Sorting Algorithms I

Basic Sorts

Why do we need to know sorting?

- Fundamental routine
- Many applications use them
- Many sorting algorithms choices
- Consider constraints & tradeoffs
  - dataset type and size
  - stability requirements
  - ease of understanding
  - time and space complexity

Basic Sorting Algorithms

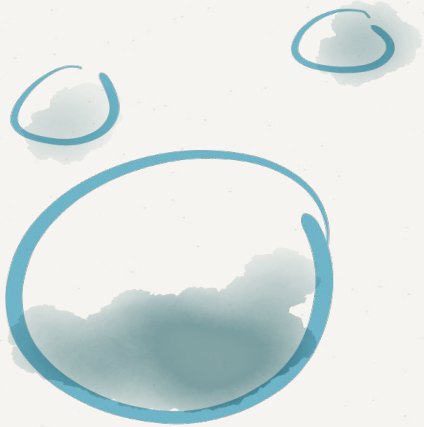
- Bubble Sort
- Selection Sort
- Insertion Sort

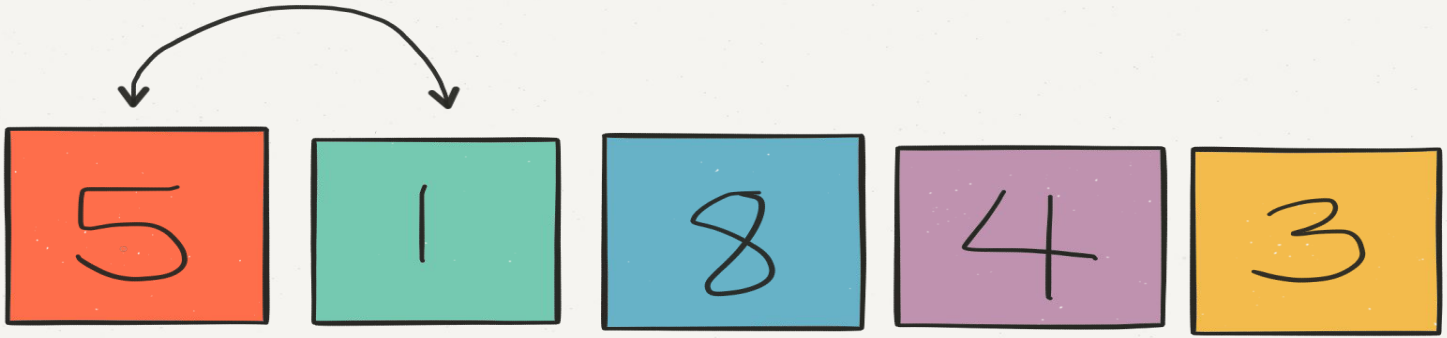
Purpose?

- Easy to understand
- Good starting point
- Useful for small inputs (n < 10)

5 1 8 4 3

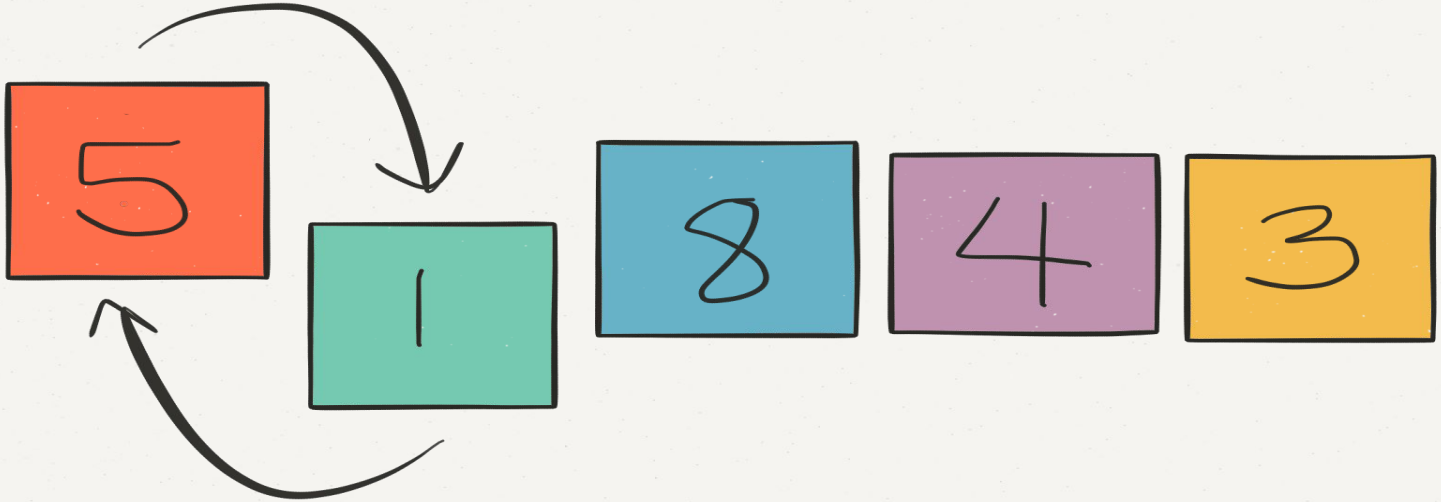
Bubble





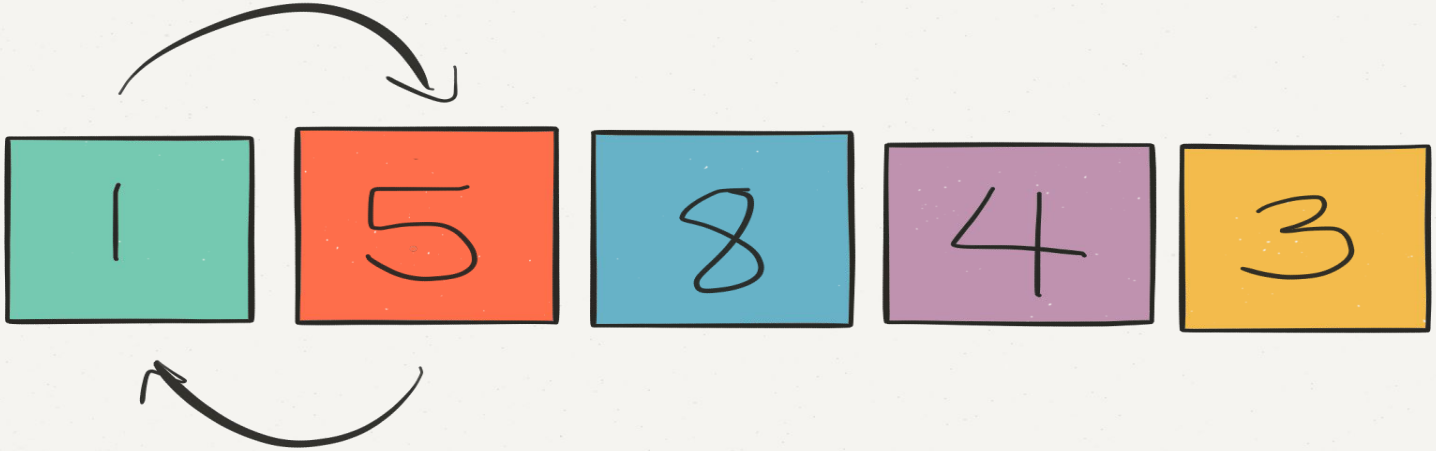
$5 > 1$

SWAP?

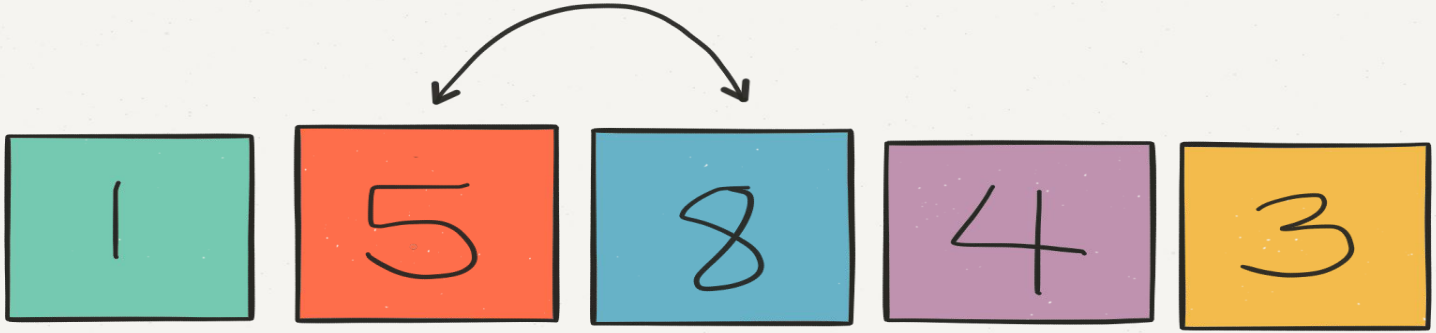


5 > 1

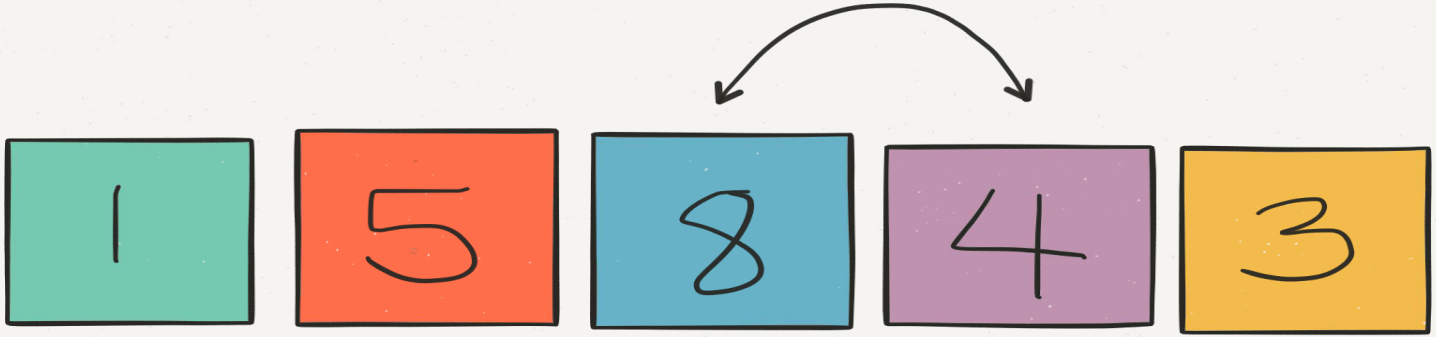
SWAP : TRUE



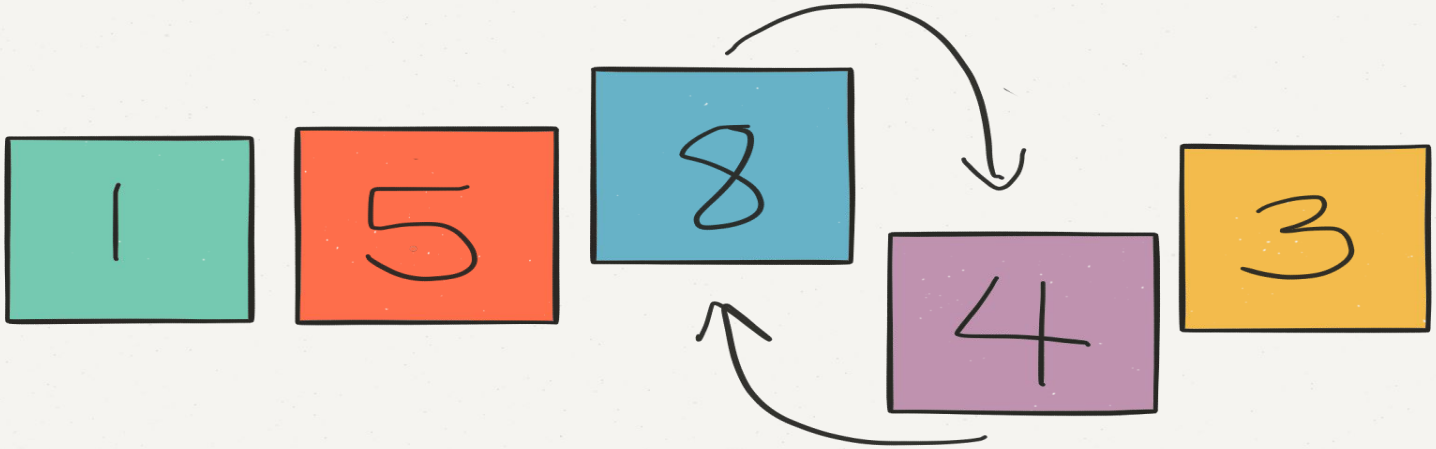
$5 > 1$



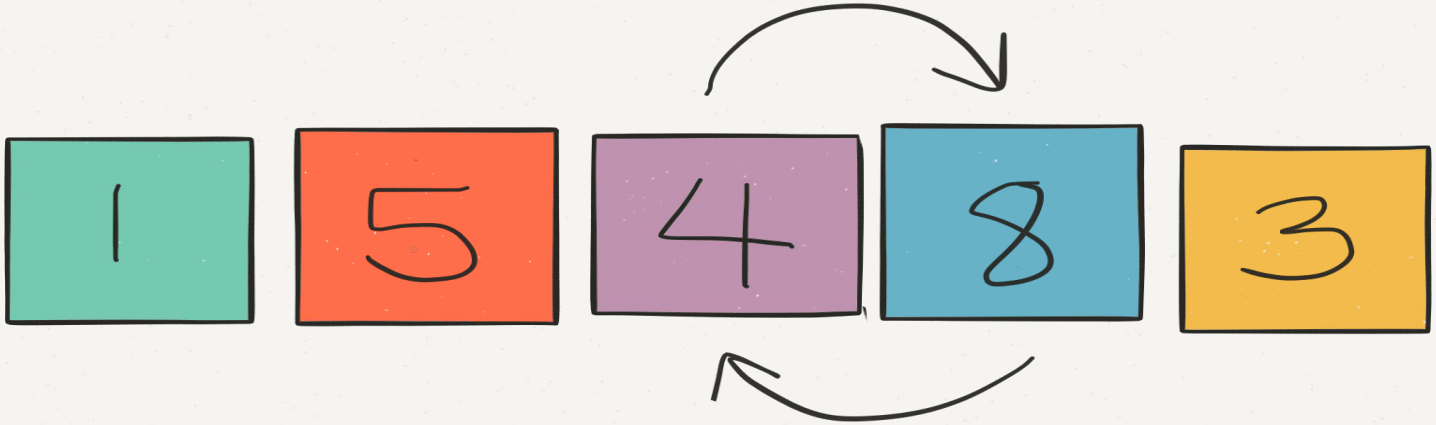
$5 < 8$



$8 > 4$

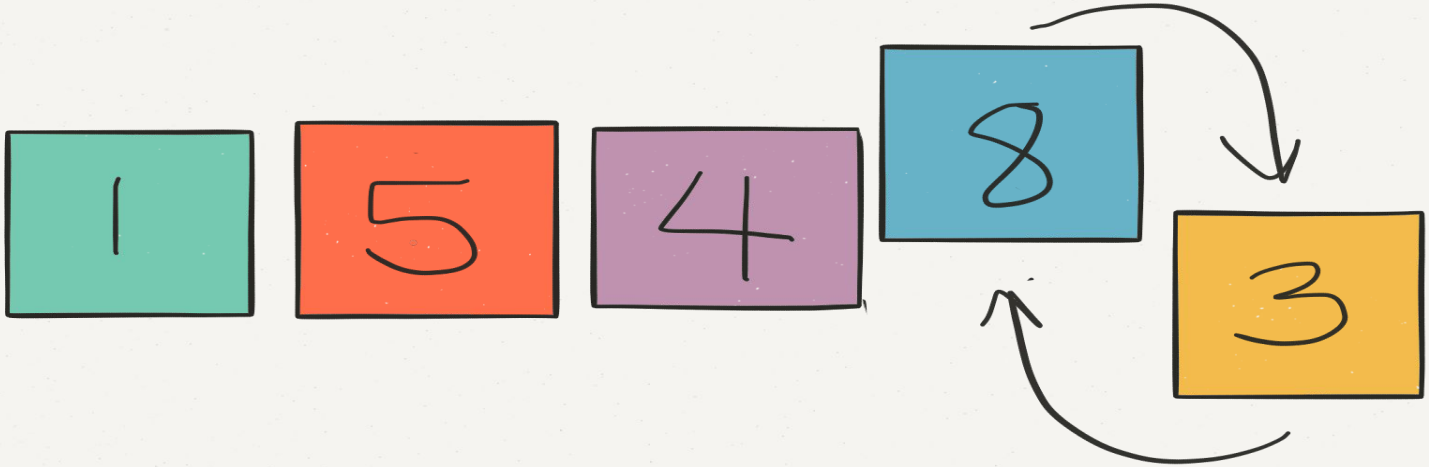


$8 > 4$

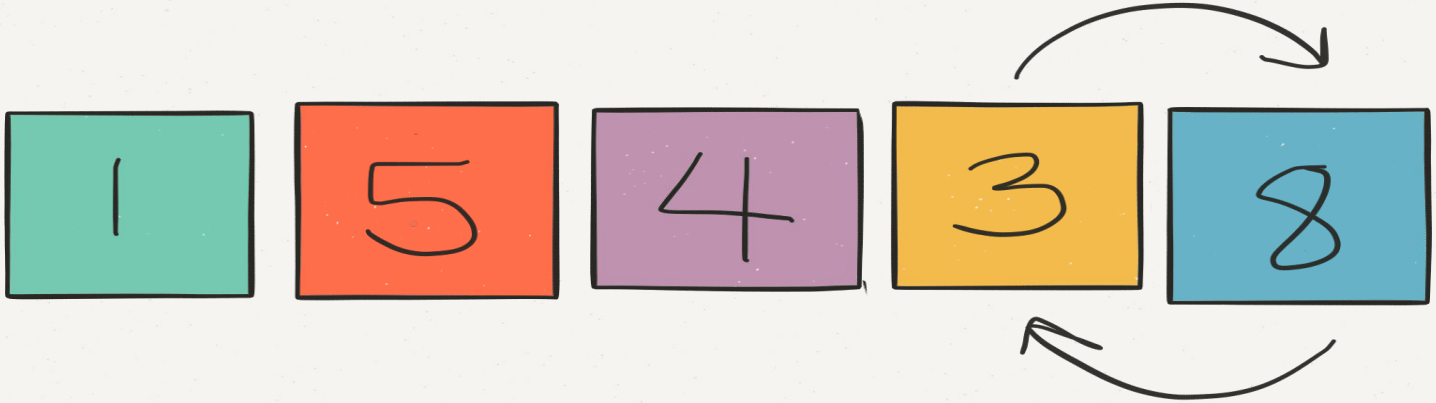


$8 > 4$

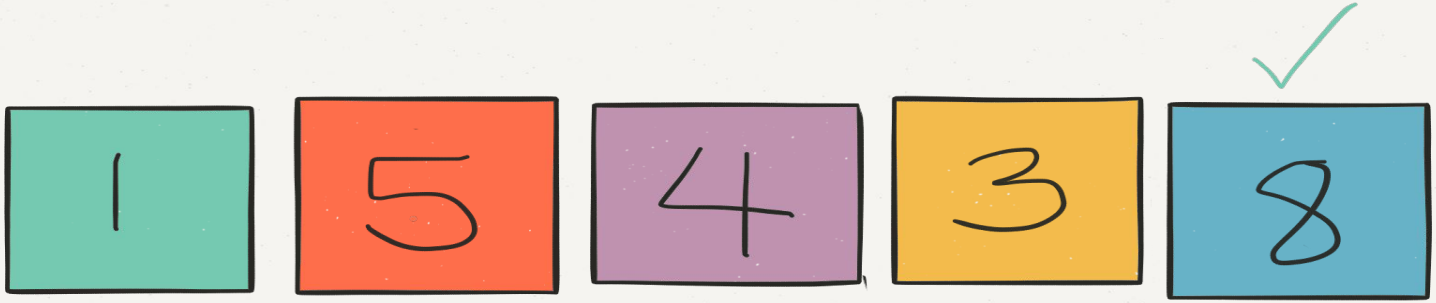
$8 > 3$



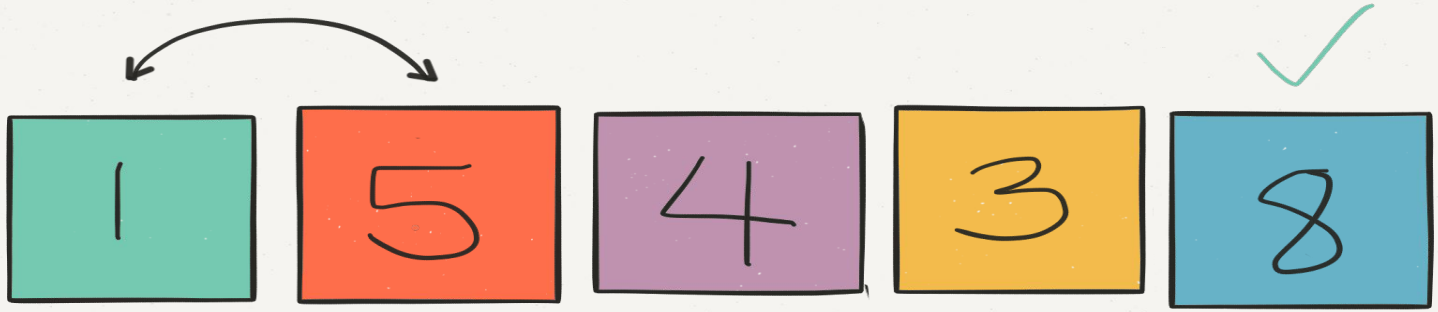
$8 > 3$



$8 > 3$

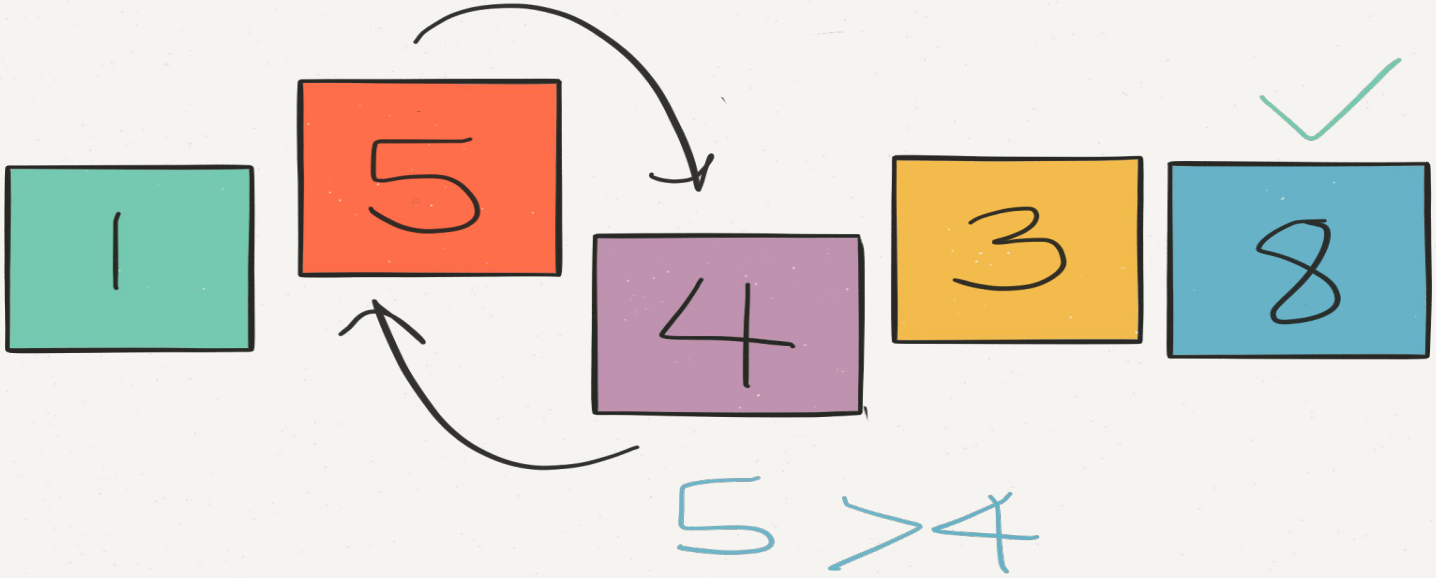


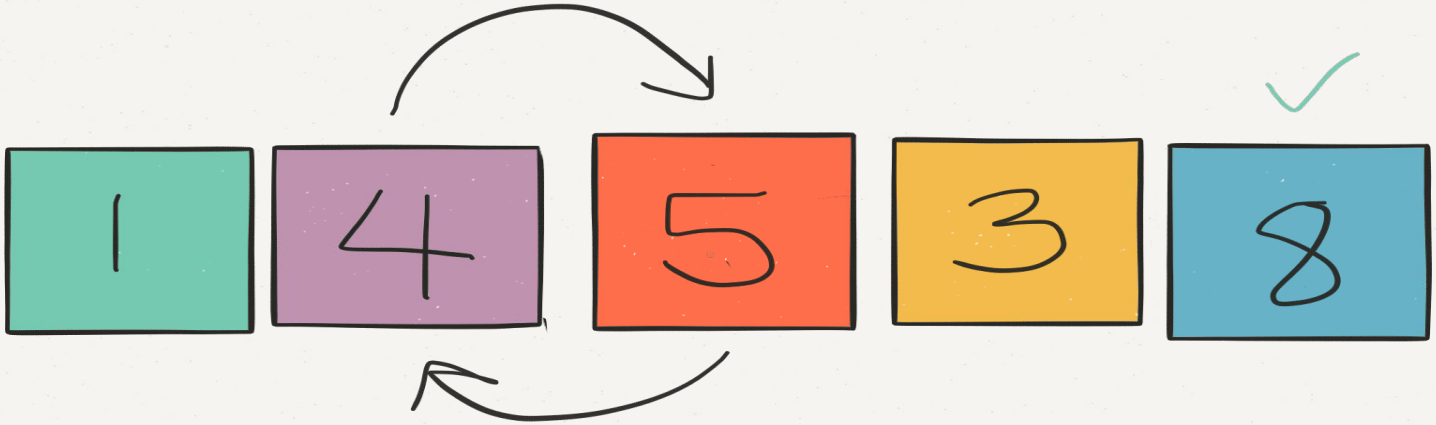
SWAP : TRUE . . . REPEAT!



$1 < 5$

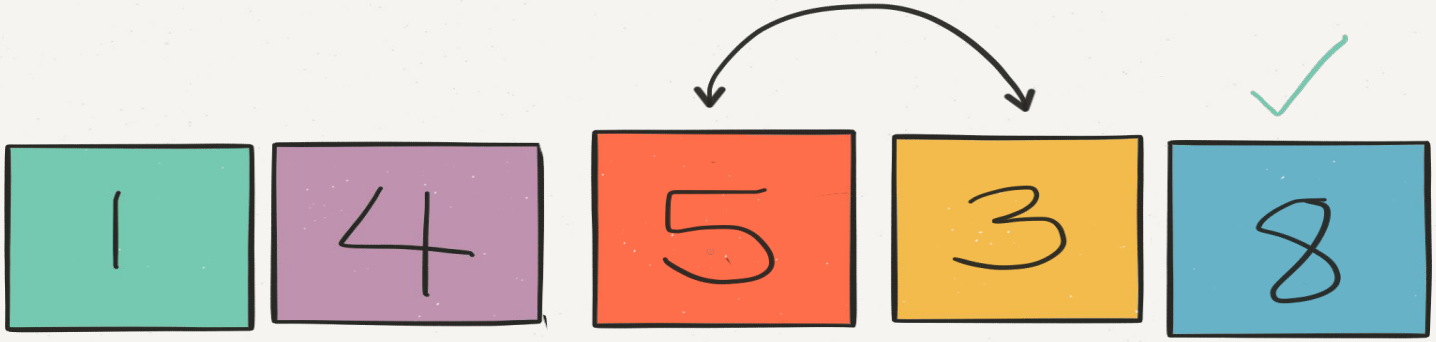
SWAP : ?


1
5
4
3
8
5 > 4
SWAP : TRUE



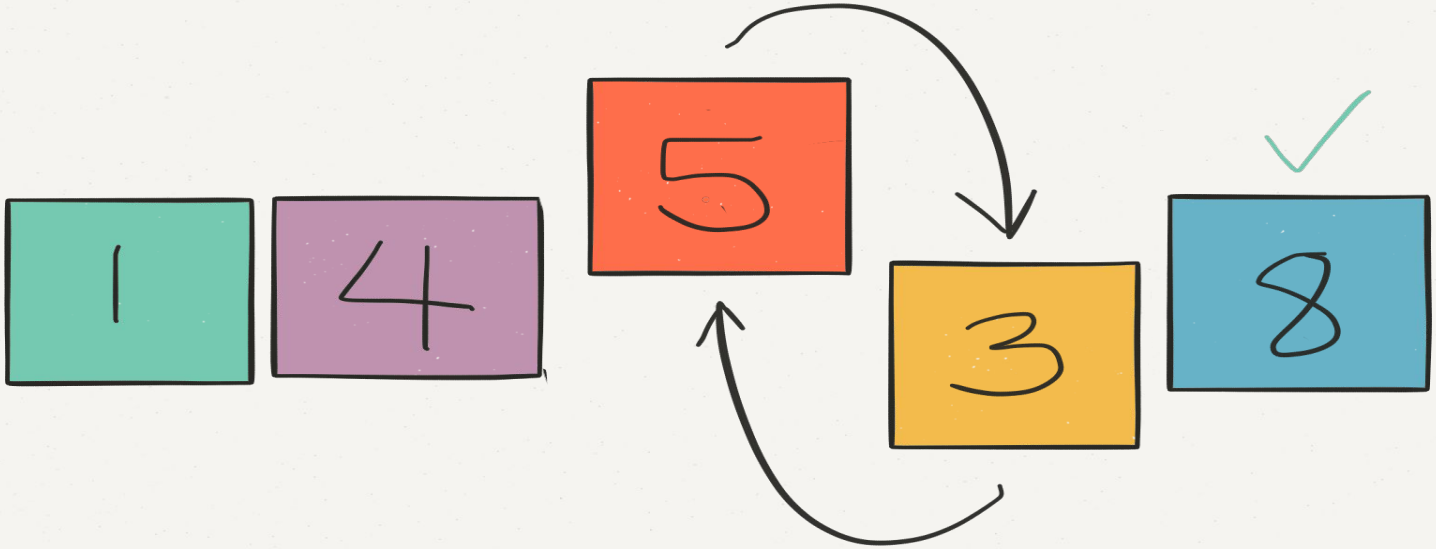
5 > 4

SWAP : TRUE

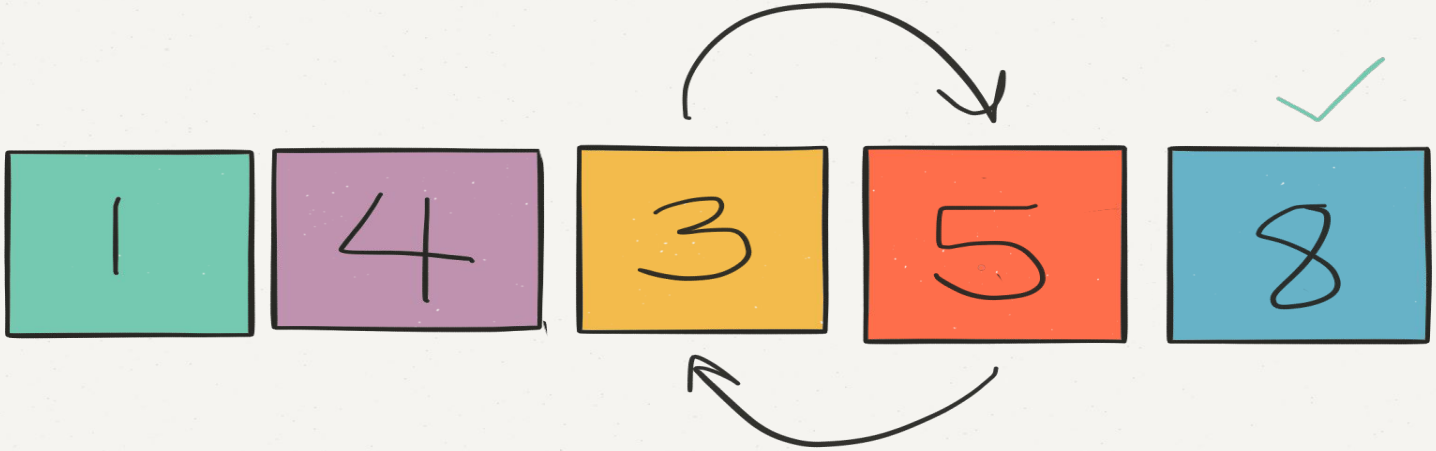


5 > 3

SWAP : TRUE



SWAP : TRUE



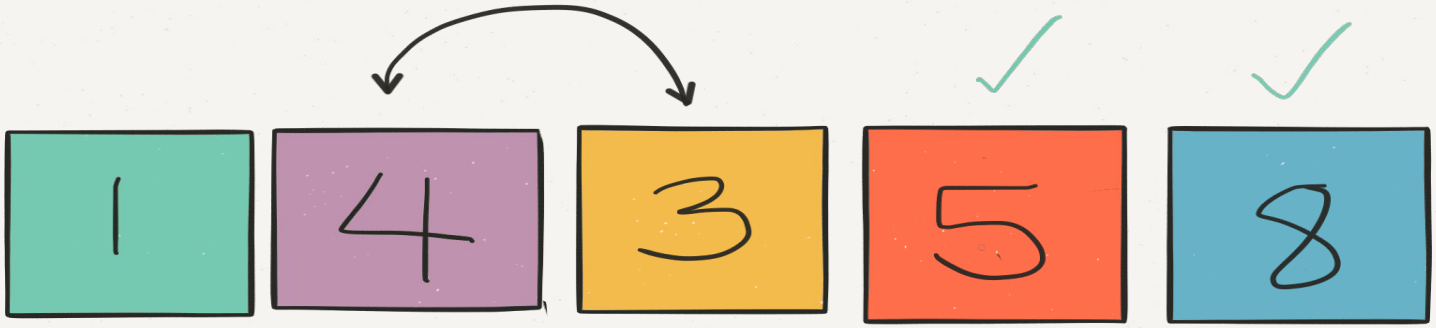
SWAP : TRUE

REPEAT!

SWAP: TRUE

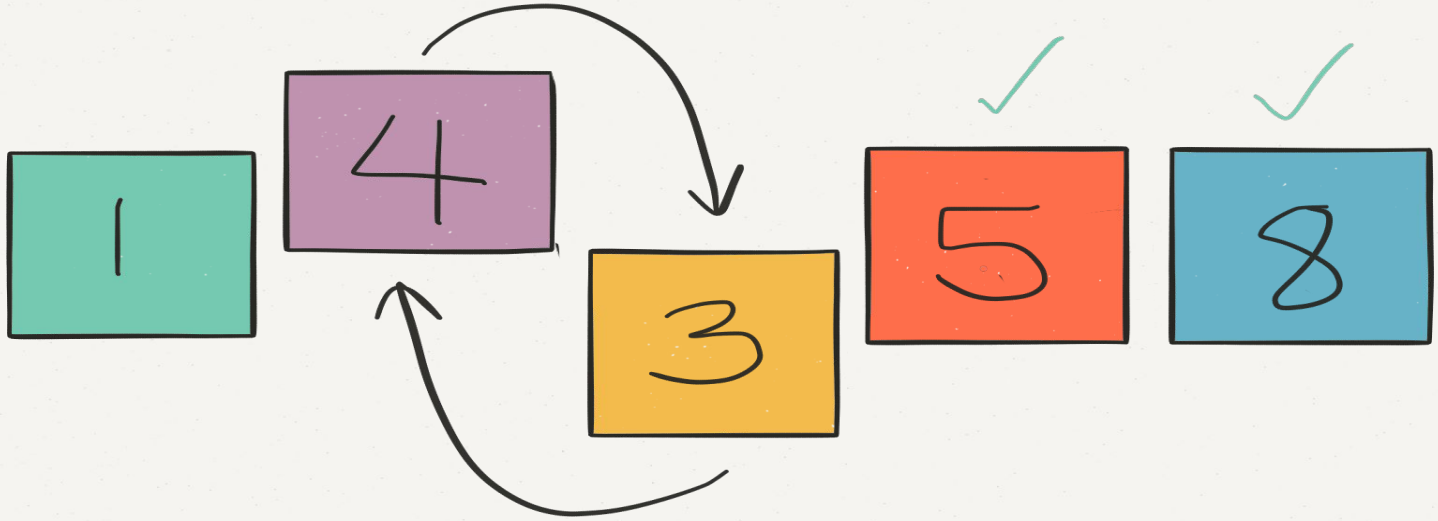
$1 < 4$

SWAP: ?

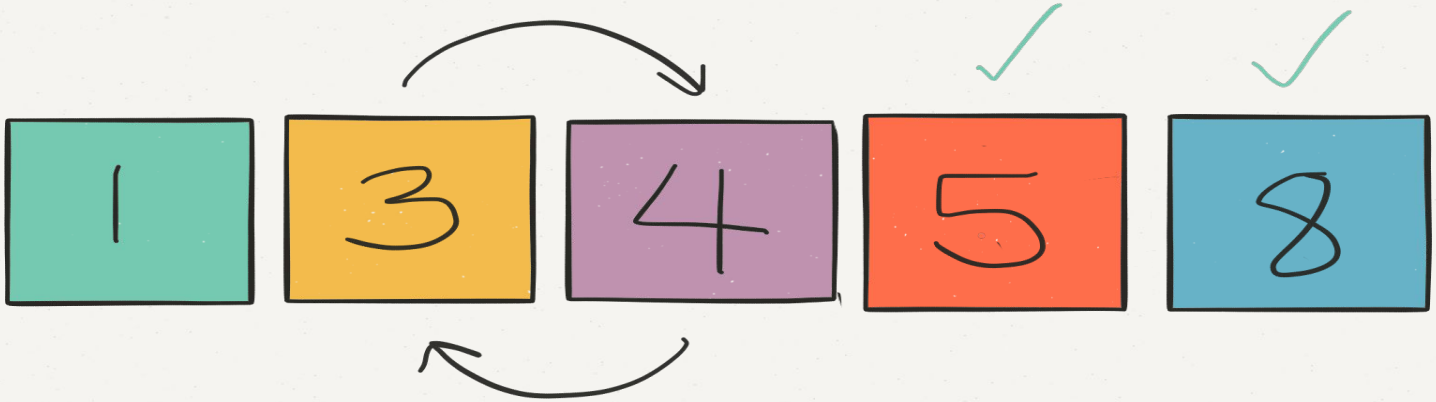


473

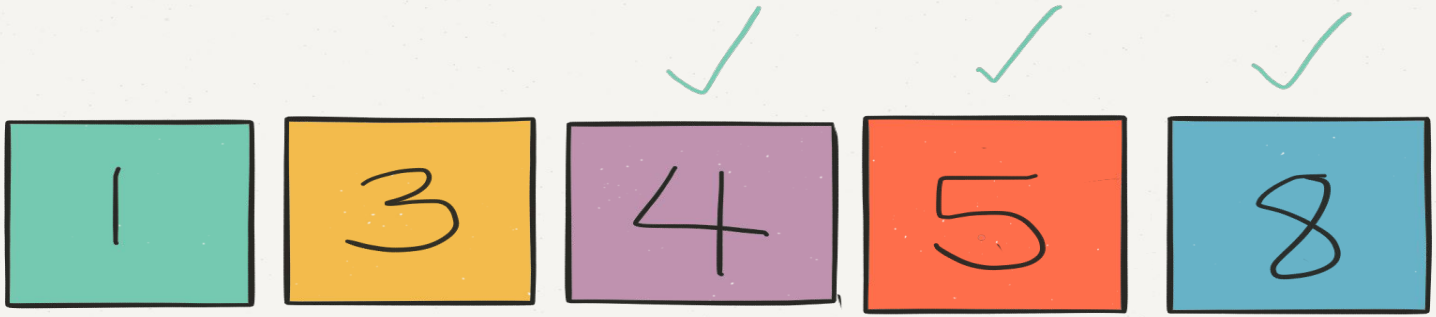
SWAP: TRUE



SWAP: TRUE

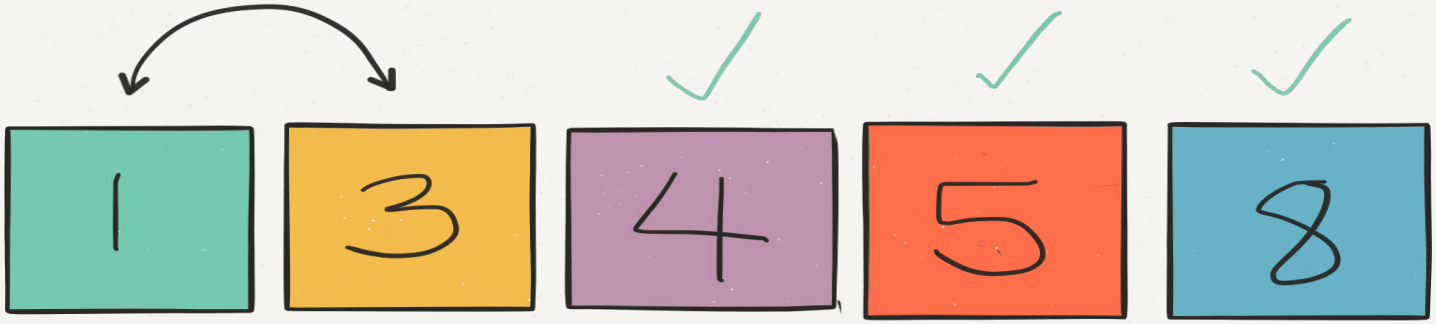


SWAP : TRUE



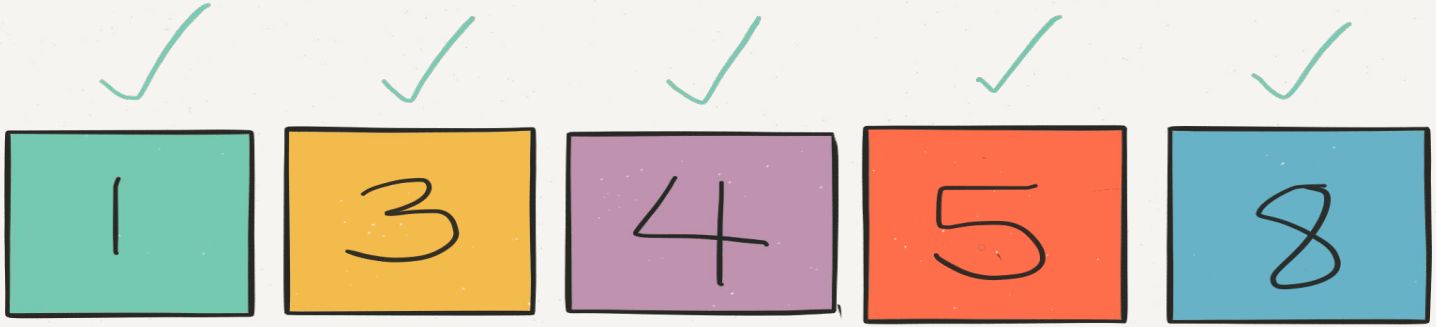
REPEAT!

SWAP: TRUE



1 < 3

SWAP : ?



DONE!

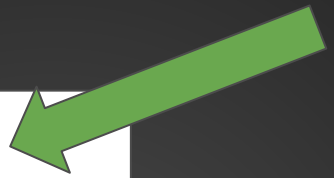
SWAP: FALSE

Bubble Sort

```
def bubble(arr):
```

Bubble Sort

loop backwards

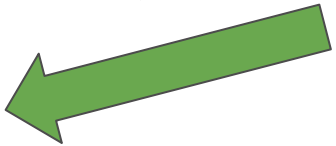


```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
```

Bubble Sort

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):

    for i in range(0, end):
```

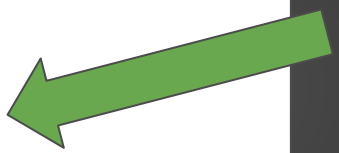


loop from 0 to “end”

Bubble Sort

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):

    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
```



perform a swap if current and next is out of order

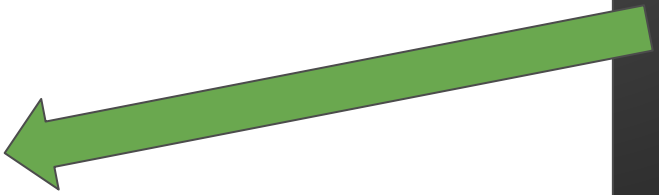
Bubble Sort

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):

    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]



  return arr
```



return sorted array

Bubble Sort

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]




  return arr
```

add flag to see if there was a swap within that loop

Bubble Sort

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True


  return arr
```

if there is a swap,
change to ‘True’

Bubble Sort

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

if no swaps were made in one outer loop, array is sorted

Bubble Sort

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

n

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

n
1

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

```
# n
  # 1
  # n/2
```

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

```
# n
  # 1
  # n/2
    # 1
    # 1
    # 1
```

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

```
# n
  # 1
  # n/2
    # 1
    # 1
    # 1
  # 1
  # 1
```

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

```
# n
  # 1
  # n/2
    # 1
    # 1
    # 1
  # 1
  # 1
# 1
```

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

```
# n
  # 1
  # n/2
    # 3


  # 1
  # 1
# 1
```

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

```
# n
  # 1
  # 3/2n


  # 1
  # 1
# 1
```

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

n

3/2n + 3

1

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

$3/2n^2 + 3n$

1

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

$3/2n^2 + 3n + 1$

Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

$3/2n^2 + 3n + 1$

drop coefficients and lower order terms

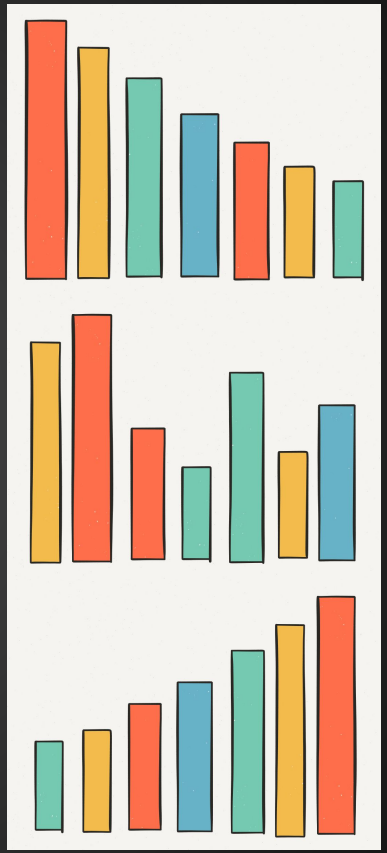
Bubble Sort - Time Complexity

```
def bubble(arr):
  for end in range(len(arr)-1,-1,-1):
    swap = False
    for i in range(0, end):
      if arr[i] > arr[i+1]:
        arr[i], arr[i+1] = arr[i+1], arr[i]
        swap = True
    if swap == False:
      break
  return arr
```

Quadratic $O(n^2)$

Time Complexity

- Worse case
  - $O(n^2)$
- Average Case
  - $O(n^2)$
- Best Case
  - $\Omega(n)$



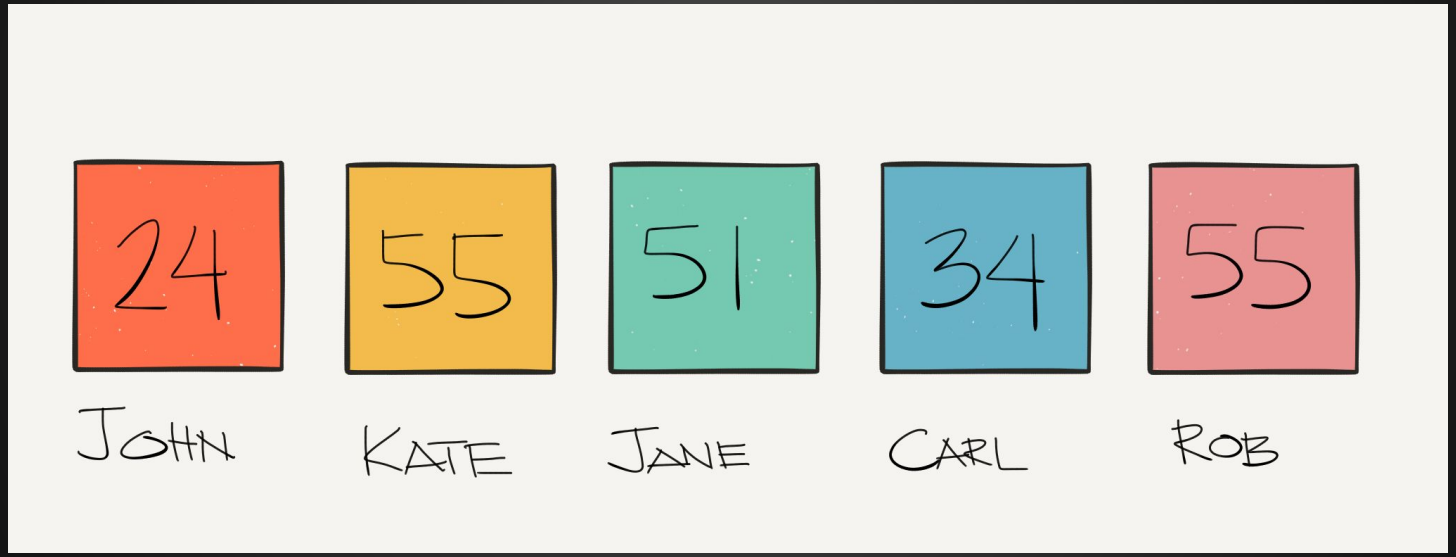
Advantages/Disadvantage

- Simple to understand
- Only linear time if sorted
- Constant auxiliary space
- Stable
- Impractical for virtually all applications
  - Insertion sort is faster

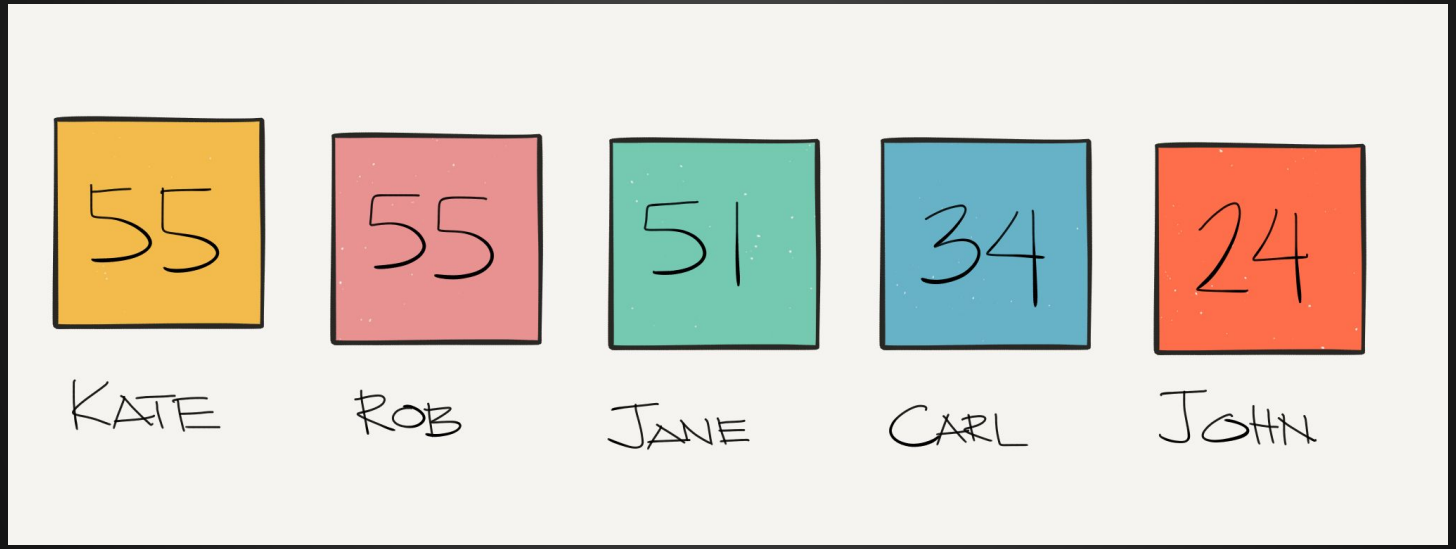
Stable vs Non-Stable

- Stable sorts leave equal values in original order.
- Non-stable sorts can not guarantee original order of equal values

High scores

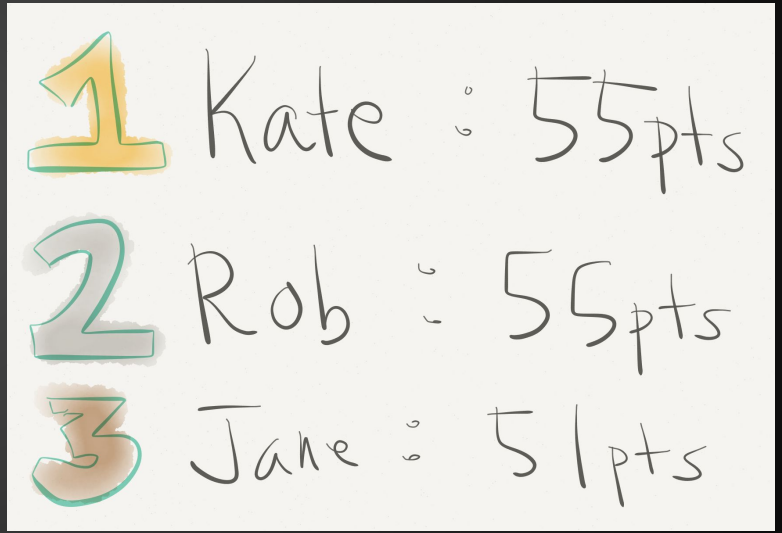


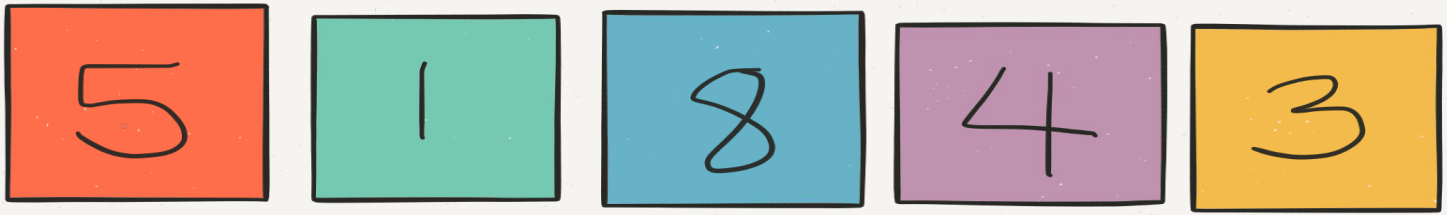
High scores - sorted



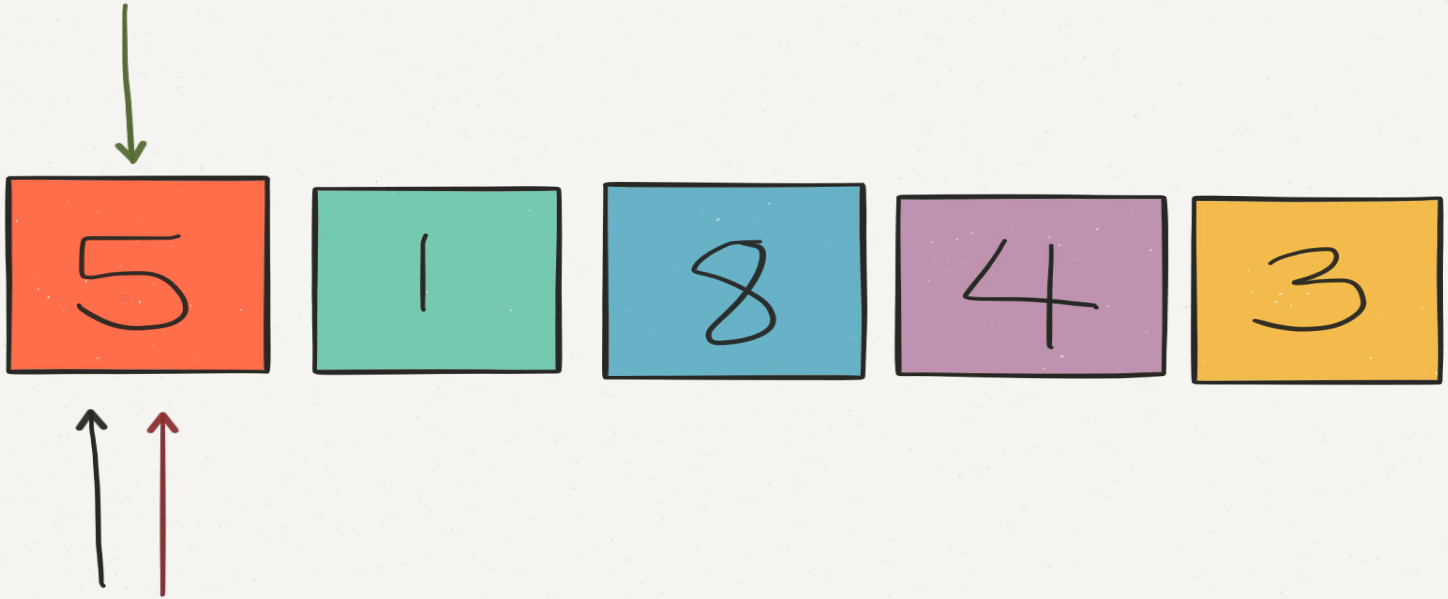
When would we need a stable sort?

- Tiebreakers
  - Last, First Name
- Maintain order
  - top scores

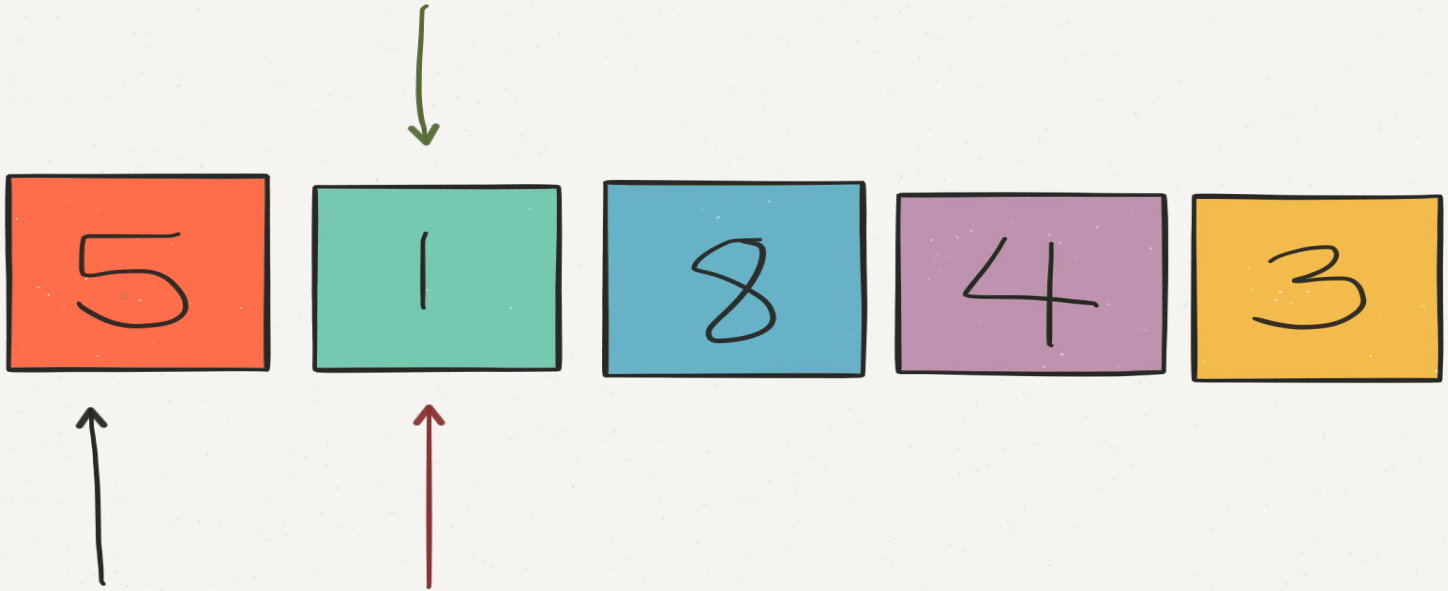


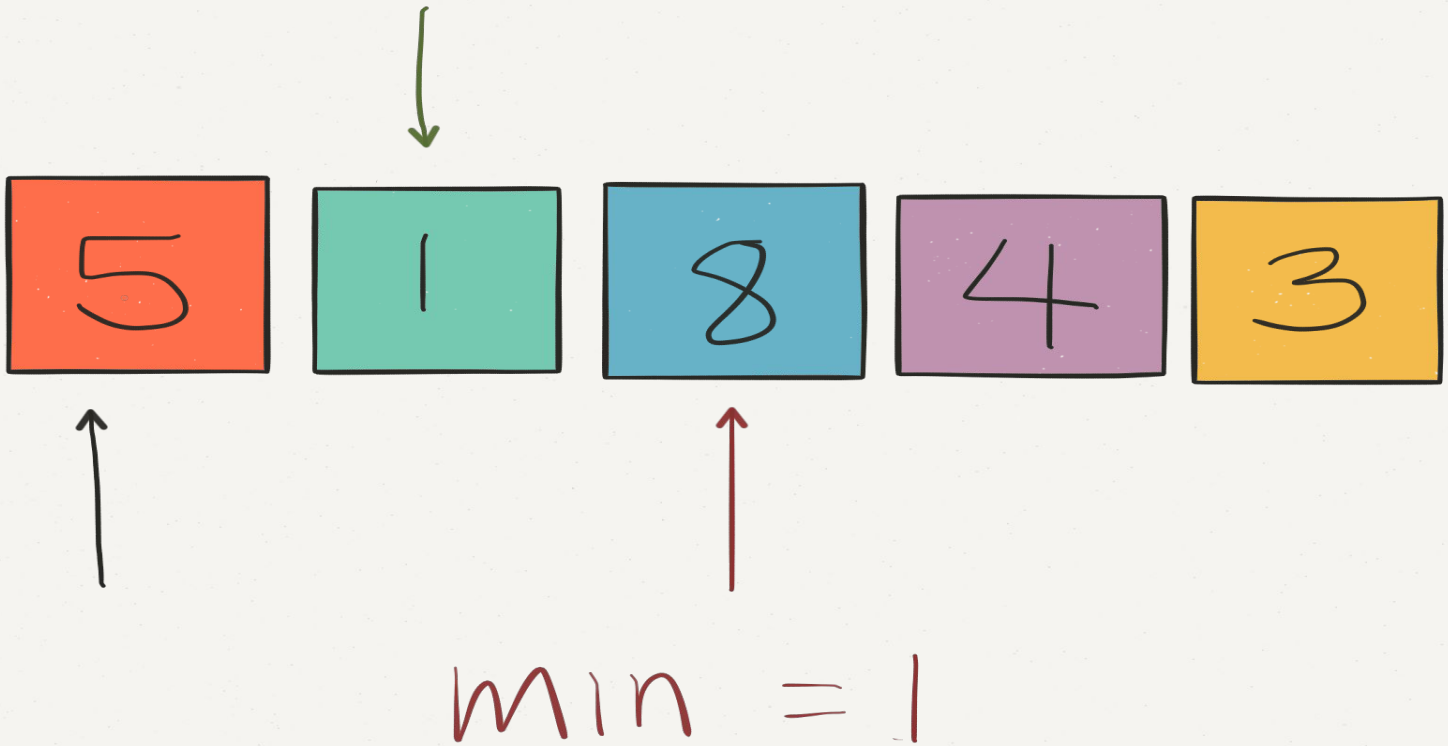


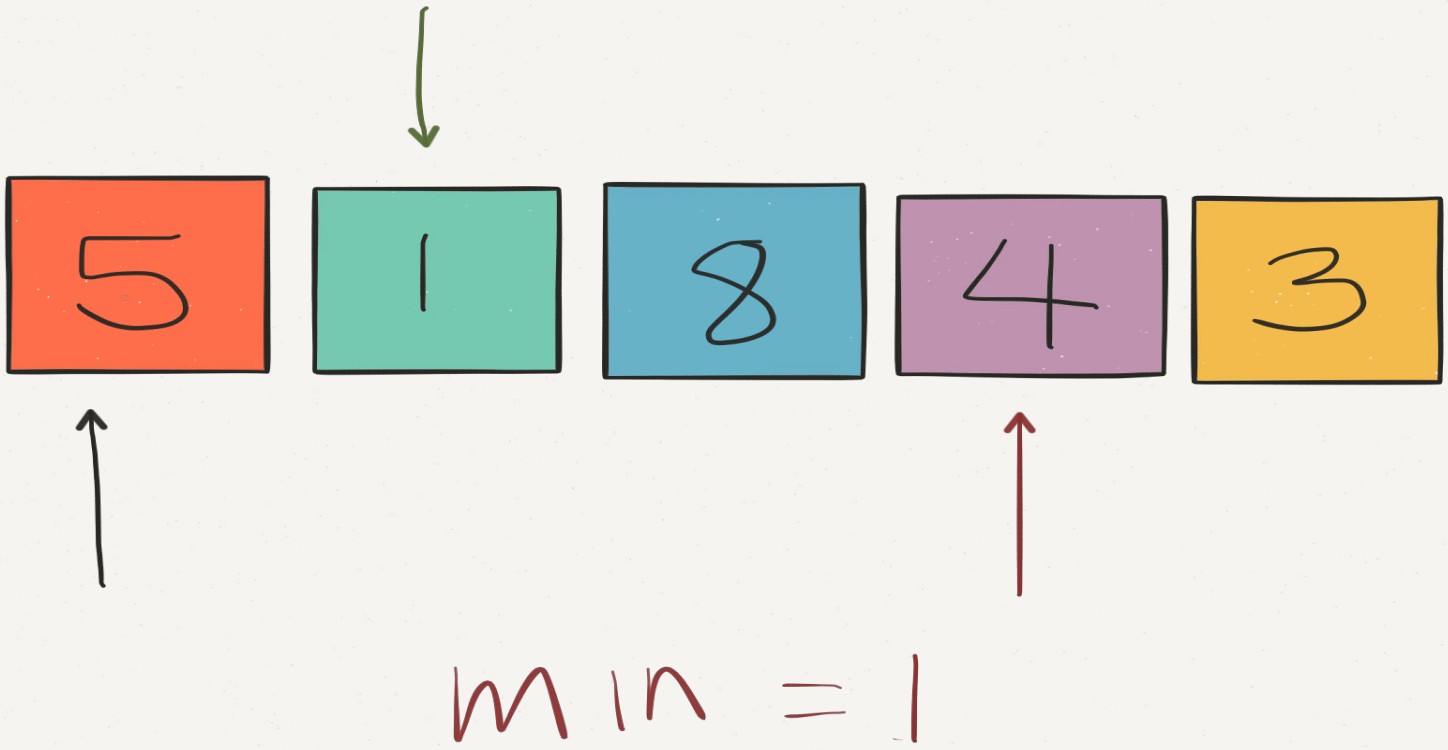
SELECTION

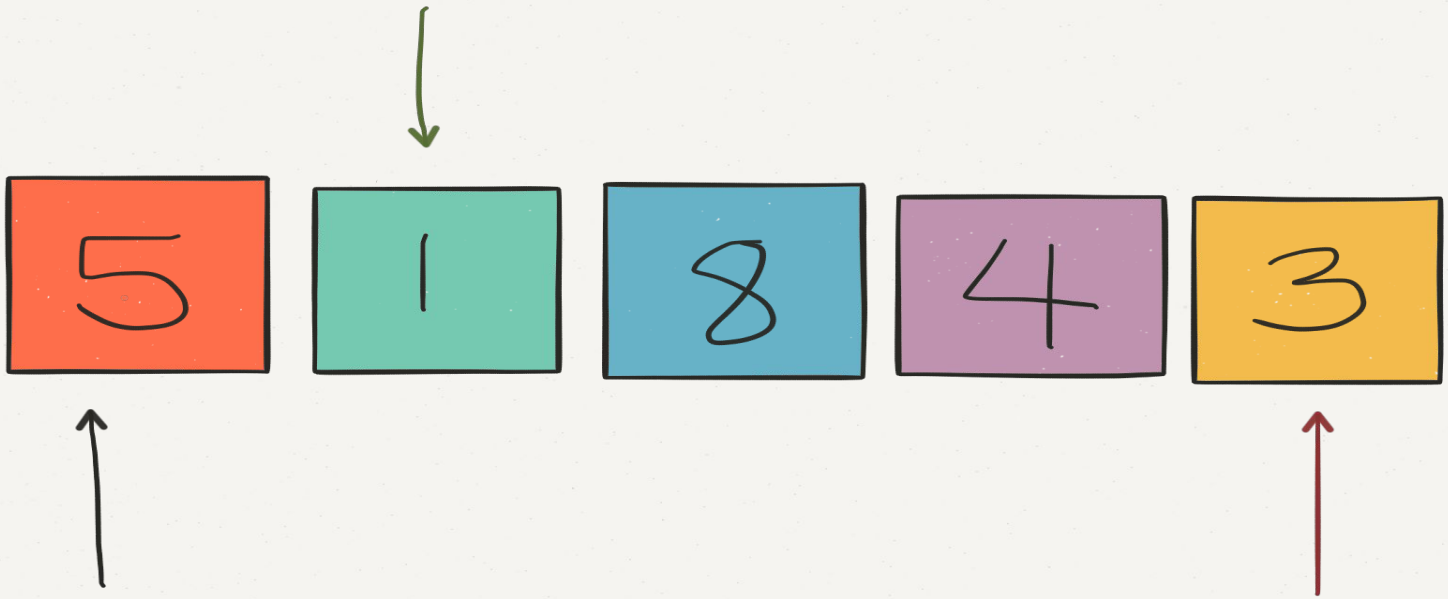


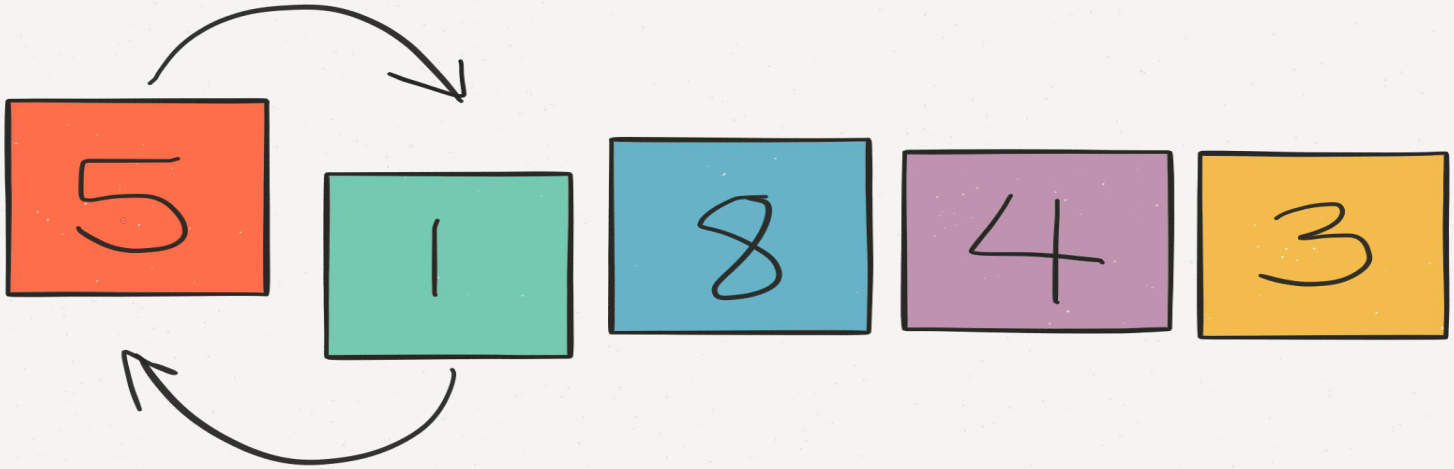
min = 5



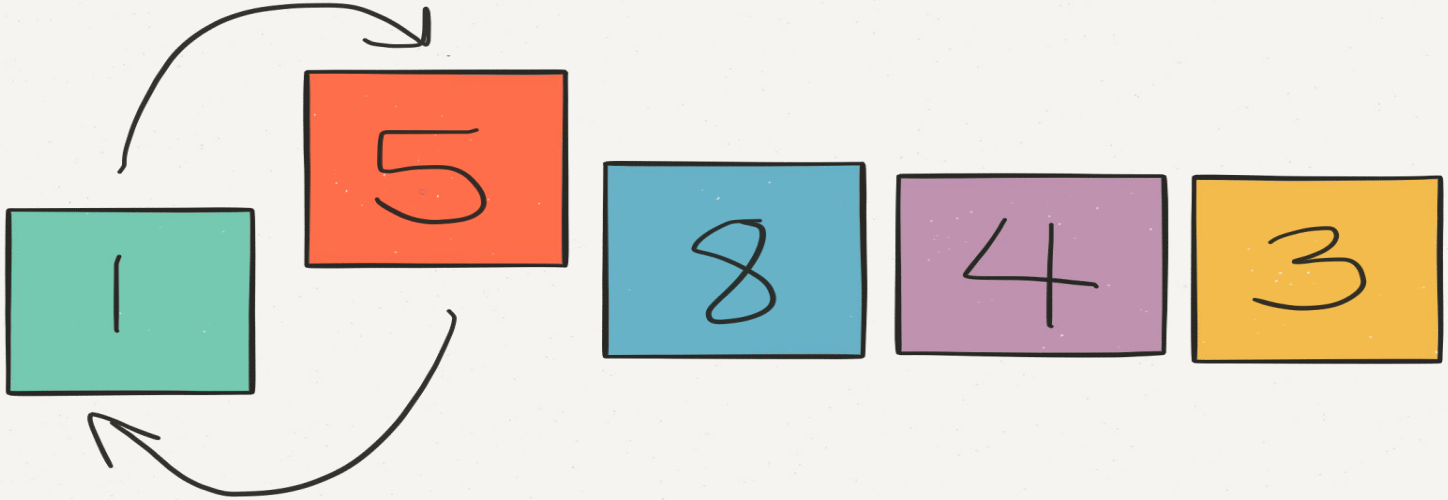



5
1
8
4
3
min = 1

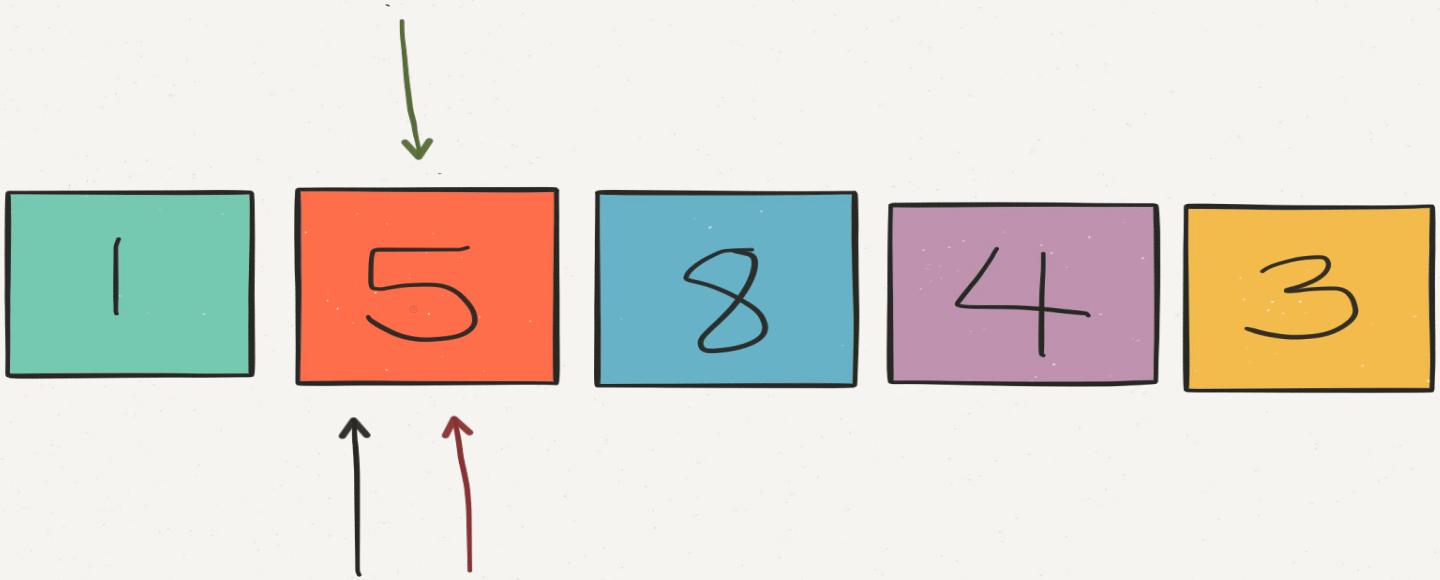




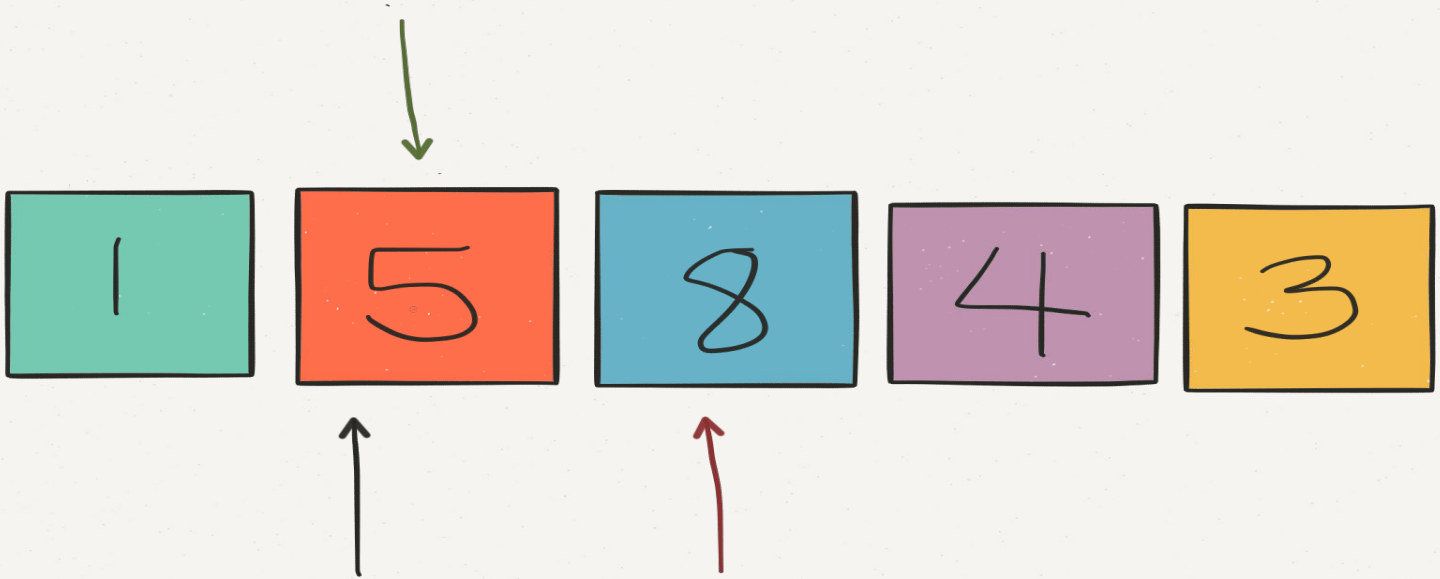
min = 1

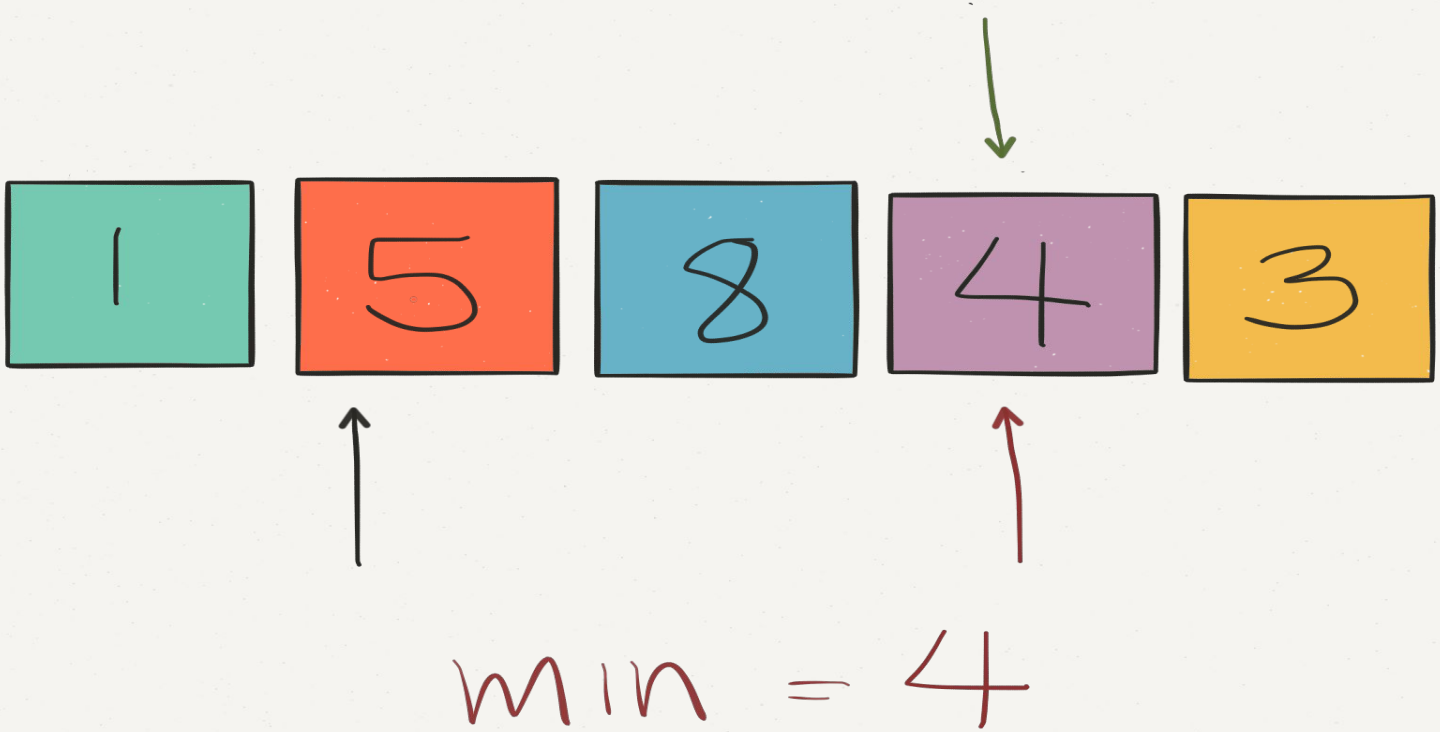


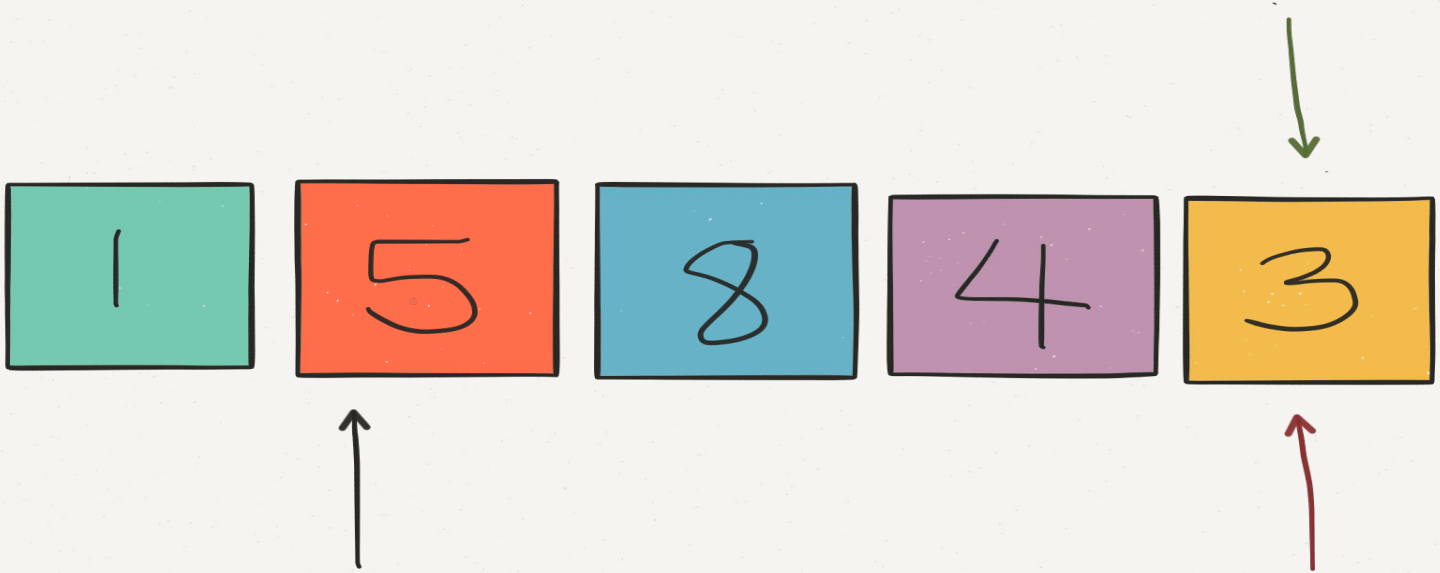
min = 1

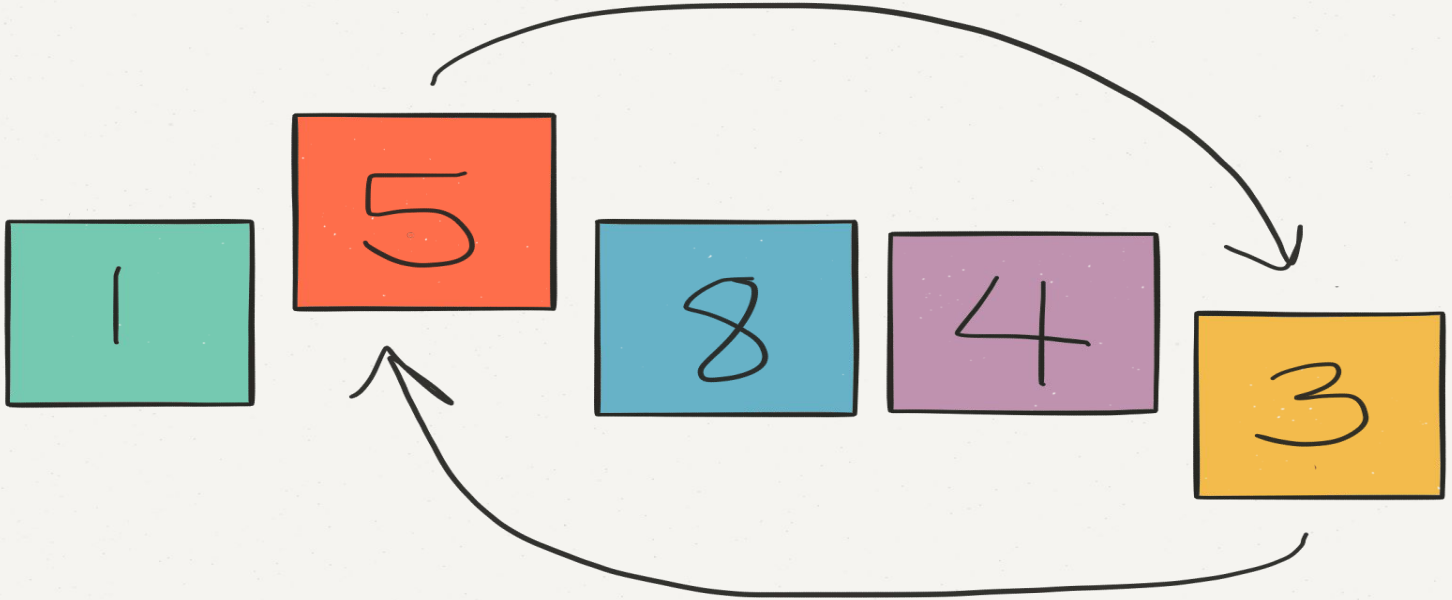


min = 5

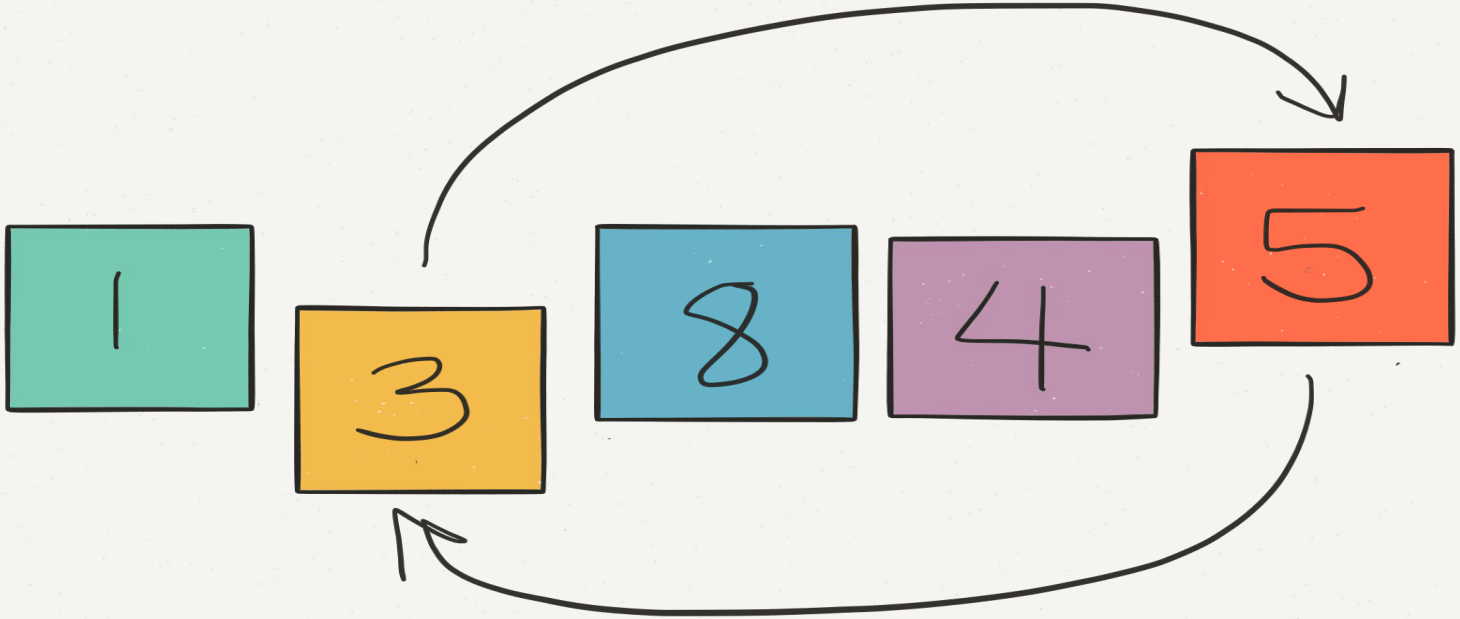




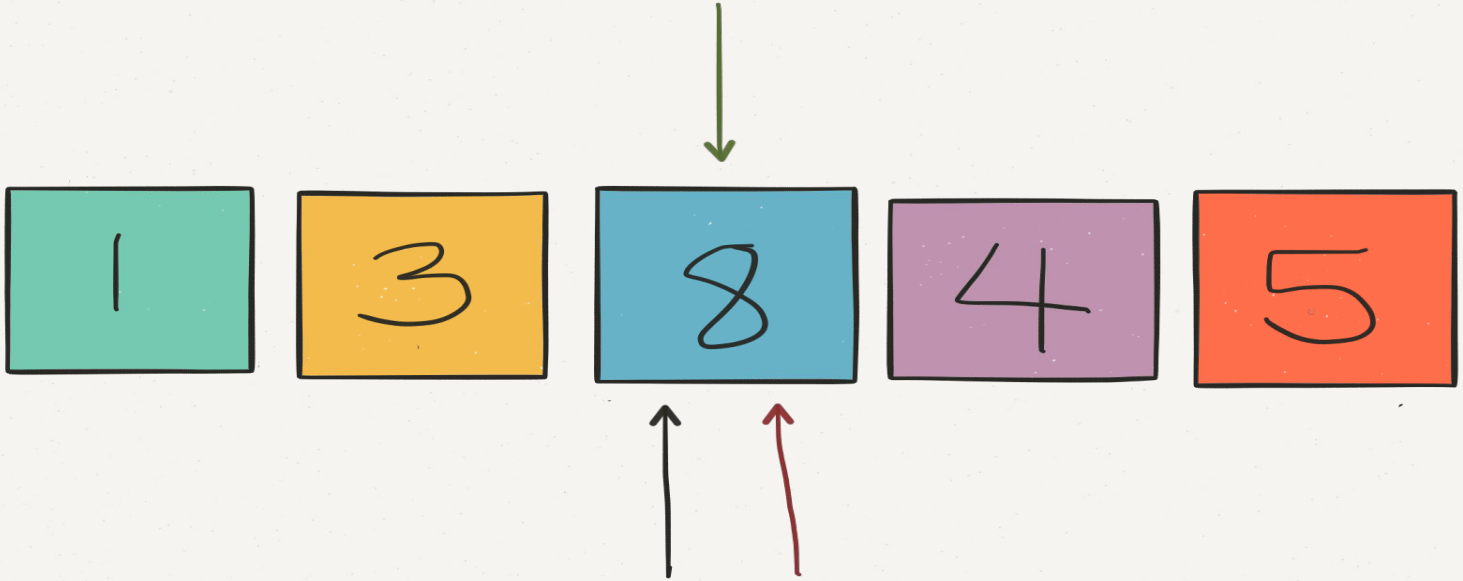


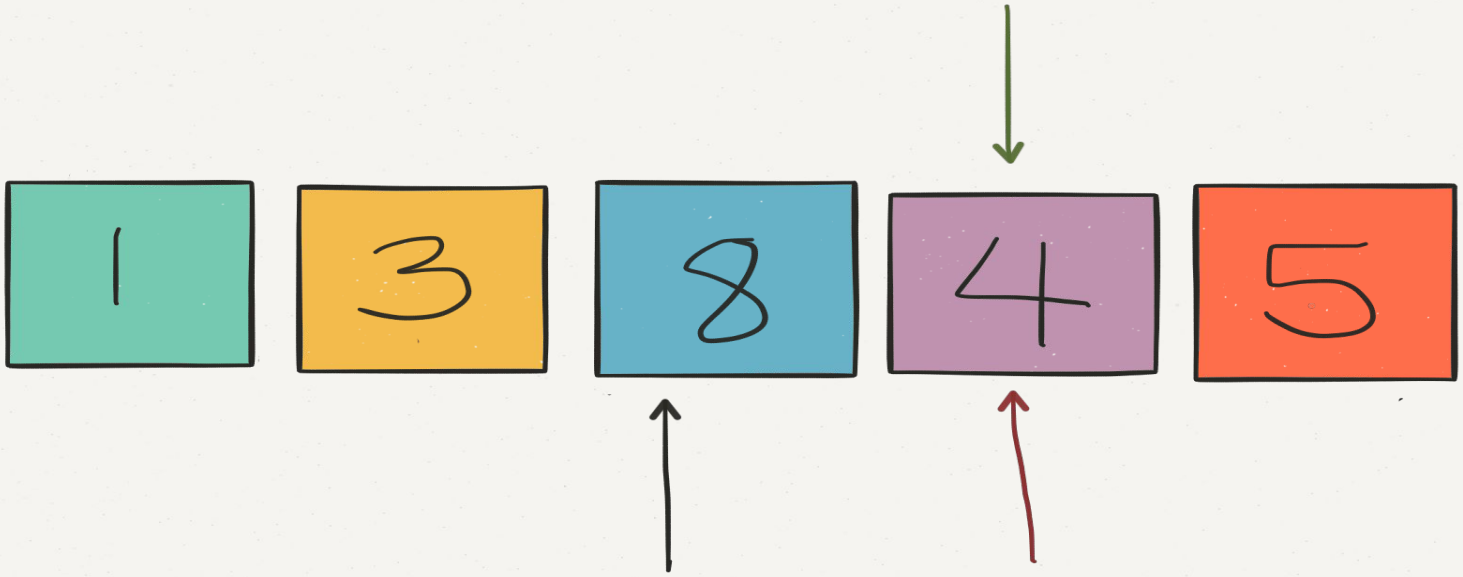


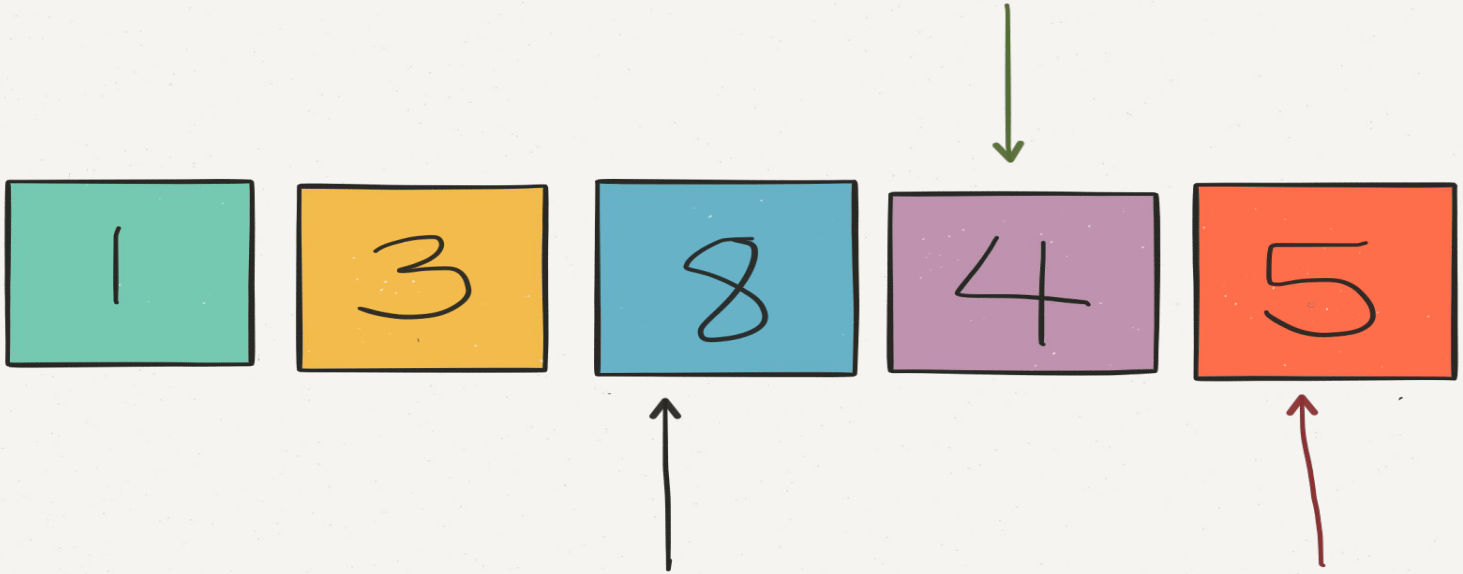
min = 3



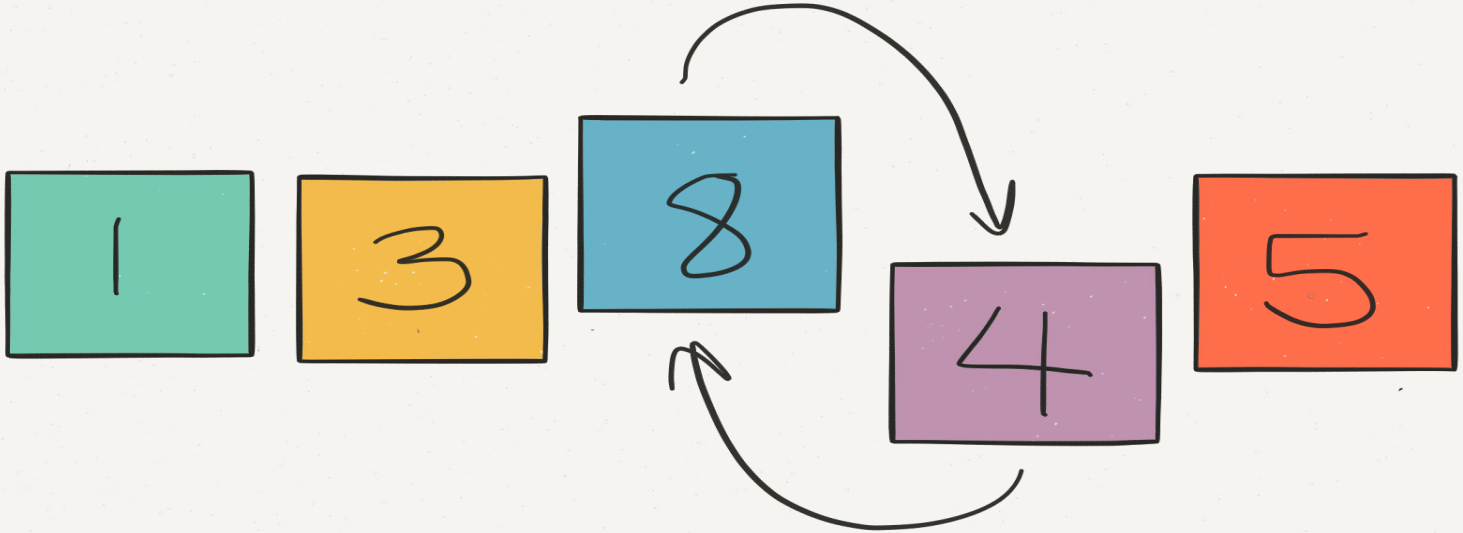
min = 3



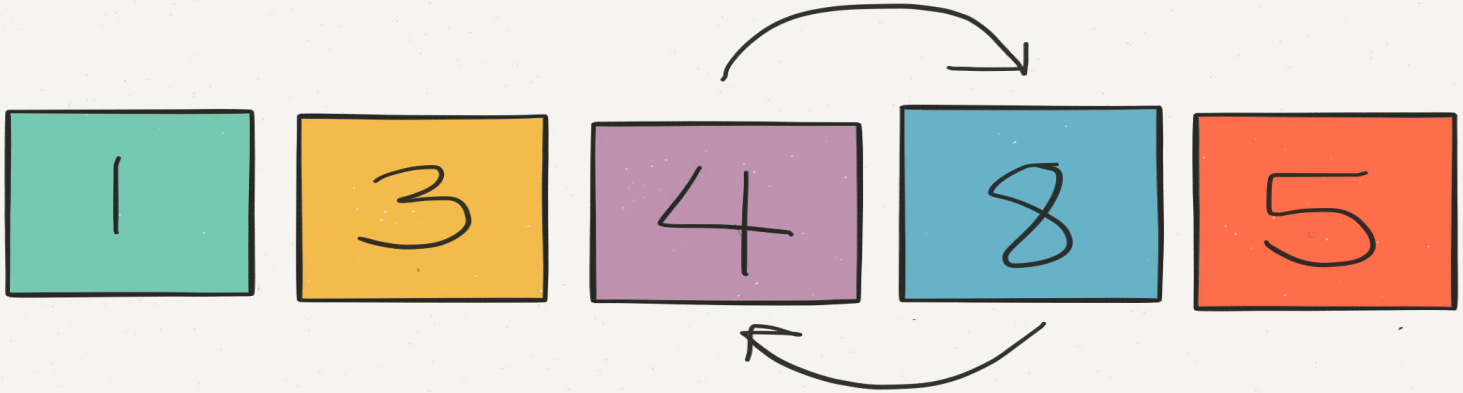




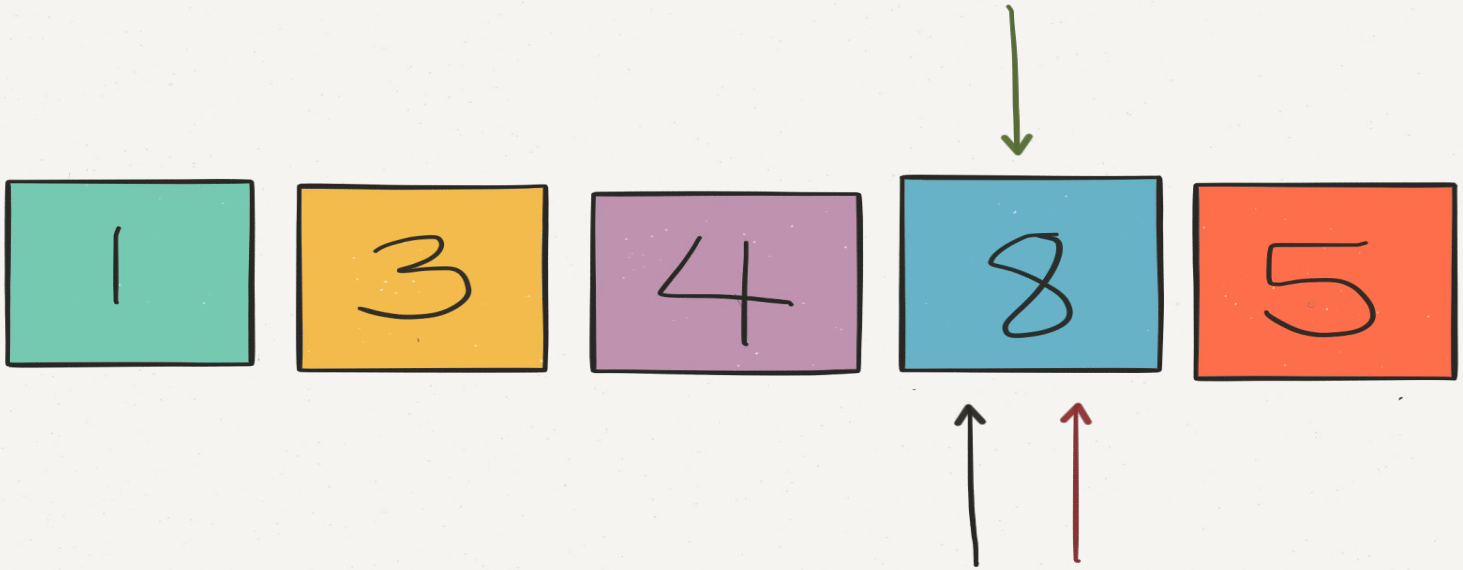
min = 4



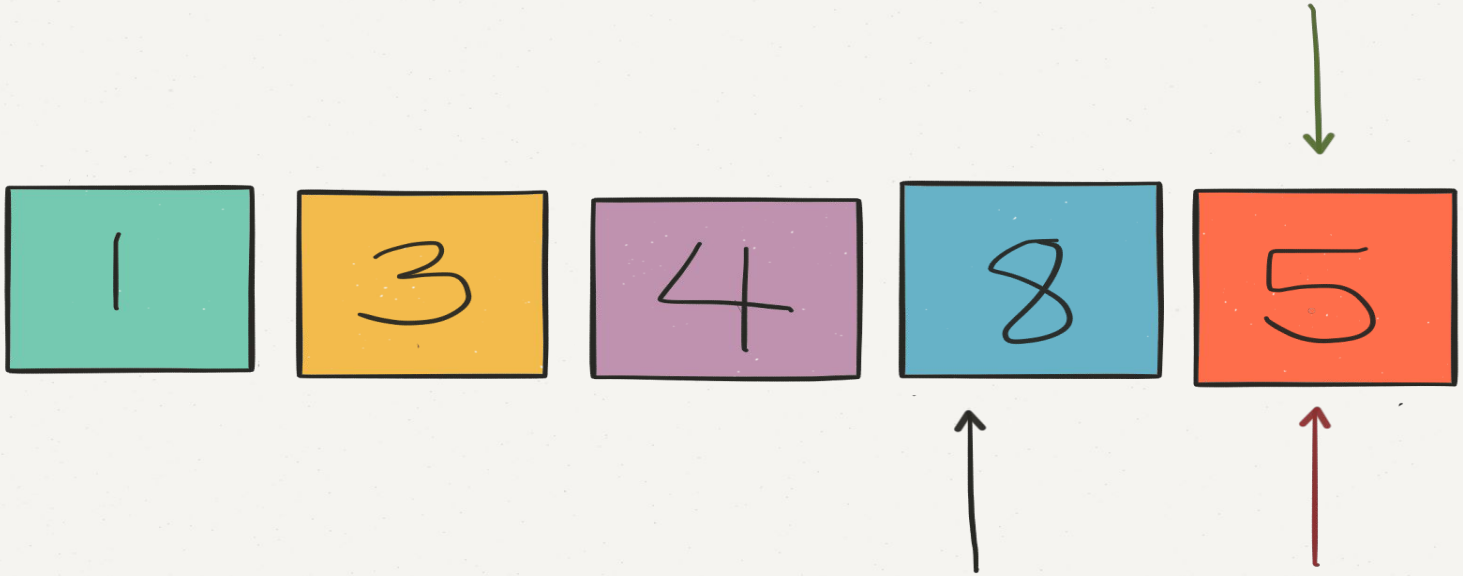
min = 4

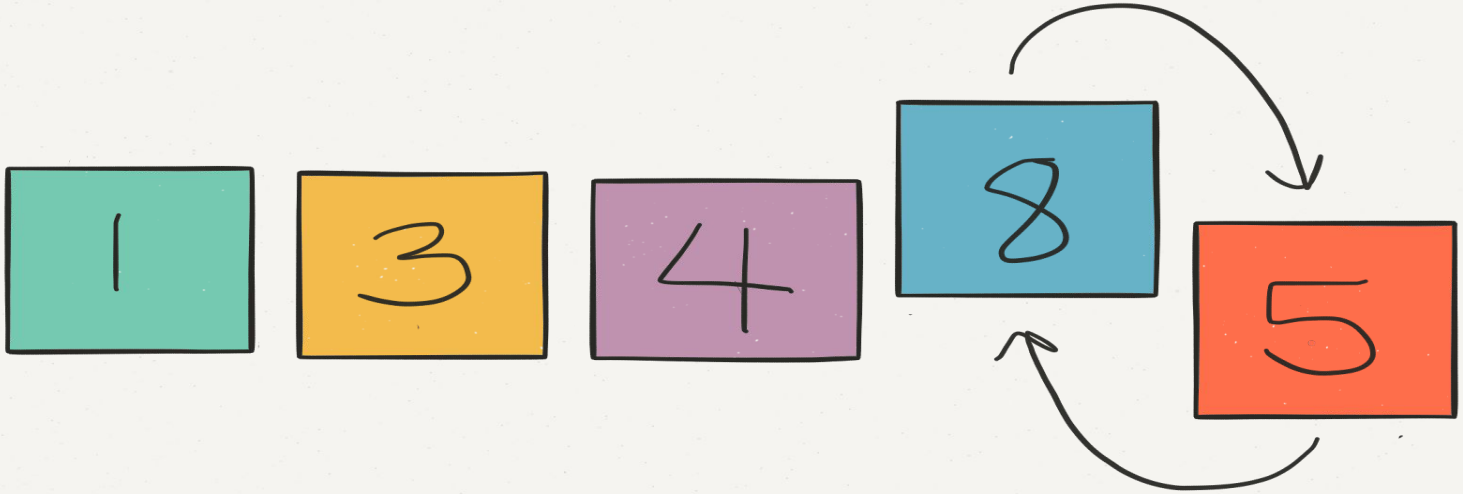


min = 4

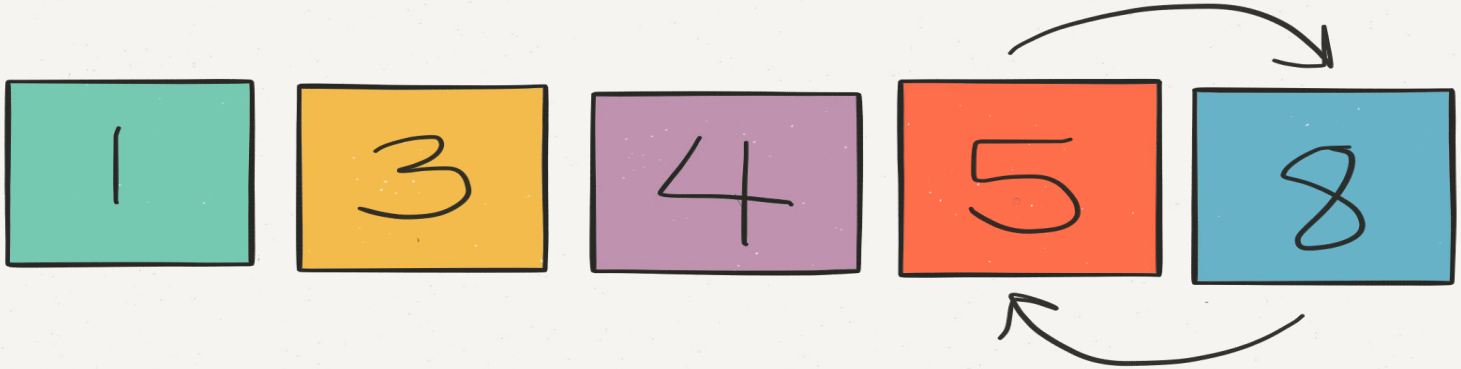


min = 8

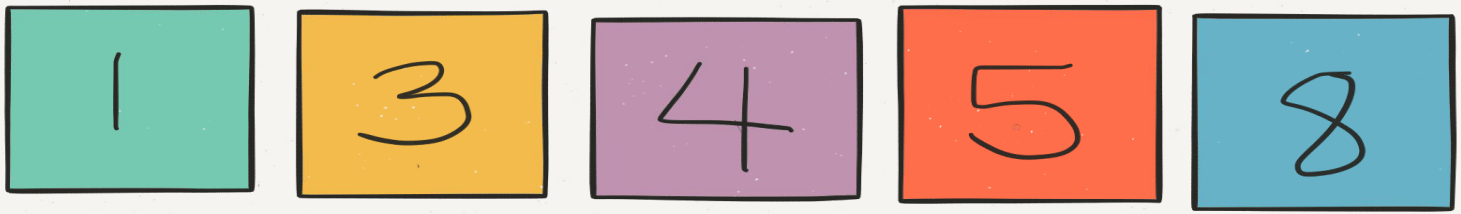

1
3
4
8
5
min = 5



min = 5

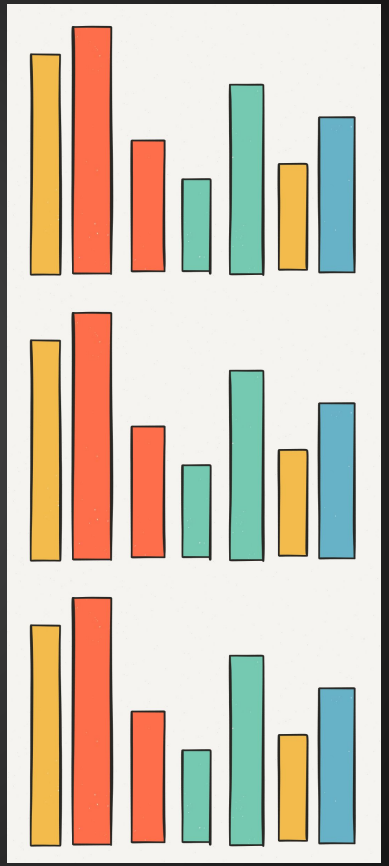


min = 5


1
3
4
5
8

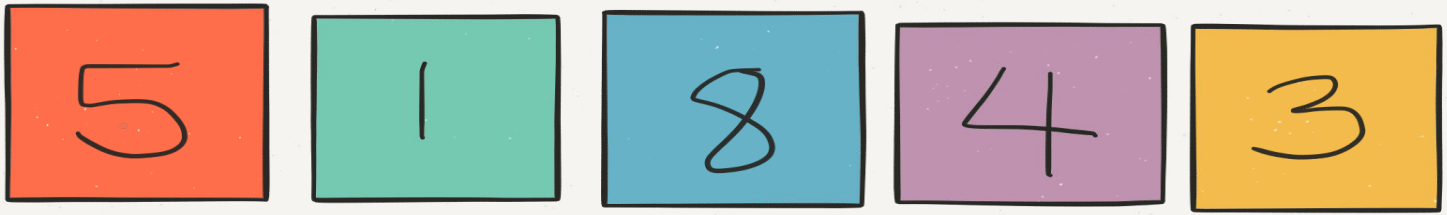
Time Complexity

- Worse case
  - $O(n^2)$
- Average Case
  - $O(n^2)$
- Best Case
  - $\Omega(n^2)$

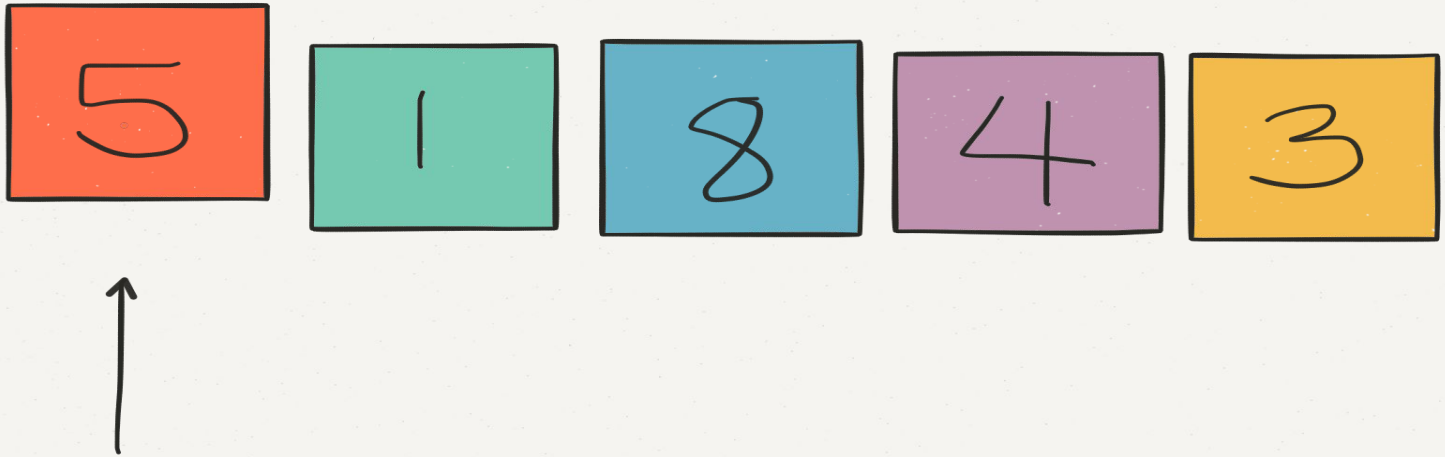


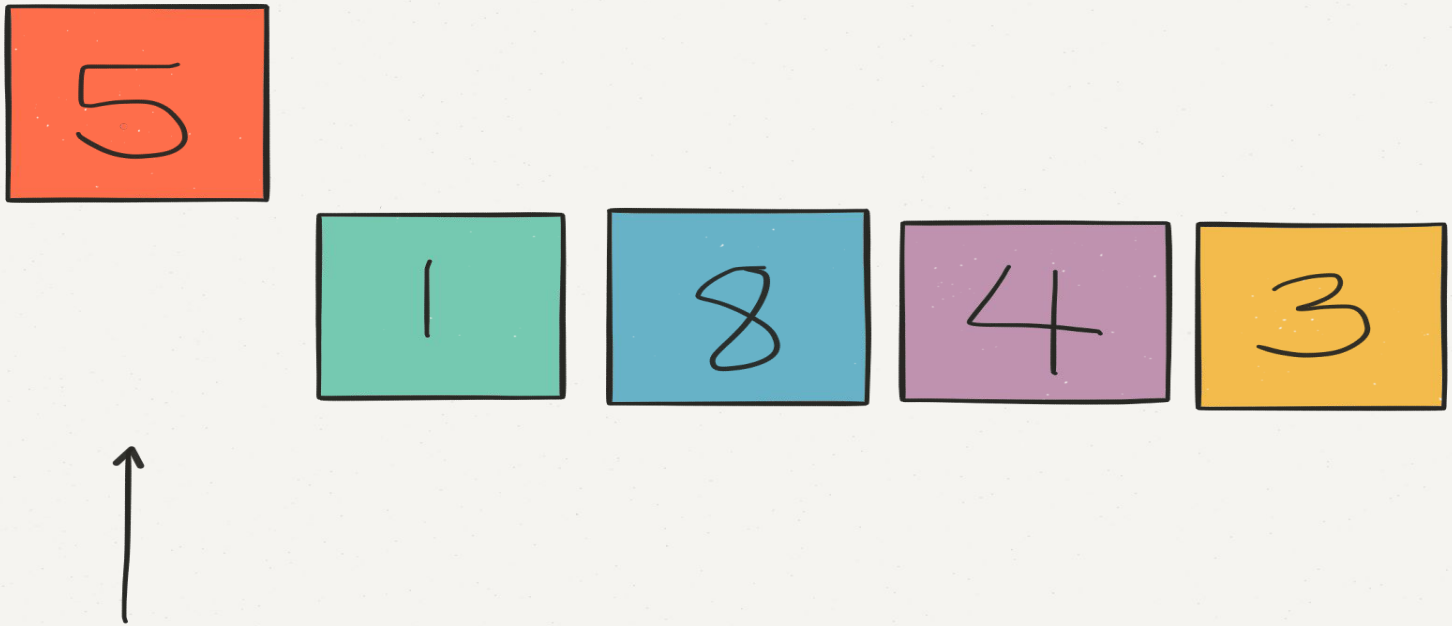
Advantages/Disadvantage

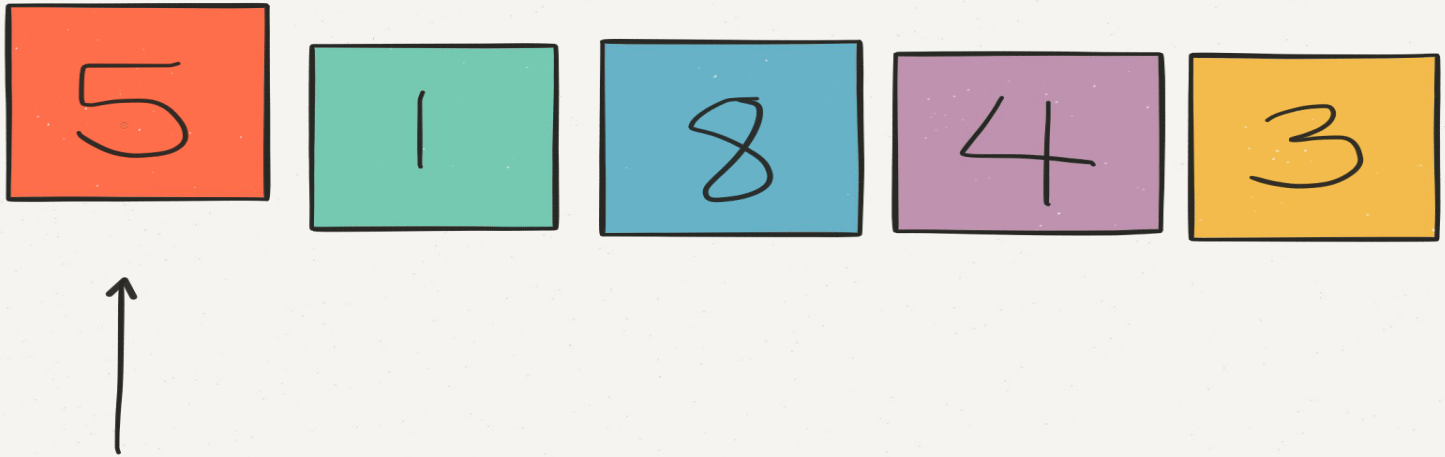
- Simple to understand
- Constant auxiliary space
- Quadratic time at best
- Not stable

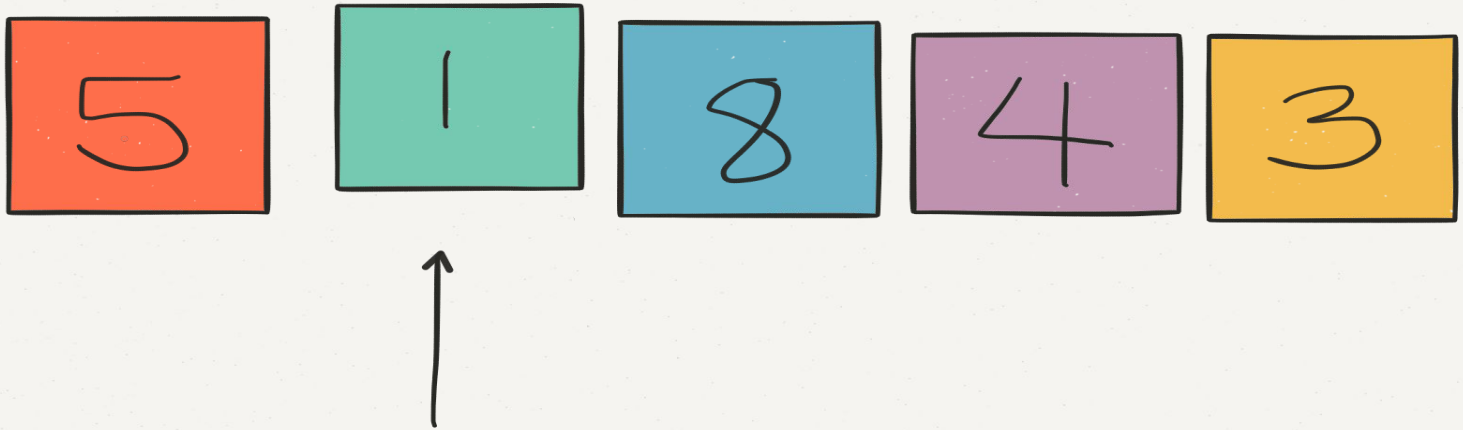


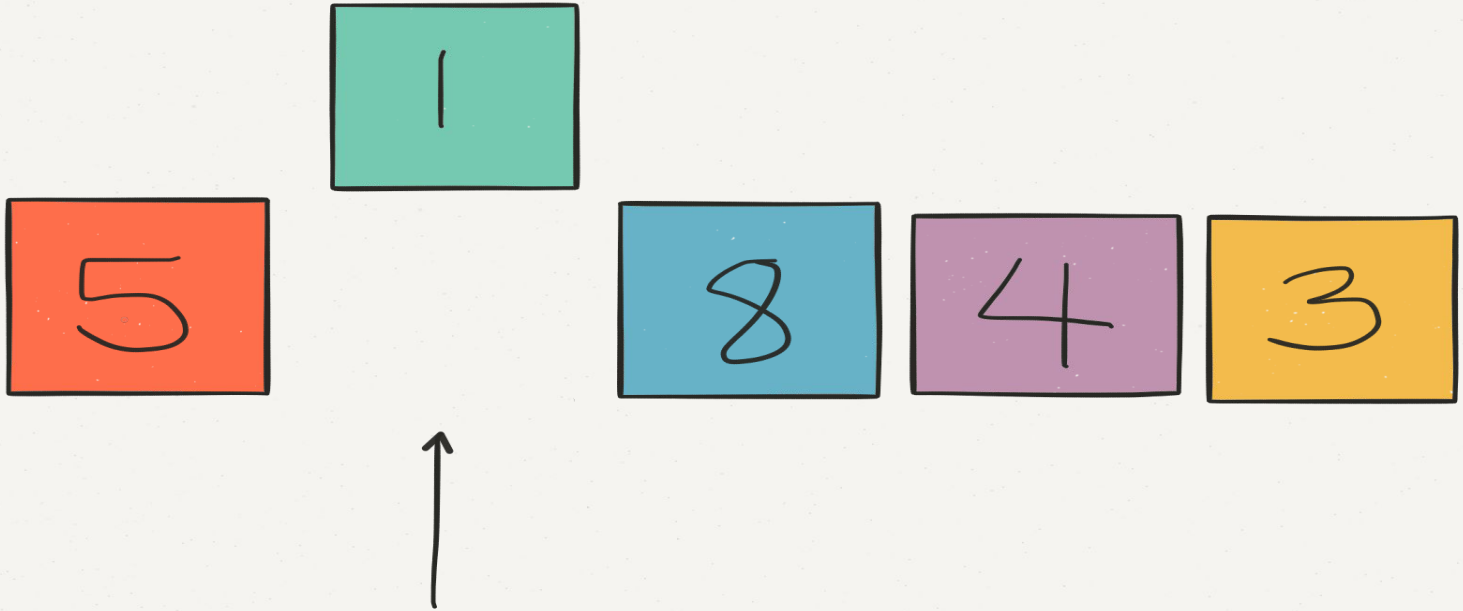
INSERTION

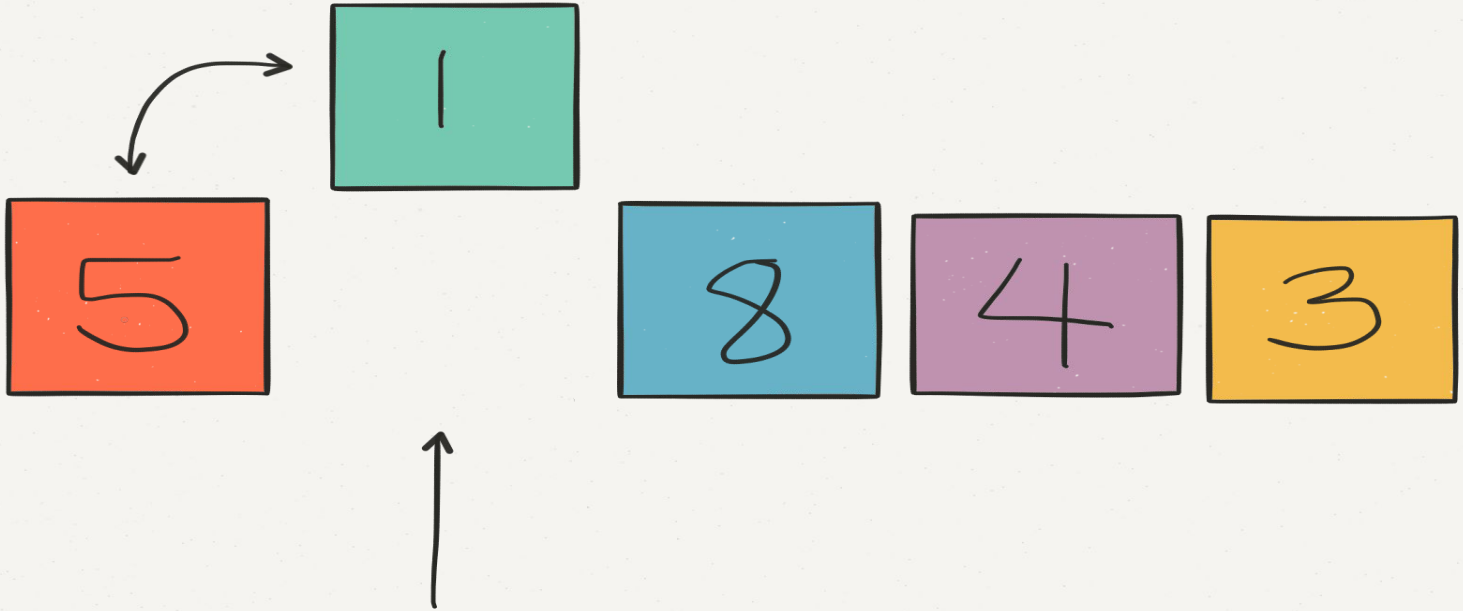

5
1
8
4
3

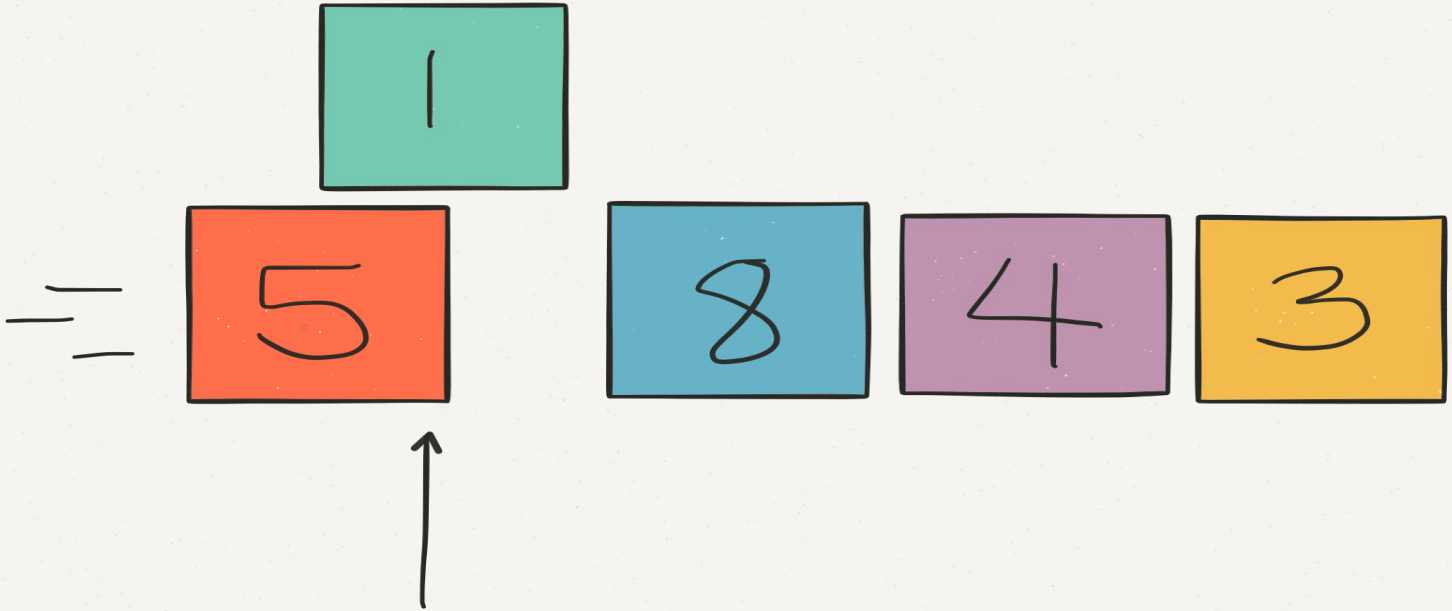

5
1
8
4
3

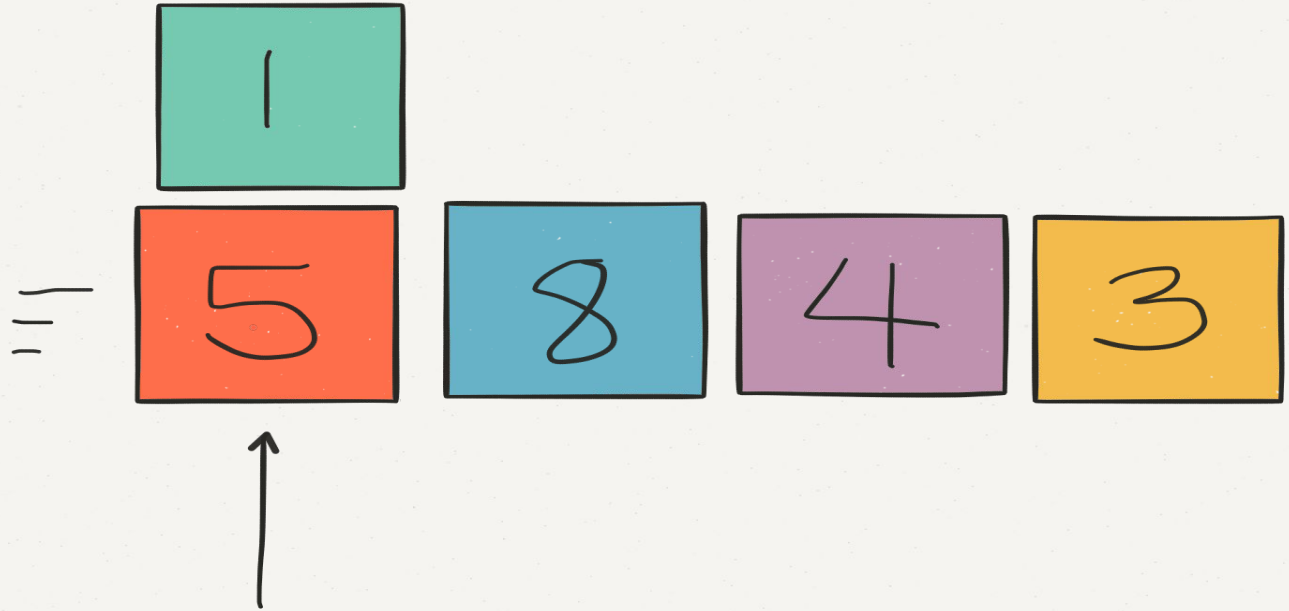

5
1
8
4
3

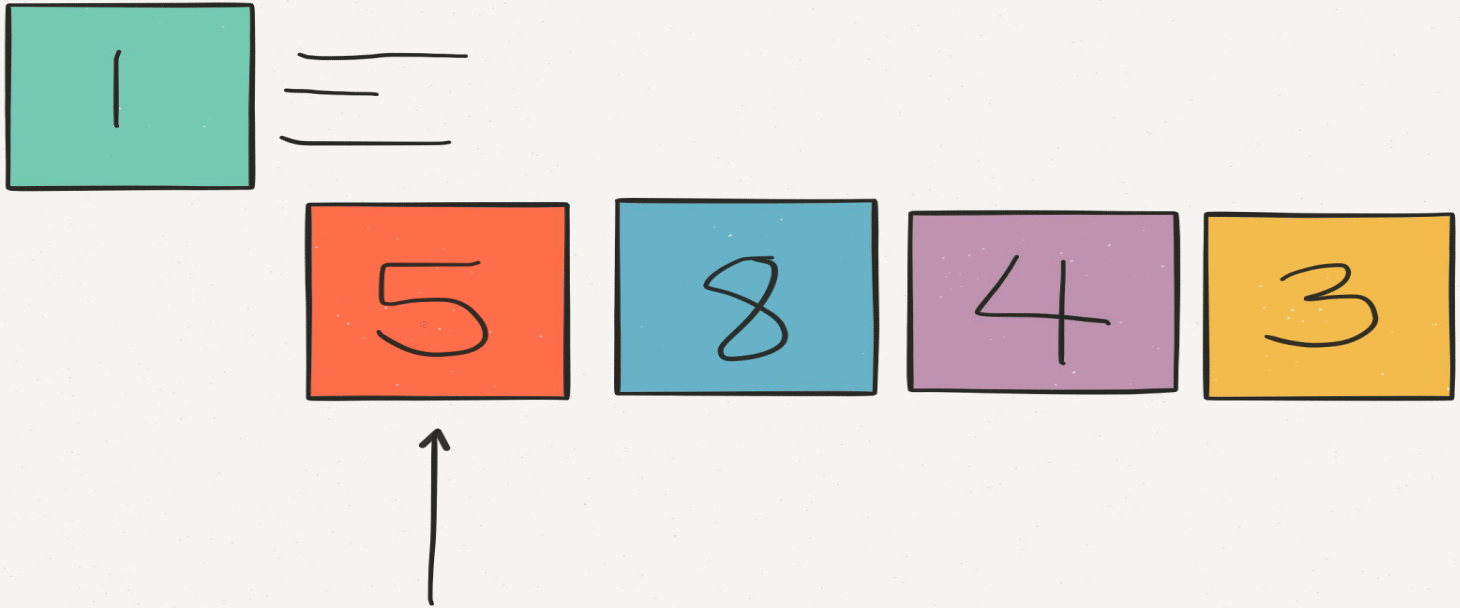

5
1
8
4
3

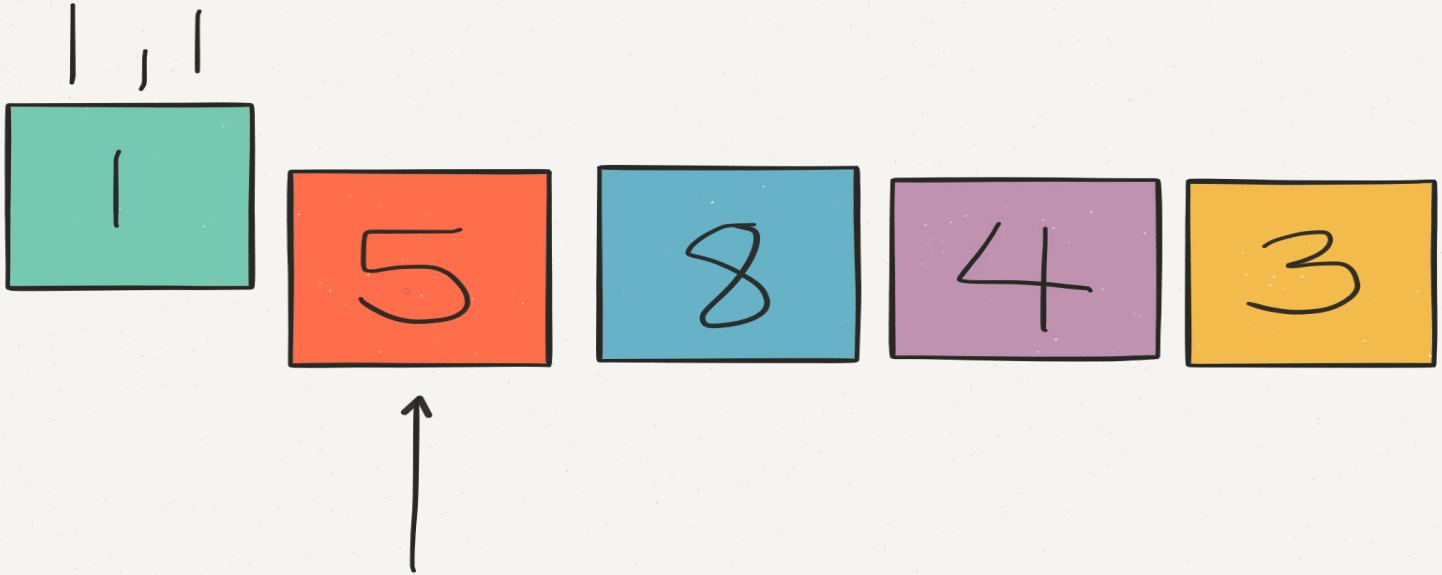

1
5
8
4
3

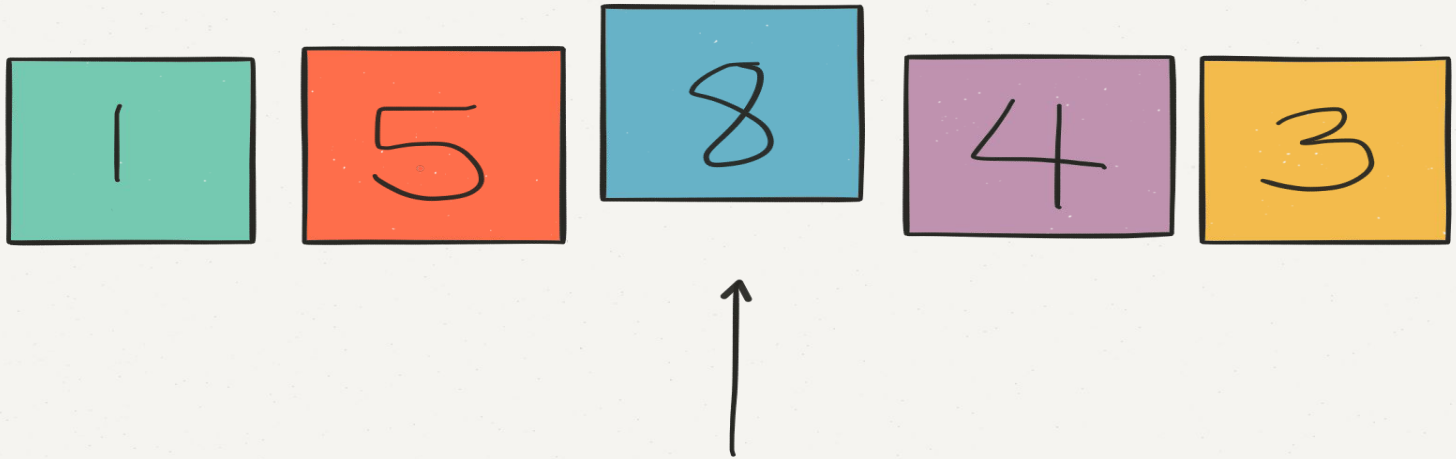

1
5
8
4
3

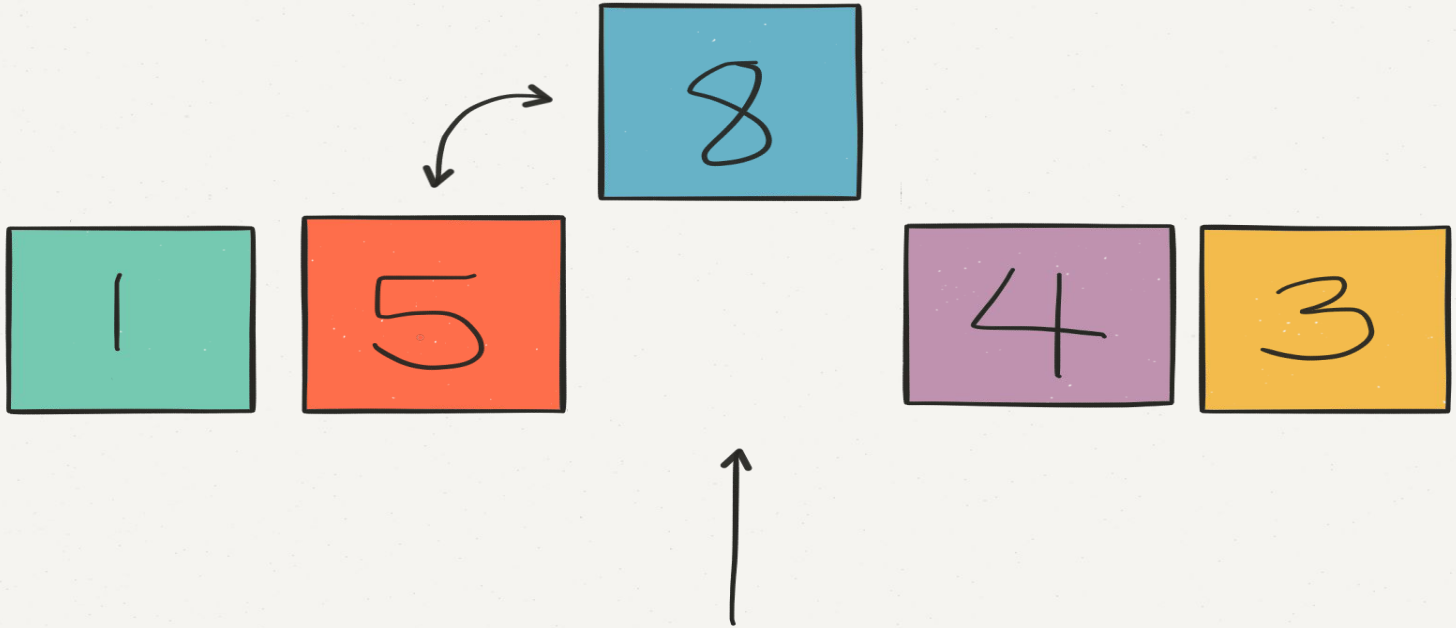

1
5
8
4
3

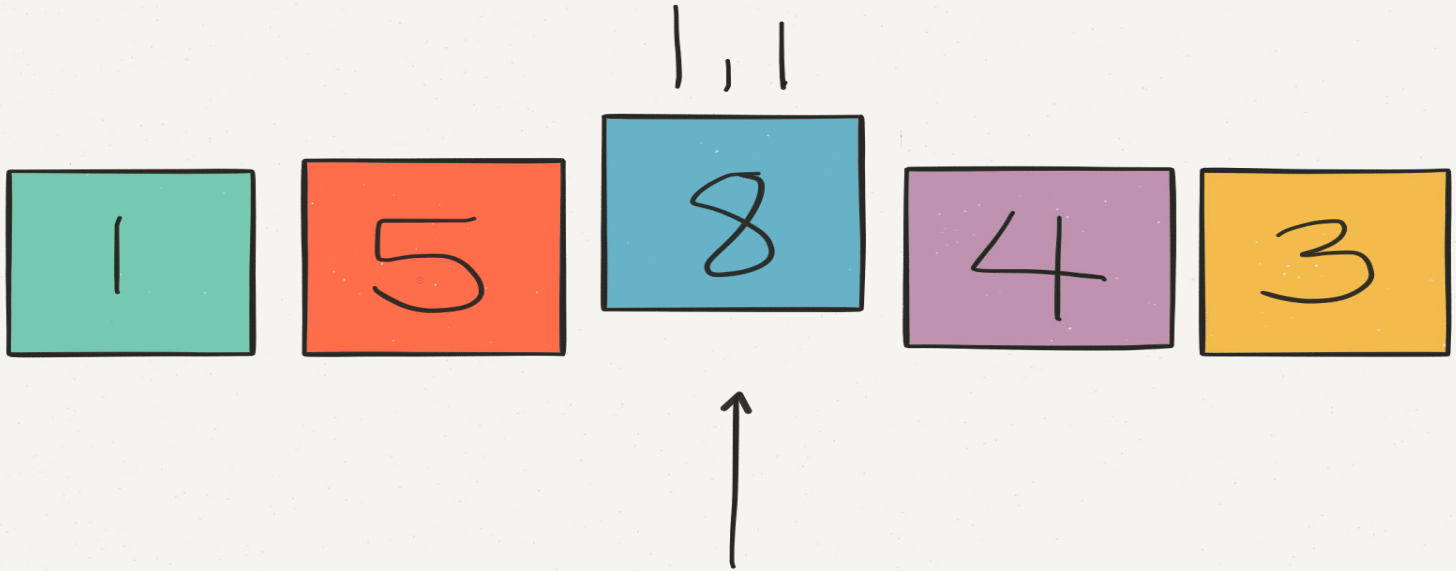

1
5
8
4
3

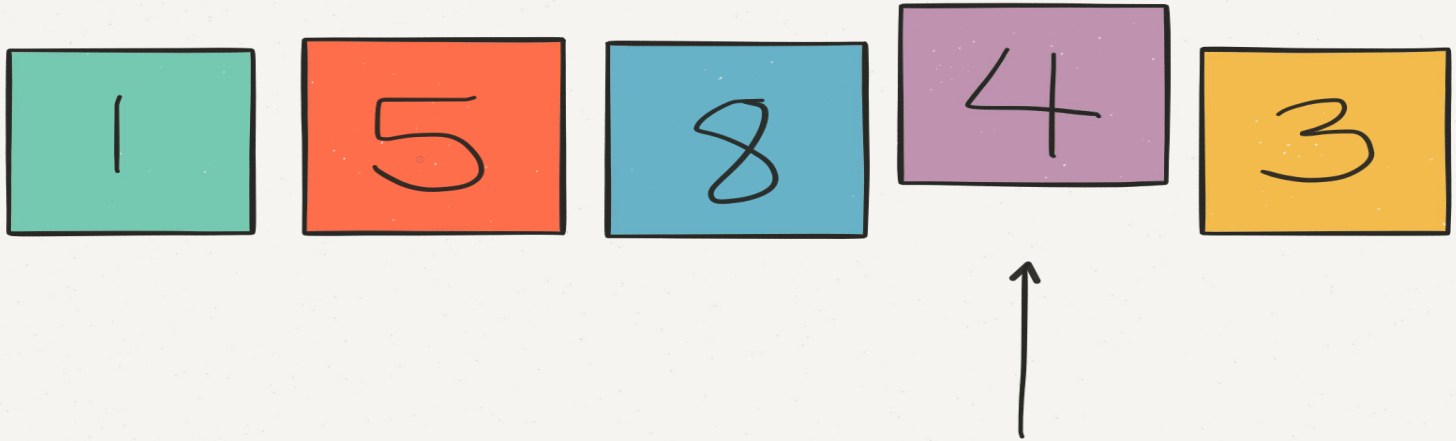

1
5
8
4
3

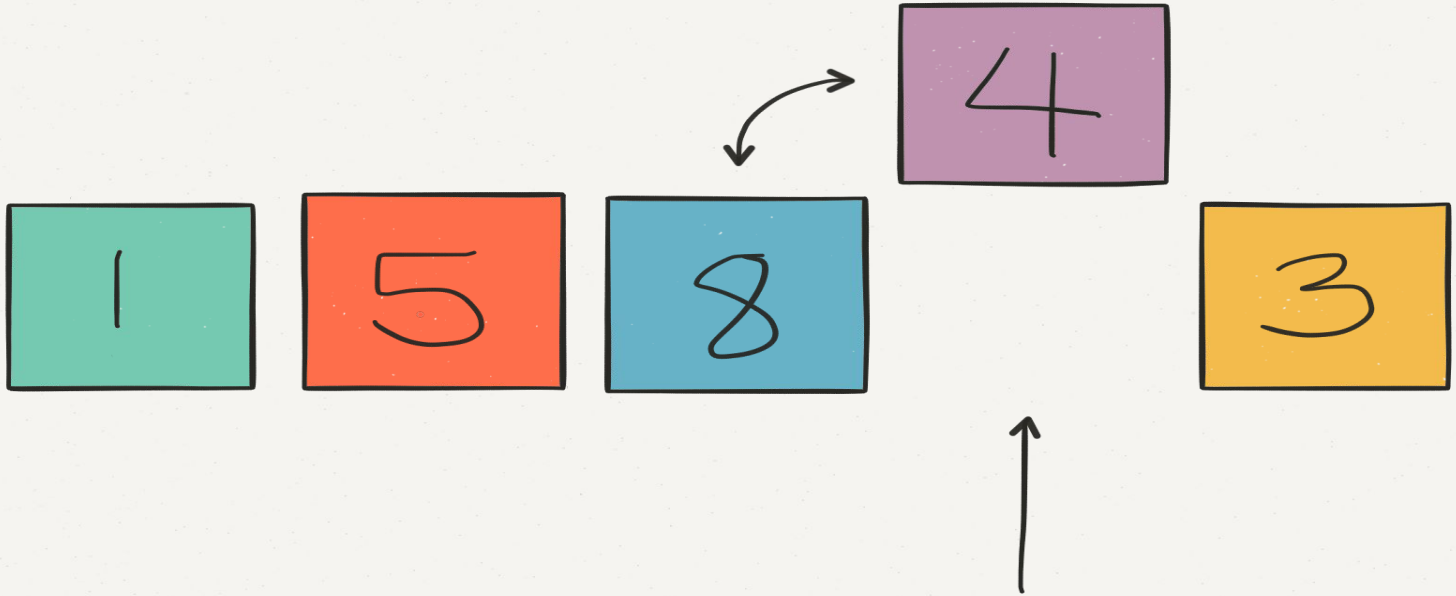


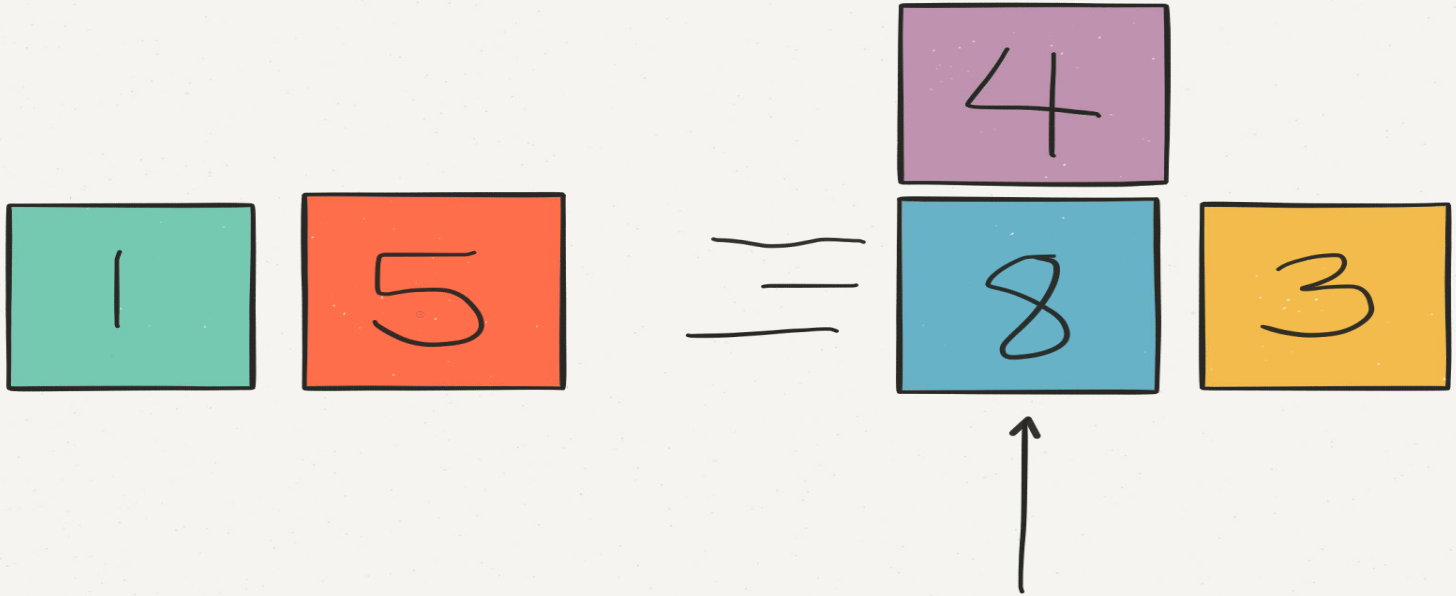

1
5
8
4
3

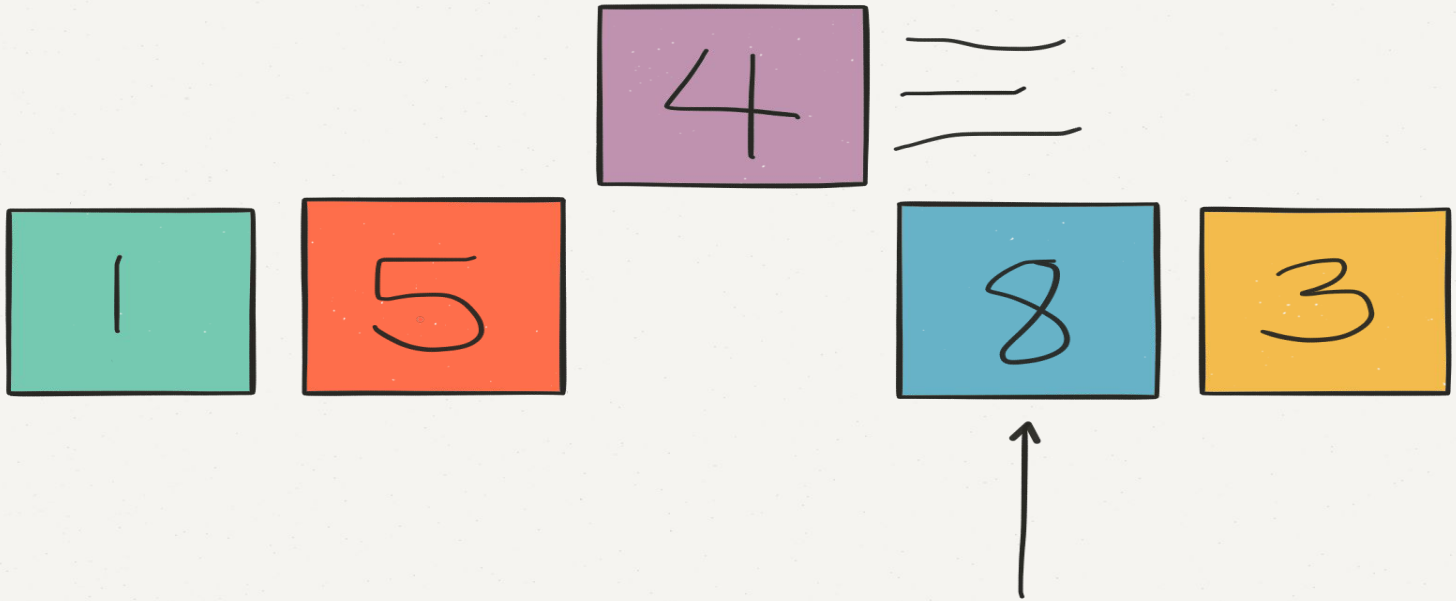

8
1
5
4
3

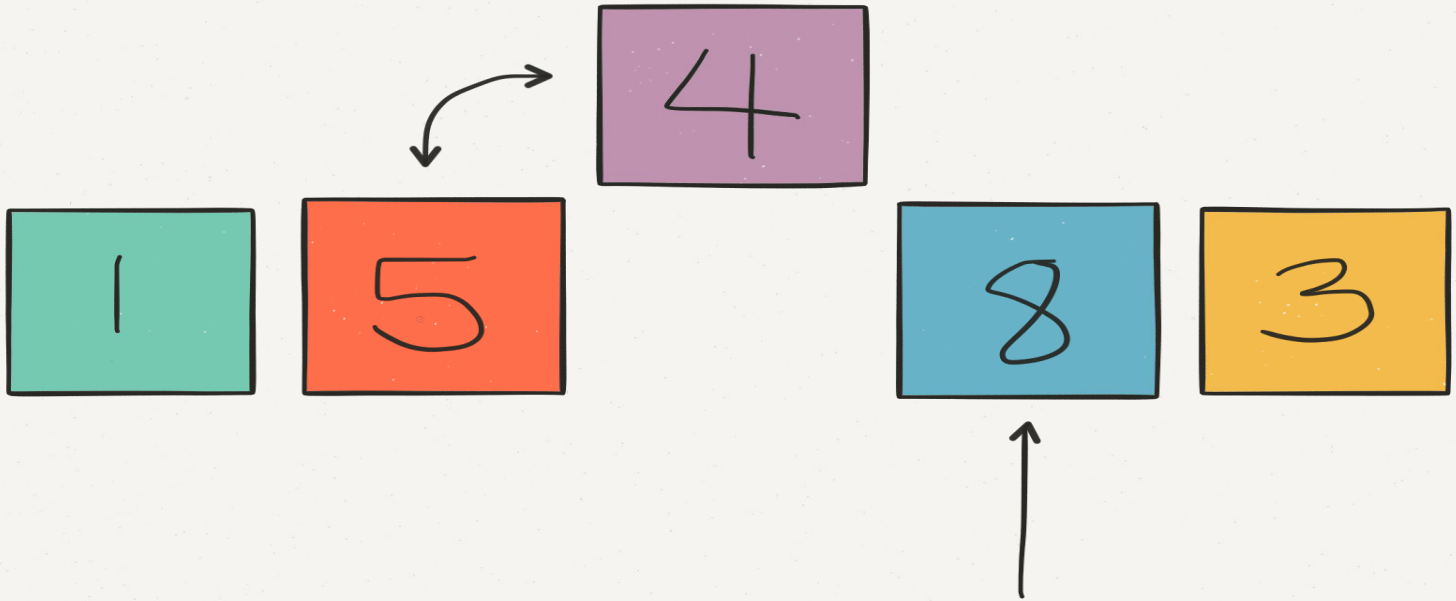

1, 1
1
5
8
4
3

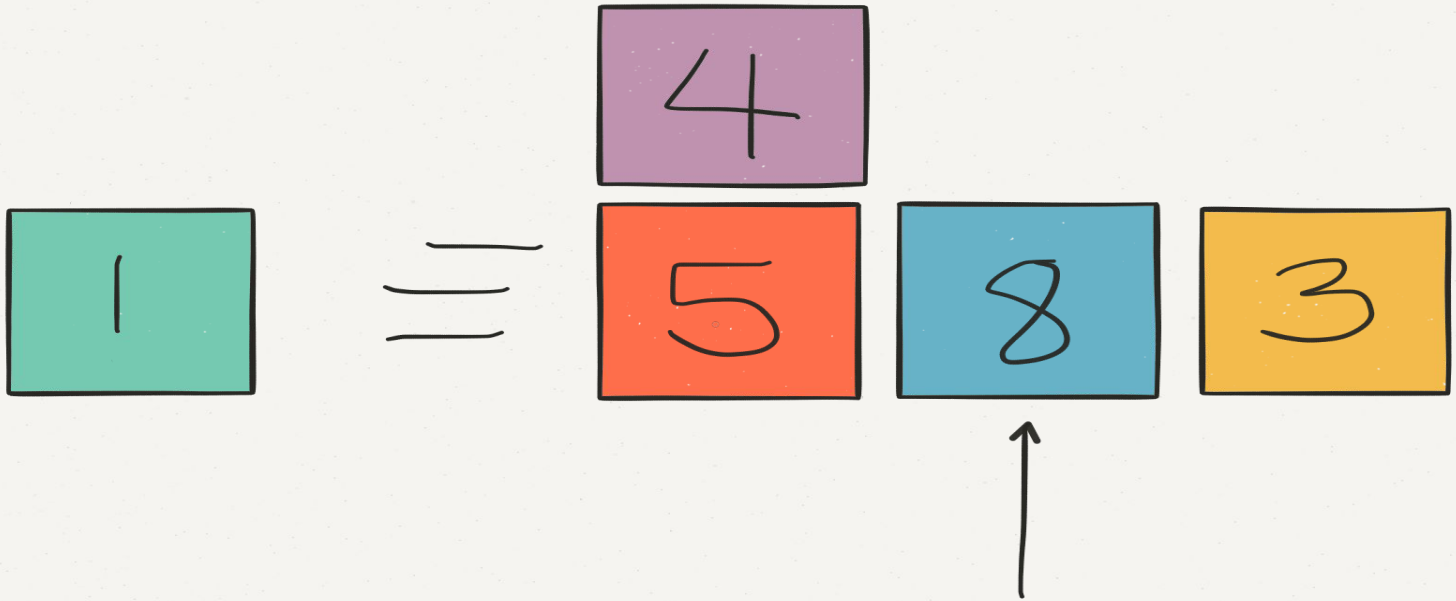


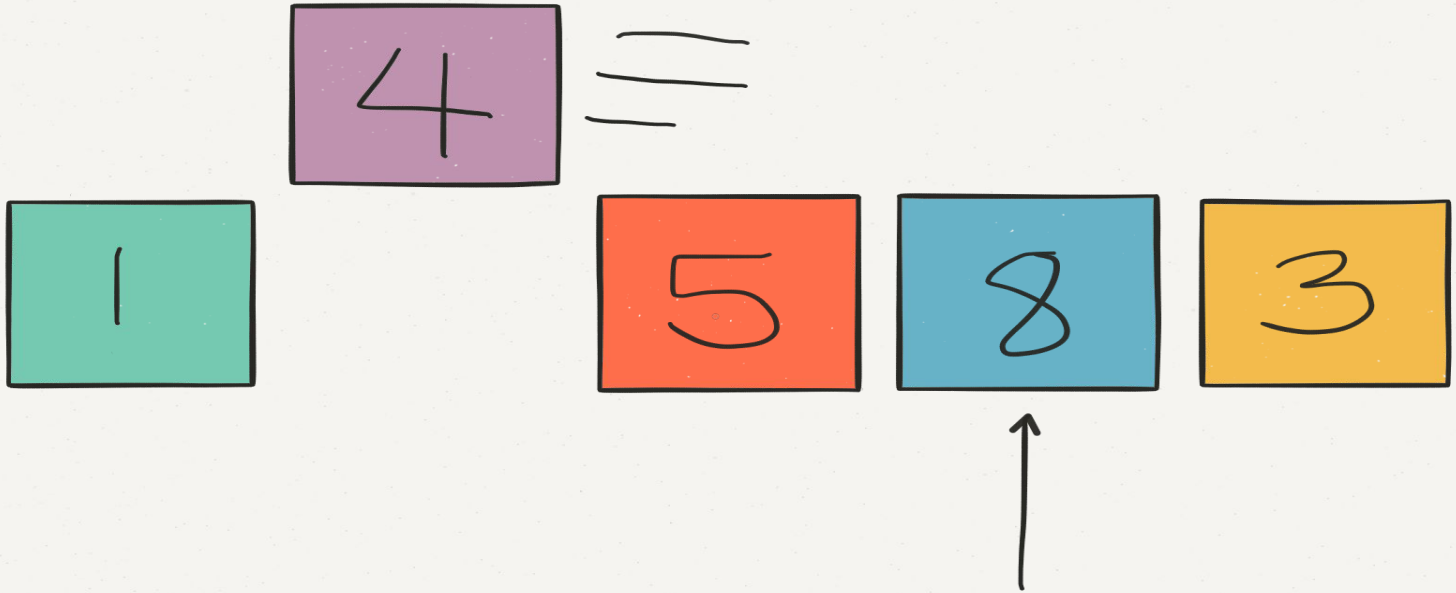

4
1
5
8
3

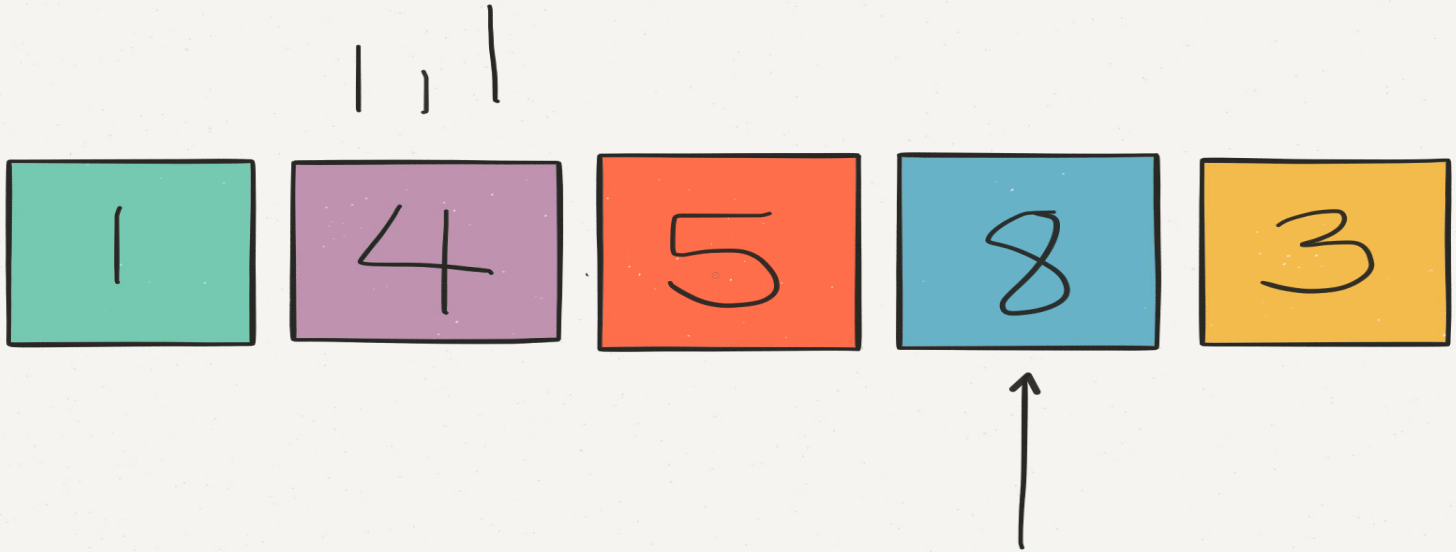

1
5
4
8
3

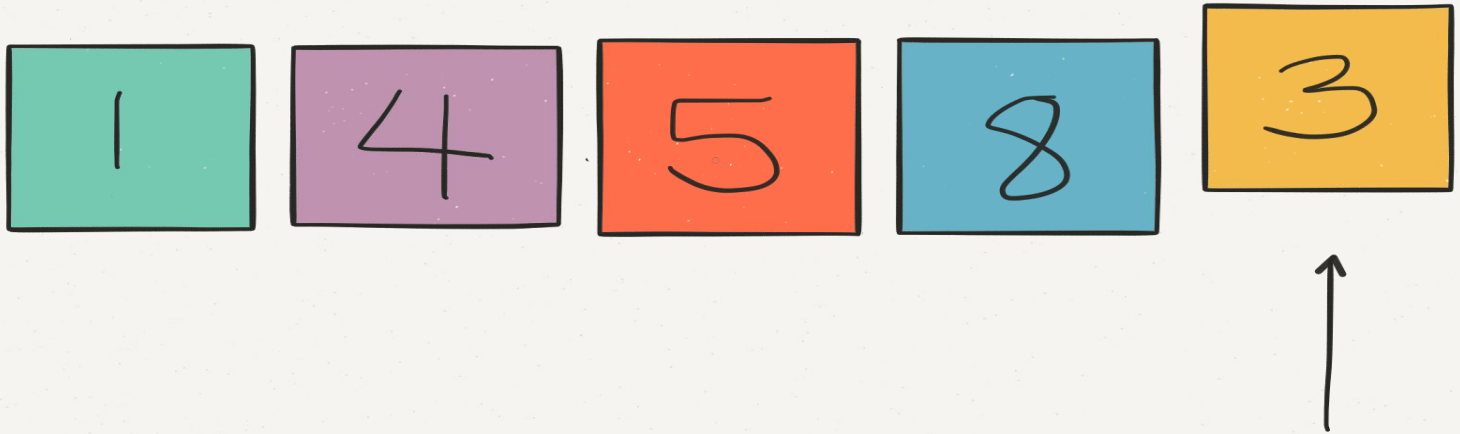

4
1
5
8
3

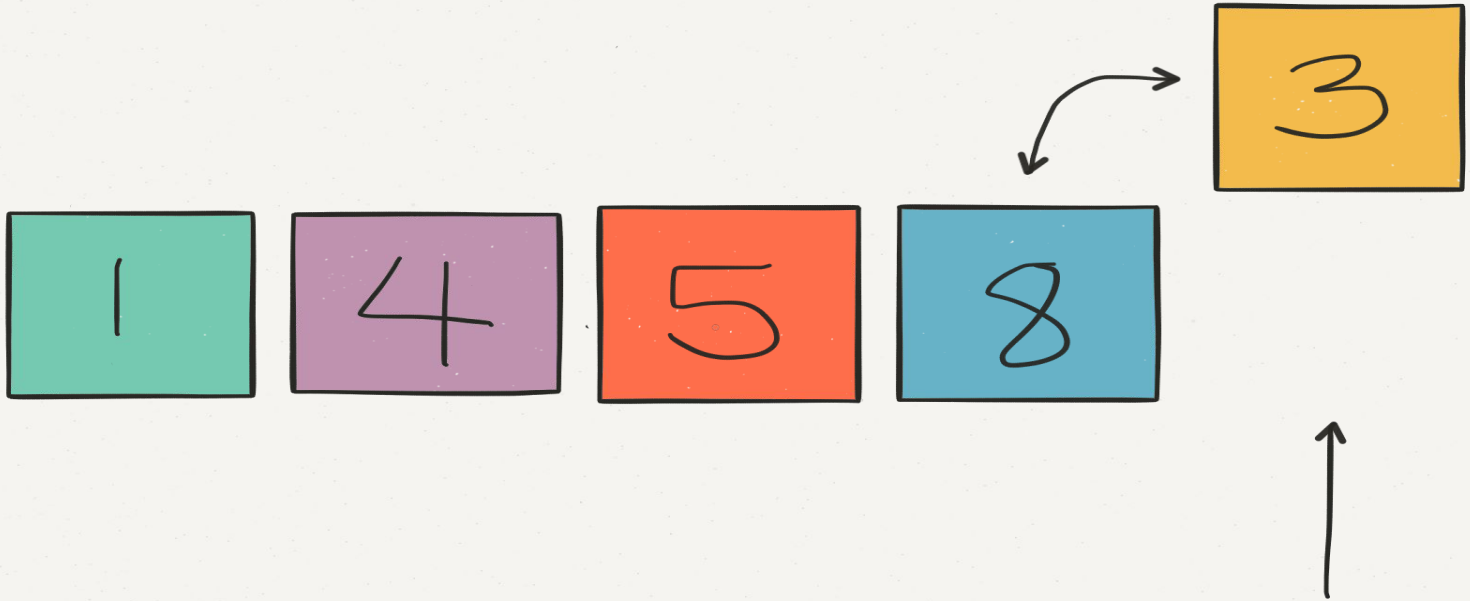

4
1
5
8
3

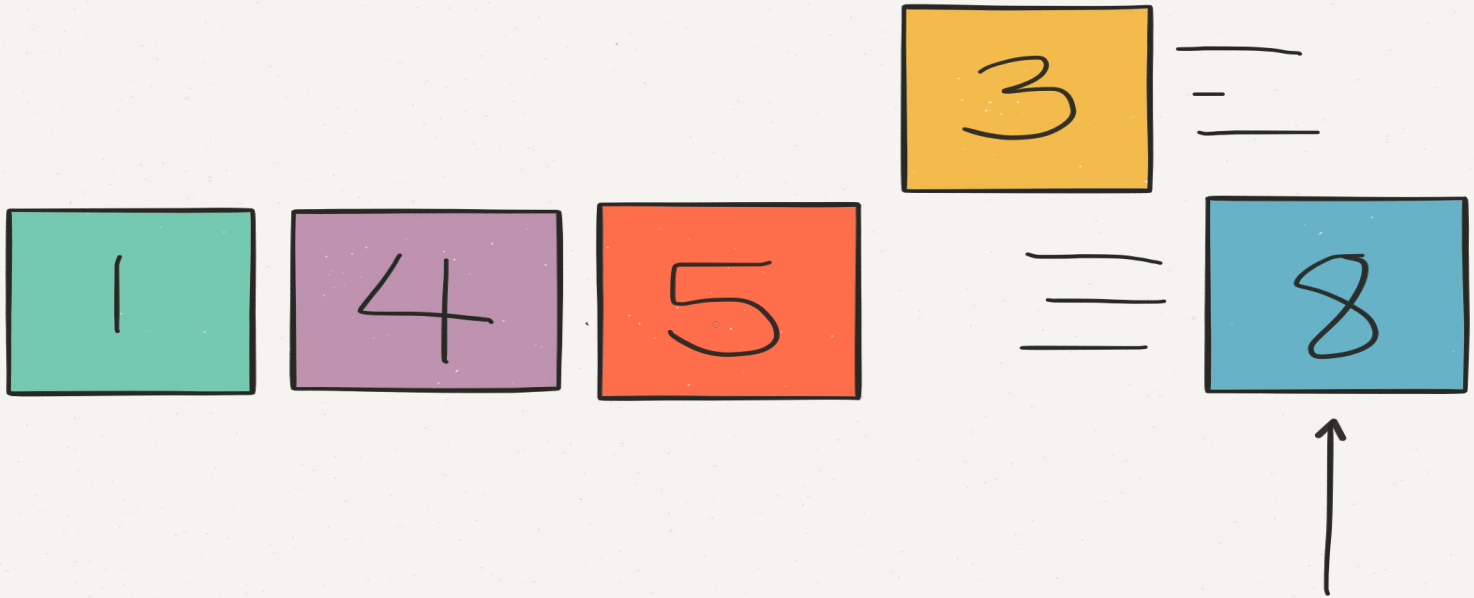

4
1
5
8
3

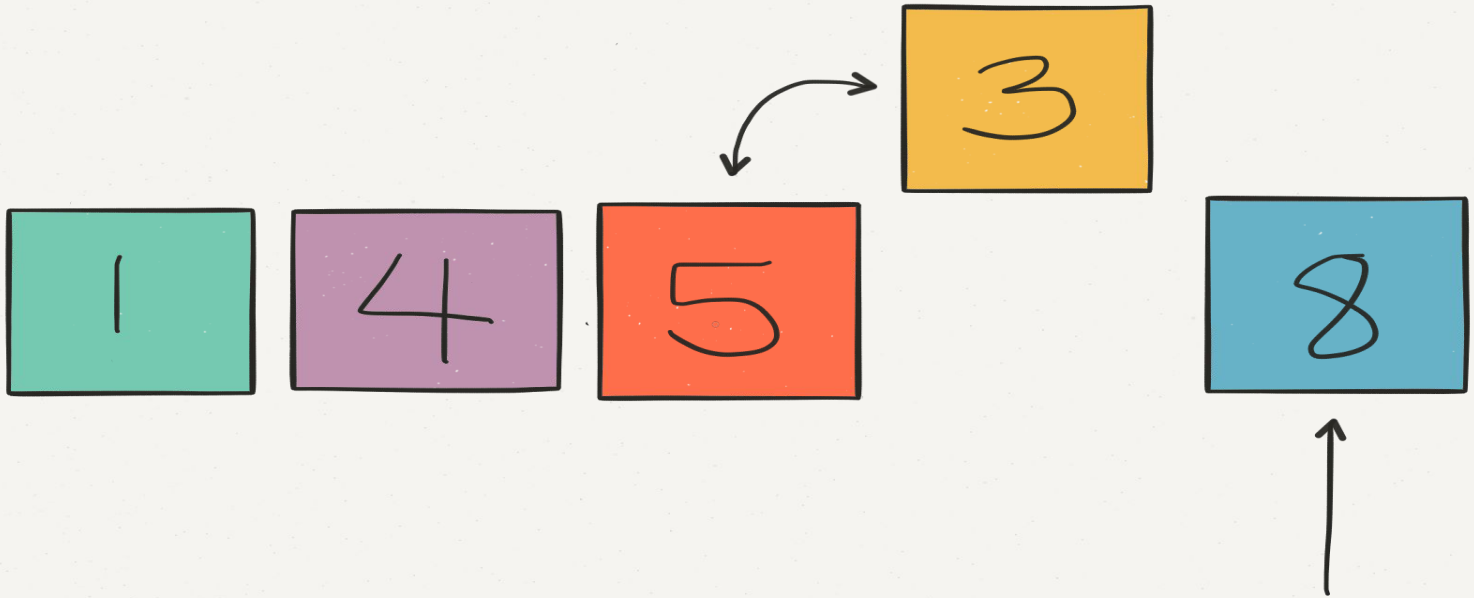

4
1
5
8
3

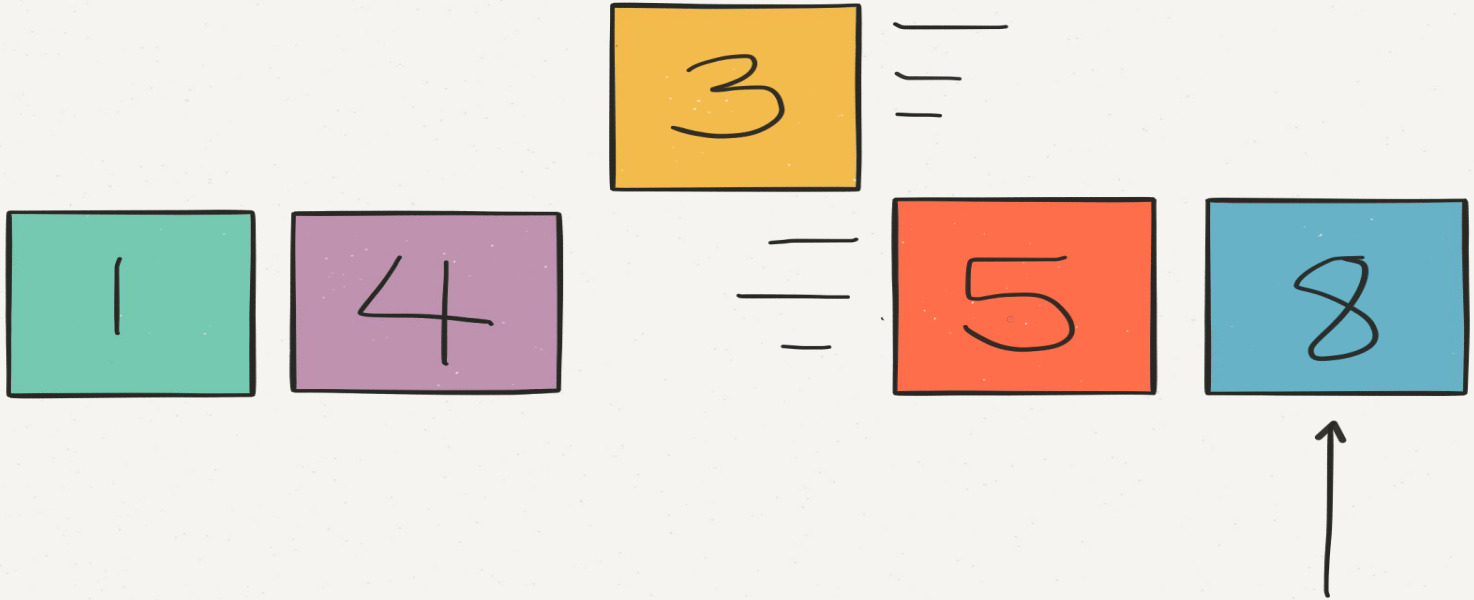

1
4
5
8
3

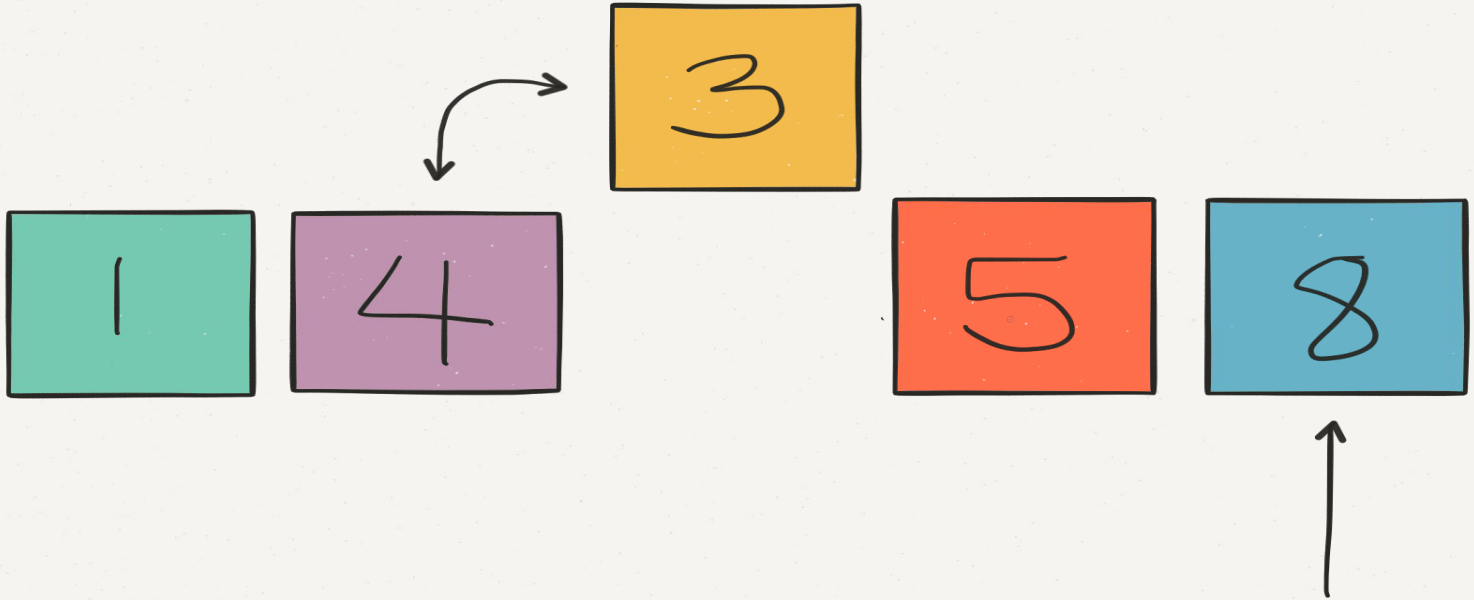


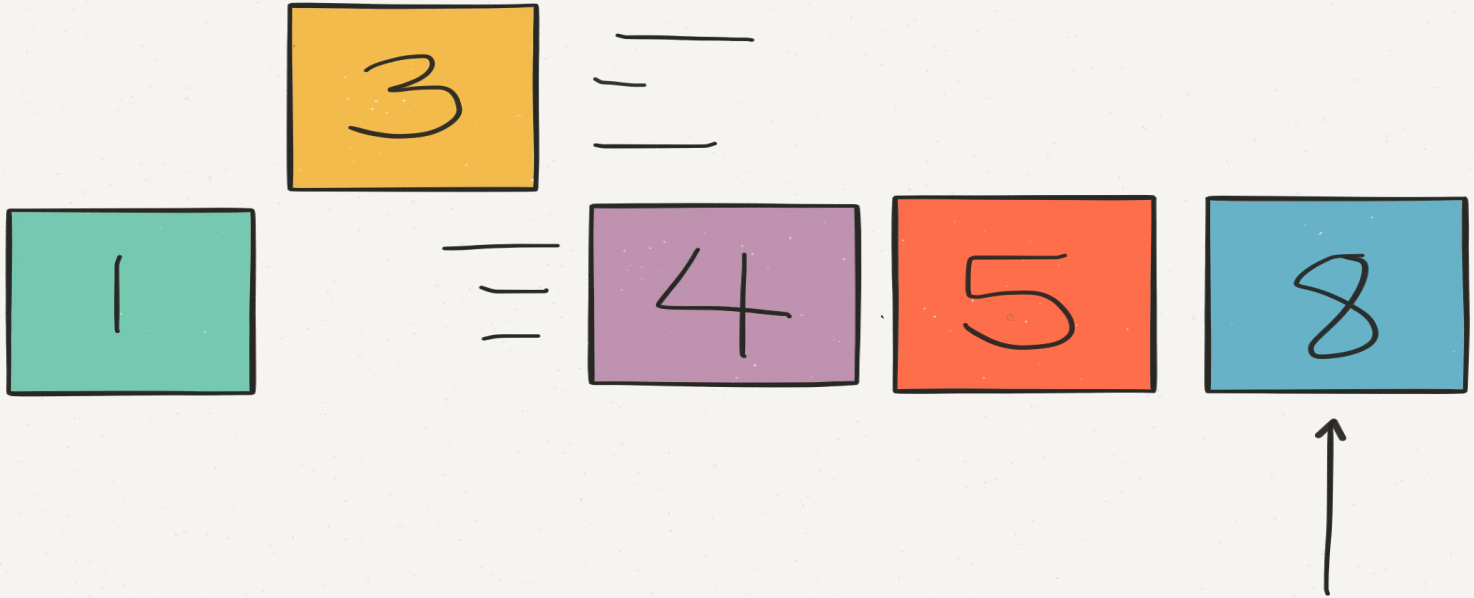

3
1
4
5
8

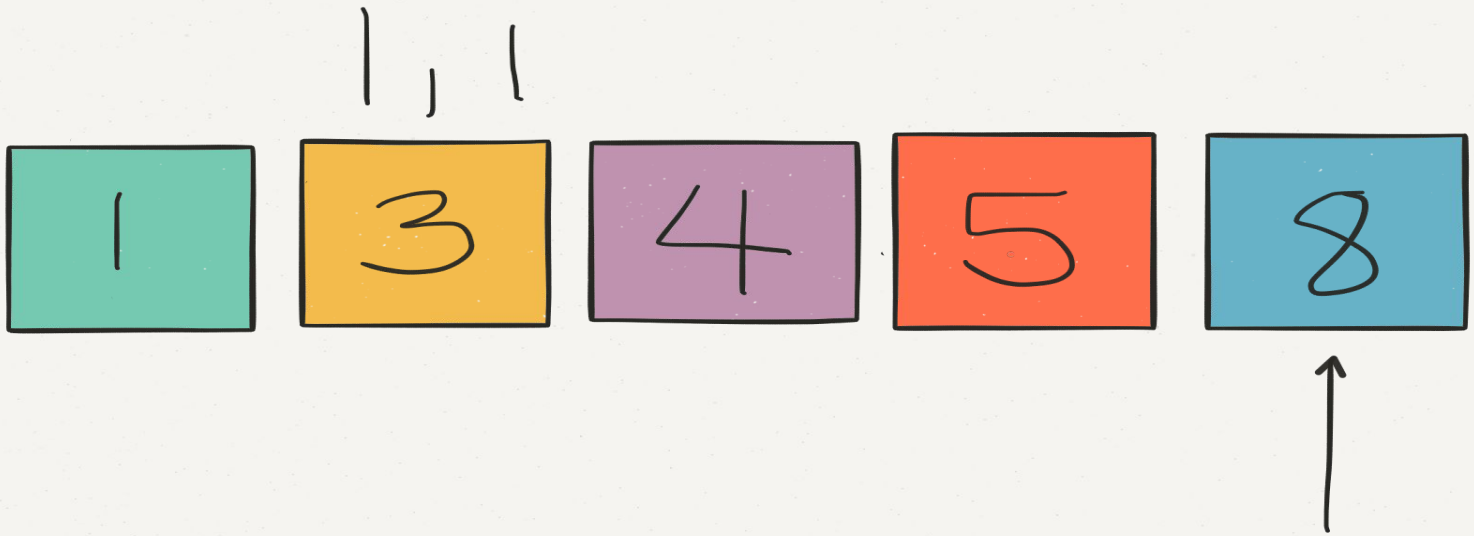

3
1
4
5
8

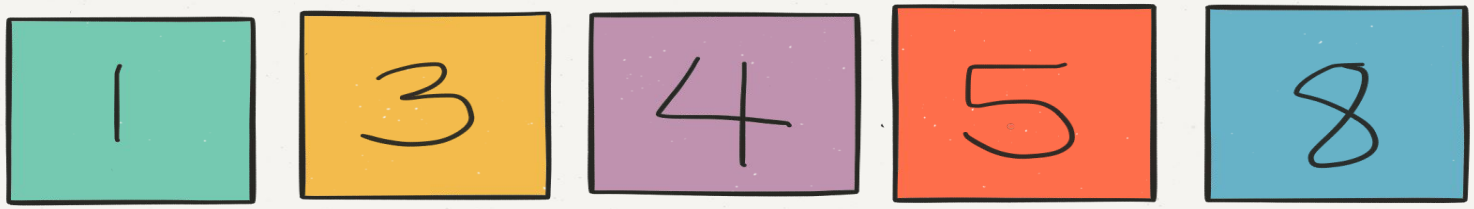

3
1
4
5
8


3
1
4
5
8


3
1
4
5
8

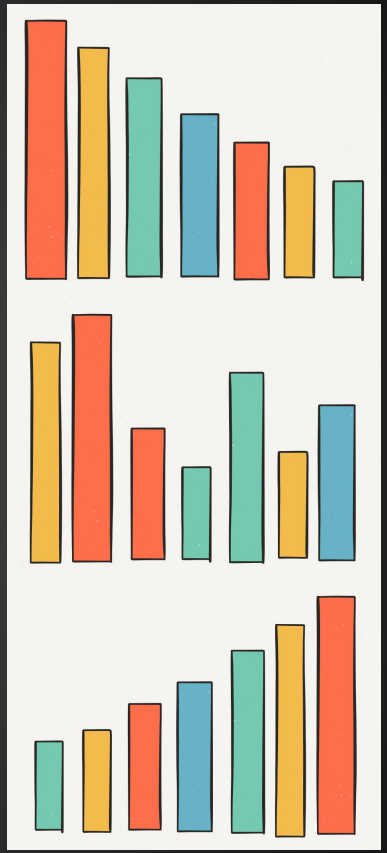

3
1
4
5
8


1, 1
1
3
4
5
8


1
3
4
5
8

Time Complexity

- Worse case
  - $O(n^2)$
- Average Case
  - $O(n^2)$
- Best Case
  - $\Omega(n)$



Advantages

- Simple to understand
- Constant auxiliary space
- Average twice as fast as bubble & selection
- Stable sort
- Streamable
- Fast for nearly sorted data O(kn)
- Linear time for sorted arrays

Disadvantage

- Still quadratic time at worse
  - Inefficient for large data sets
  - Not as fast compared to mergesort or quicksort

Which basic sort should I use?

- Insertion sort
  - For n < 10
  - When the dataset is almost sorted
  - Want to maintain simplicity

Resources

- Khan Academy Algorithms Course
  - Javascript
  - https://www.khanacademy.org/computing/computer-science/algorithms/sorting-algorithms/a/sorting
  - https://www.khanacademy.org/computing/computer-science/algorithms/insertion-sort/a/insertion-sort
- Visualgo.net
  - http://visualgo.net/sorting.html
- USF Computer Science
  - https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html