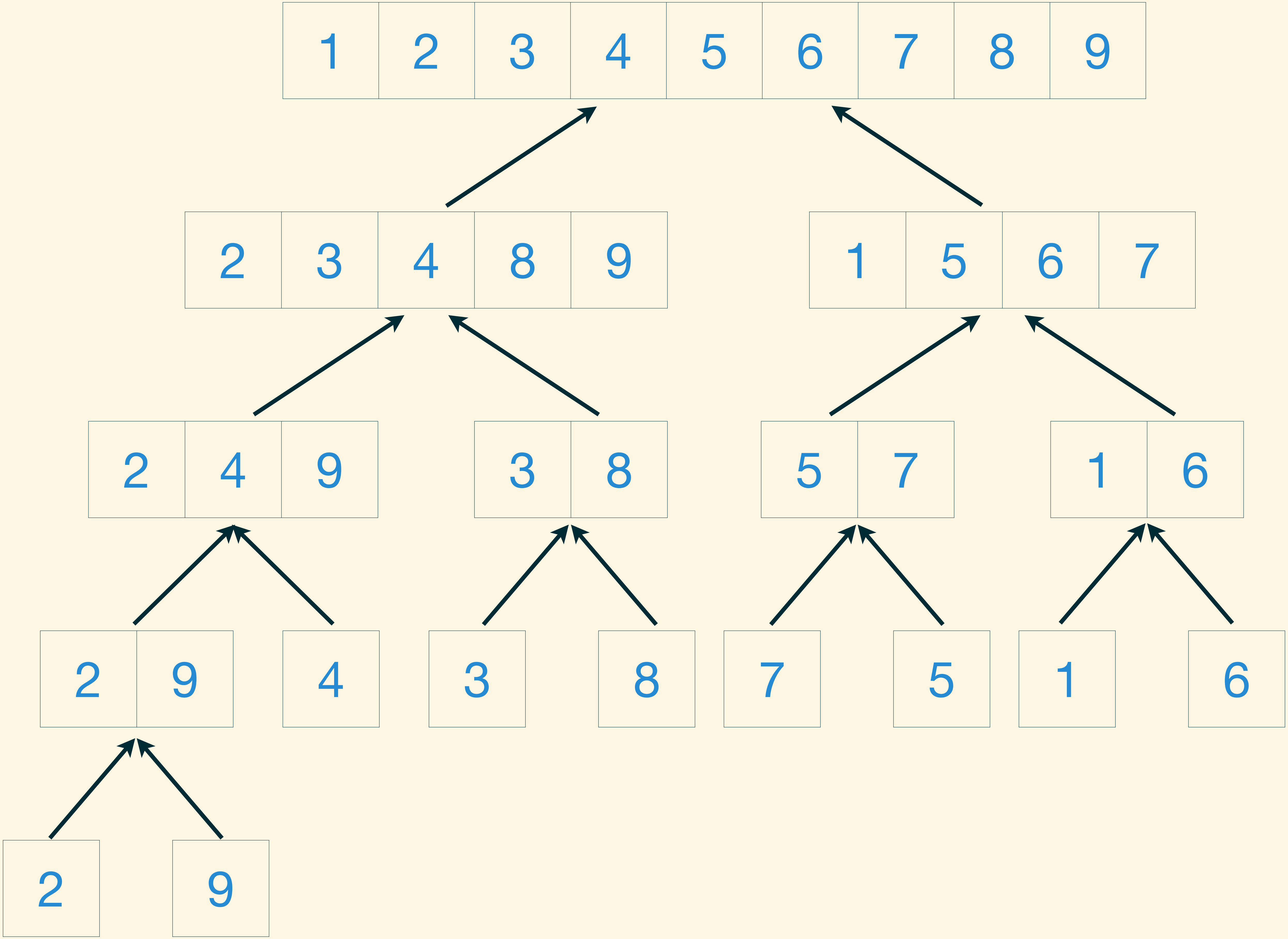
Merge Two Sequence



Merge Two Sequence



1	2	3	4	5	6	7	8	9


1 2 3 4 5 6 7 8 9
2 3 4 8 9
1 5 6 7
2 4 9
3 8
5 7
1 6
2 9
4
3
8
7
5
1
6
2
9

Source Code

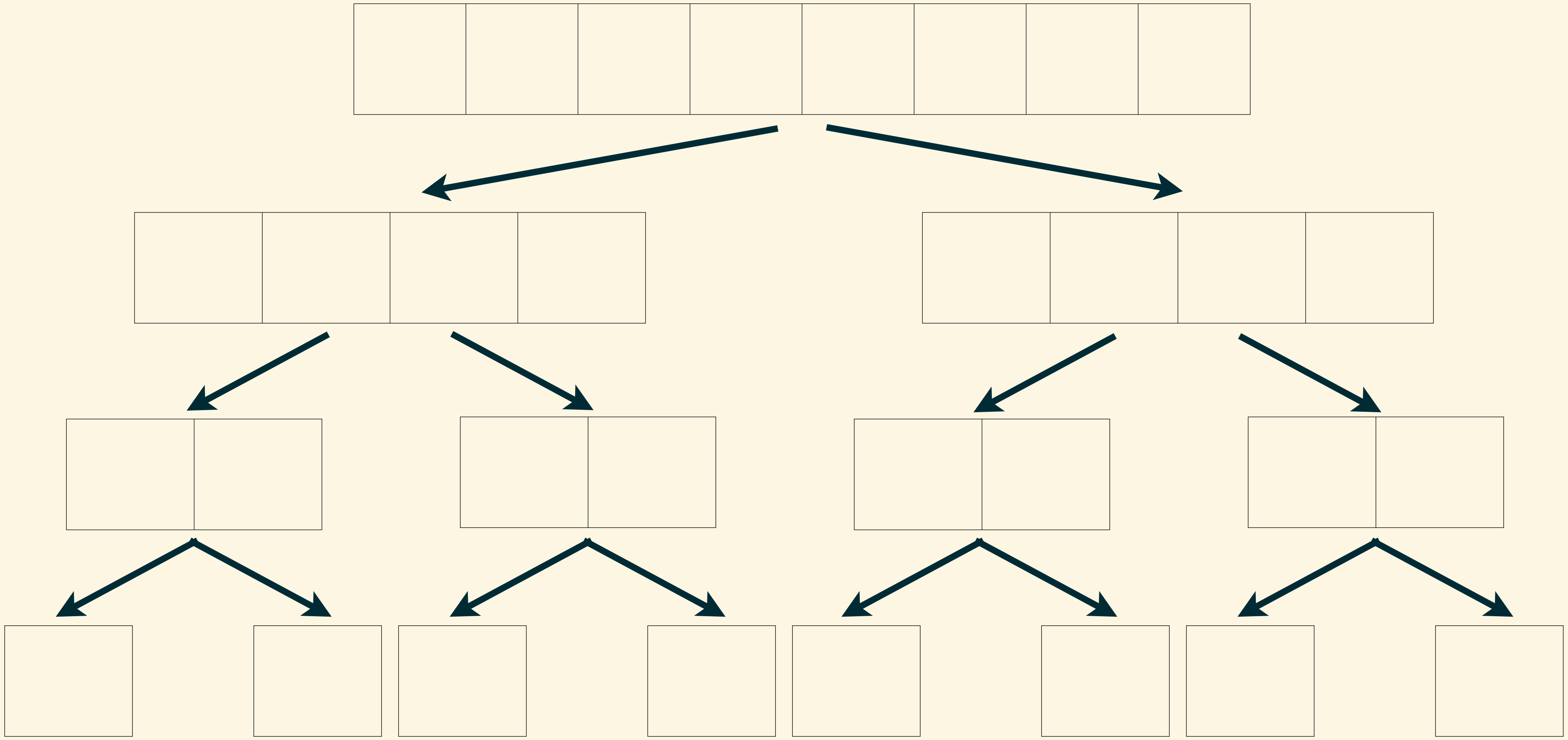
```
int mergeSort( int L, int R ) {
  if ( R - L == 1 )
    return;
  int mid = ( R + L ) / 2, C[ R - L ];
  mergeSort( L, mid );
  mergeSort( mid, R );
  for ( int p1 = L, p2 = mid, p = 0; p < R - L; ++p ) {
    if ( ( p1 != mid && A[ p1 ] < A[ p2 ] ) || p2 == R )
      C[ p ] = A[ p1++ ];
    else
      C[ p ] = A[ p2++ ];
  }
  for ( int i = L; i < R; ++i )
    A[ i ] = C[ i - L ];
}
```

Time Complexity

total time

Time Complexity

total time



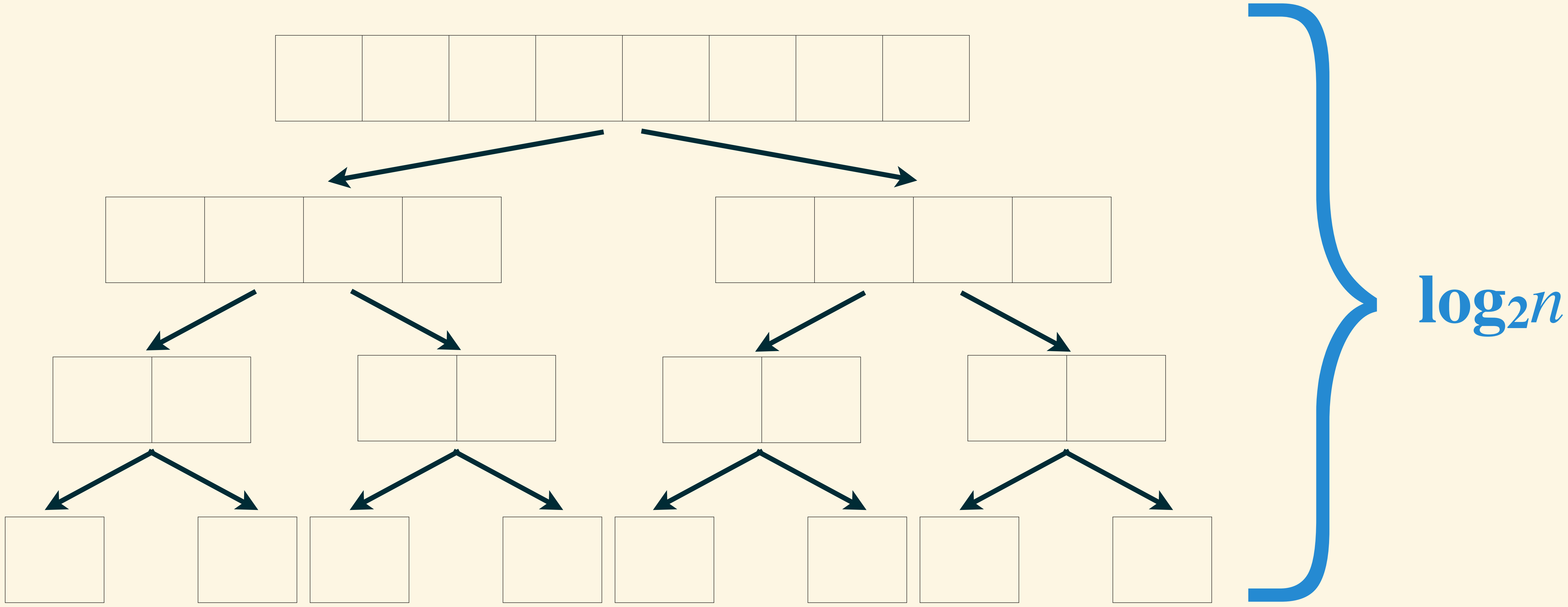
n

2 * (n/2) = n

...

n * (n/n) = n

Best height



Time Complexity: O($n\mathbf{log}_2 n$)

Quick Sort

快速排序法

Algorithm

採用 divide & conquer 策略

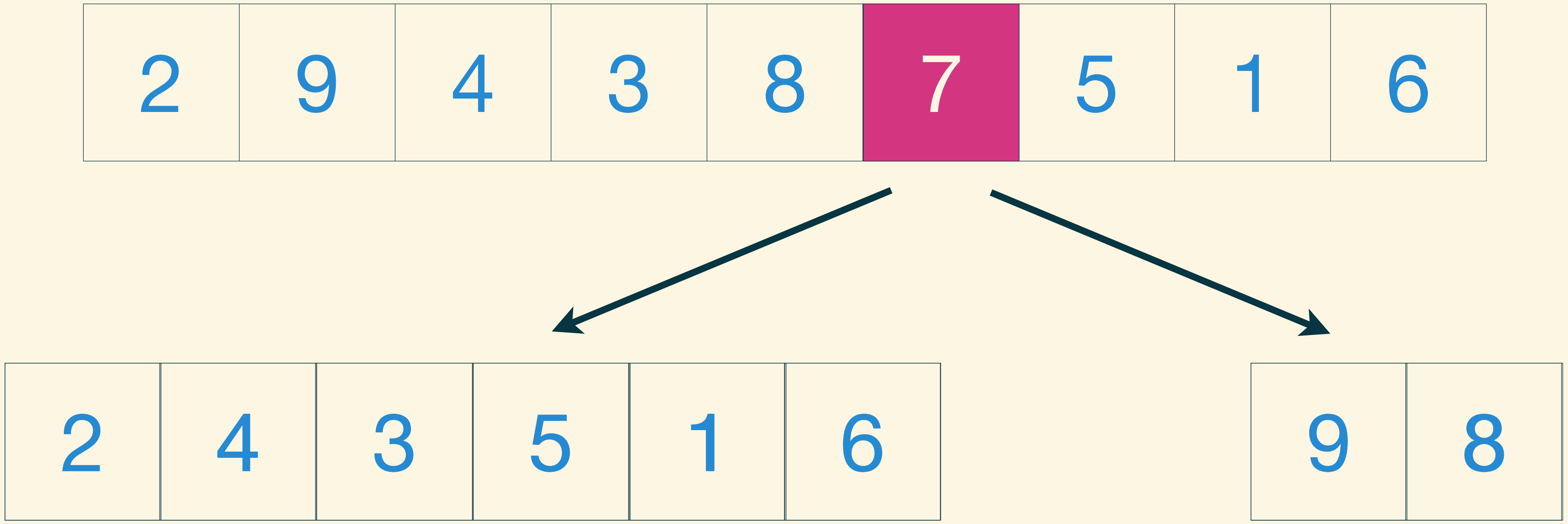
Divide

從數列中挑出一個元素作為 pivot， 利用 pivot 將原數列分為兩個子數列：

- 所有元素小於 pivot 的數列 A
- 所有元素大於 pivot 的元素的數列 B
- 等於 pivot 的元素可放置在任一個中

2	9	4	3	8	7	5	1	6

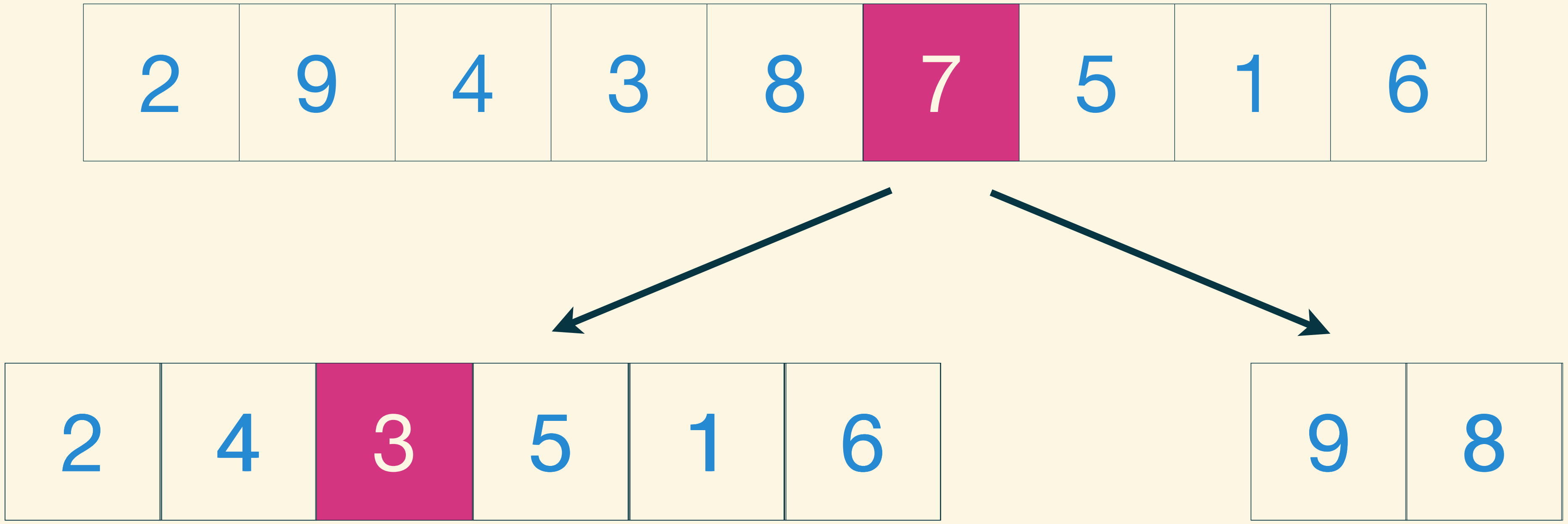
2	9	4	3	8	**7**	5	1	6



2	9	4	3	8	7	5	1	6

2	4	3	5	1	6

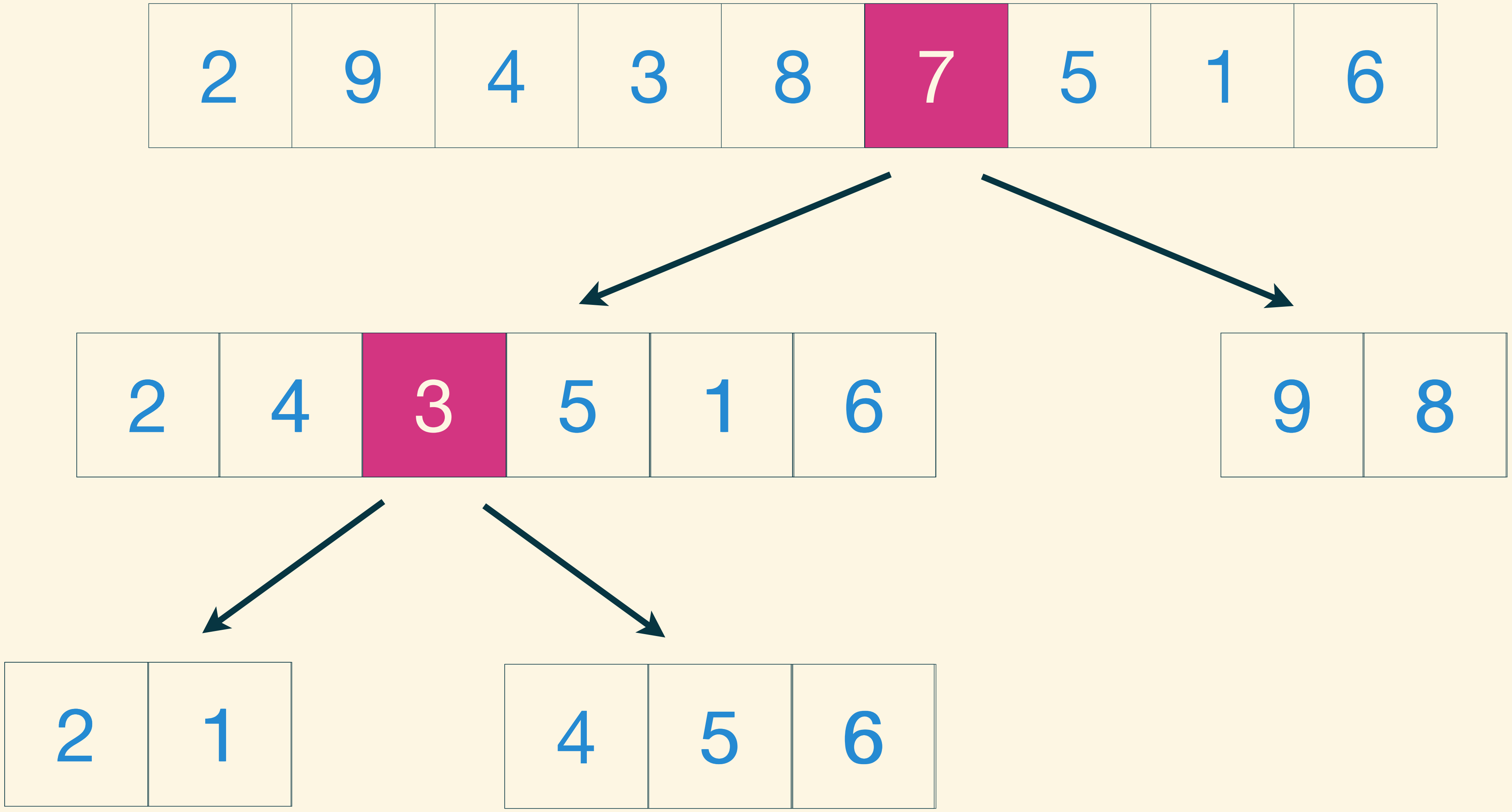
9	8



2 9 4 3 8 7 5 1 6

2 4 3 5 1 6

9 8



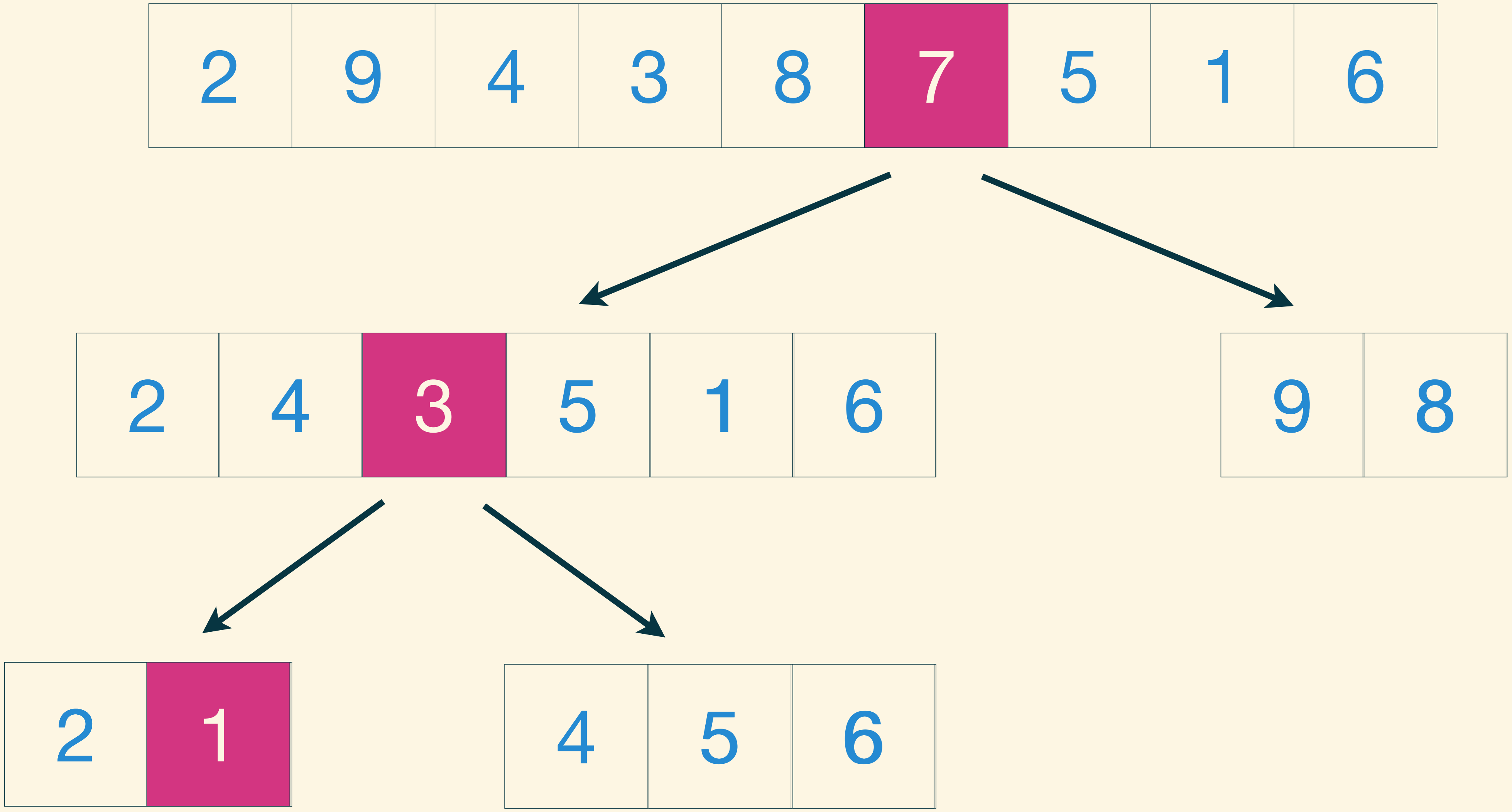
2 9 4 3 8 7 5 1 6

2 4 3 5 1 6

9 8

2 1

4 5 6



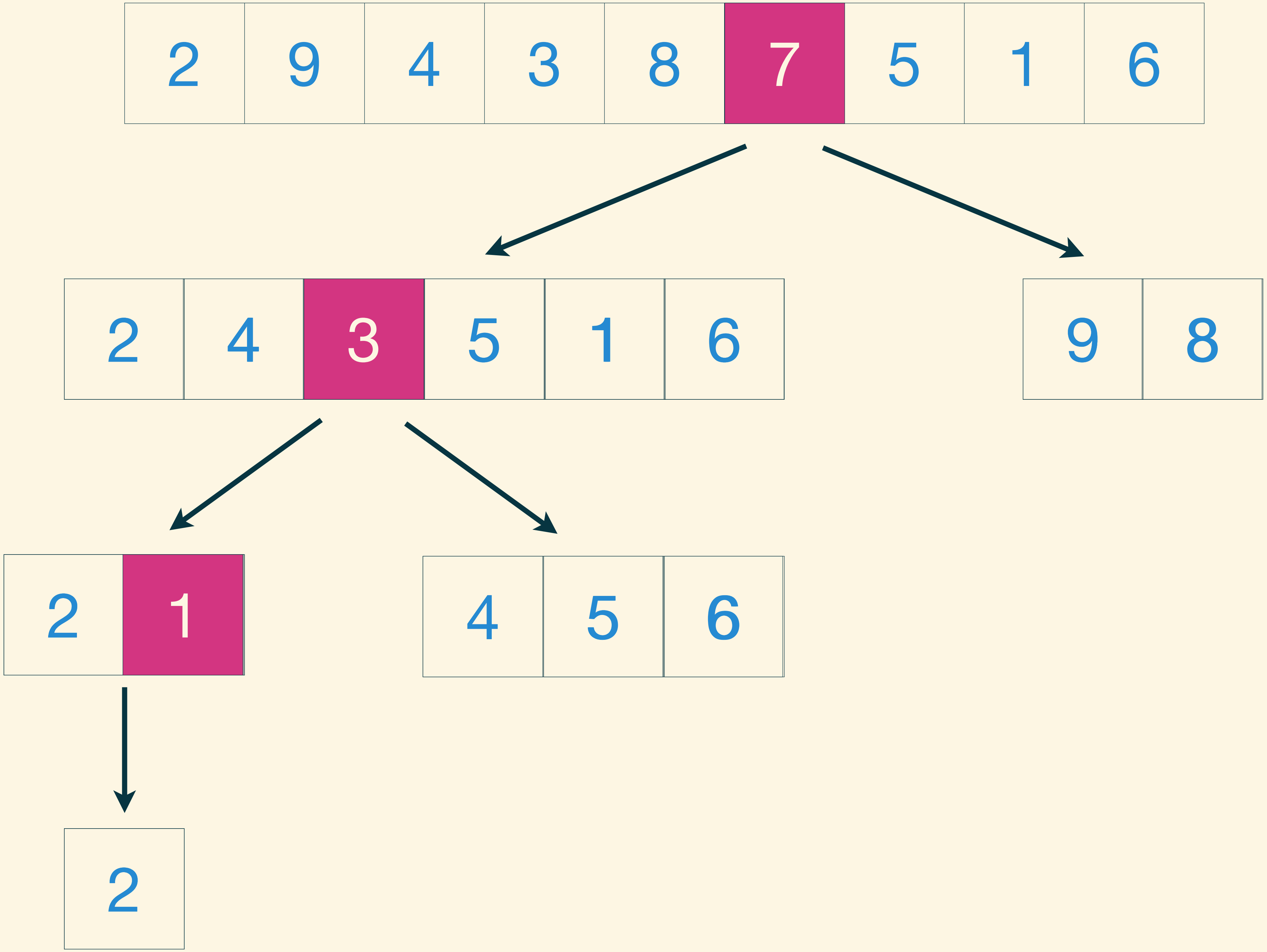
2 9 4 3 8 7 5 1 6

2 4 3 5 1 6

9 8

2 1

4 5 6



2 9 4 3 8 7 5 1 6

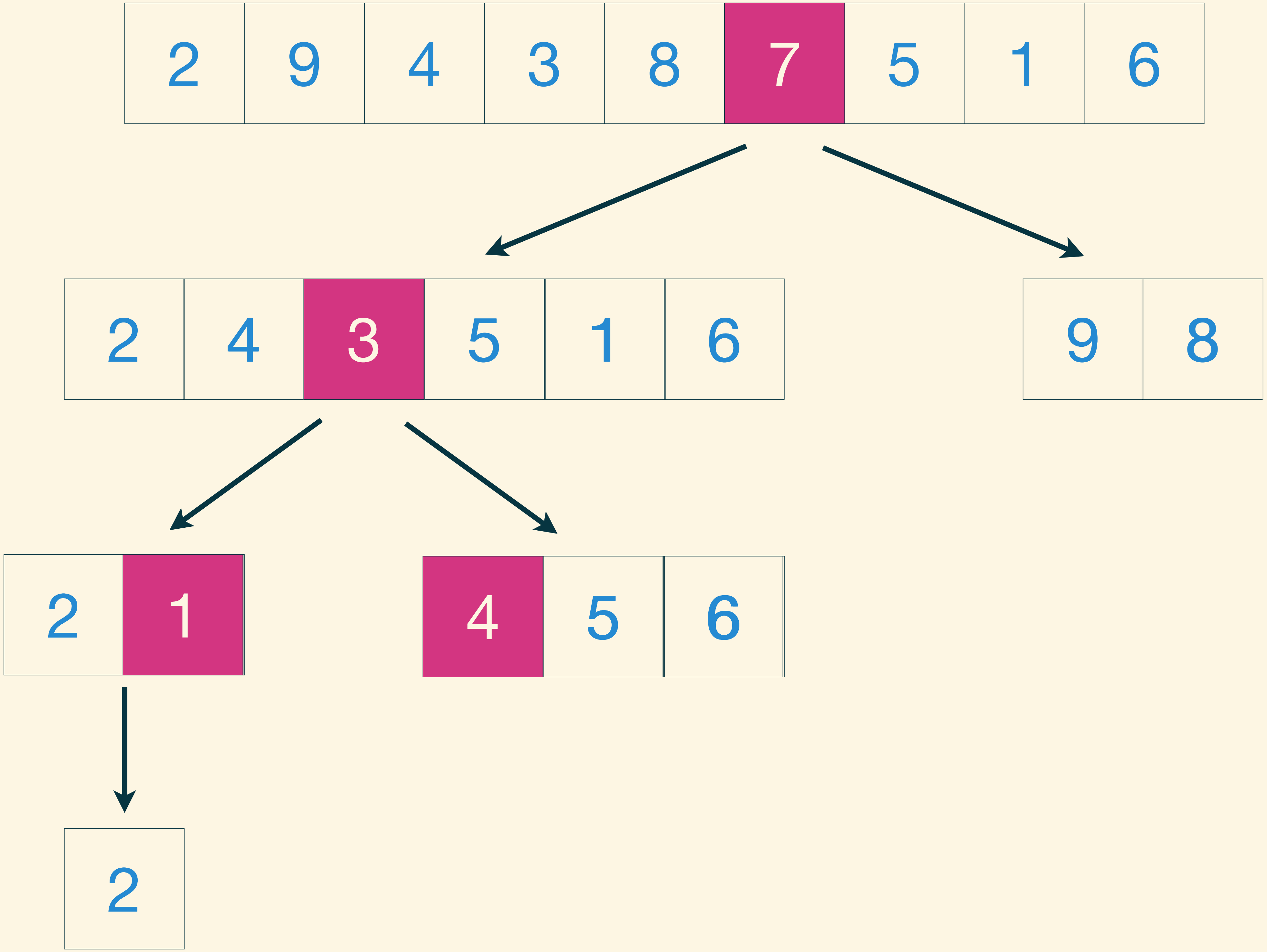
2 4 3 5 1 6

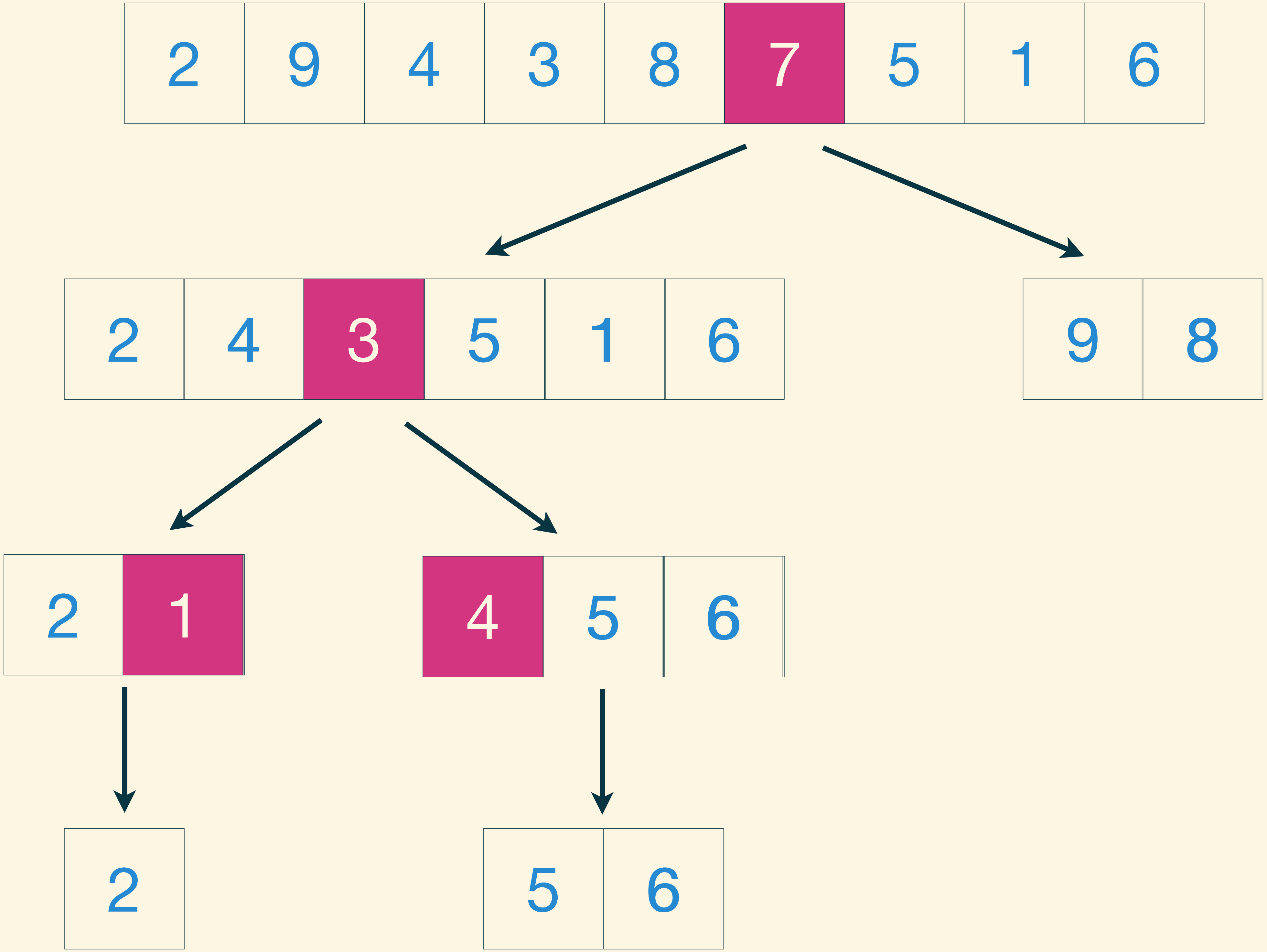
9 8

2 1

4 5 6

2


2 9 4 3 8 7 5 1 6
2 4 3 5 1 6
9 8
2 1
4 5 6
2



2 9 4 3 8 7 5 1 6

2 4 3 5 1 6

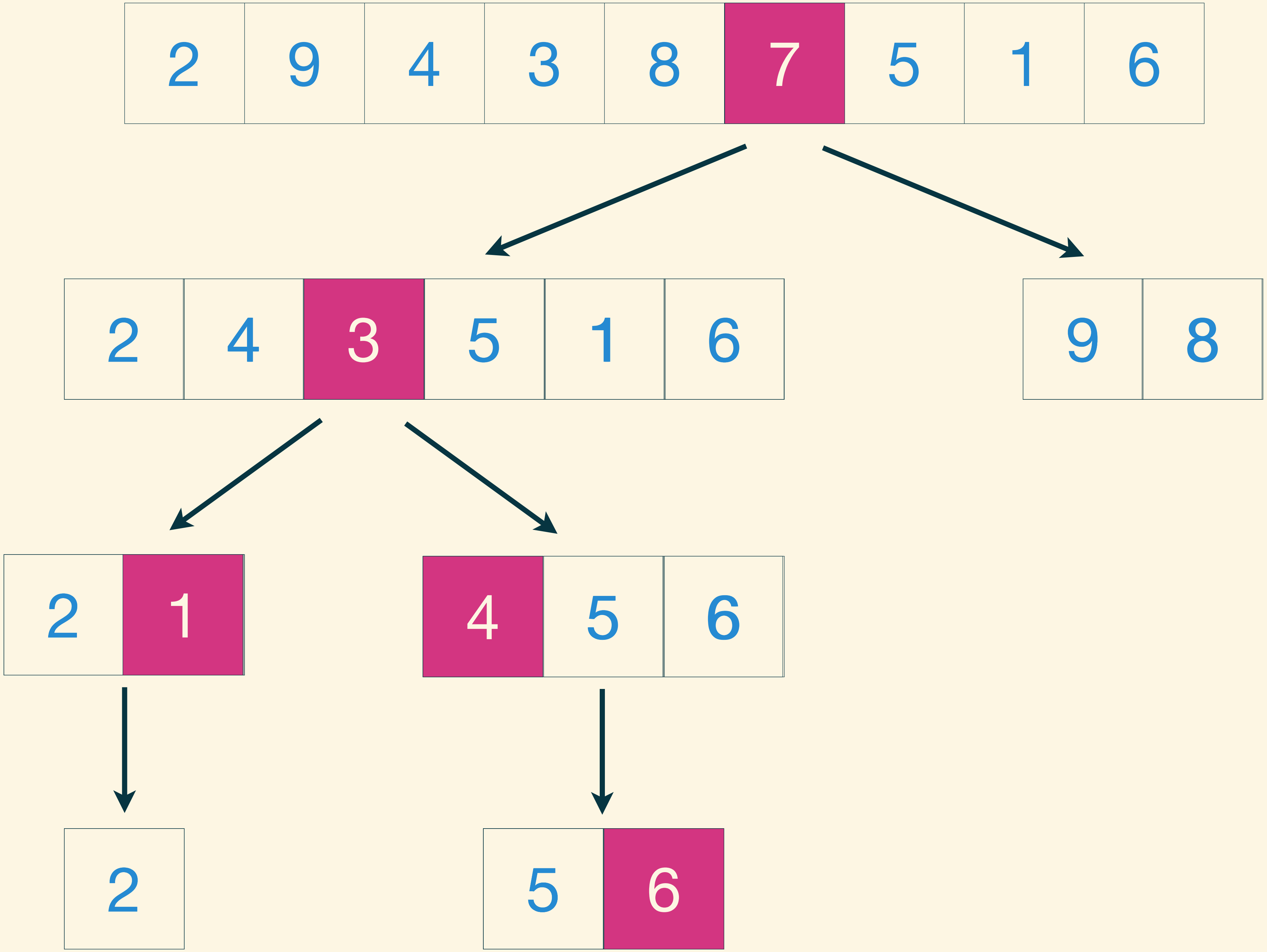
9 8

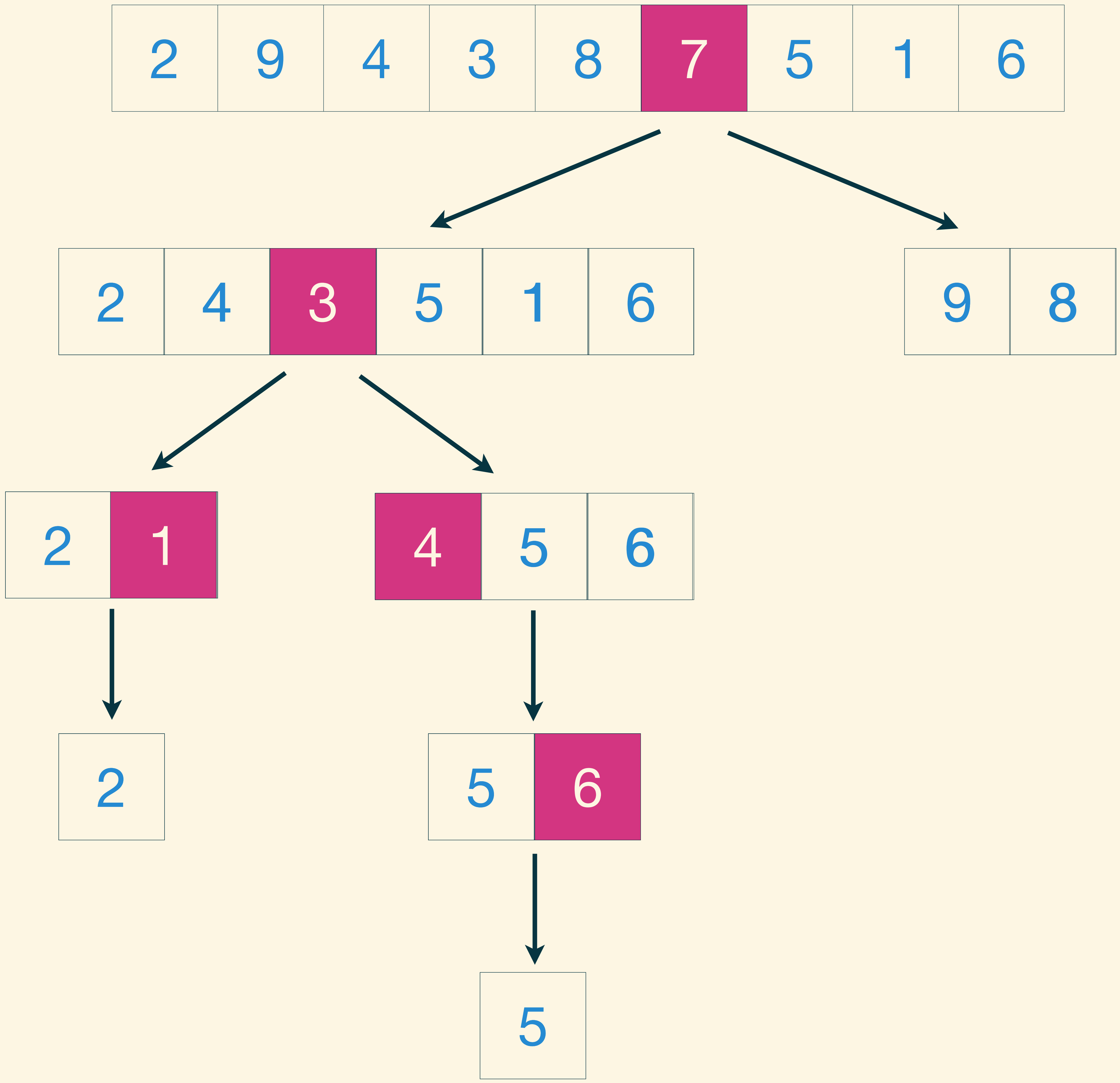
2 1

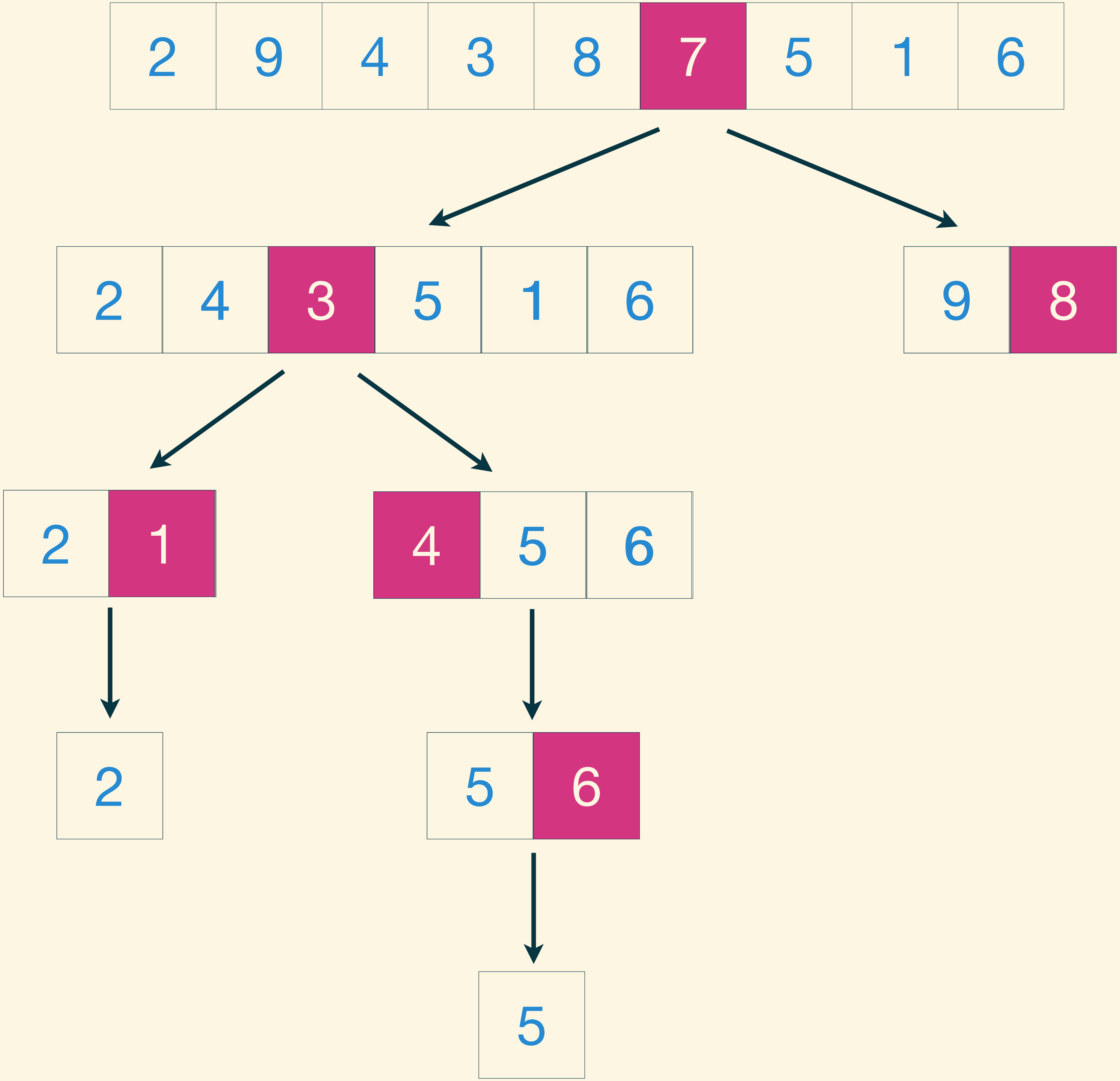
4 5 6

2

5 6


2 9 4 3 8 7 5 1 6
2 4 3 5 1 6
9 8
2 1
4 5 6
2
5 6


2 9 4 3 8 7 5 1 6
2 4 3 5 1 6
9 8
2 1
4 5 6
2
5 6
5



2 9 4 3 8 7 5 1 6

2 4 3 5 1 6

9 8

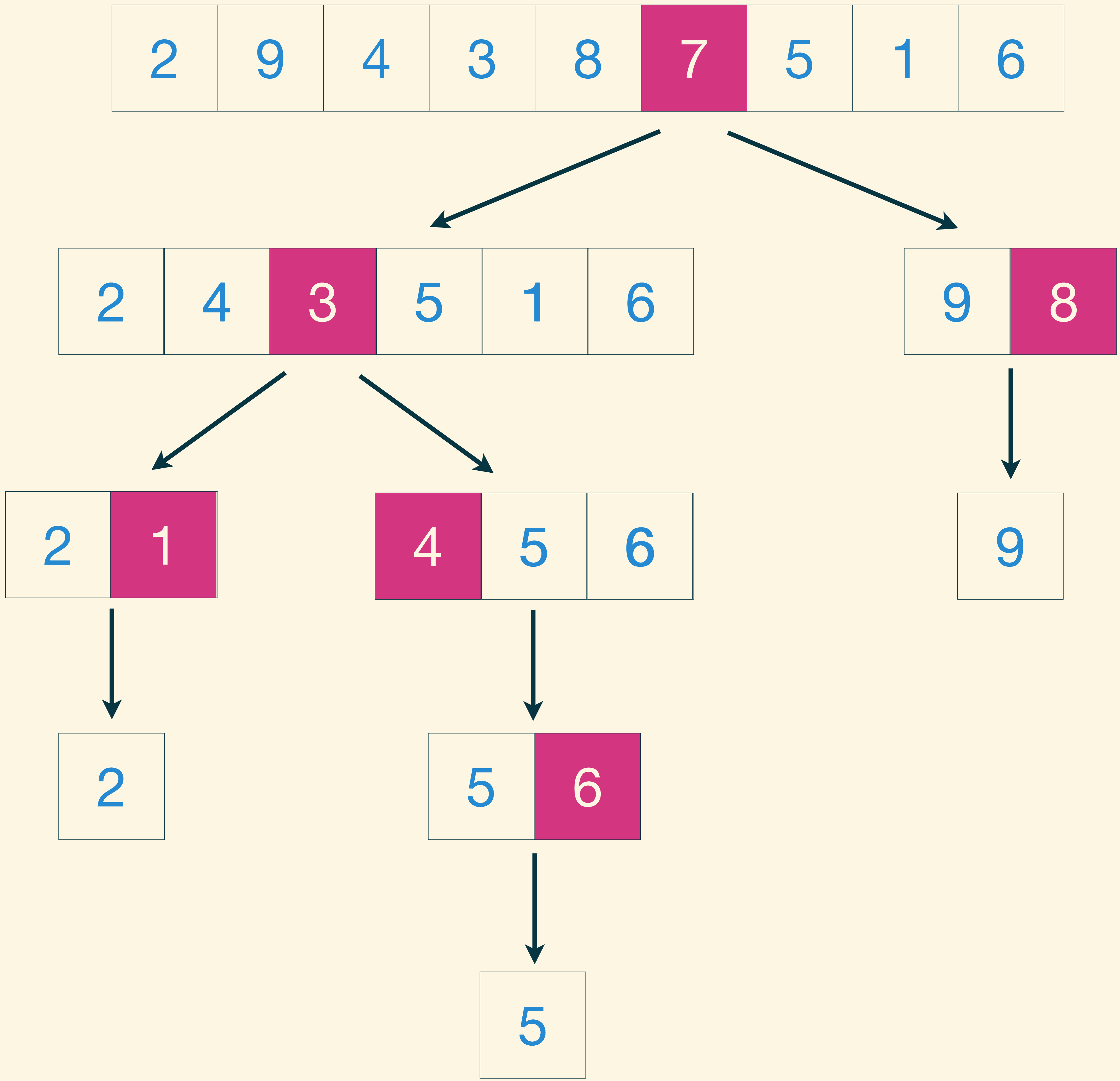
2 1

4 5 6

2

5 6

5

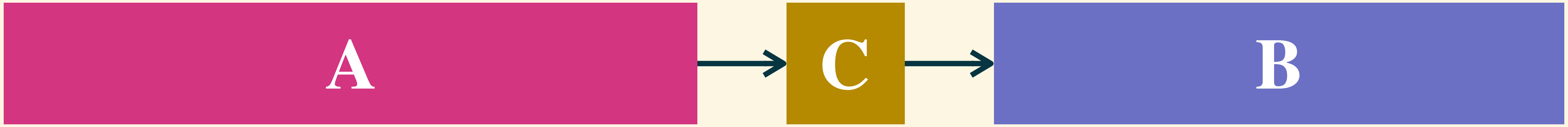

2 9 4 3 8 7 5 1 6
2 4 3 5 1 6
9 8
2 1
4 5 6
9
2
5 6
5

Conquer

由於子數列已被 pivot 分為兩段，一段小於等於 pivot 的數列 A，另一段大於等於 pivot 的數列 B。

Conquer

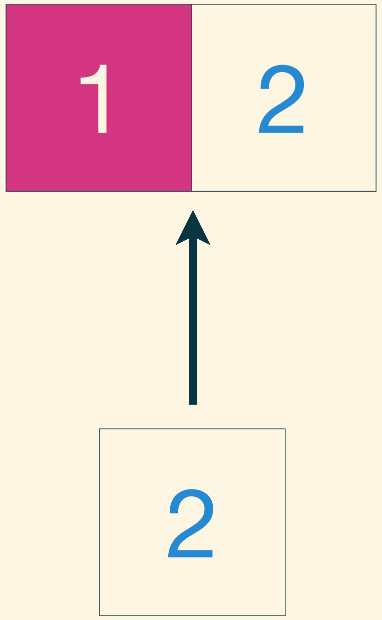
由於子數列已被 pivot 分為兩段，一段小於等於 pivot 的數列 A，另一段大於等於 pivot 的數列 B。

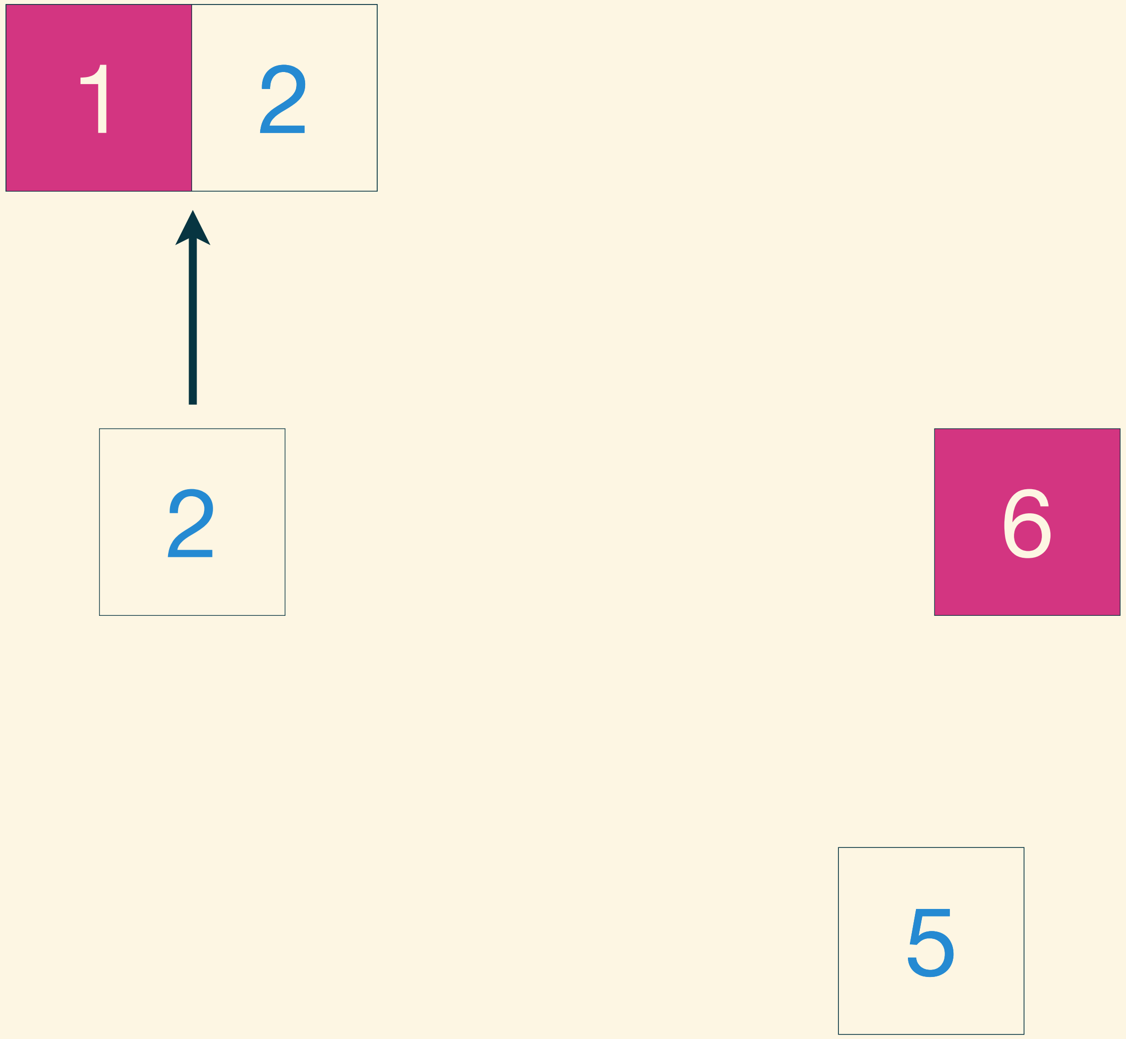


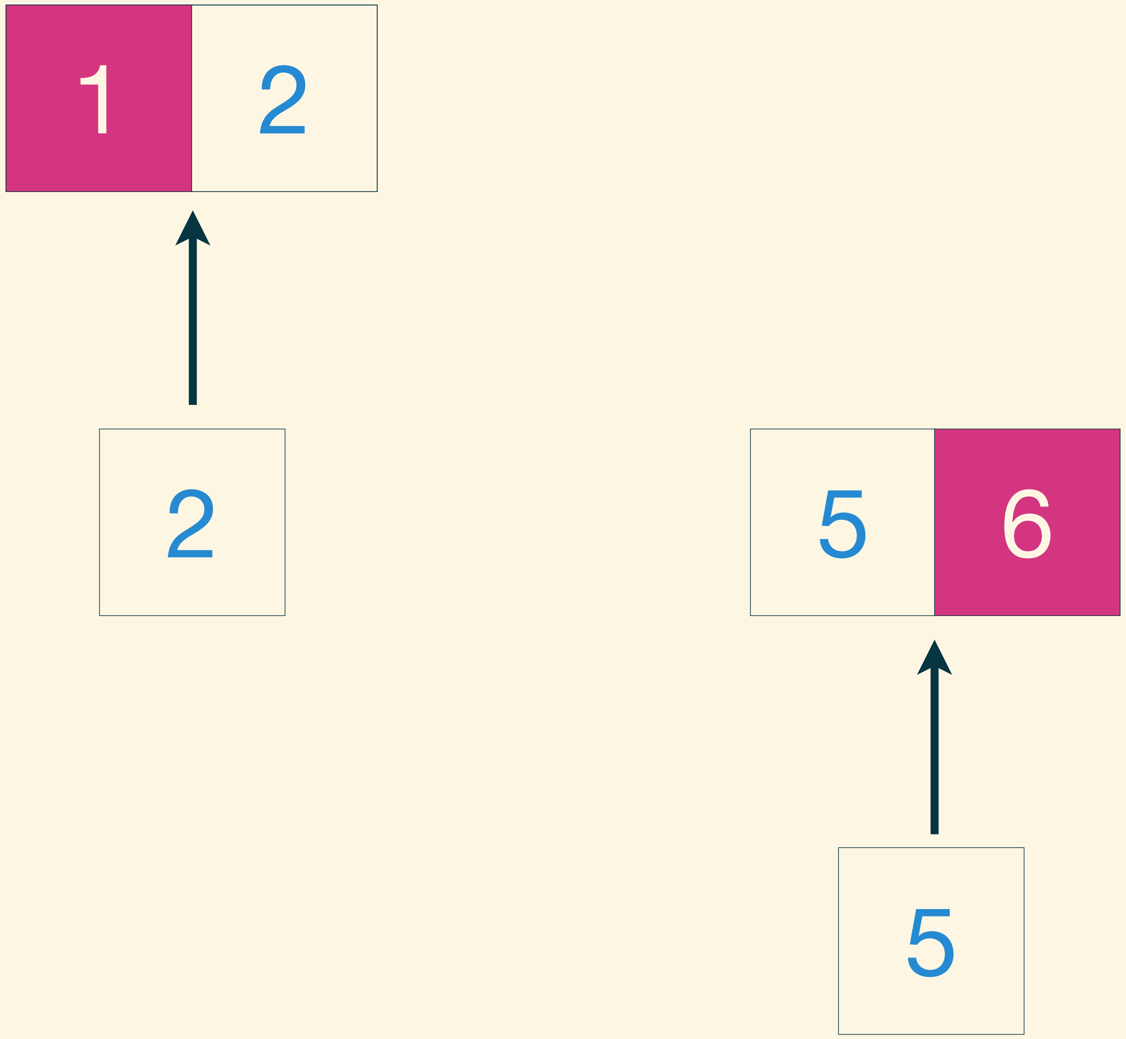
2

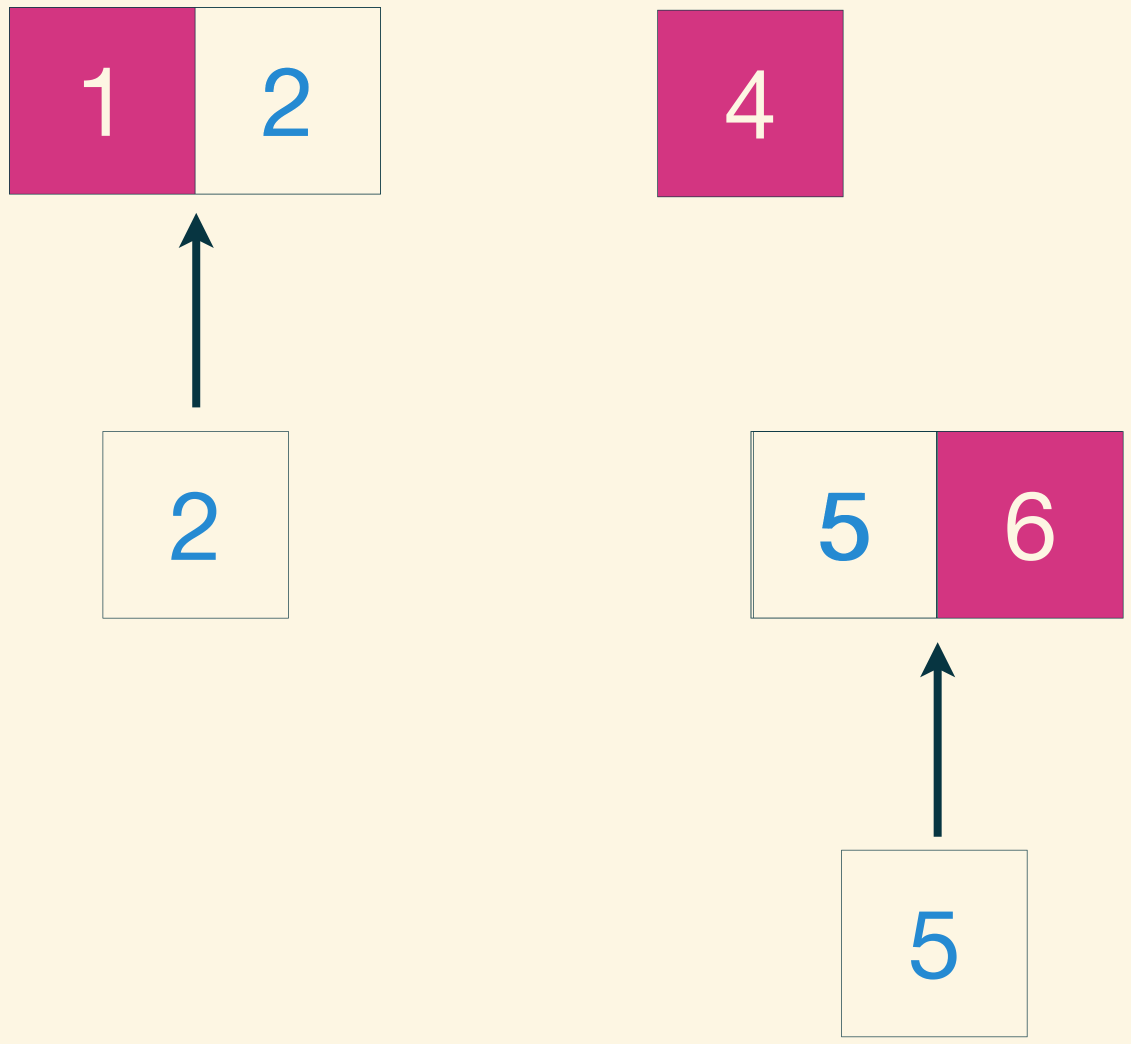
1

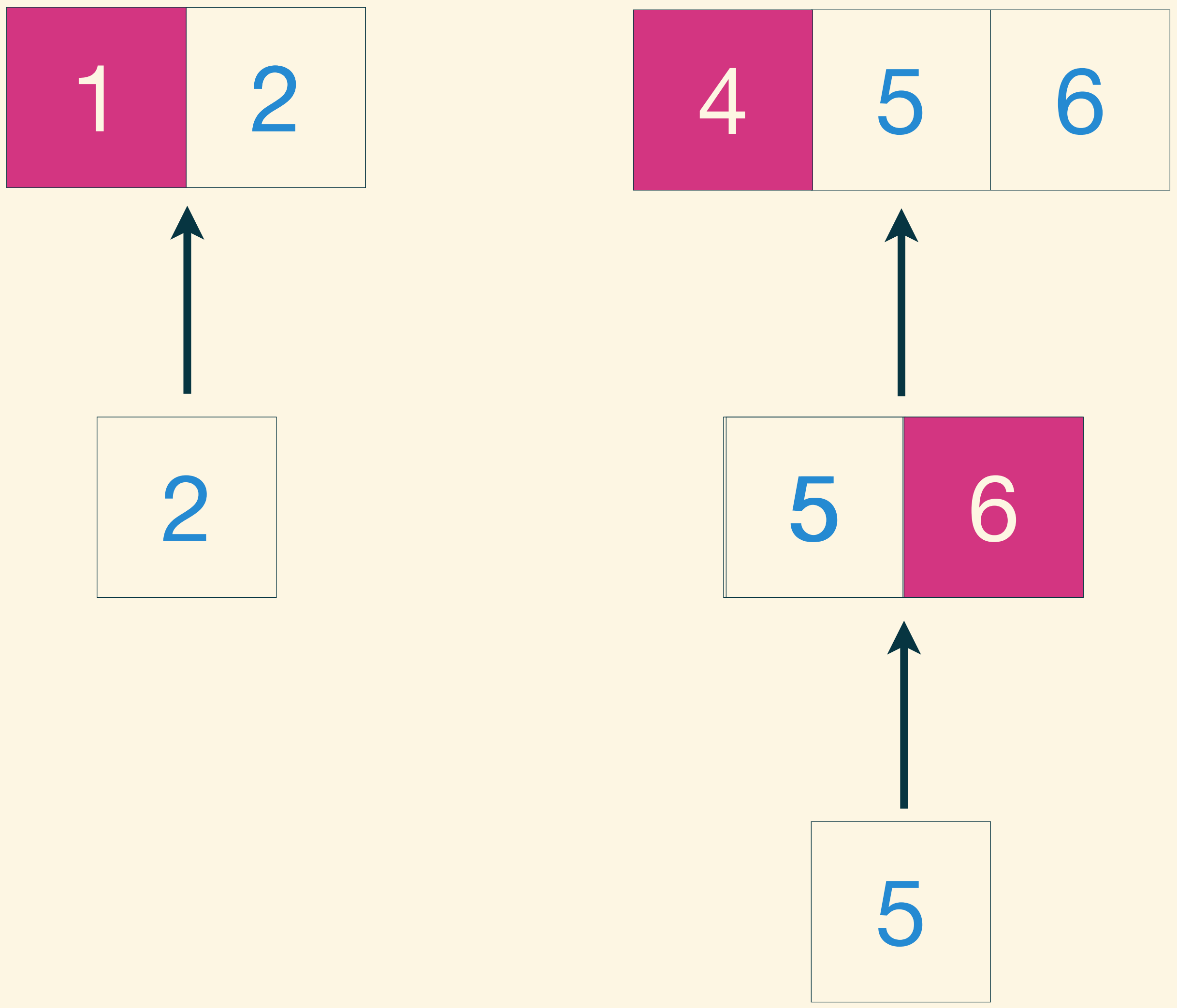
2

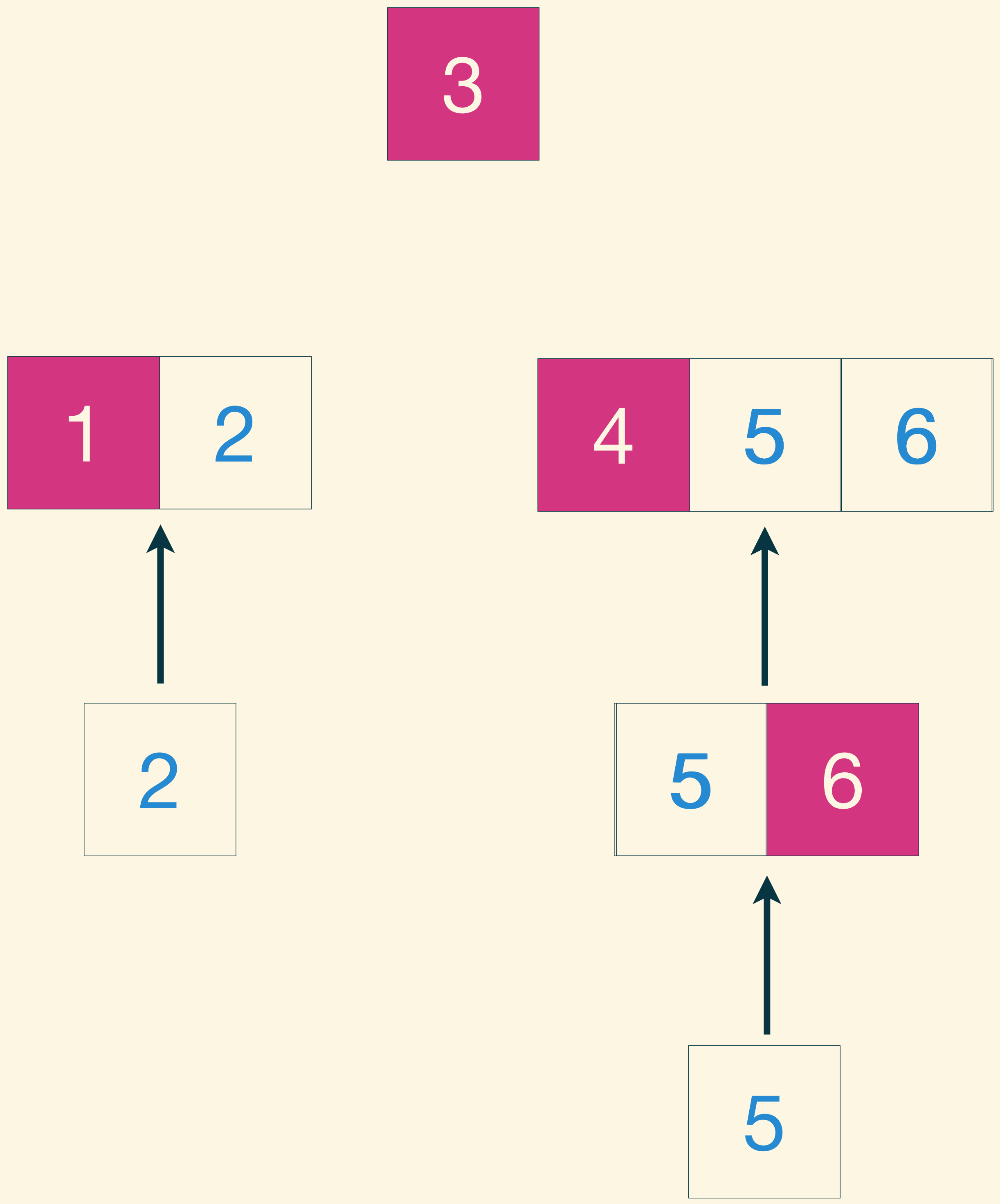

1
2
2


1
2
2
6
5

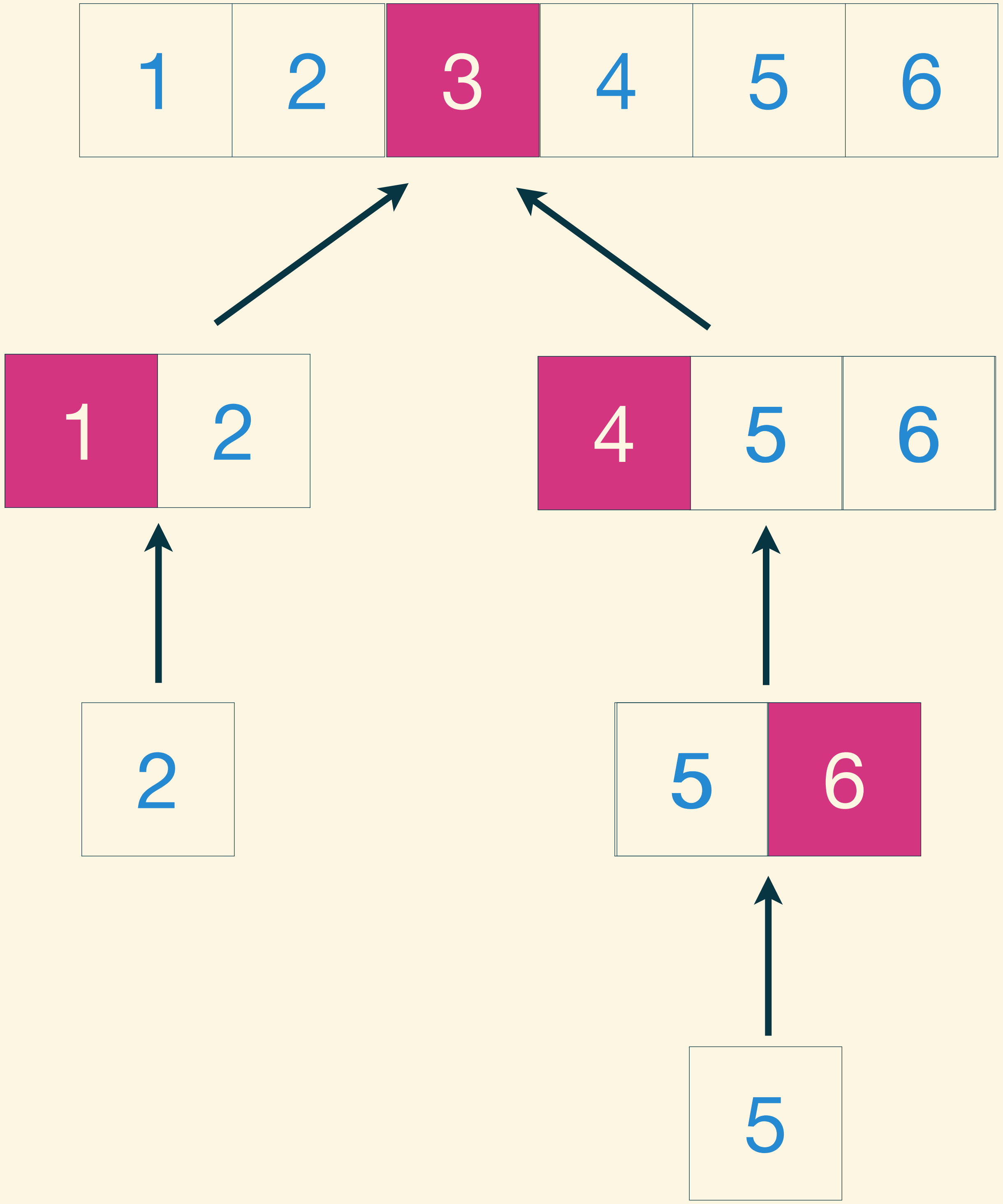

1
2
2
5
6
5


1
2
4
2
5
6
5


1
2
2
4
5
6
5
6
5



3
1
2
4
5
6
2
5
6
5



1 2 3 4 5 6

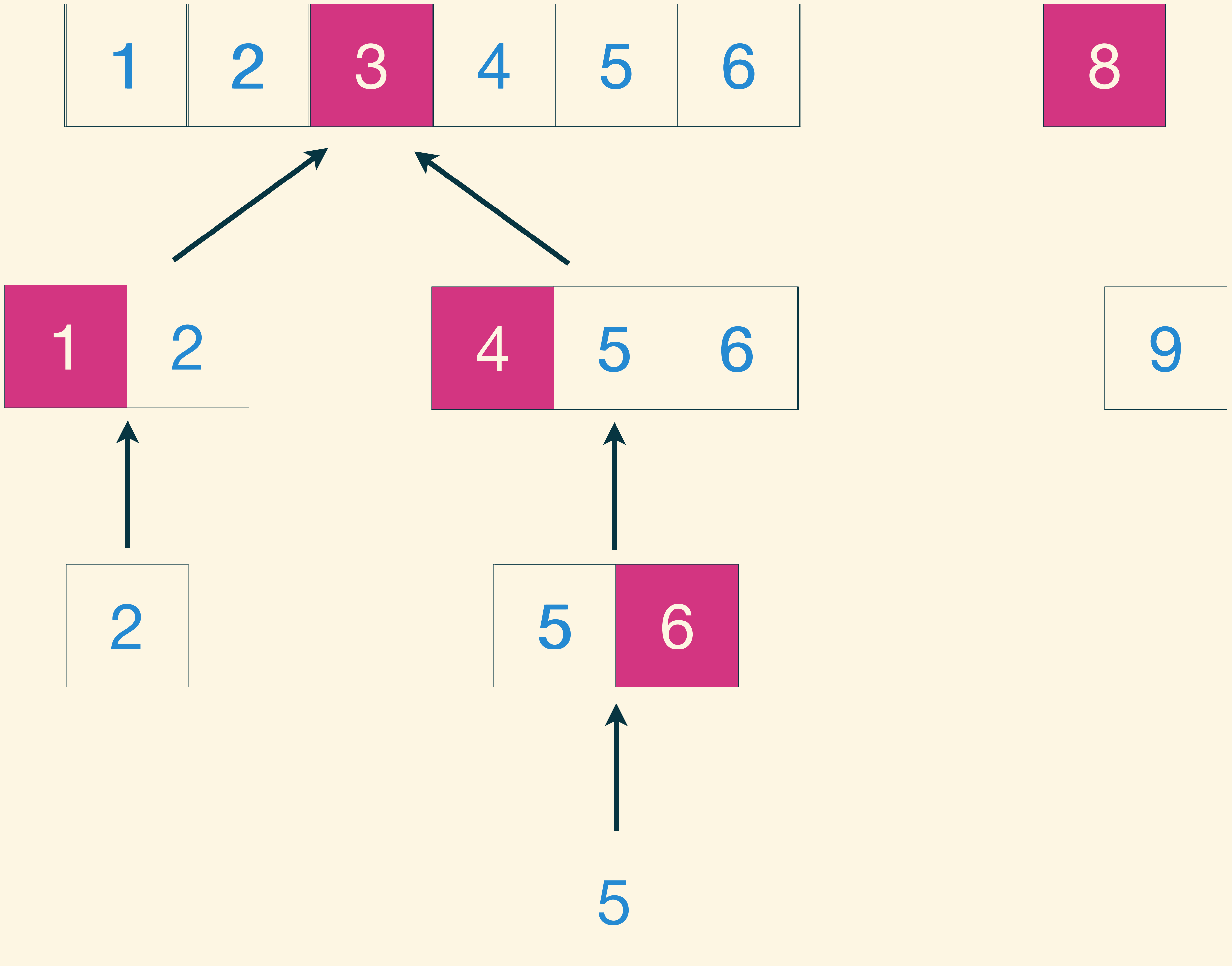
1 2

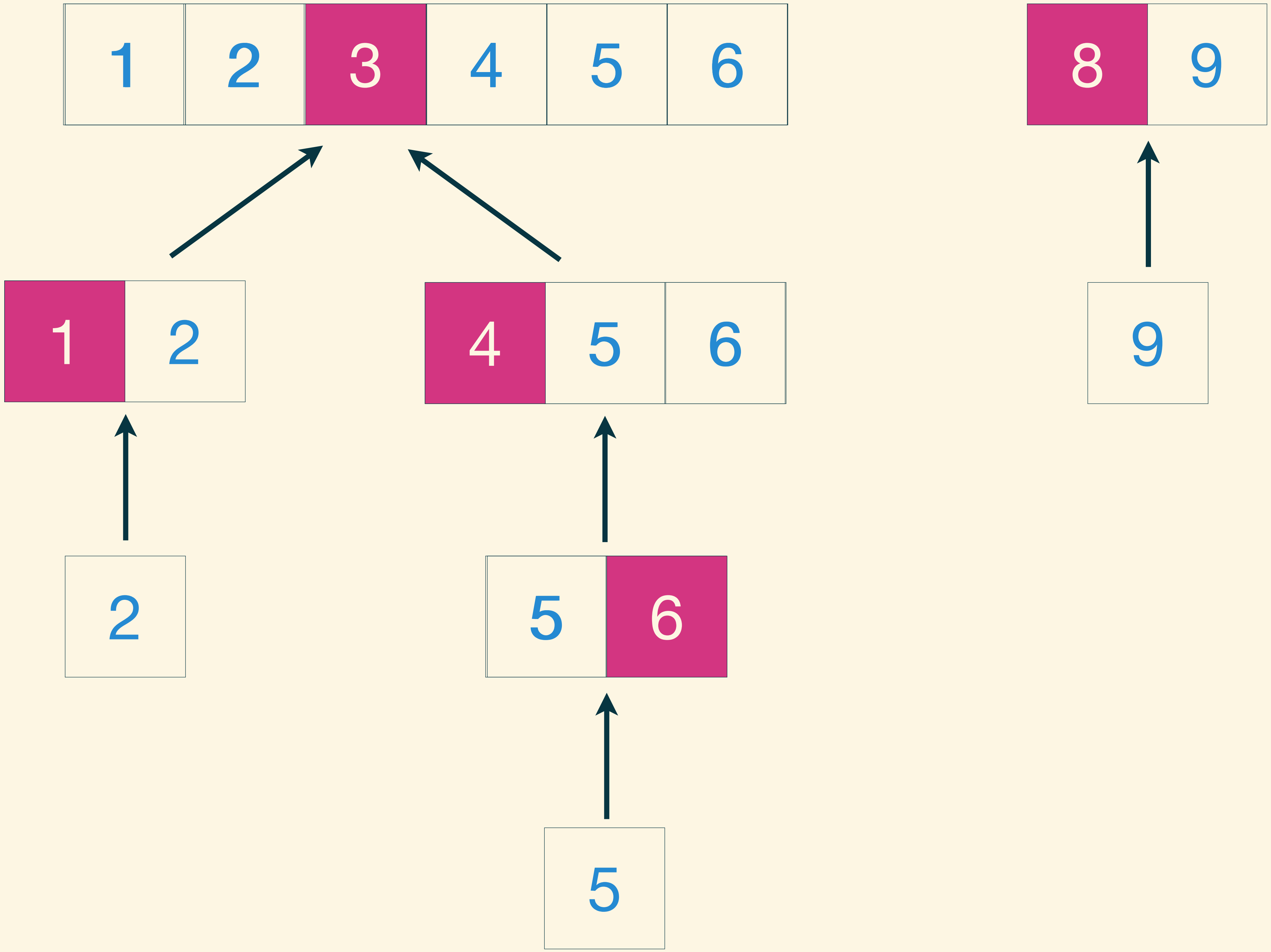
4 5 6

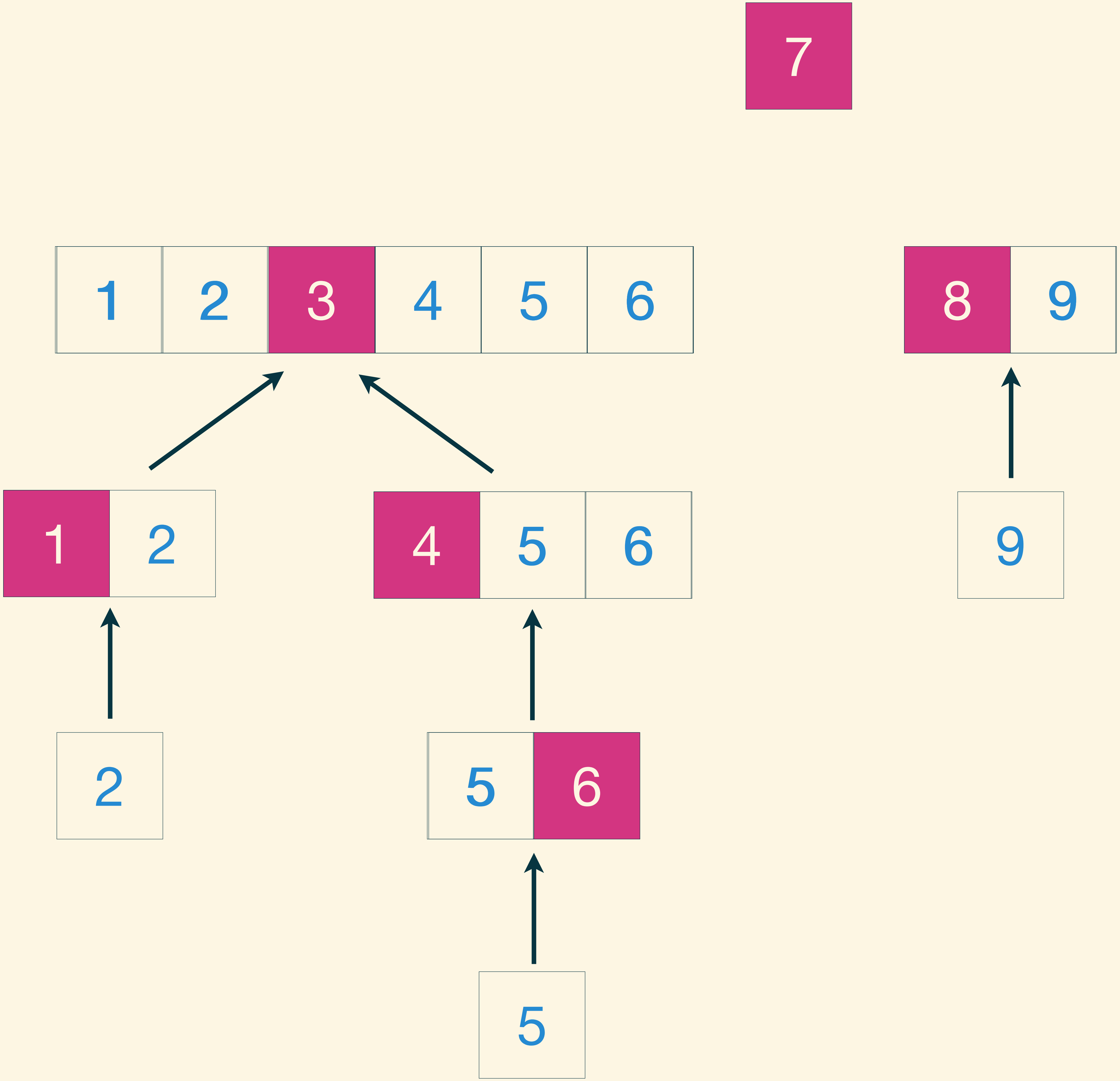
2

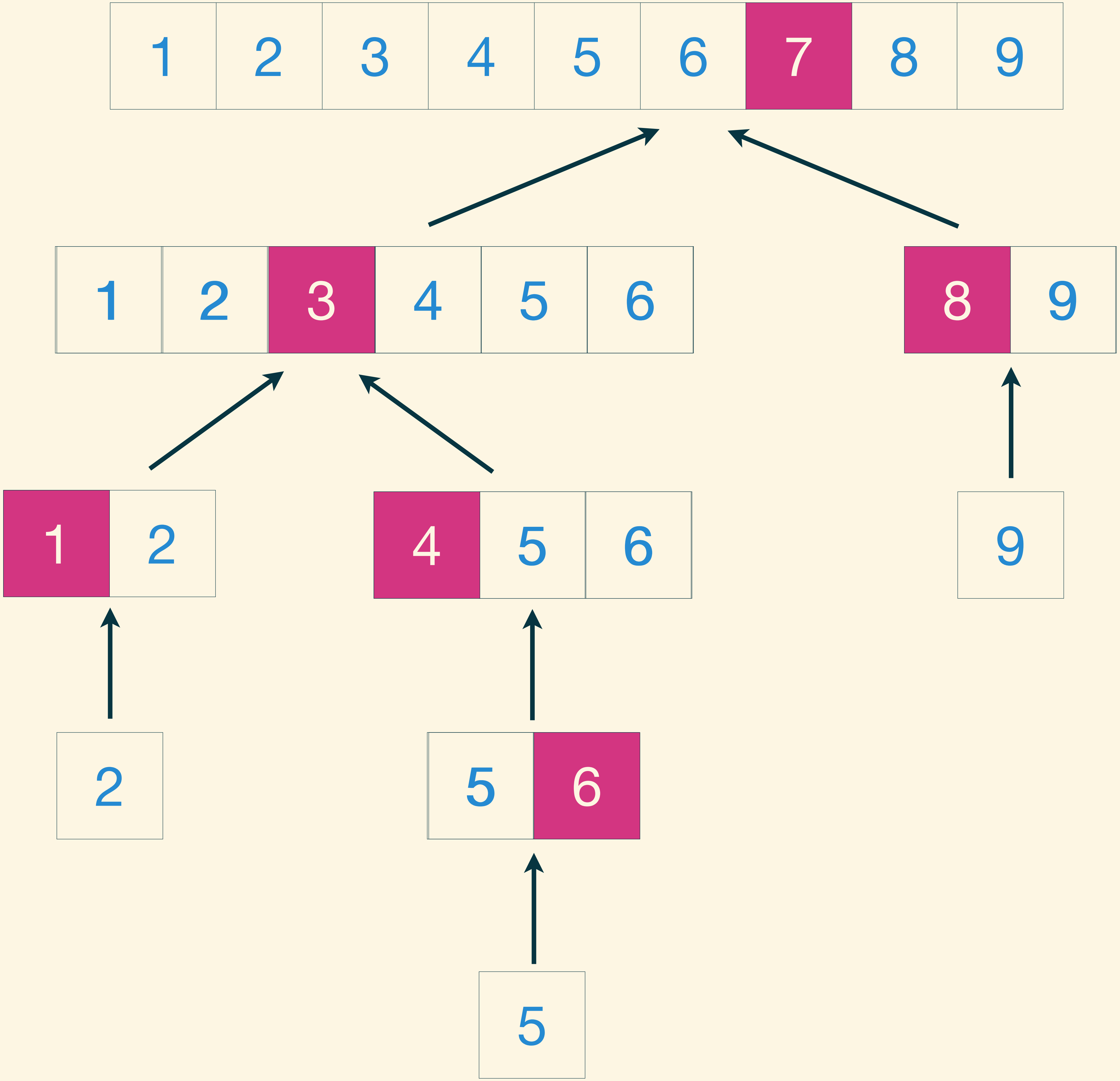
5 6

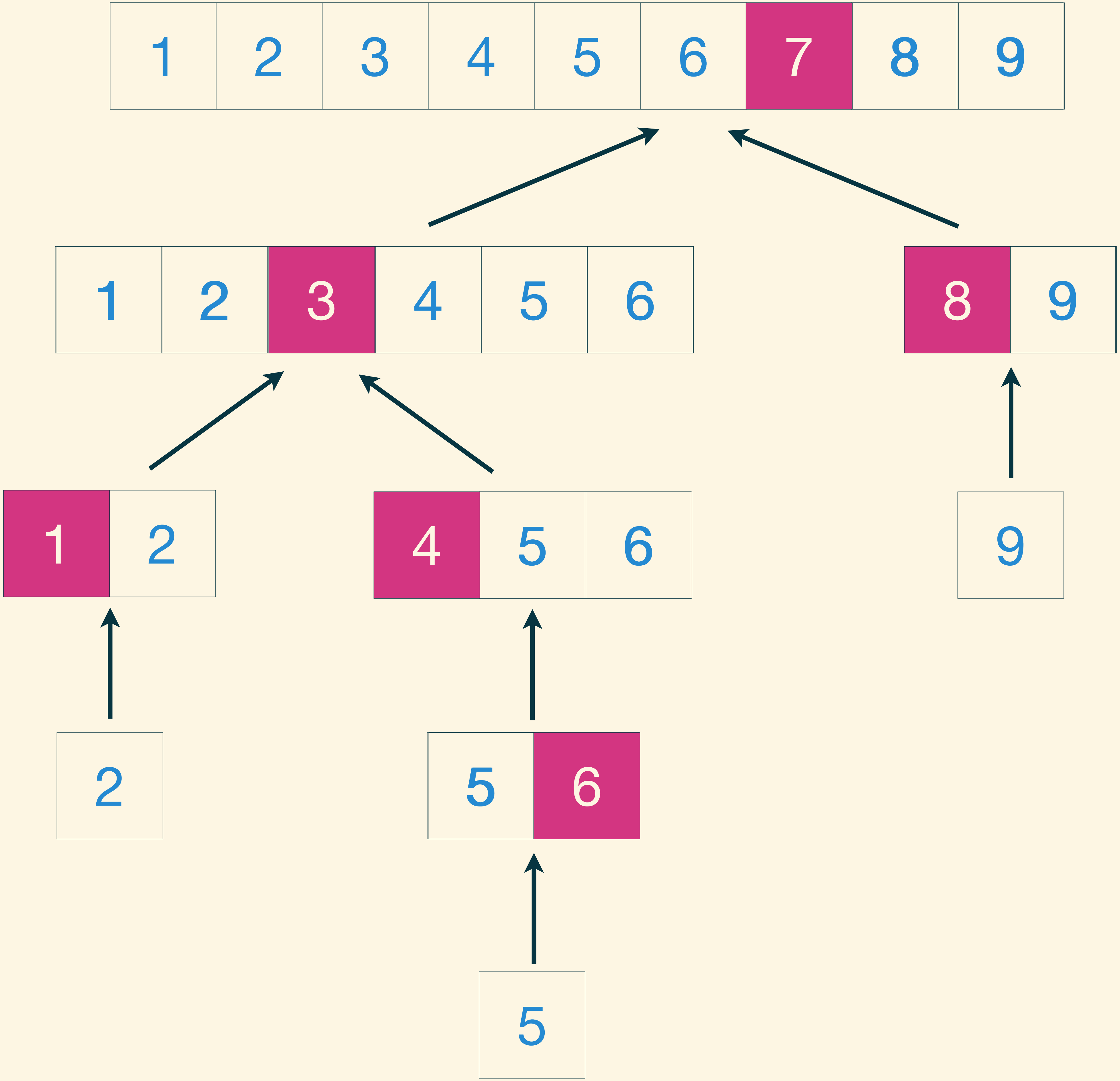
5


1 2 3 4 5 6
8
1 2
4 5 6
9
2
5 6
5


1 2 3 4 5 6
8 9
1 2
4 5 6
9
2
5 6
5


7
1 2 3 4 5 6
8 9
1 2
4 5 6
9
2
5 6
5


1 2 3 4 5 6 7 8 9
1 2 3 4 5 6
8 9
1 2
4 5 6
9
2
5 6
5


1 2 3 4 5 6 7 8 9
1 2 3 4 5 6
8 9
1 2
4 5 6
9
2
5 6
5

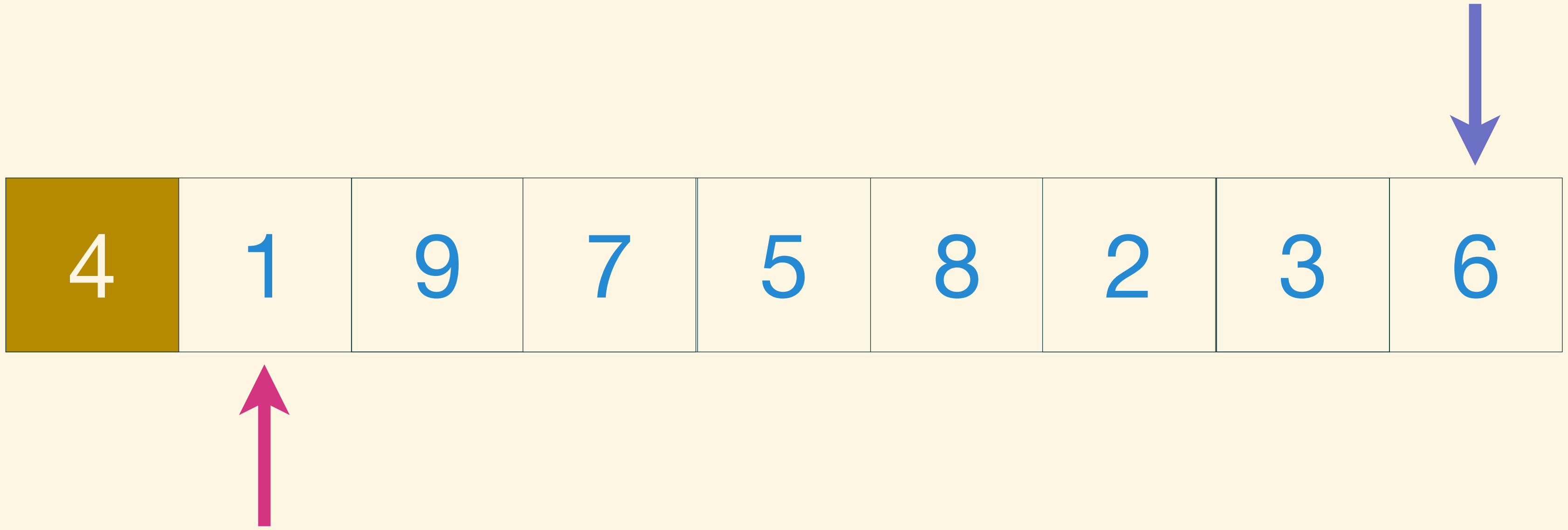
Source Code

```
void quickSort( int L, int R ) {
    if ( R - L <= 1 )
        return;
    int pivot = N[ L ], p1 = L + 1, p2 = R - 1;
    do {
        while ( N[ p1++ ] <= pivot )
            ;
        while ( N[ p2-- ] > pivot )
            ;
        if ( p1 < p2 )
            swap( N[ p1 ], N[ p2 ] );
    } while ( p1 < p2 );
    quickSort( L + 1, p1 );
    quickSort( p1, R );
    for ( int i = L + 1; i < p1; ++i )
        swap( N[ i - 1 ], N[ i ] );
}
```

How to Divide

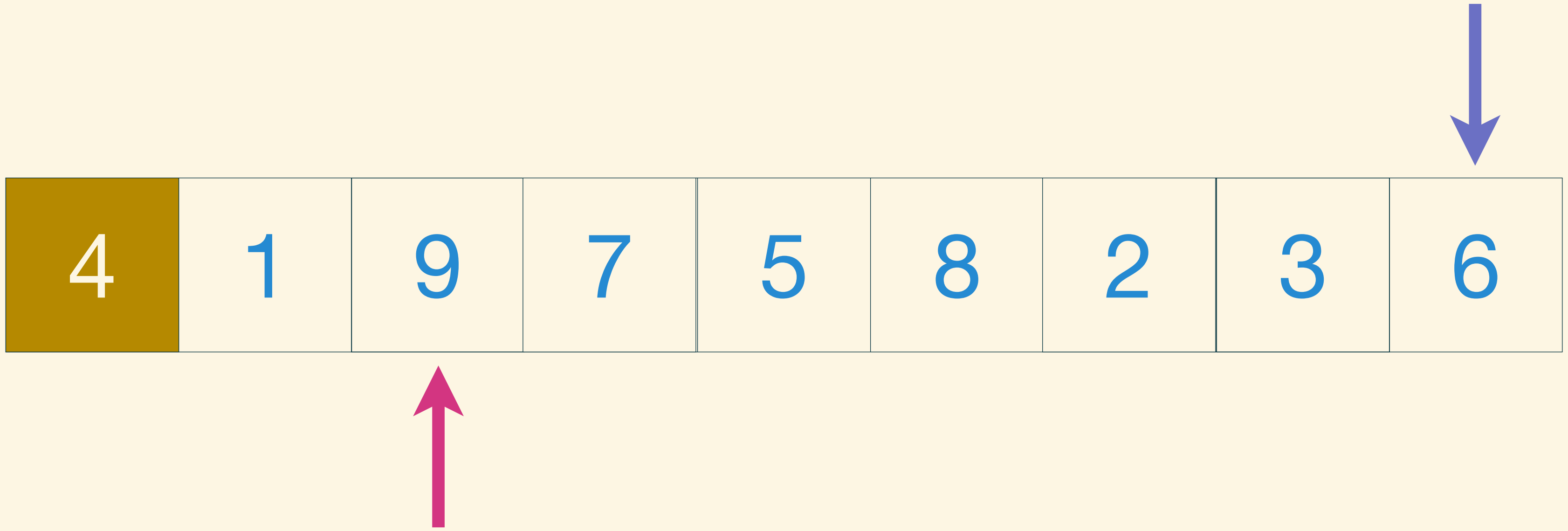
4	1	9	7	5	8	2	3	6

How to Divide



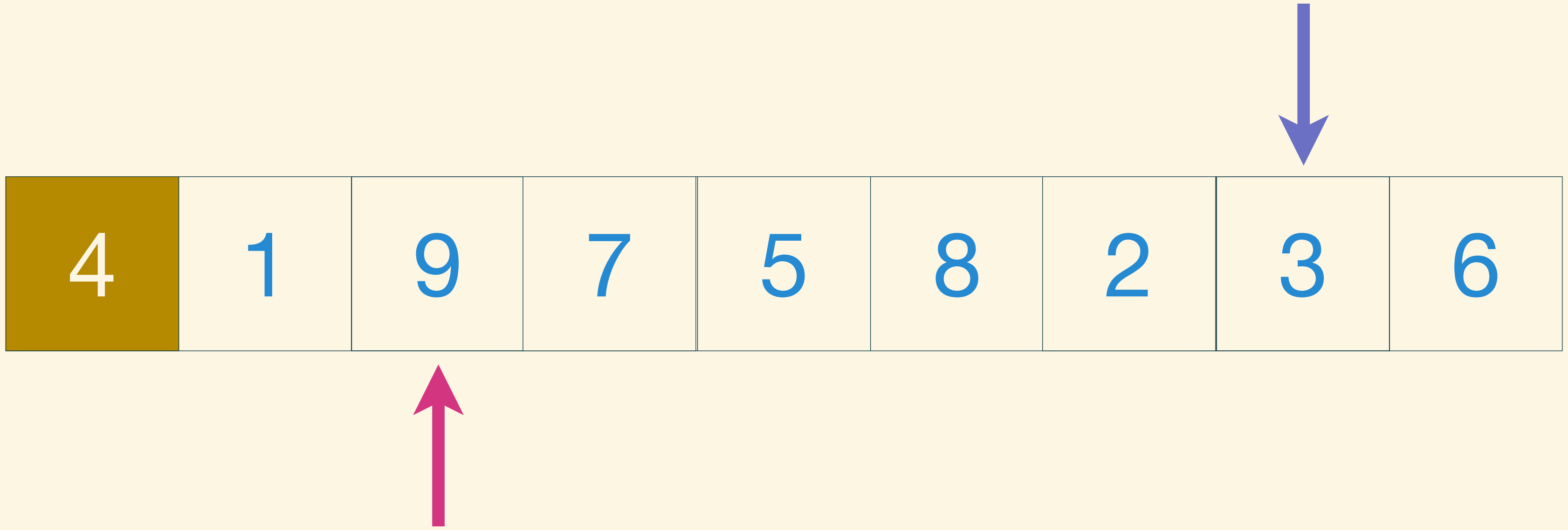
4	1	9	7	5	8	2	3	6

How to Divide



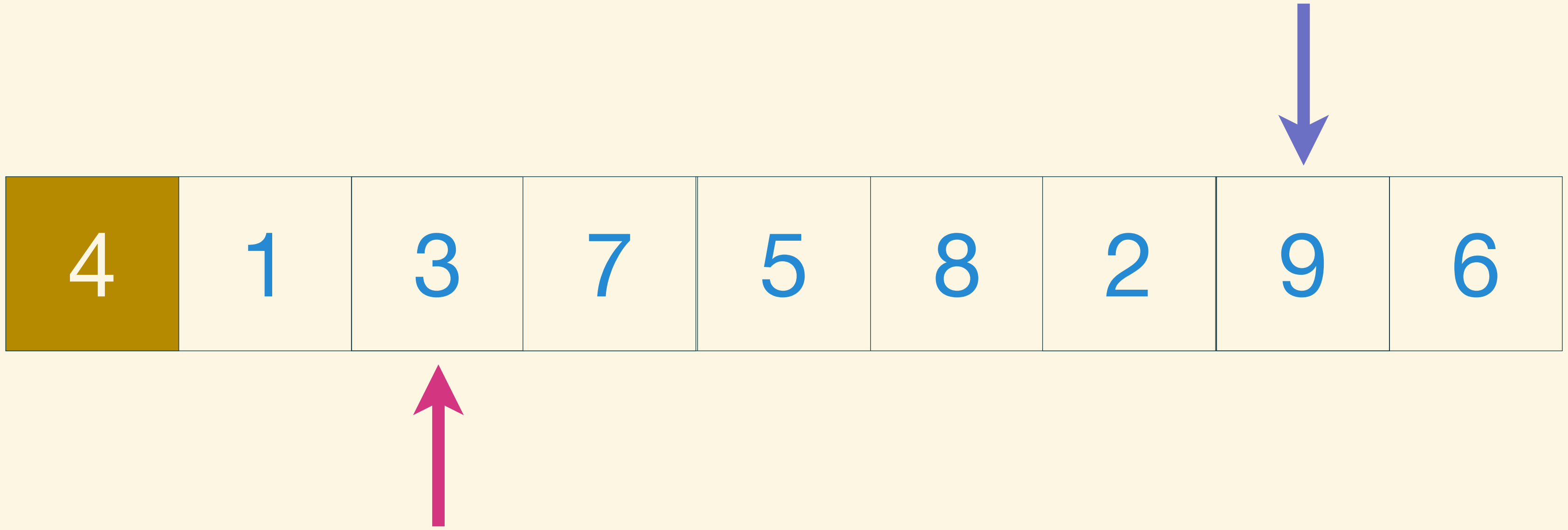
4	1	9	7	5	8	2	3	6

How to Divide



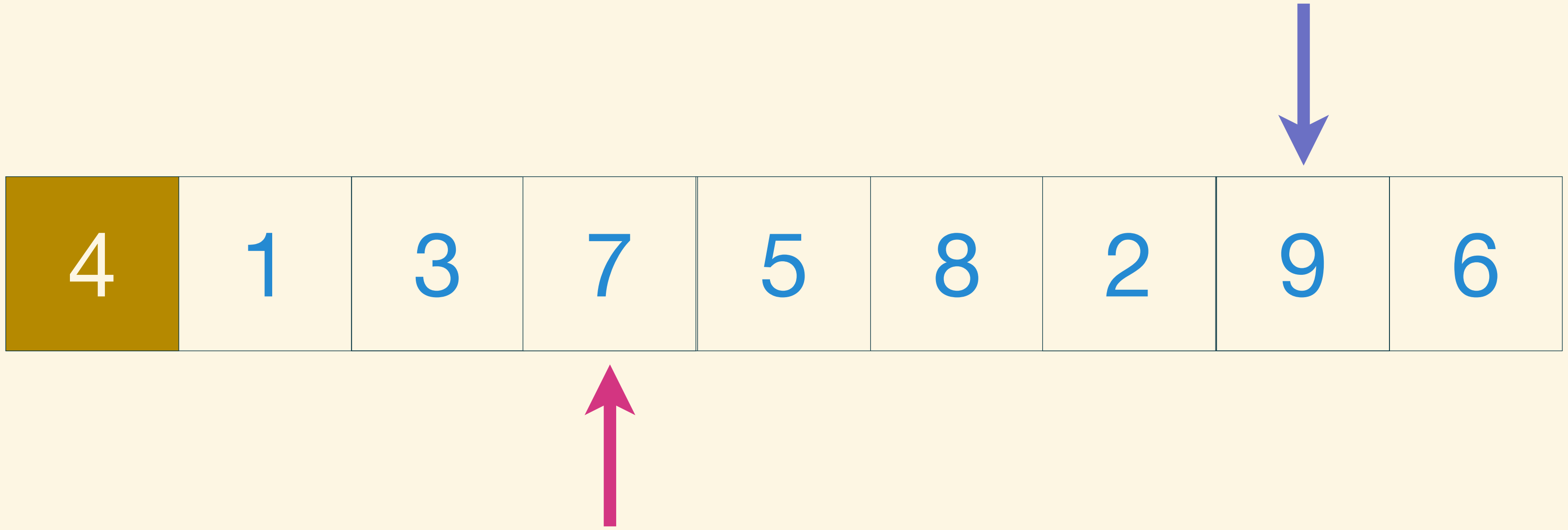
4	1	9	7	5	8	2	3	6

How to Divide



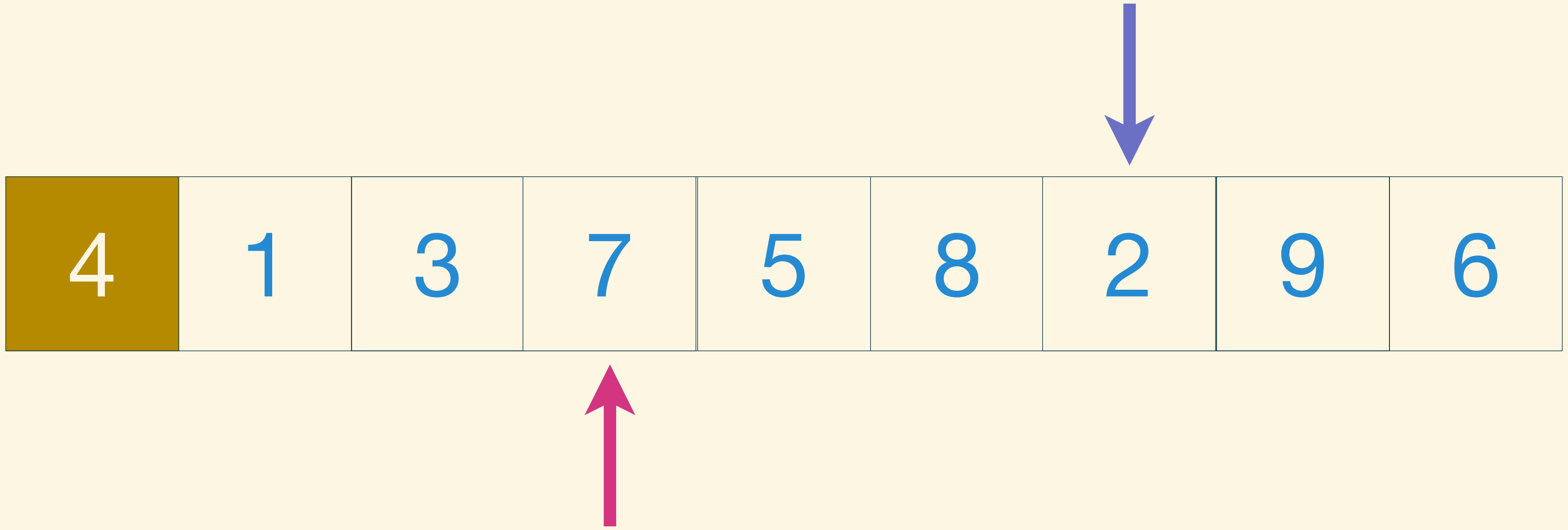
4	1	3	7	5	8	2	9	6

How to Divide



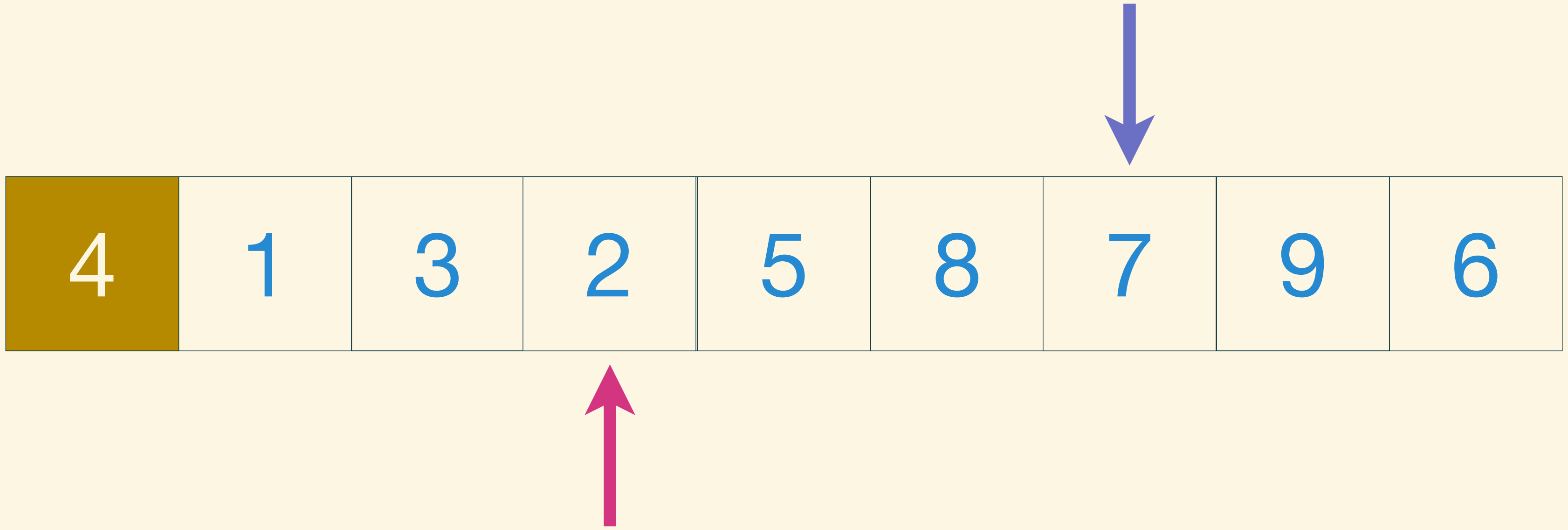
4	1	3	7	5	8	2	9	6

How to Divide



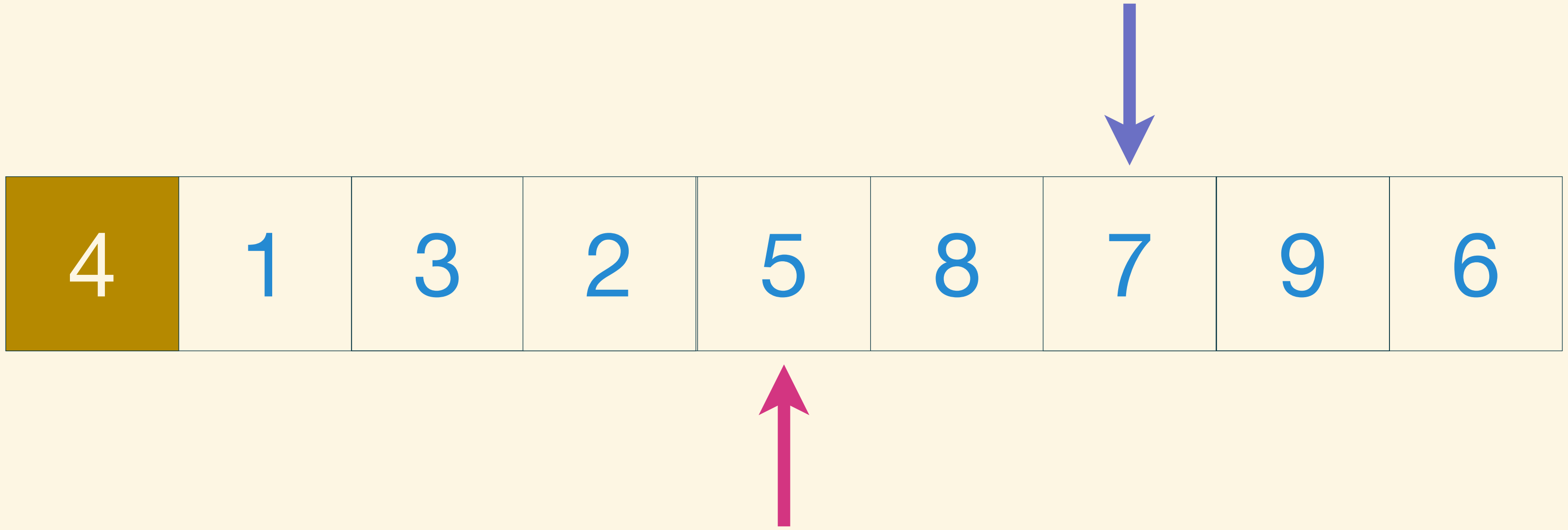
4	1	3	7	5	8	2	9	6

How to Divide



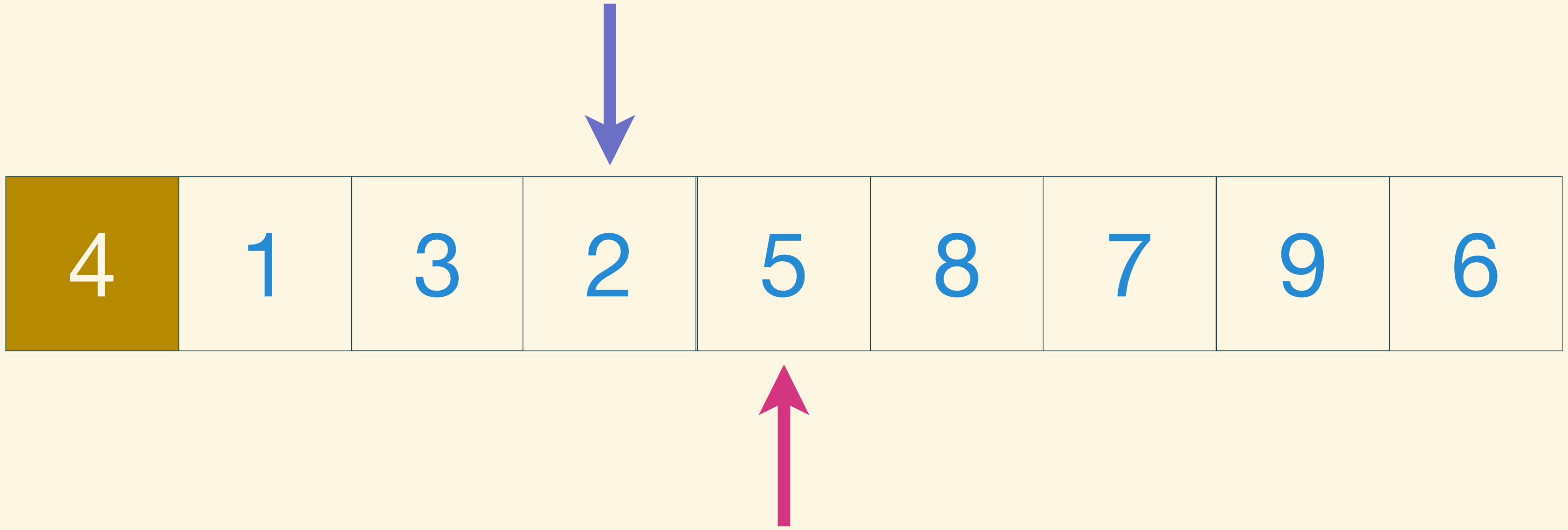
4	1	3	2	5	8	7	9	6

How to Divide



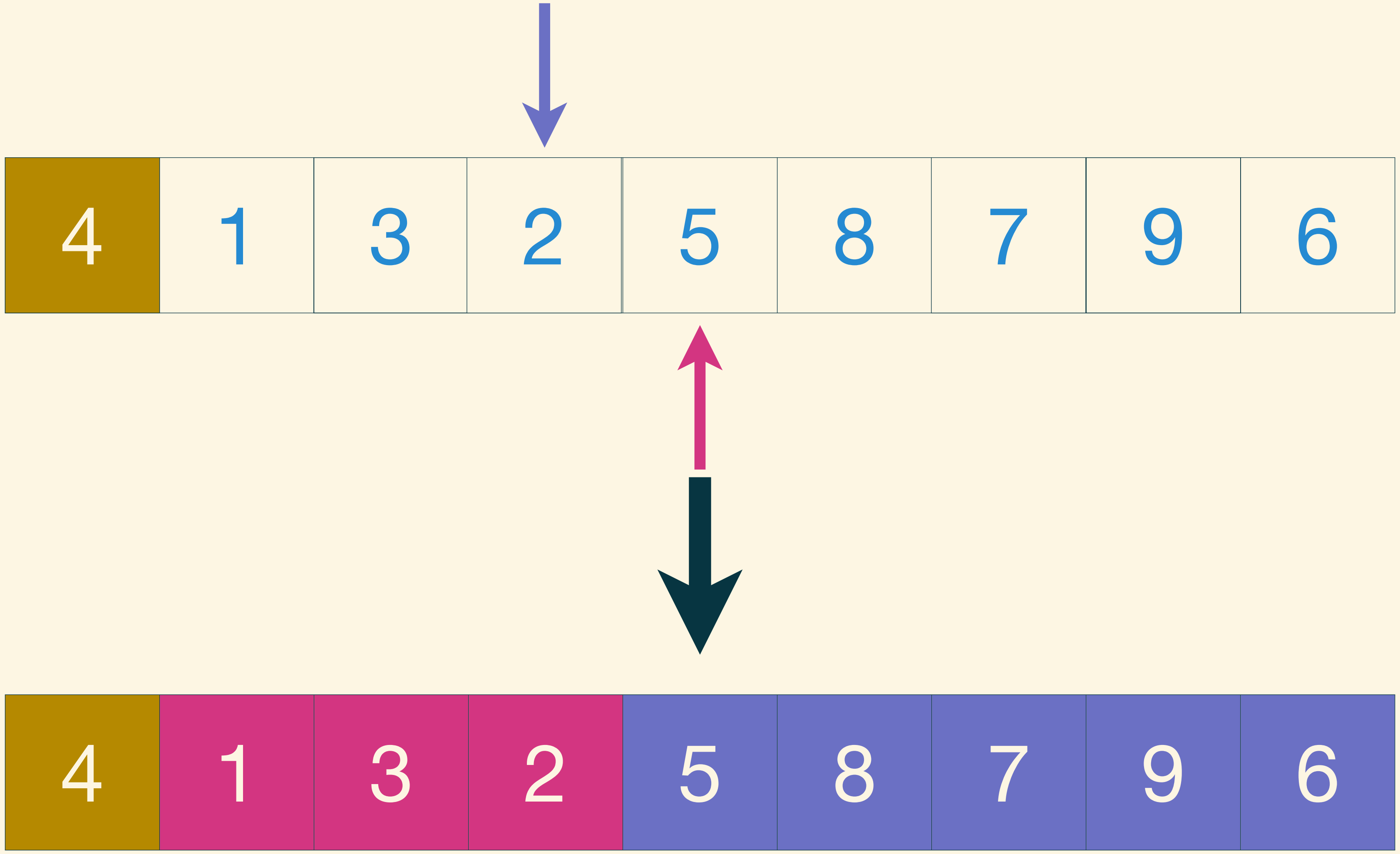
4	1	3	2	5	8	7	9	6

How to Divide



4	1	3	2	5	8	7	9	6

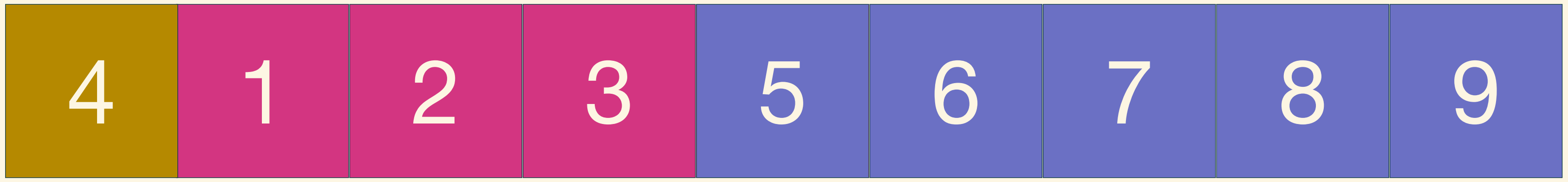
How to Divide



4	1	3	2	5	8	7	9	6

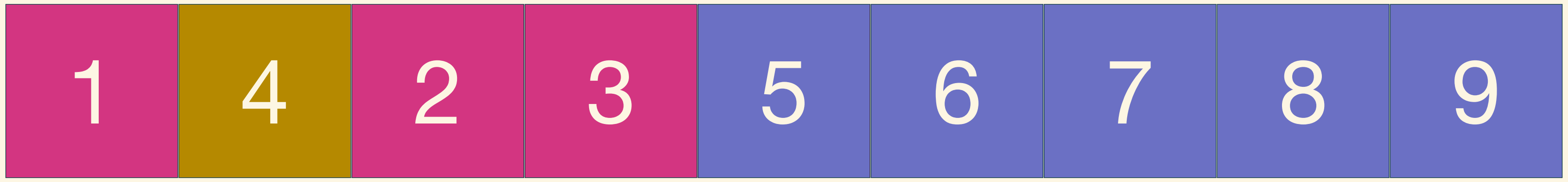
4	1	3	2	5	8	7	9	6

How to Conquer



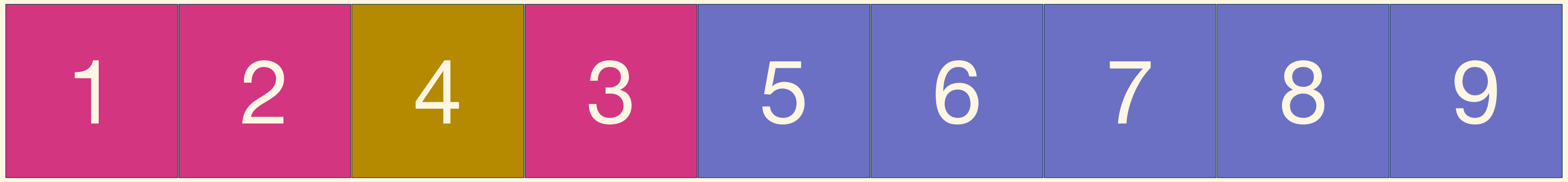
4	1	2	3	5	6	7	8	9

How to Conquer



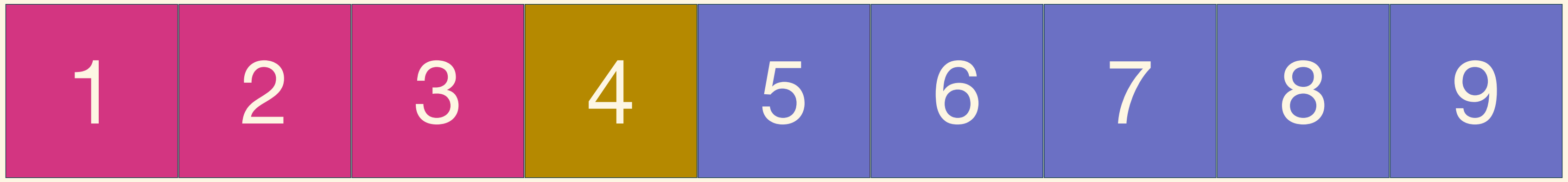
1 4 2 3 5 6 7 8 9

How to Conquer



1	2	4	3	5	6	7	8	9

How to Conquer



1	2	3	4	5	6	7	8	9

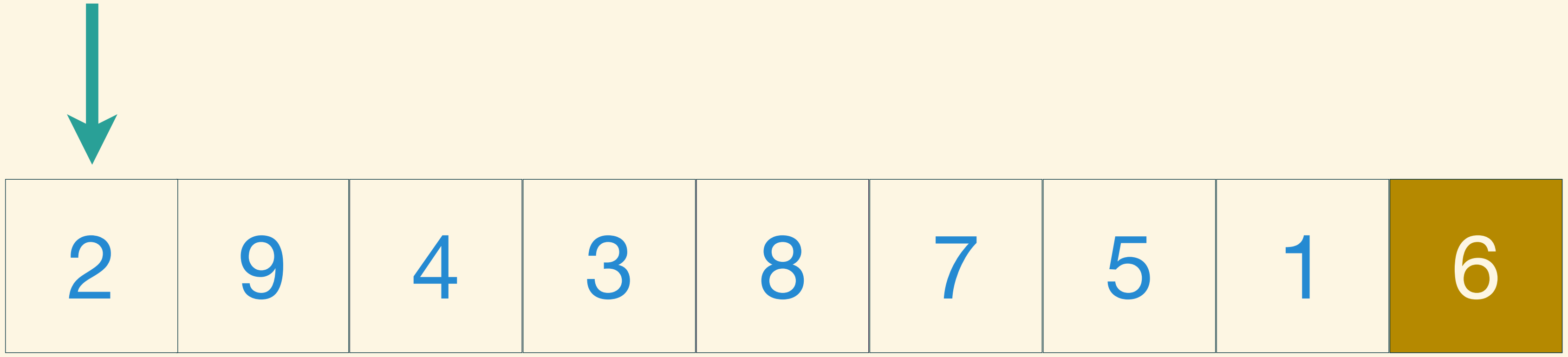
In-place Version

```
void quickSort( int L, int R ) {
   if ( R - L <= 1 )
      return;
   int pivot = N[ R - 1 ], p = L;
   for ( int i = L; i < R - 1; ++i ) {
      if ( N[ i ] <= pivot ) {
         swap( N[ i ], N[ p ] );
         ++p;
      }
   }
   swap( N[ R - 1 ], N[ p ] );
   quickSort( L, p );
   quickSort( p + 1, R );
}
```

How to Divide

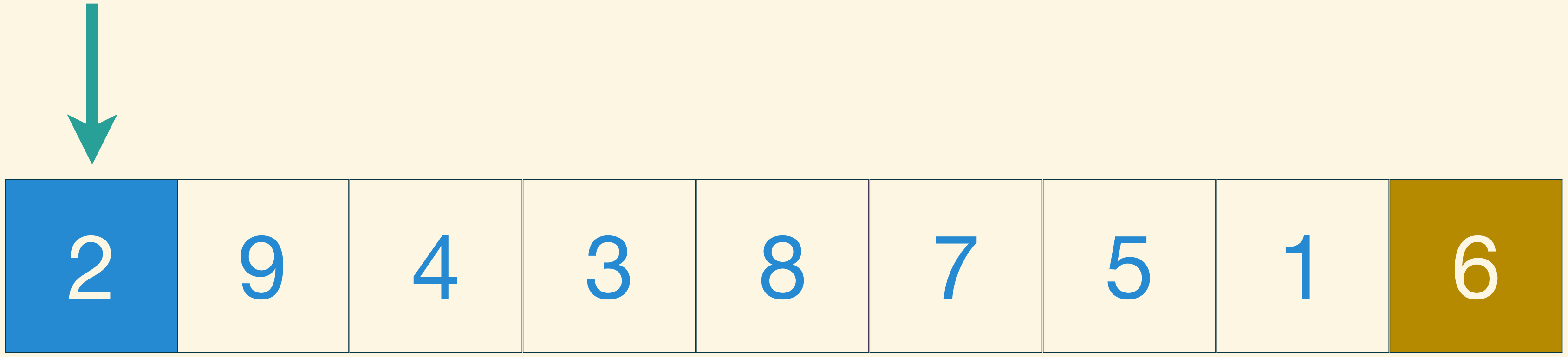
2	9	4	3	8	7	5	1	6

How to Divide



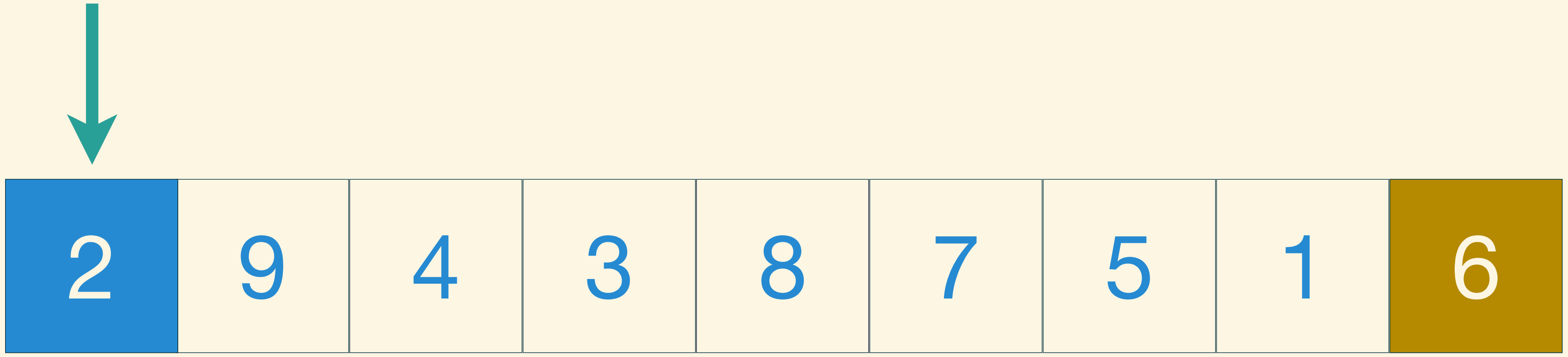
2	9	4	3	8	7	5	1	6

How to Divide



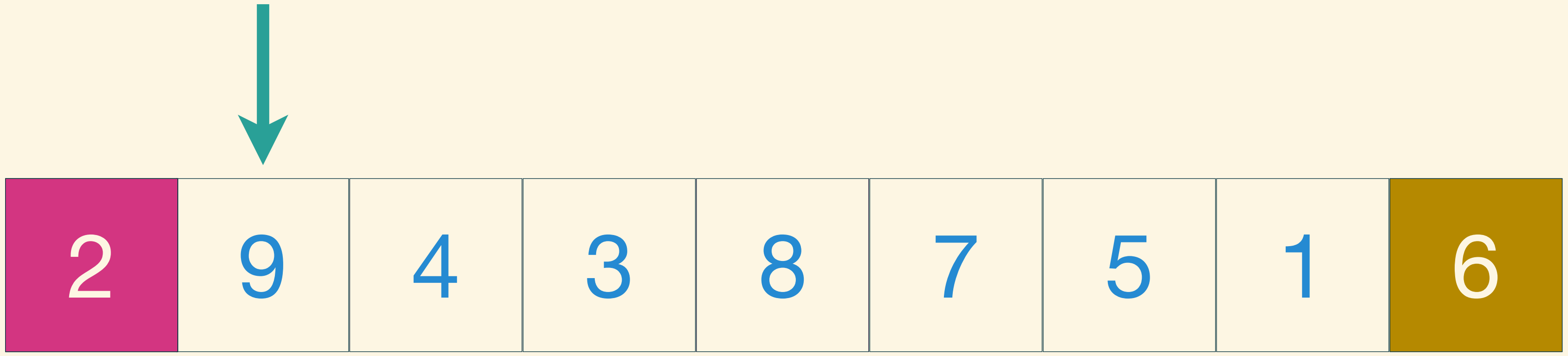
2	9	4	3	8	7	5	1	6

How to Divide



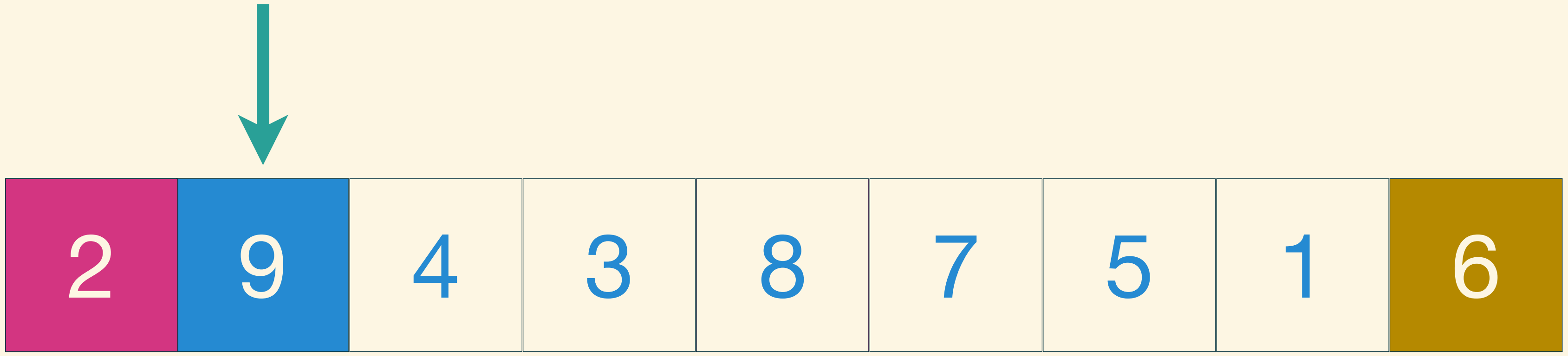
2	9	4	3	8	7	5	1	6

How to Divide



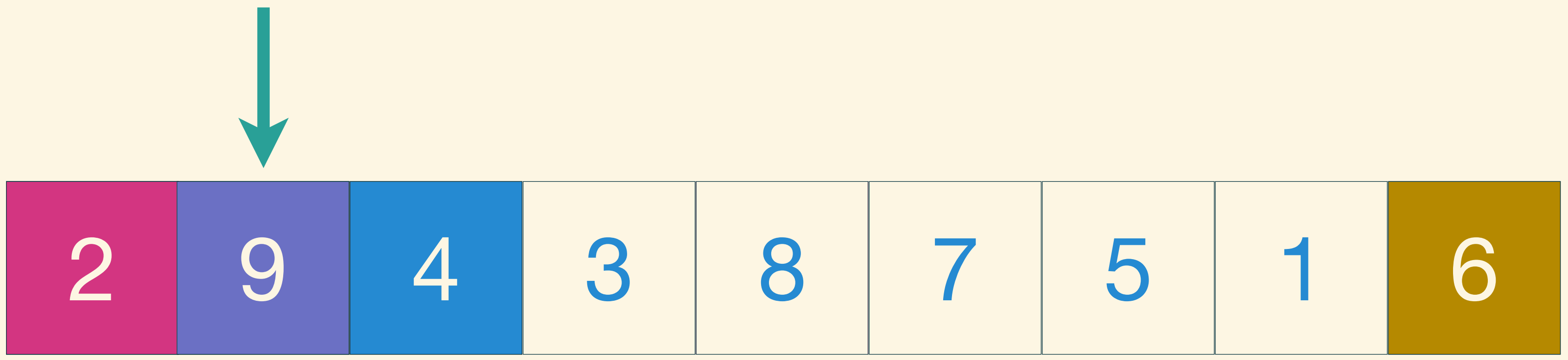
2	9	4	3	8	7	5	1	6

How to Divide



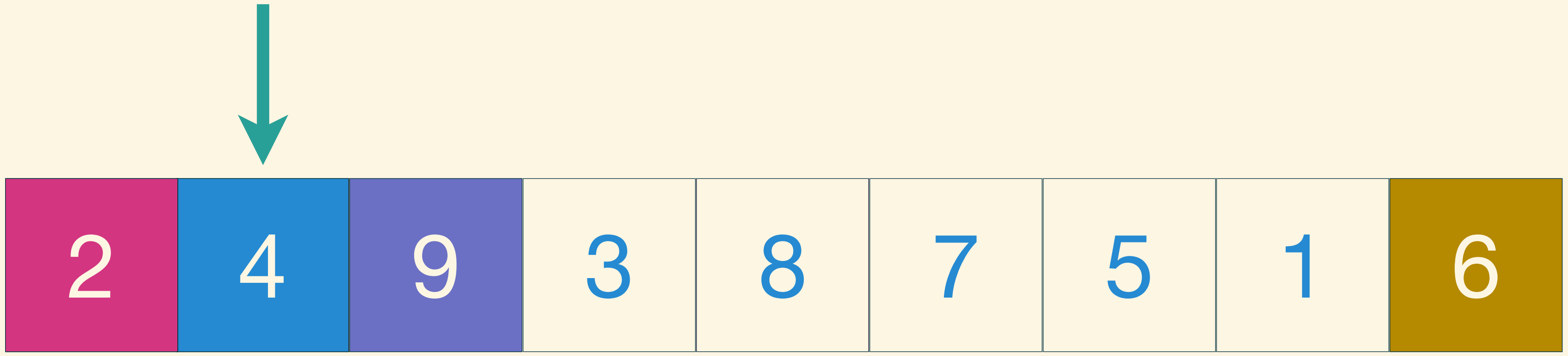
2	9	4	3	8	7	5	1	6

How to Divide



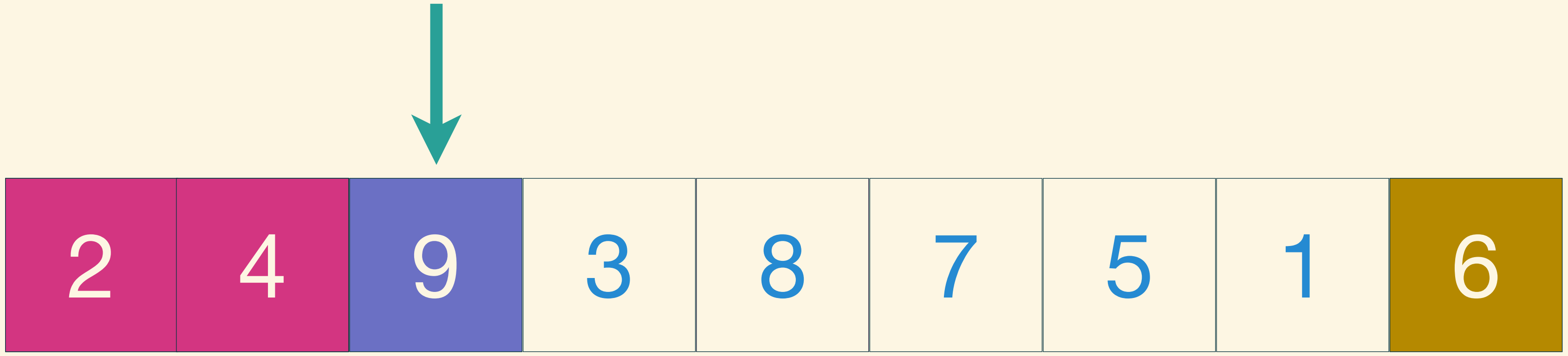
2	9	4	3	8	7	5	1	6

How to Divide



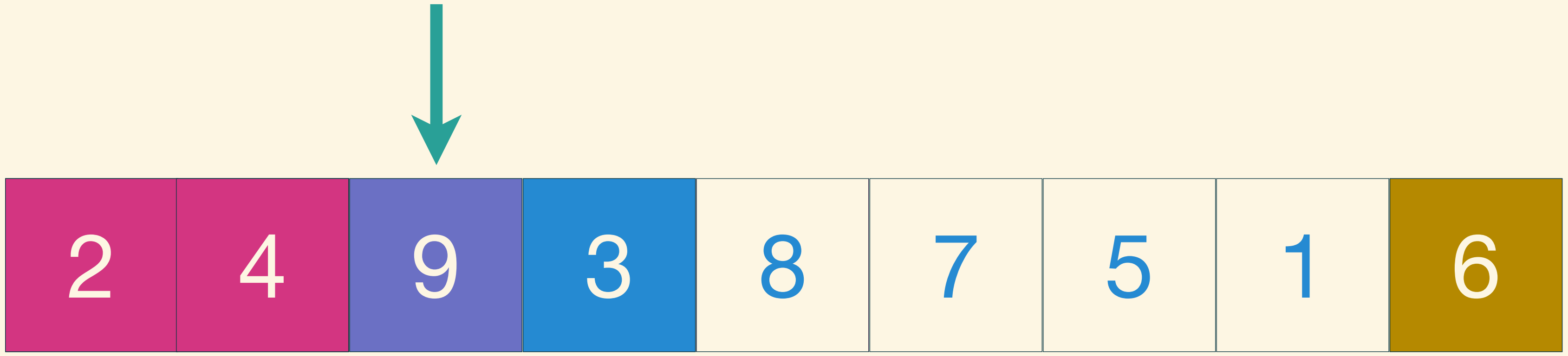
2	4	9	3	8	7	5	1	6

How to Divide



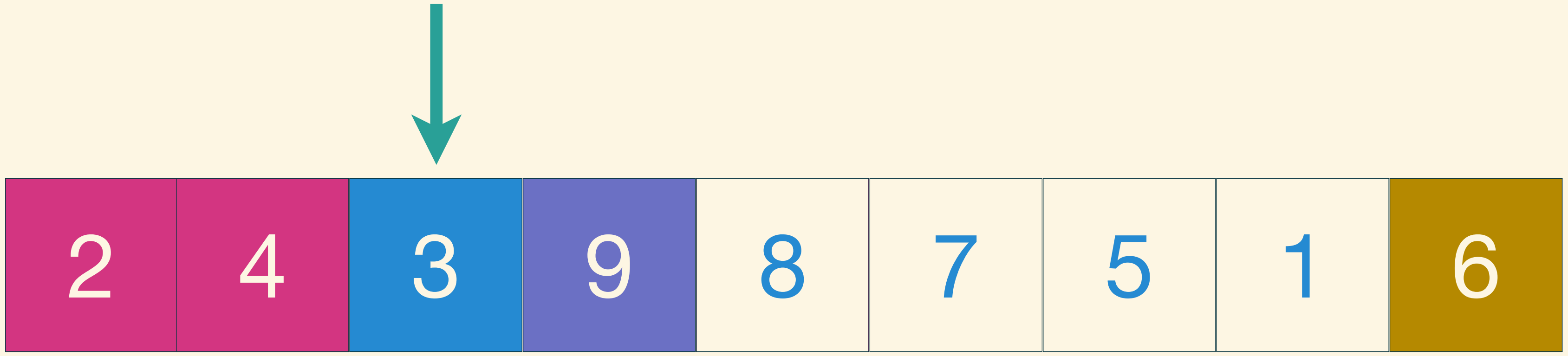
2	4	9	3	8	7	5	1	6

How to Divide



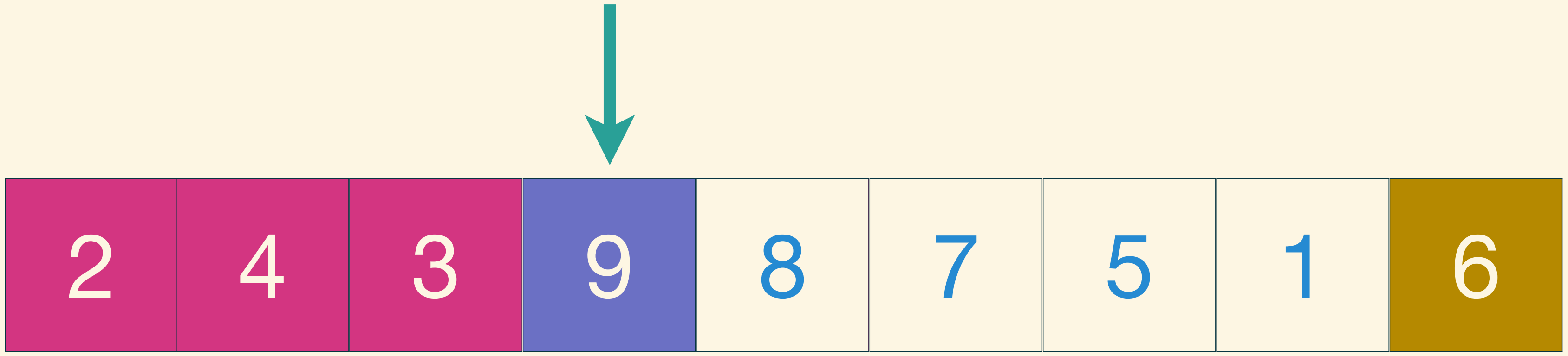
2	4	9	3	8	7	5	1	6

How to Divide



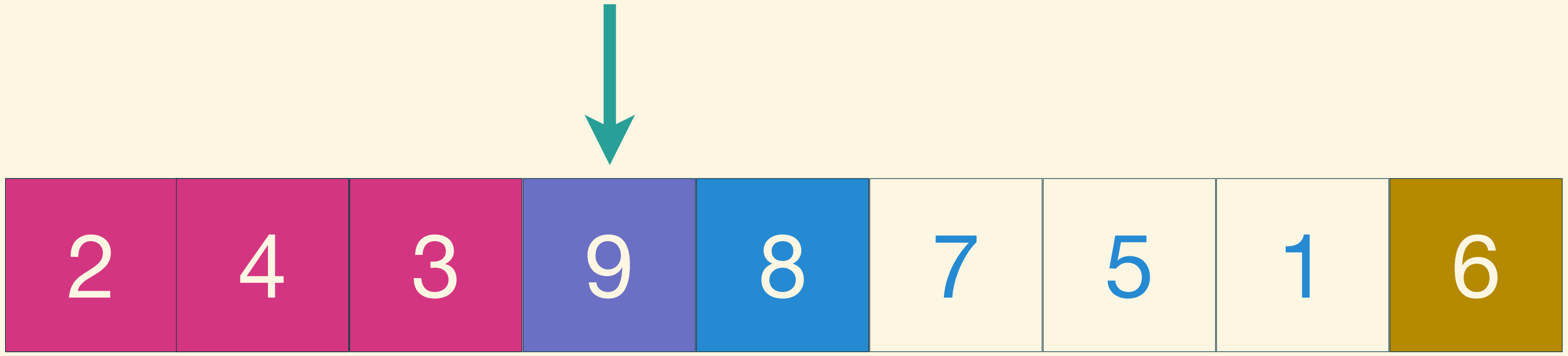
2	4	3	9	8	7	5	1	6

How to Divide



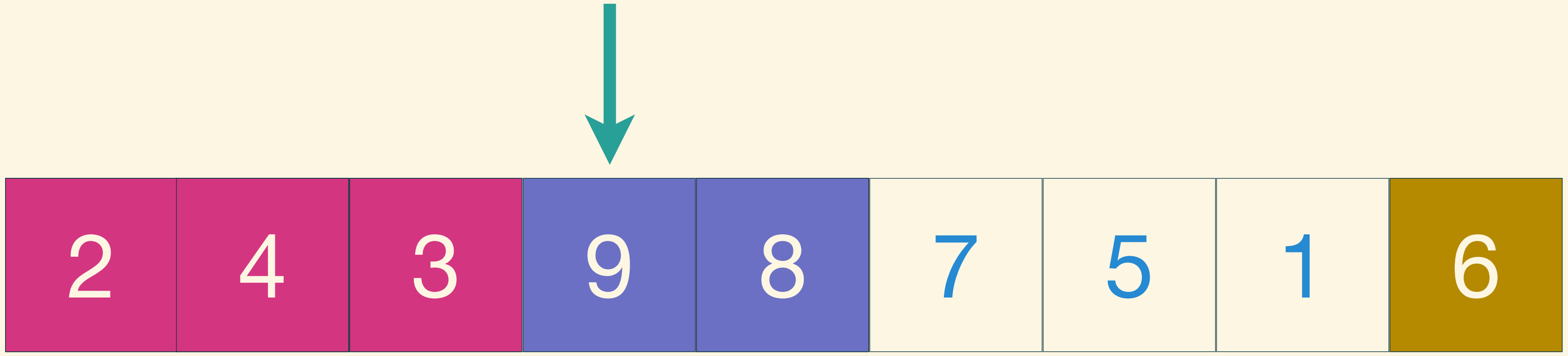
2	4	3	9	8	7	5	1	6

How to Divide



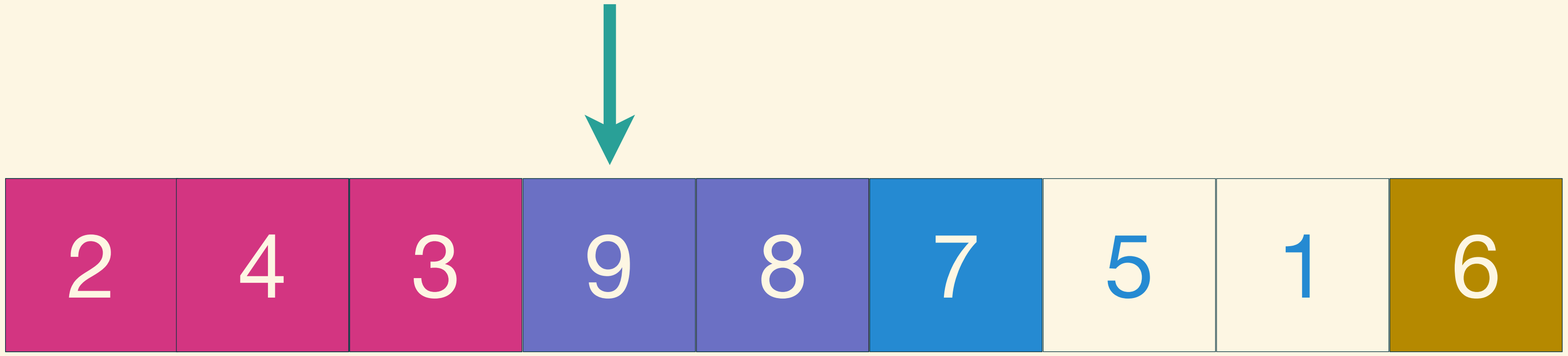
2	4	3	9	8	7	5	1	6

How to Divide



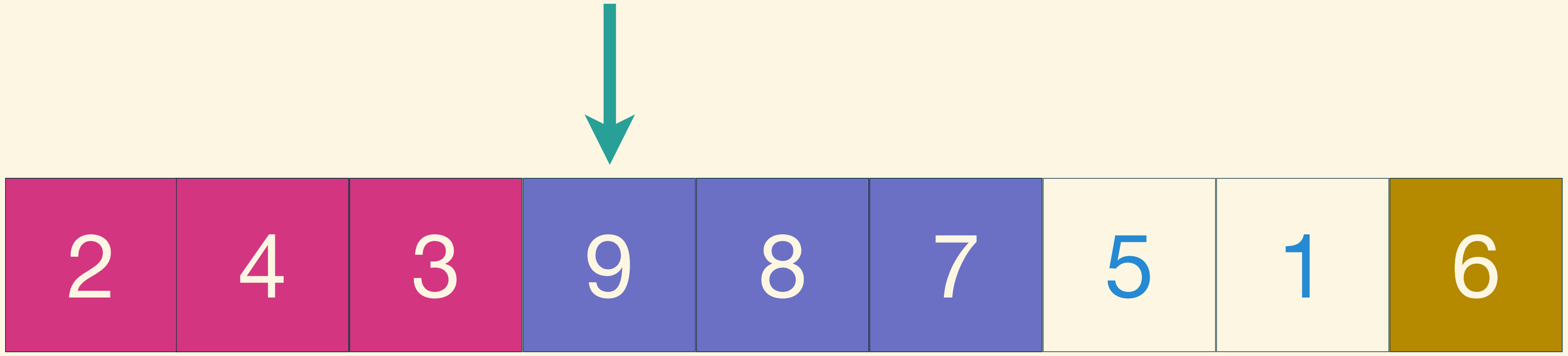
2	4	3	9	8	7	5	1	6

How to Divide



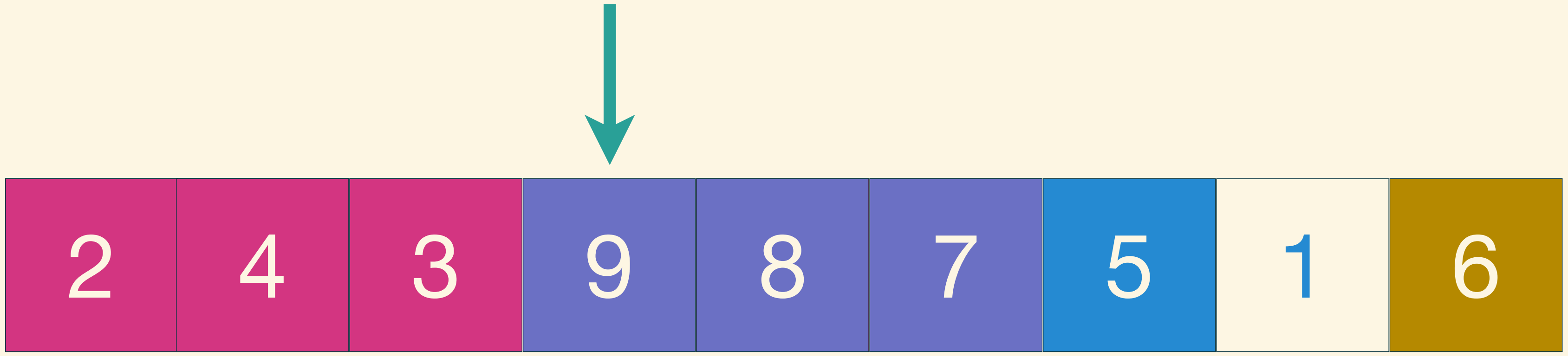
2	4	3	9	8	7	5	1	6

How to Divide



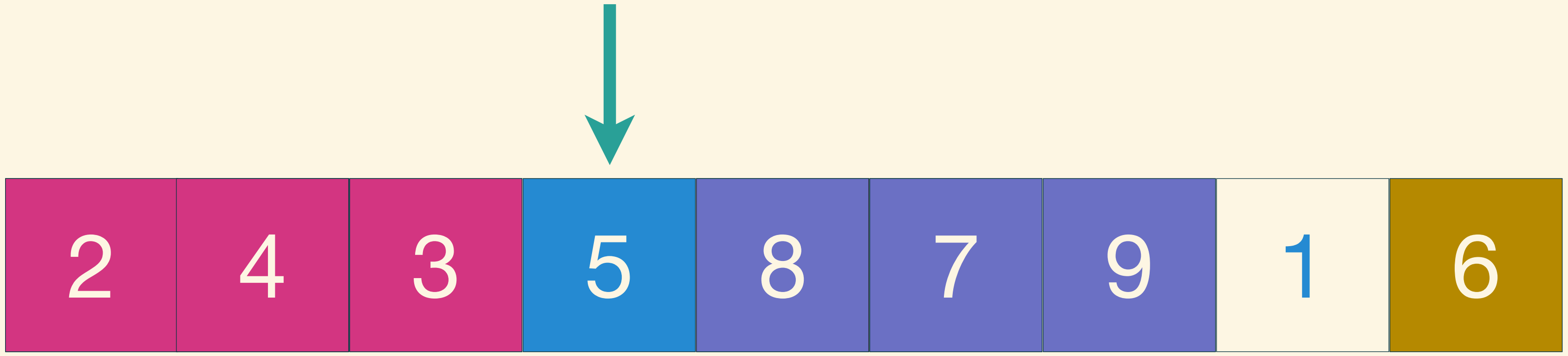
2	4	3	9	8	7	5	1	6

How to Divide



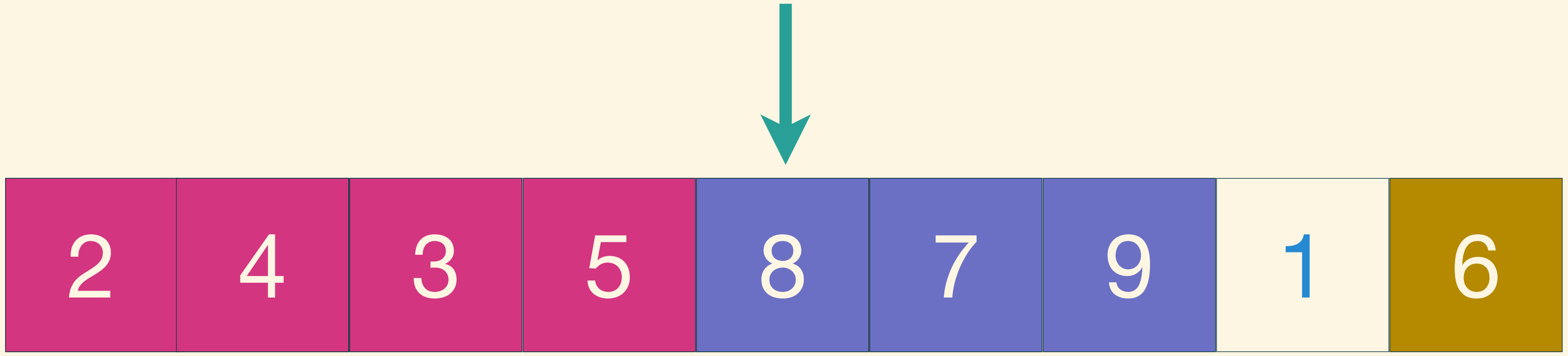
2	4	3	9	8	7	5	1	6

How to Divide



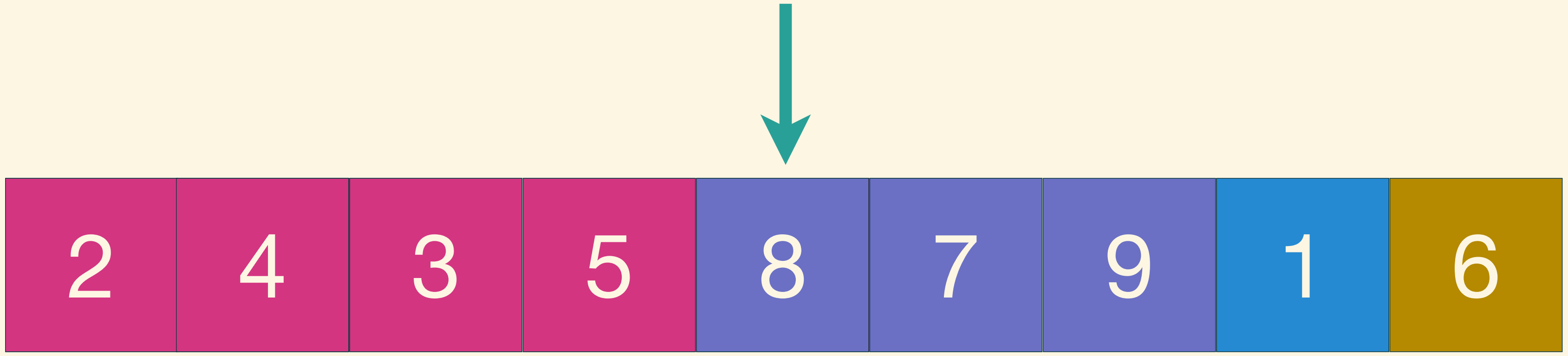
2	4	3	5	8	7	9	1	6

How to Divide



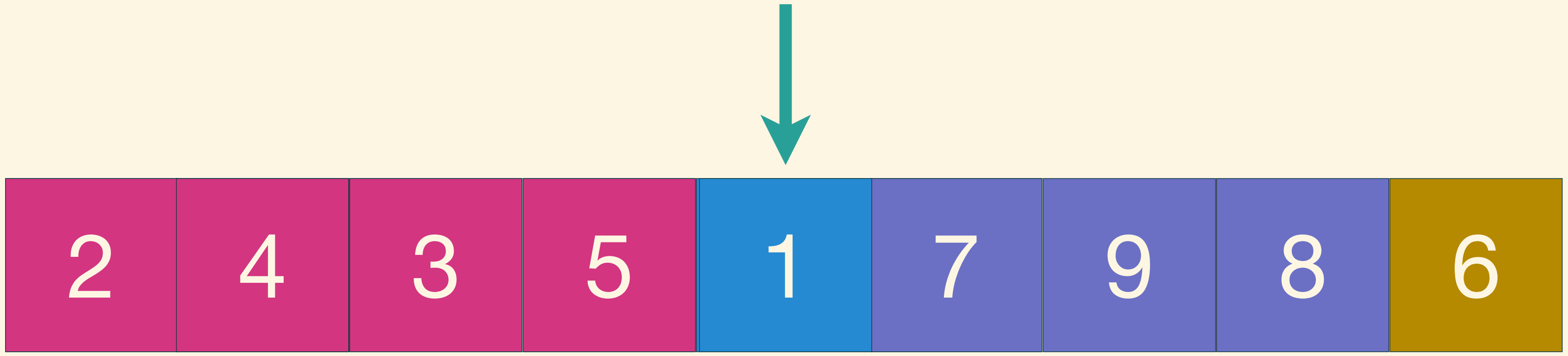
2	4	3	5	8	7	9	1	6

How to Divide



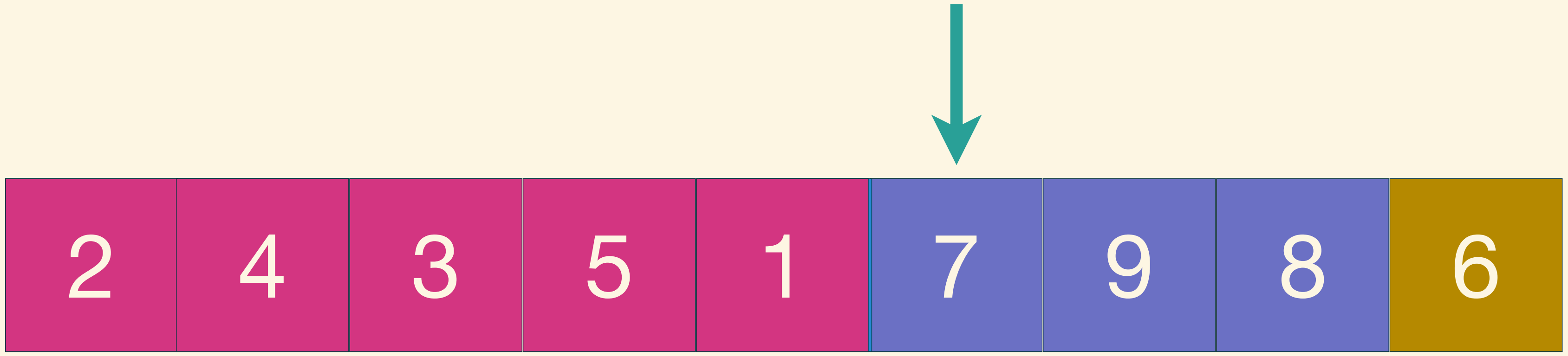
2	4	3	5	8	7	9	1	6

How to Divide



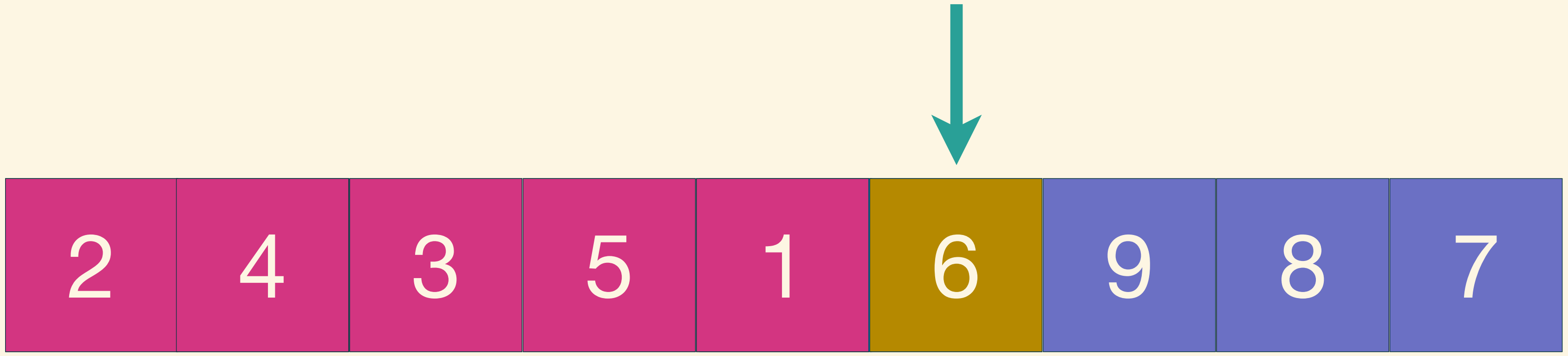
2	4	3	5	1	7	9	8	6

How to Divide



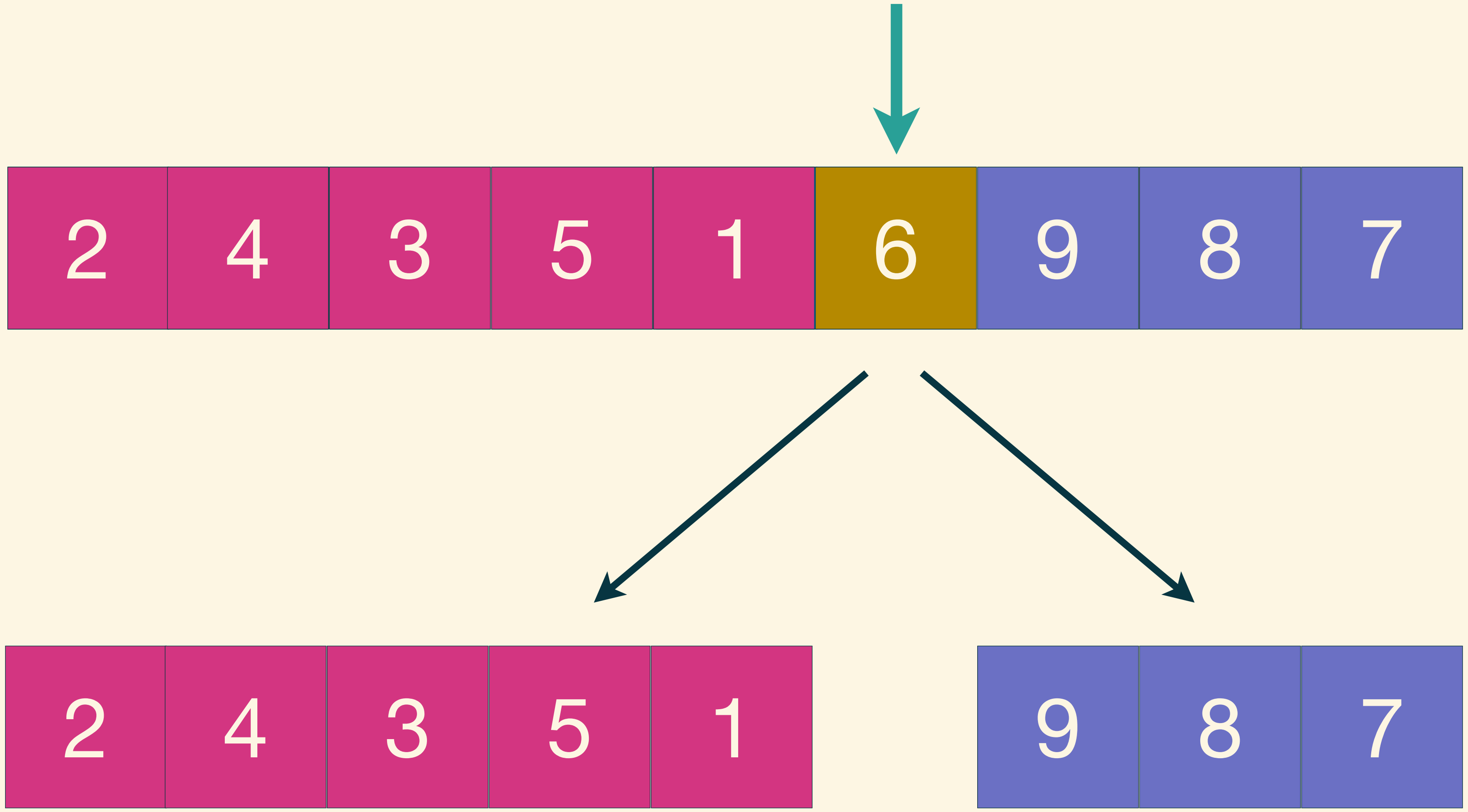
2	4	3	5	1	7	9	8	6

How to Divide



2 4 3 5 1 6 9 8 7

How to Divide

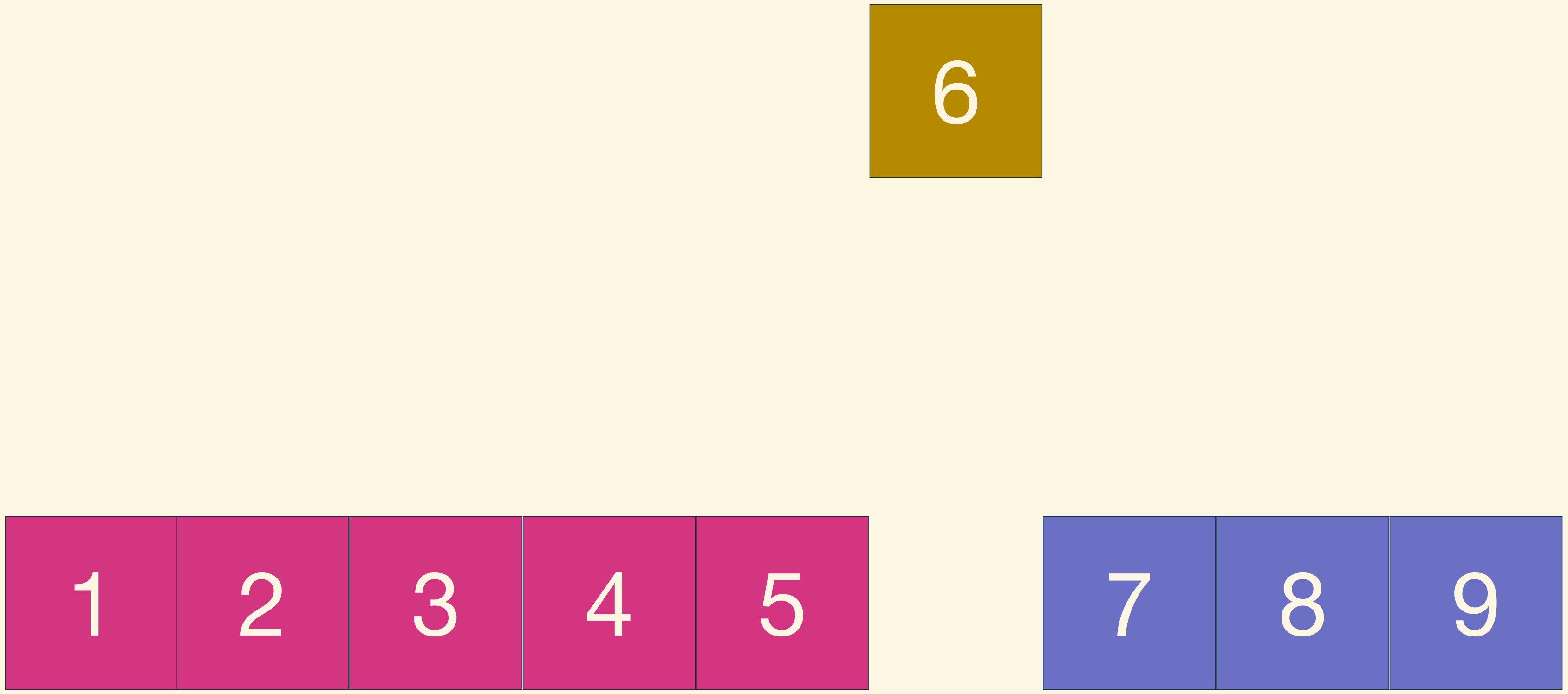


2 4 3 5 1 6 9 8 7

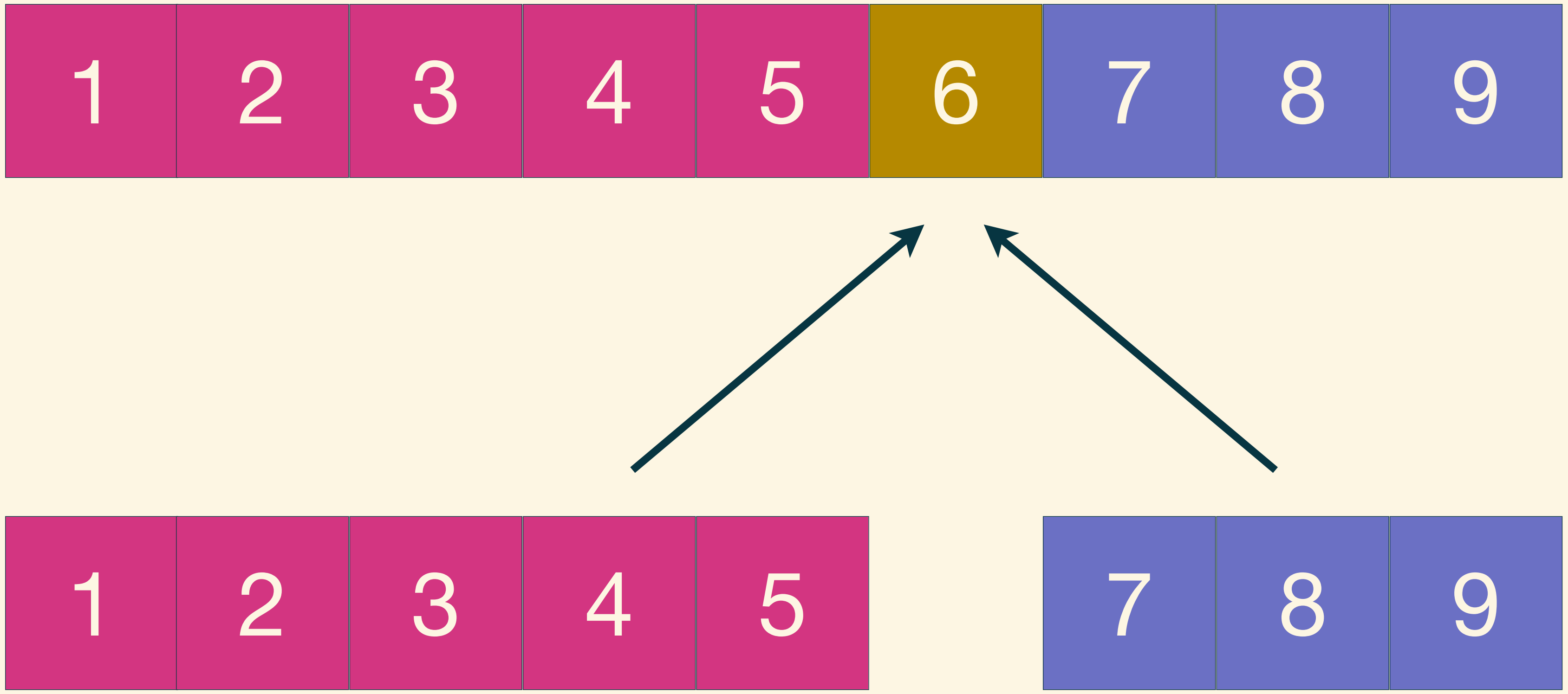
2 4 3 5 1

9 8 7

How to Conquer



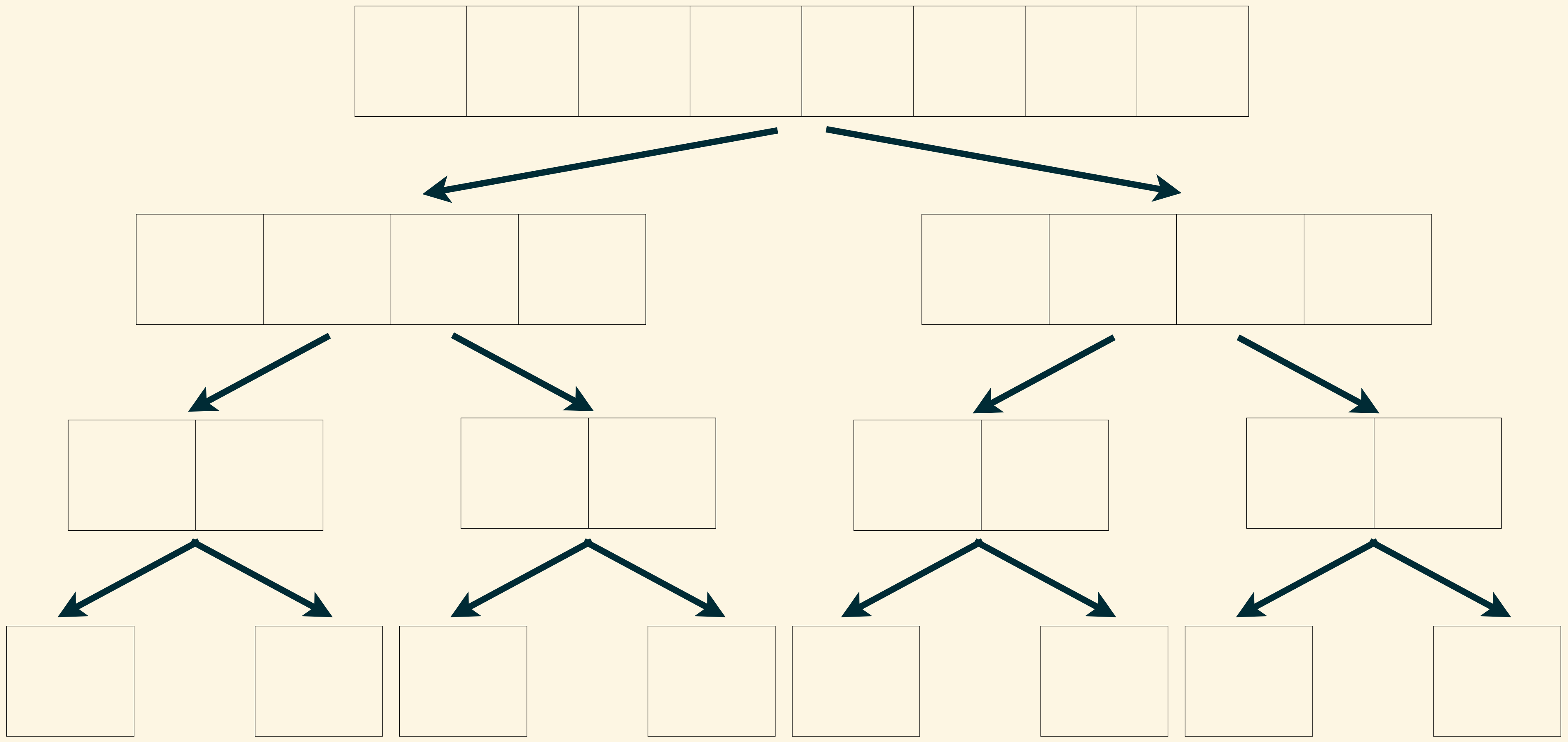
How to Conquer



Time Complexity

total time

Time Complexity



total time

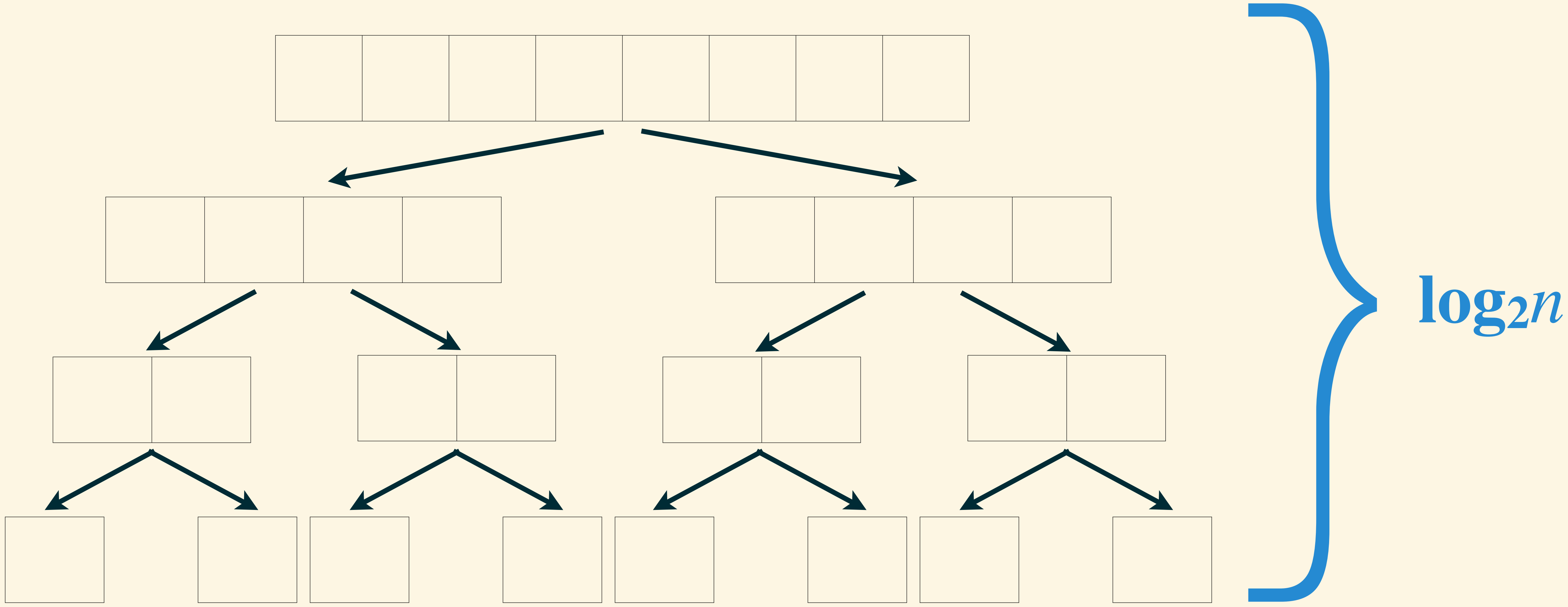
n

2 * (n/2) = n

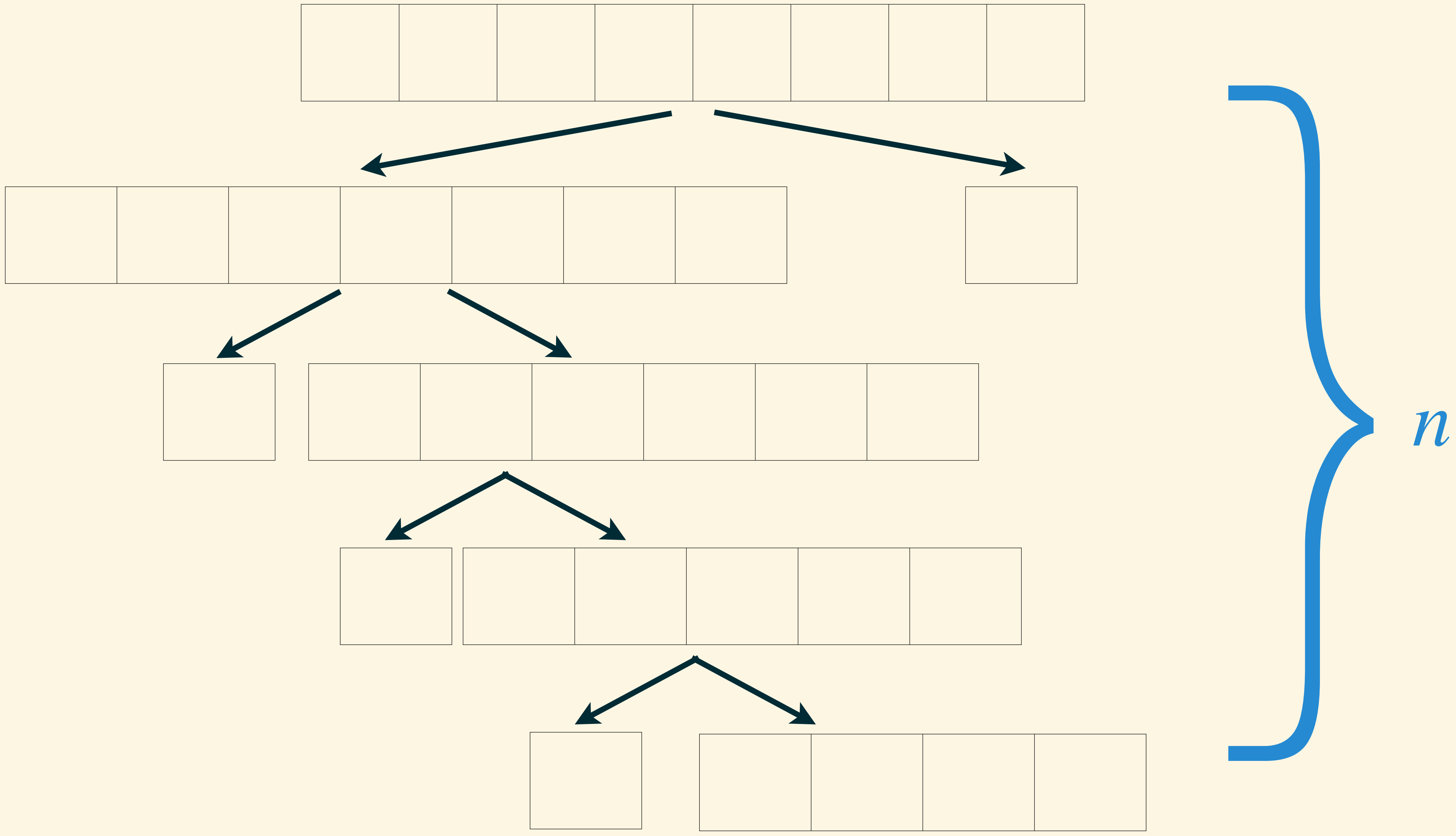
...

n * (n/n) = n

Best height



Worst height



Time Complexity:

$O(n\mathbf{log}_2 n) \sim O(n^2)$

Average Time Complexity:
$O(n\mathbf{log}_2 n)$

Builtin Function

如果你純粹想要排序罷了...

qsort (C/C++)

- include <stdlib.h> or <cstdlib>
- compare function

```
void qsort (void* base, size_t num, size_t
size, int (*compar)(const void*,const void*));
```

```
void qsort (void* base, size_t num, size_t
size, int (*compar)(const void*,const void*));
```

base: 指向欲排序列表起始位置之指標

num: 欲排序的元素數量

size: 各元素大小

compar: 比較函式的函式指標

Compare Function

Function prototype:

```
int function_name(const void* p1, const void* p2);
```

return value	means
less than 0	p1 < p2
equal to 0	p1 == p2
greater than 0	p1 > p2

Example

```
int cmp( const void* p1, const void* p2 ) {
  return *(int*)p1 - *(int*)p2;
}

int main() {
  int n, N[ 10010 ];
  while ( scanf( “%d”, &n ) != EOF ) {
    int x;
    for ( int i = 0; i < n; ++i ) {
      scanf( “%d”, &x );
      N[ i ] = x;
    }

    qsort( N, n, sizeof( int ), cmp );
  }
  return 0;
}
```

sort (STL)

- include <algorithm>
- compare function for customized behavior

```
template <class RandomAccessIterator>
void sort (RandomAccessIterator first,
RandomAccessIterator last);
```

```
template <class RandomAccessIterator, class Compare>
void sort (RandomAccessIterator first,
RandomAccessIterator last, Compare comp);
```

Sort by operator <

```
template <class RandomAccessIterator>
void sort (RandomAccessIterator first,
RandomAccessIterator last);
```

first: 指向欲排序列表起始位置之 iterator

last: 指向欲排序列表結尾位置之 iterator

指標可被轉型為 iterator!

依照該資料型別的小於運算子 (<) 作為排序依據！

Customized Data Type

Function prototype for operator <:

```
bool operator< ( const type_name &p ) const;
```

return value	means
TRUE	this < p
FALSE	this >= p

Example (builtin type)

```
int main() {
  int n, N[ 10010 ];
  while ( scanf( “%d”, &n ) != EOF ) {
    int x;
    for ( int i = 0; i < n; ++i ) {
      scanf( “%d”, &x );
      N[ i ] = x;
    }
    sort( N, N + n );
  }
  return 0;
}
```

Example (custom type)

```
struct T {
  int x, y;
  bool operator< ( const T &p ) const {
    return x == p.x ? x < p.x : y < p.y;
  }
};
T pt[ 10010 ];
int main() {
  int n;
  while ( scanf( “%d”, &n ) != EOF ) {
    int x, y;
    for ( int i = 0; i < n; ++i ) {
      scanf( “%d %d”, &x, &y );
      pt[ i ].x = x, pt[ i ].y = y;
    }
    sort( pt, pt + n );
  }
  return 0;
}
```

Sort by Compare Function

```
template <class RandomAccessIterator, class Compare>
void sort (RandomAccessIterator first,
RandomAccessIterator last, Compare comp);
```

first: 指向欲排序列表起始位置之 iterator

last: 指向欲排序列表結尾位置之 iterator

comp: 比較函式

指標可被轉型為 iterator!

Customized Data Type

Function prototyp:

```
bool function_name ( type_name p1, type_name p2 );
```

return value	means
TRUE	p1 < p2
FALSE	p1 >= p2

Example (descending)

```
bool descending( int p1, int p2 ) {
  return p1 >= p2;
}

int main() {
  int n, N[ 10010 ];
  while ( scanf( “%d”, &n ) != EOF ) {
    int x;
    for ( int i = 0; i < n; ++i ) {
      scanf( “%d”, &x );
      N[ i ] = x;
    }
    sort( N, N + n, descending );
  }
  return 0;
}
```

Reference

- 冒泡排序 - 維基百科，自由的百科全書
- 歸併排序 - 維基百科，自由的百科全書
- 快速排序 - 維基百科，自由的百科全書
- qsort - C++ Reference
- sort - C++ Reference

Practice Now

10810 - Ultra-QuickSort

Thank You for Your Listening.