CSE321 - PS

Problem 1

Construct an algorithm to find the missing element in a regularly increasing array.

Example:

2 4 6 8 12 → missing element is 10

3 9 12 15 18 → missing element is 6

Solution 1

```
findMissingElement (array):
    differences ← [ ]
    n ← length of array
    for i from 0 to n-1:
        append (array[i+1]-array[i]) to differences
    difference ← mostly seen element in differences
    for i from 0 to n-1:
        if (array[i+1]-array[i]) != difference:
            return array[i] + difference
```

Solution 1

```
findMissingElement (array):
    differences ← [ ]
    n ← length of array
    for i from 0 to n-1:
        append (array[i+1]-array[i]) to differences
    difference ← mostly seen element in differences
    for i from 0 to n-1:
        if (array[i+1]-array[i]) != difference:
            return array[i] + difference
```

Best Case:	**Ω(n)**
Worst Case:	**O(n)**
Average Case:	**Θ(n)**

Solution 1

A solution by using the ***binary search.***

Solution 1

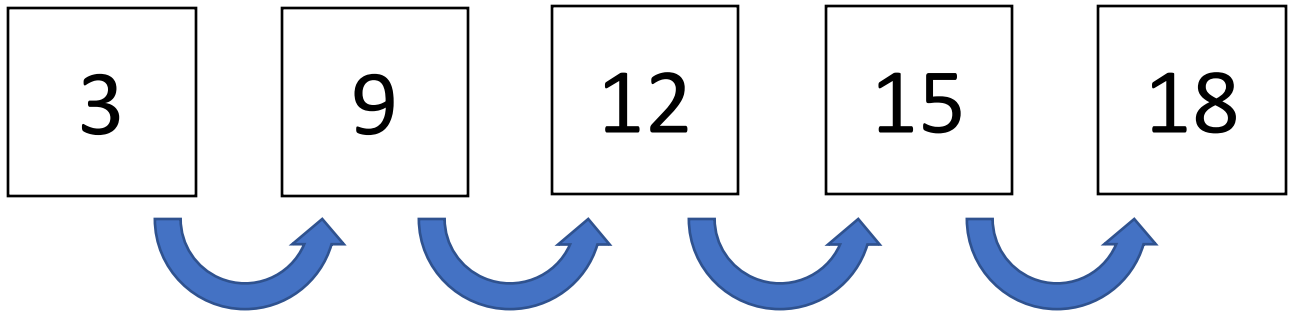
3	9	12	15	18

Solution 1

3	9	12	15	18

difference = ?

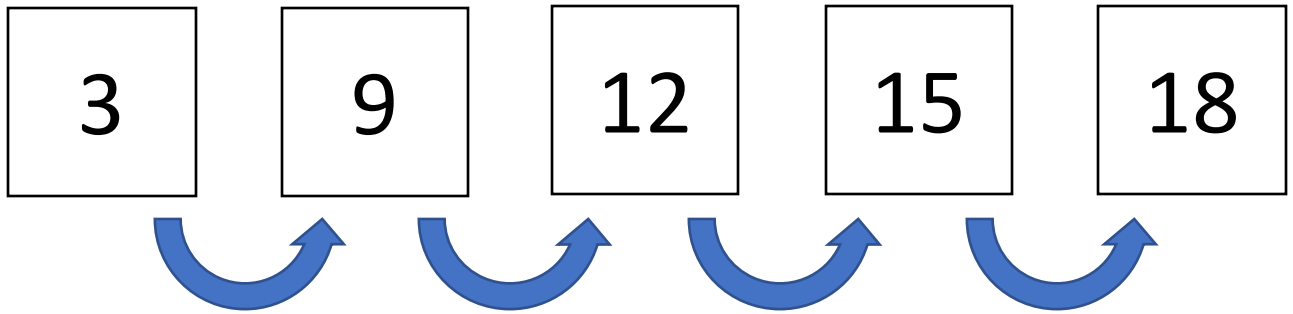
Solution 1



difference = ?

n – 1 = 4 additions

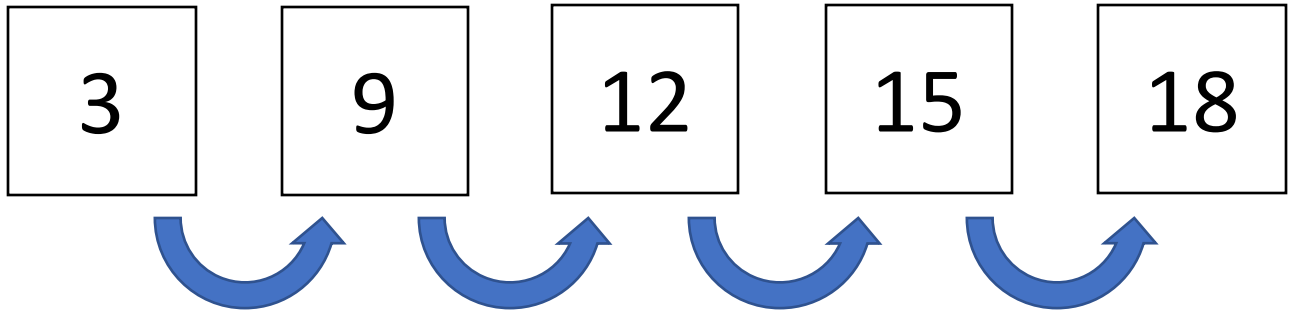
Solution 1



difference = ?

(n – 1) + 1 = n = 5 additions

Solution 1

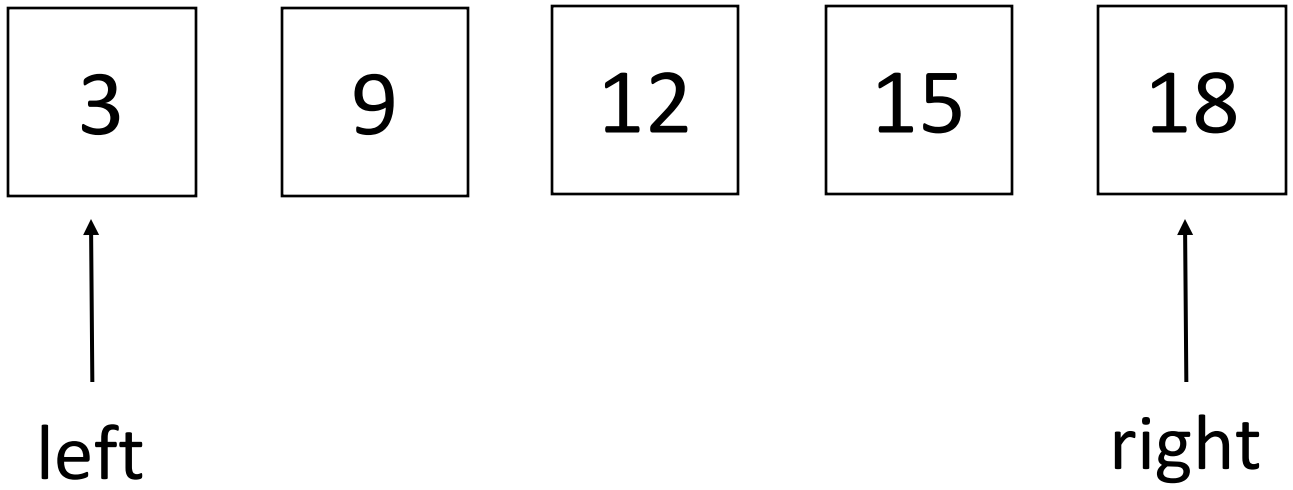


difference = ?

(n – 1) + 1 = n = 5 additions

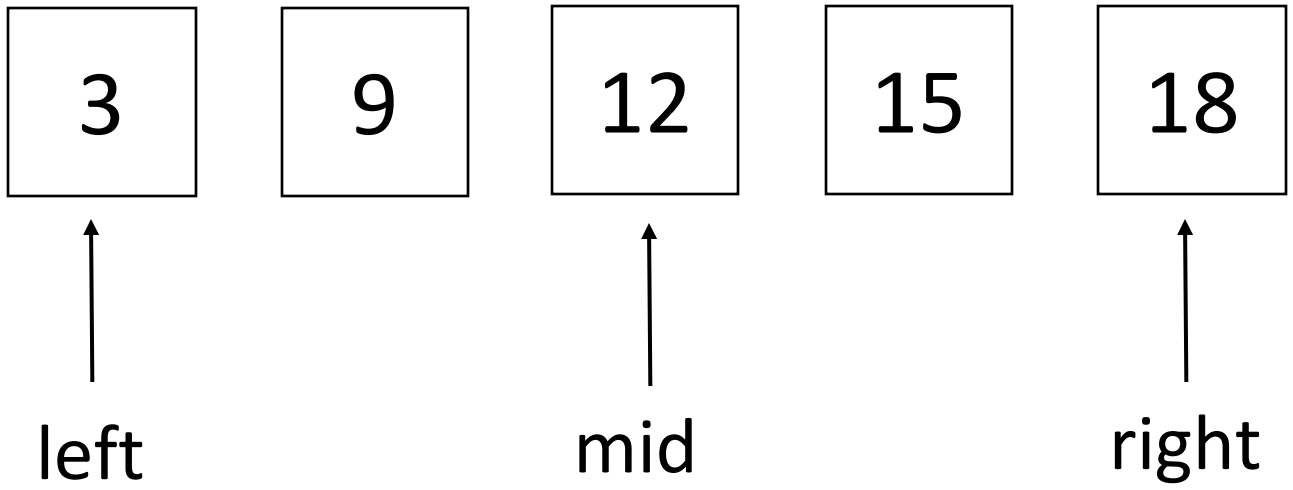
difference = (18 – 3) / 5 = 3

Solution 1



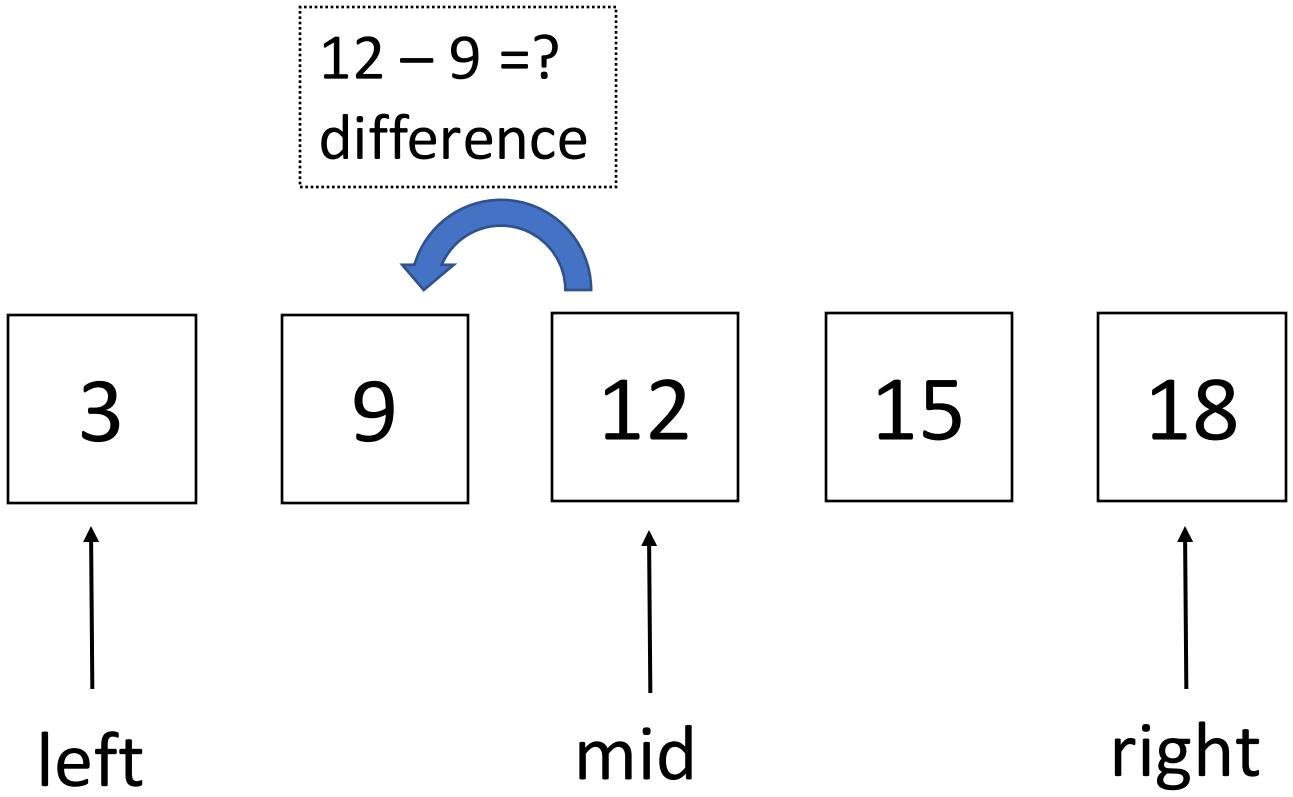
difference = 3

Solution 1



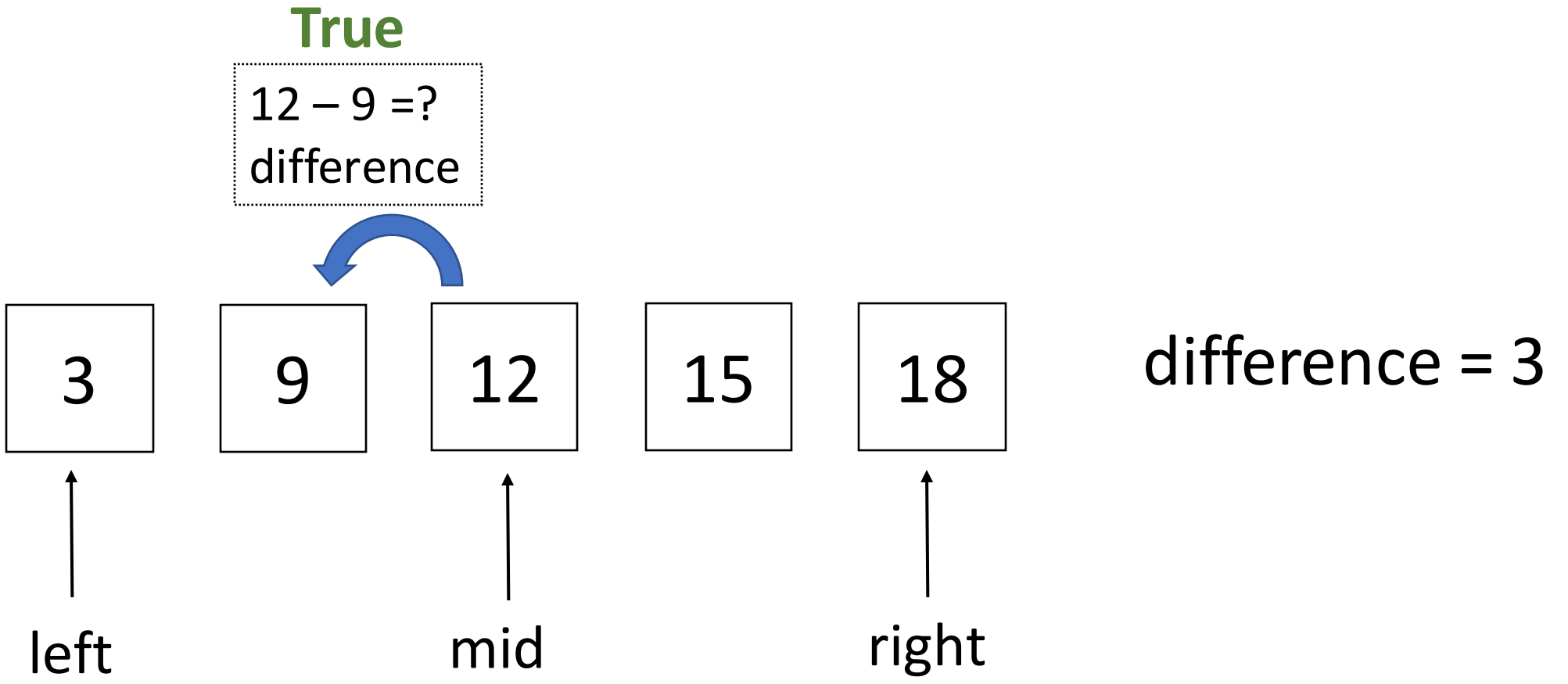
difference = 3

Solution 1

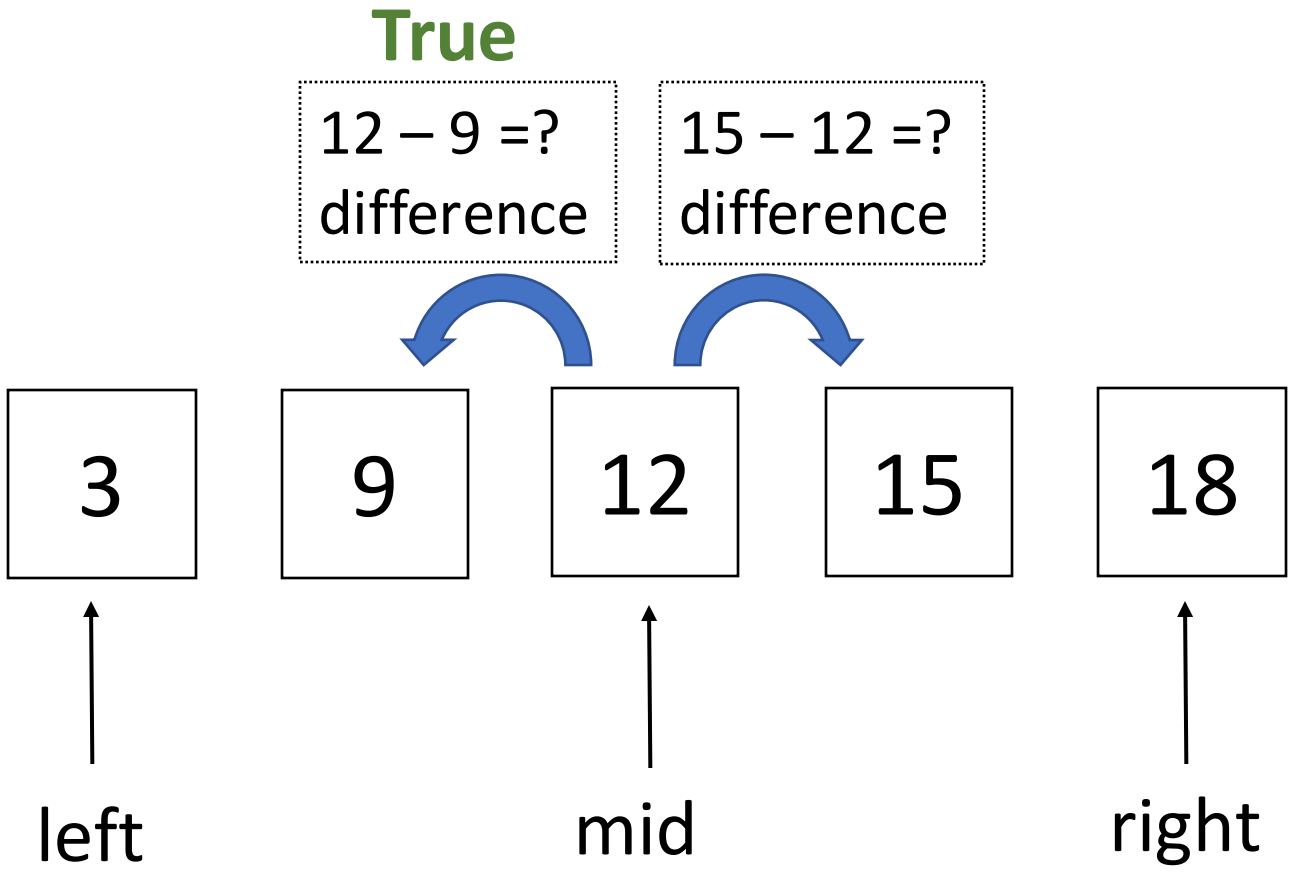


difference = 3

Solution 1

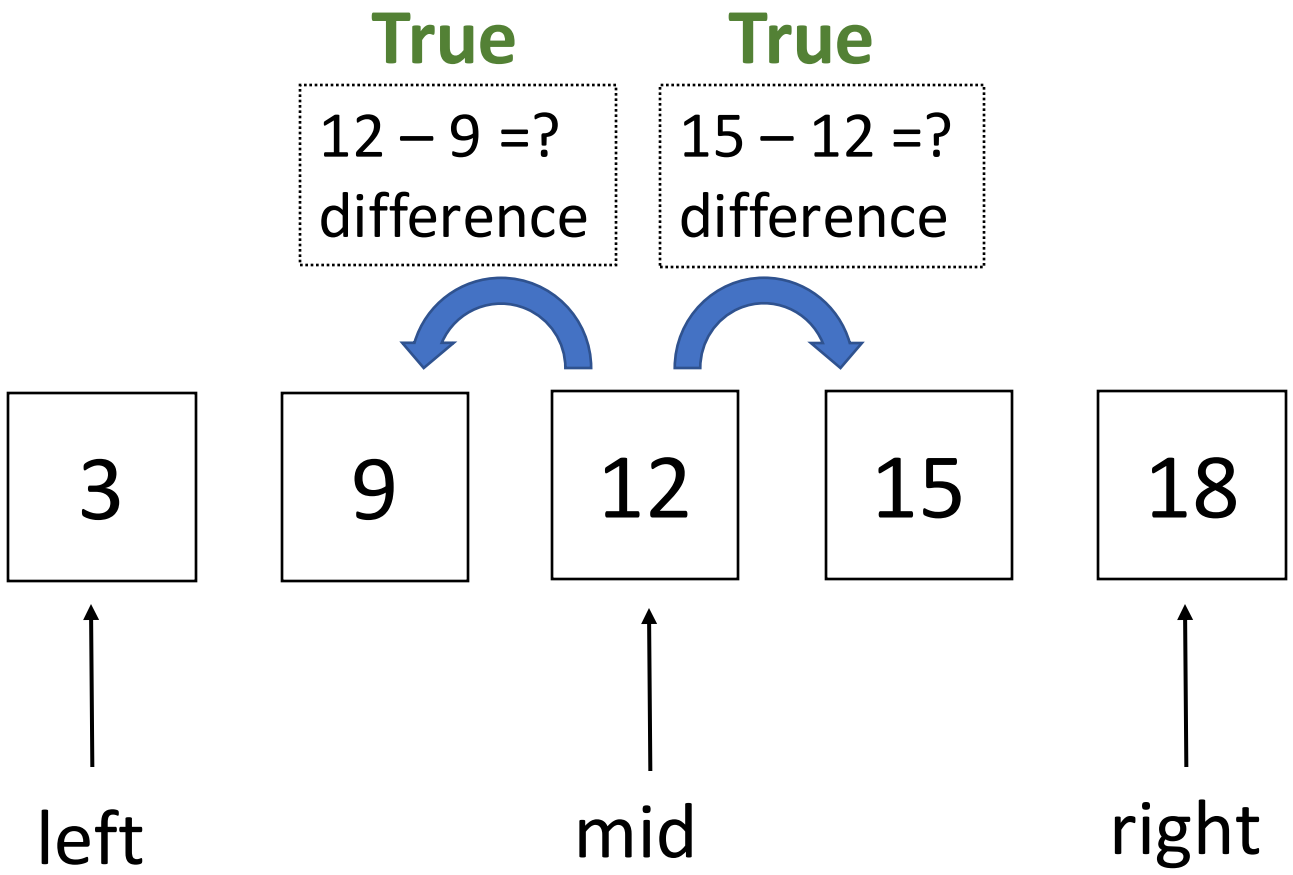


Solution 1



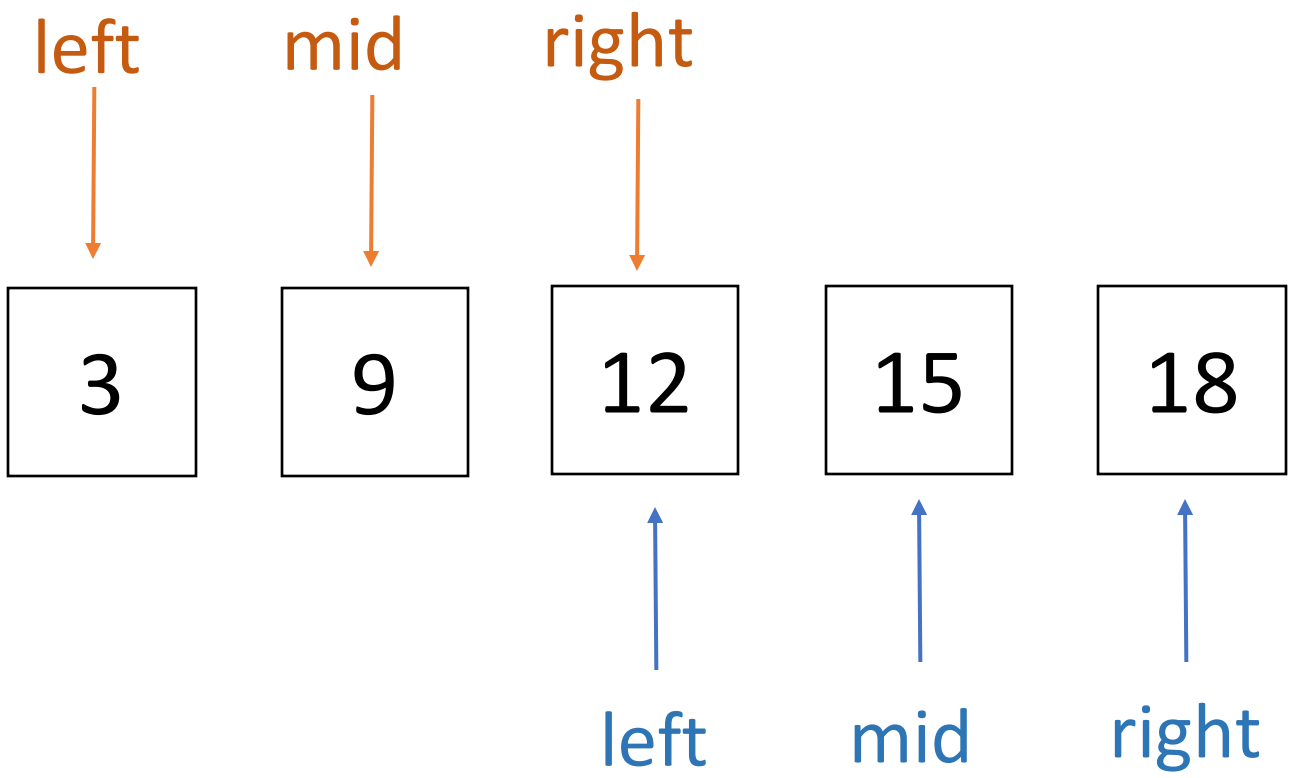
difference = 3

Solution 1



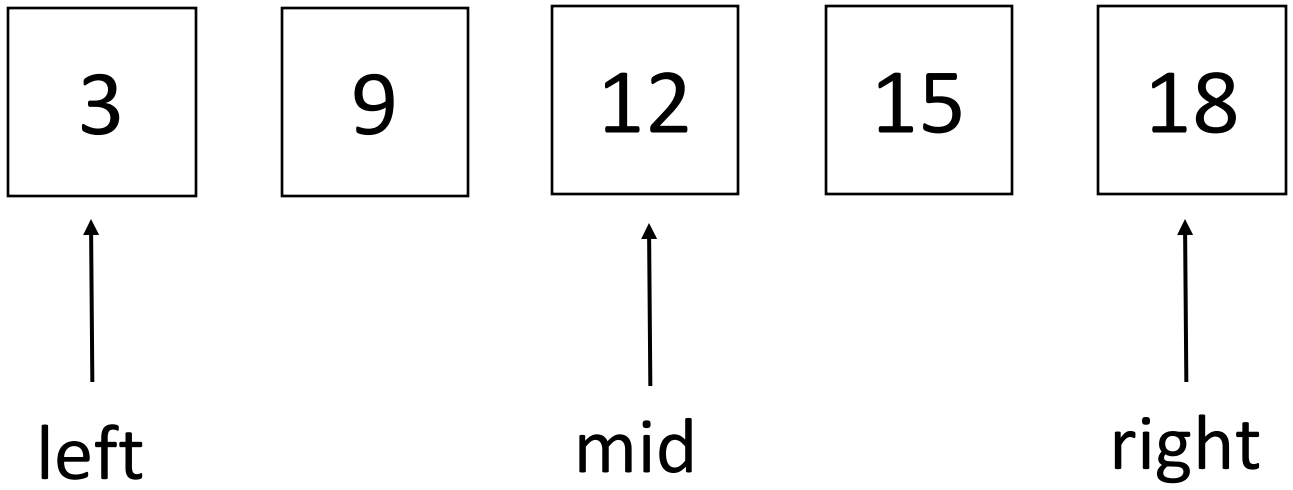
difference = 3

Solution 1



difference = 3

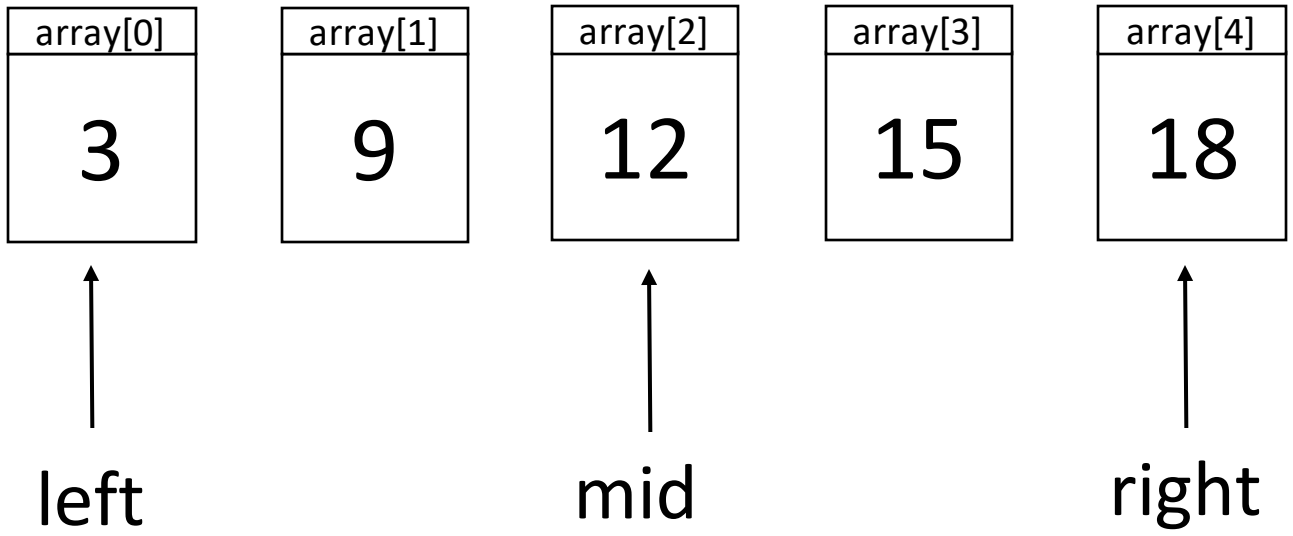
Solution 1



difference = 3

Solution 1

a = right – mid = 4 – 2 = 2 (number of additions)

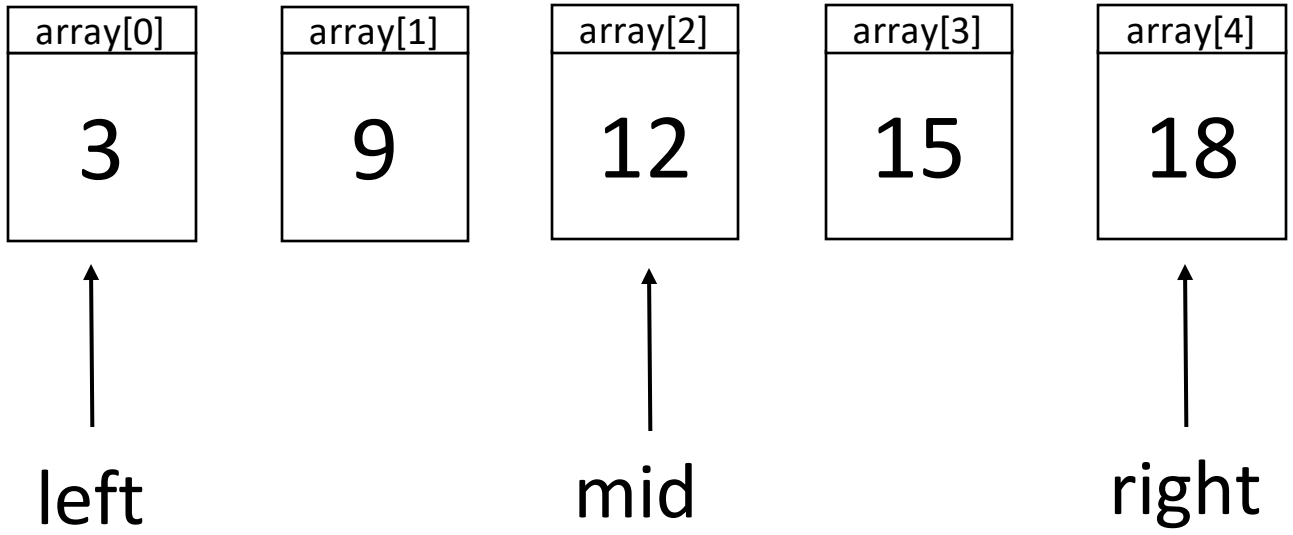


difference = 3

Solution 1

a = right – mid = 4 – 2 = 2 (number of additions)

array[mid] + a*difference =? array[right]



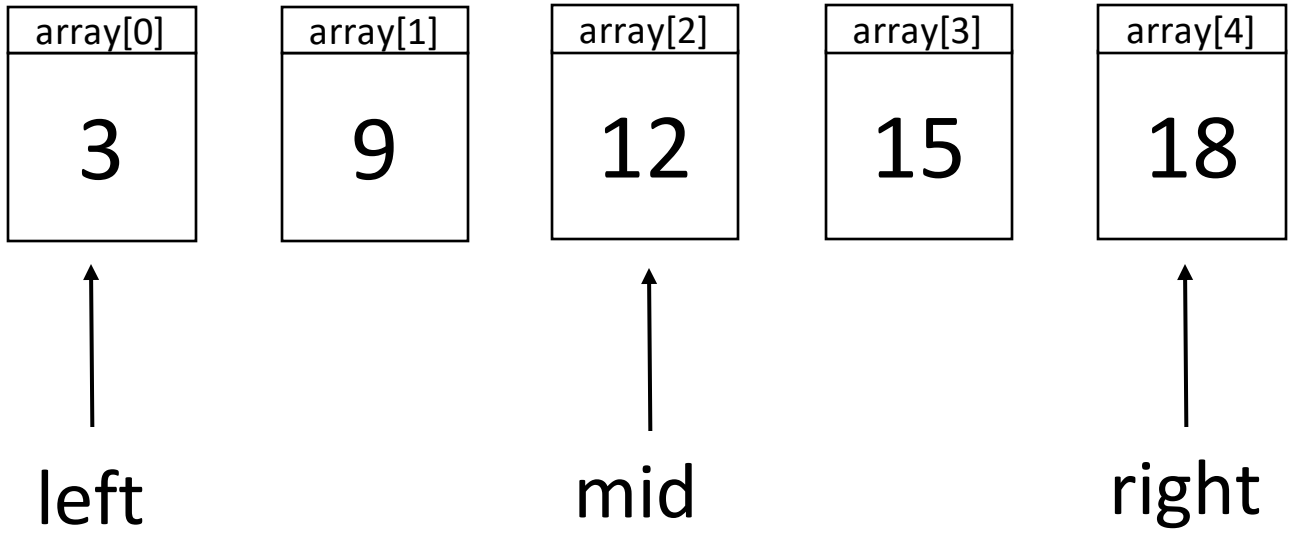
difference = 3

Solution 1

a = right – mid = 4 – 2 = 2 (number of additions)

array[mid] + a*difference =? array[right]

12 + 2*3 =? 18



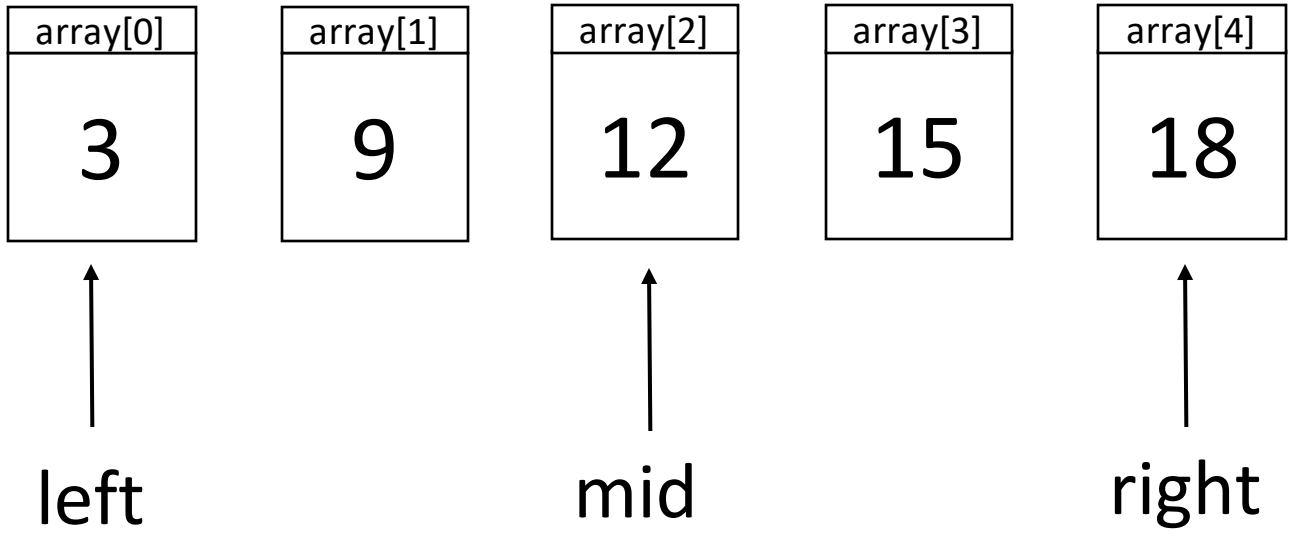
difference = 3

Solution 1

a = right – mid = 4 – 2 = 2 (number of additions)

array[mid] + a*difference =? array[right]

12 + 2*3 =? 18 **True**



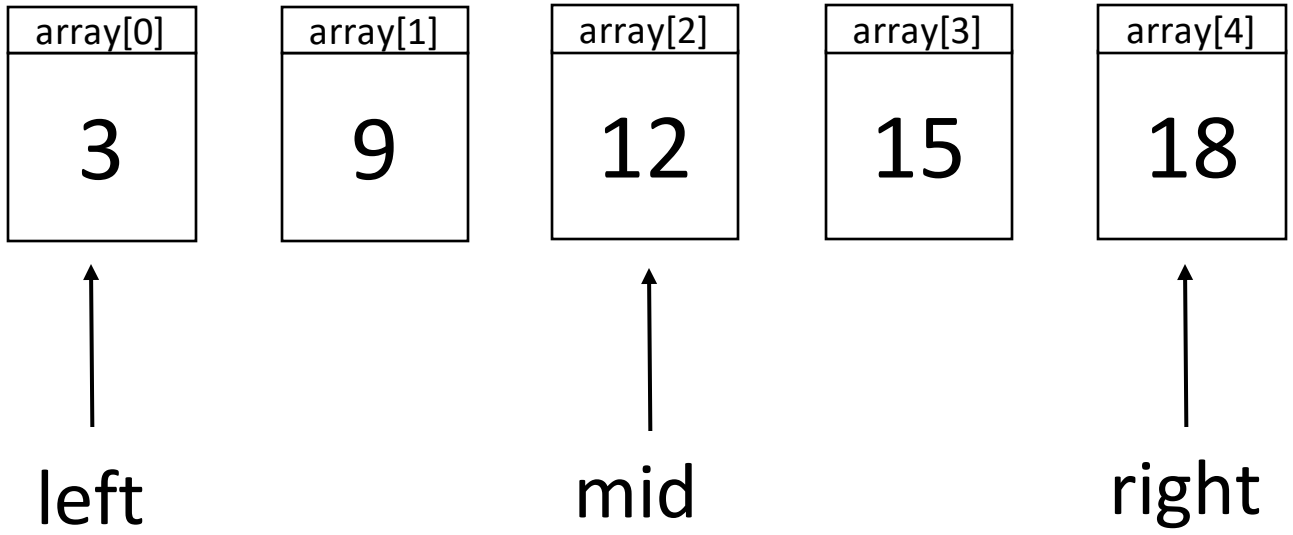
difference = 3

Solution 1

a = mid – left = 2 – 0 = 2 (number of additions)

array[mid] - a*difference =? array[left]

3 =? 12 – 2*3



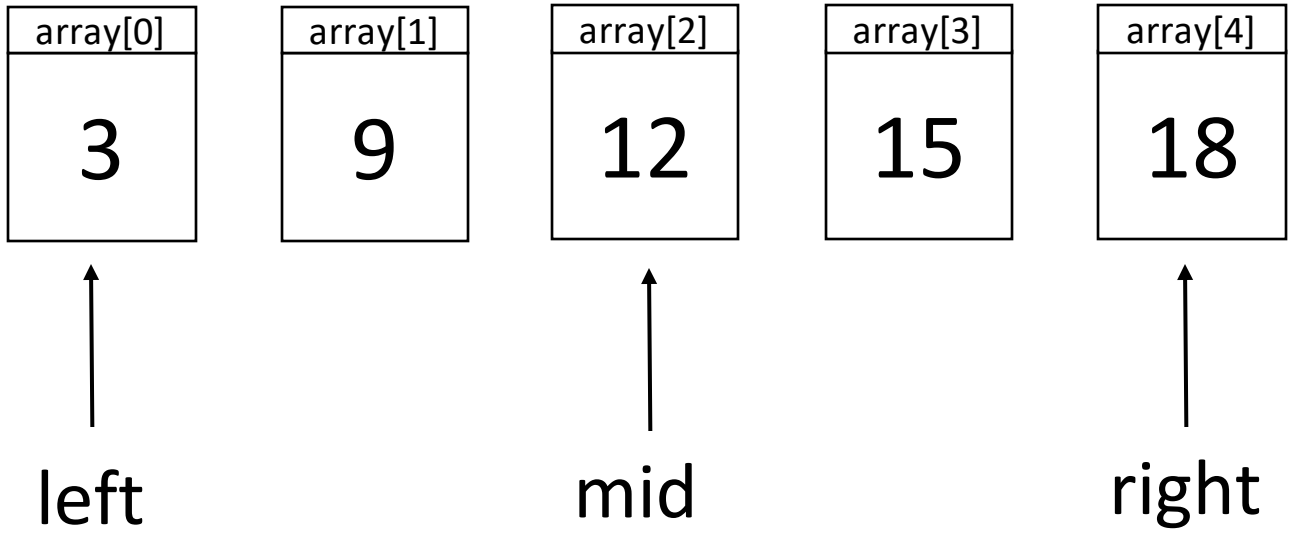
difference = 3

Solution 1

a = mid – left = 2 – 0 = 2 (number of additions)

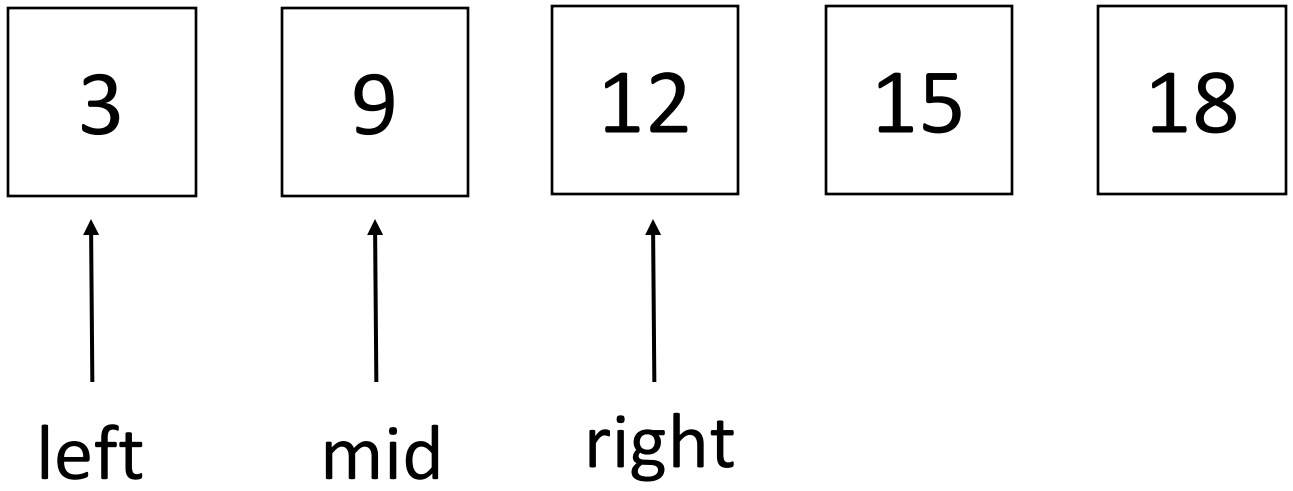
array[mid] - a*difference =? array[left]

3 =? 12 – 2*3 **False**



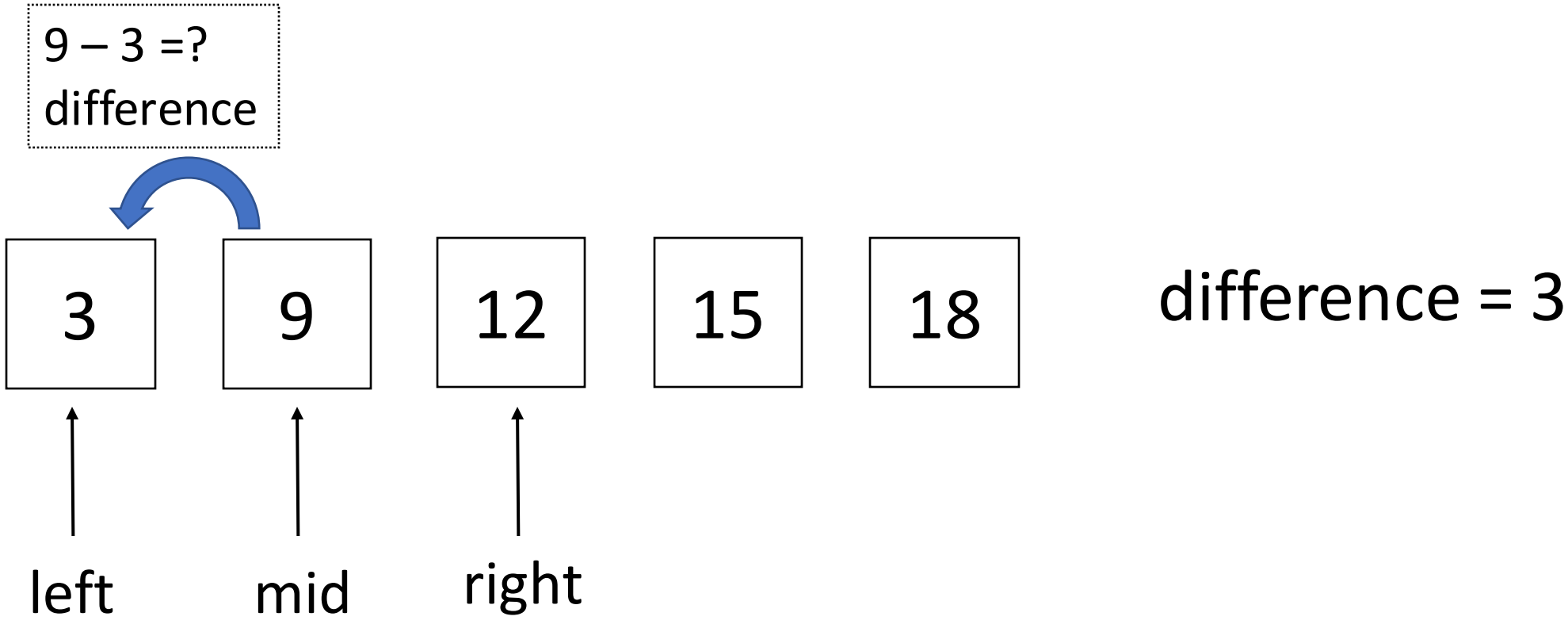
difference = 3

Solution 1



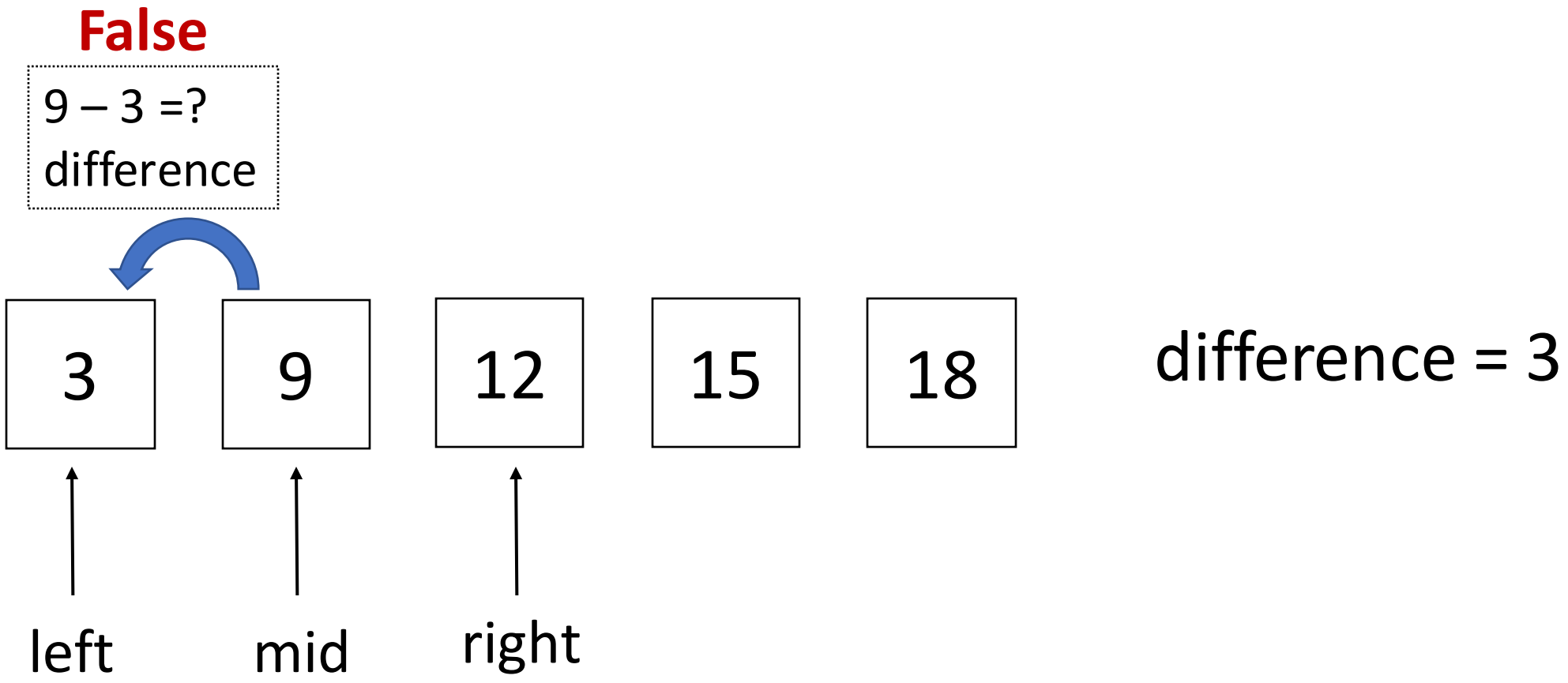
difference = 3

Solution 1



difference = 3

Solution 1

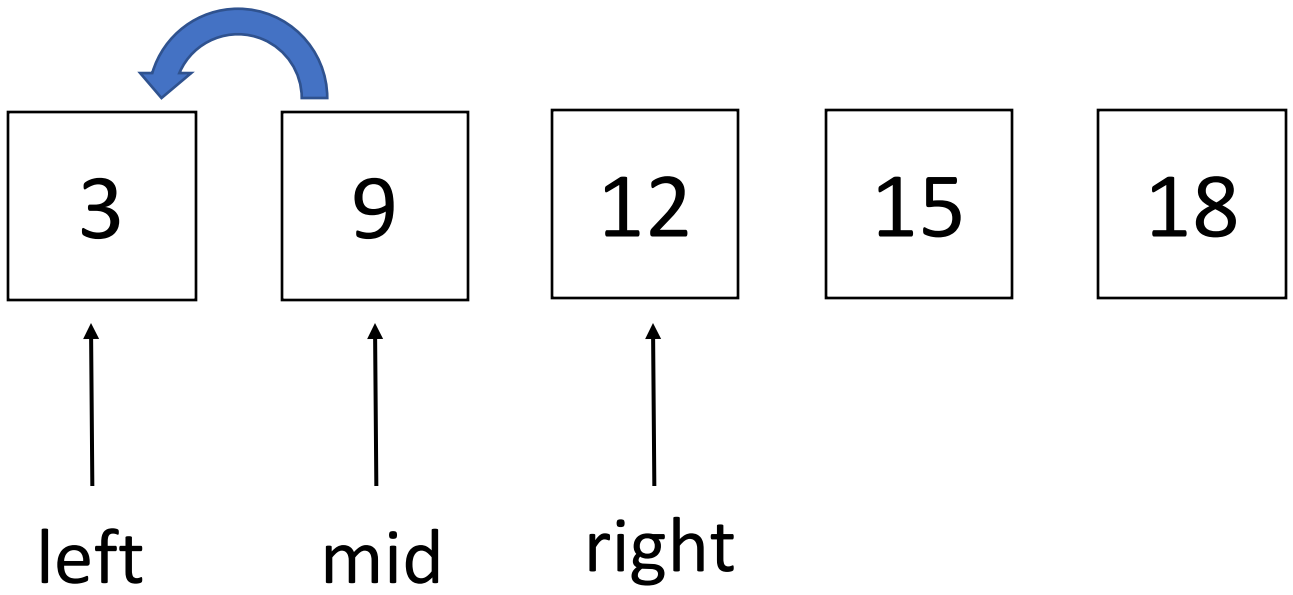


Solution 1

False

9 – 3 =?
difference

array[mid] - difference =



difference = 3

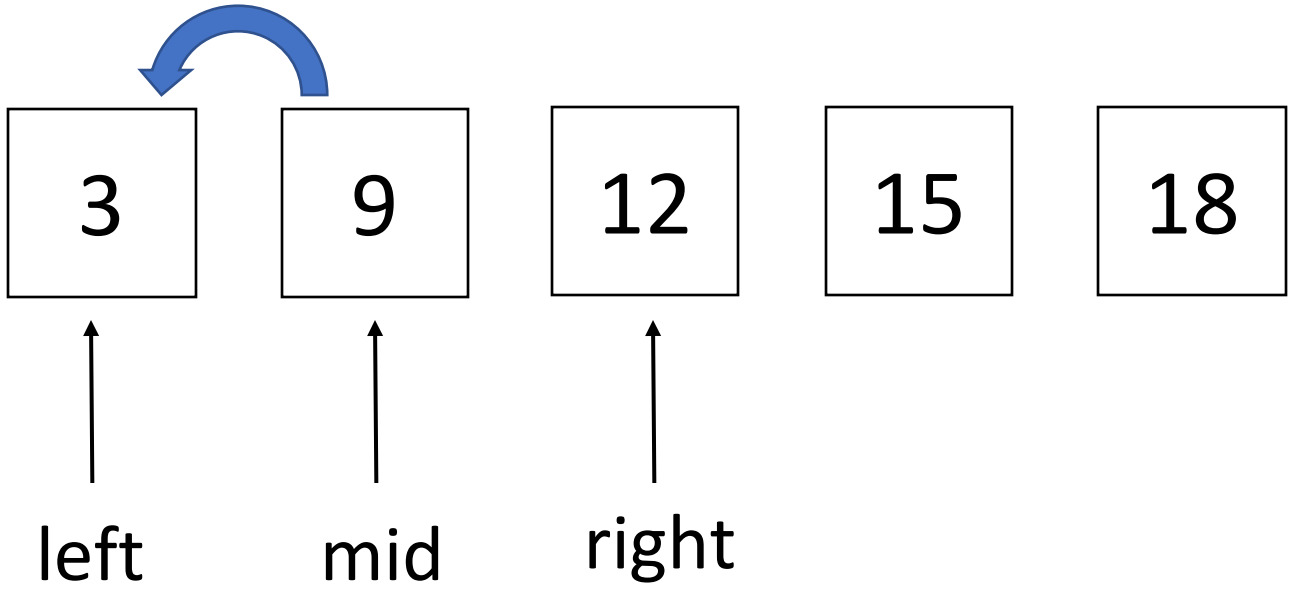
Solution 1

False

9 – 3 =?
difference

array[mid] - difference = 9 – 3 = 6

The missing element is 6



difference = 3

Solution 1

A solution by using the ***binary search***

Complexity?

Solution 1

A solution by using the ***binary search***

Best Case:	**Ω(1)**
Worst Case:	**O(logn)**
Average Case:	**Θ(logn)**

Solution 1

```
findMissingElement (array):
    differences ← [ ]
    n ← length of array
    for i from 0 to n-1:
        append (array[i+1]-array[i]) to differences
    difference ← mostly seen element in differences
    for i from 0 to n-1:
        if (array[i+1]-array[i]) != difference:
            return array[i] + difference
```

Solution 1

```
findMissingElement (array):
    difference ← (last element – first element) / (length of the array)
    for i from 0 to n-1:
        if (array[i+1]-array[i]) != difference:
            return array[i] + difference
```

Solution 1

```
findMissingElement (array):
    difference ← (last element – first element) / (length of the array)
    for i from 0 to n-1:
        if (array[i+1]-array[i])  != difference:
            return array[i] + difference
```

Complexity?

Solution 1

```
findMissingElement (array):
    difference ← (last element – first element) / (length of the array)
    for i from 0 to n-1:
        if (array[i+1]-array[i])  != difference:
            return array[i] + difference
```

Best Case:	**Ω(1)**
Worst Case:	**O(n)**
Average Case:	**Θ(n)**

Problem 2

Construct an algorithm to find the maximum difference between two elements such that the smaller element appears before the larger one.

Example:

6 10 1 7 5 → maximum difference is 7 – 1 = 6

15 6 2 3 5 → maximum difference is 5 – 2 = 3

Solution 2

A solution with O(n^2) complexity:

```
findMaxDiff (array)
    difference ← 0
    n  ← length of array
    for i from 0 to n-1:
        for j from i+1 to n:
            if array[j] > array[i]:
                difference = max(difference , A[j] - A[i])
    return difference
```

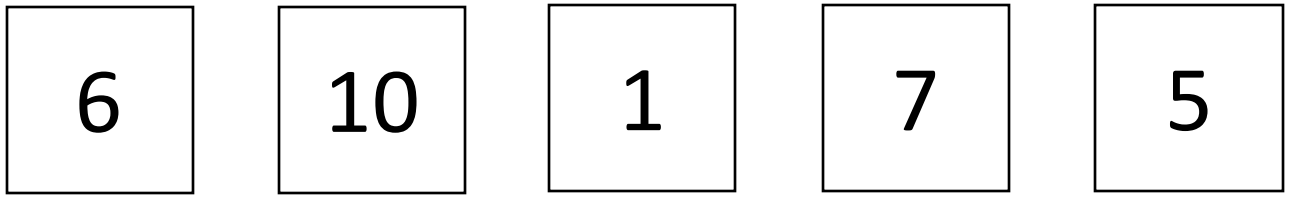
Solution 2

A solution with O(n) complexity:

```
findMaxDiff (array)
    max_difference ← 0
    max_element    ← array[-1]
    for i from n-2 to 0:
        if array[i] < max_element
            max_element = array[i]
        else
            max_difference = max ( difference,  array[i+1] - array[i] )
    return max_difference
```

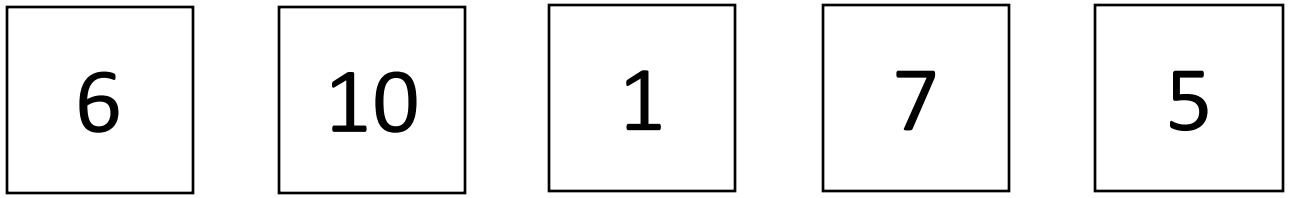
Solution 2

A solution with O(n) complexity:



Solution 2

A solution with O(n) complexity:

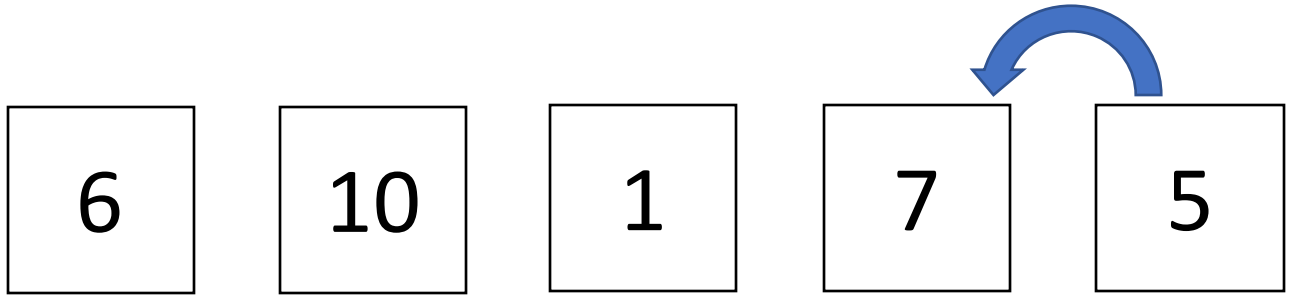


max_element = array[n-1] = 5

max_difference = 0

Solution 2

A solution with O(n) complexity:

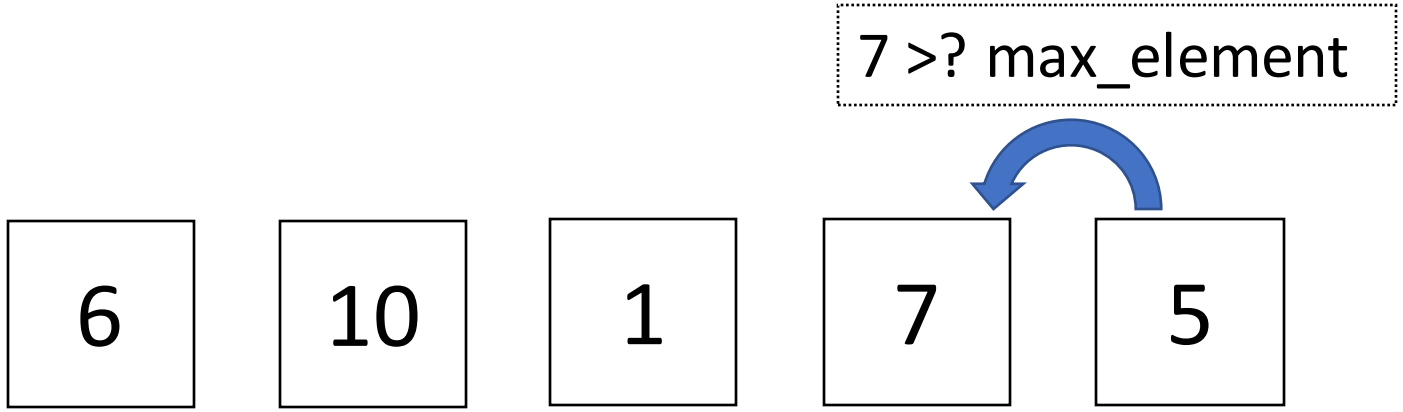


max_element = 5

max_difference = 0

Solution 2

A solution with O(n) complexity:

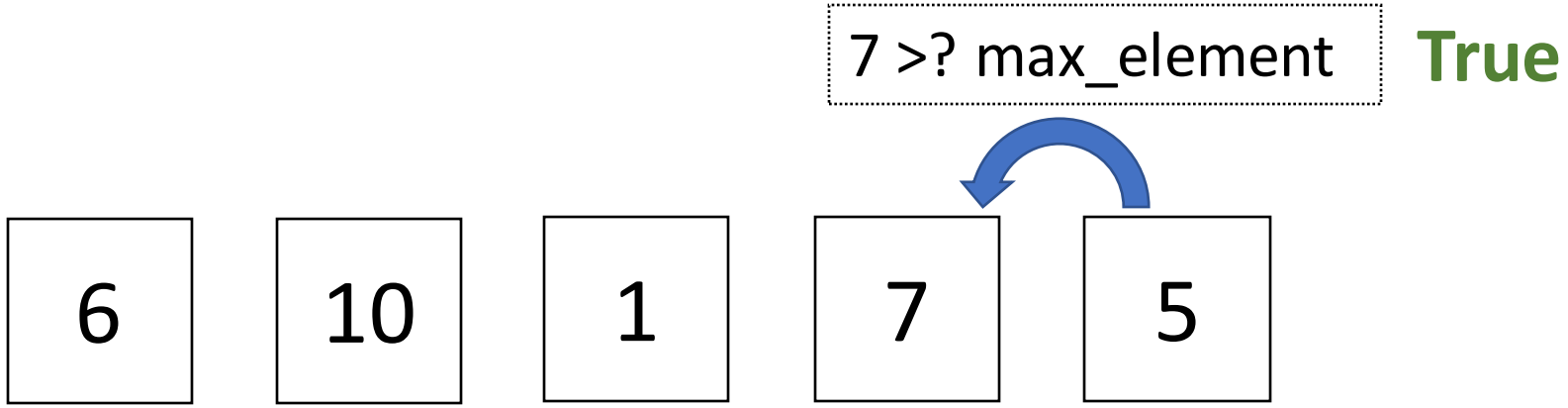


max_element = 5

max_difference = 0

Solution 2

A solution with O(n) complexity:

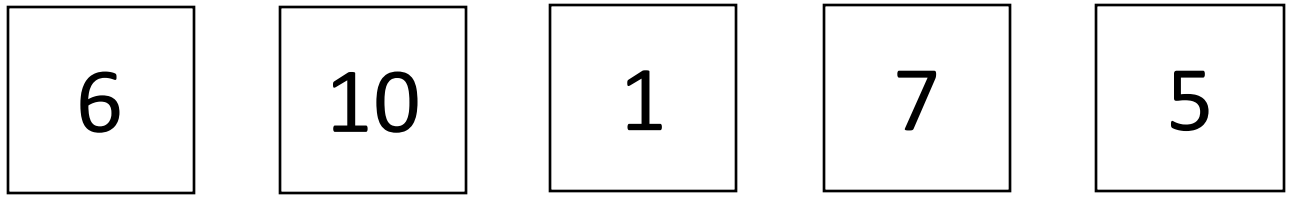


max_element = current_element = 7

max_difference = 0

Solution 2

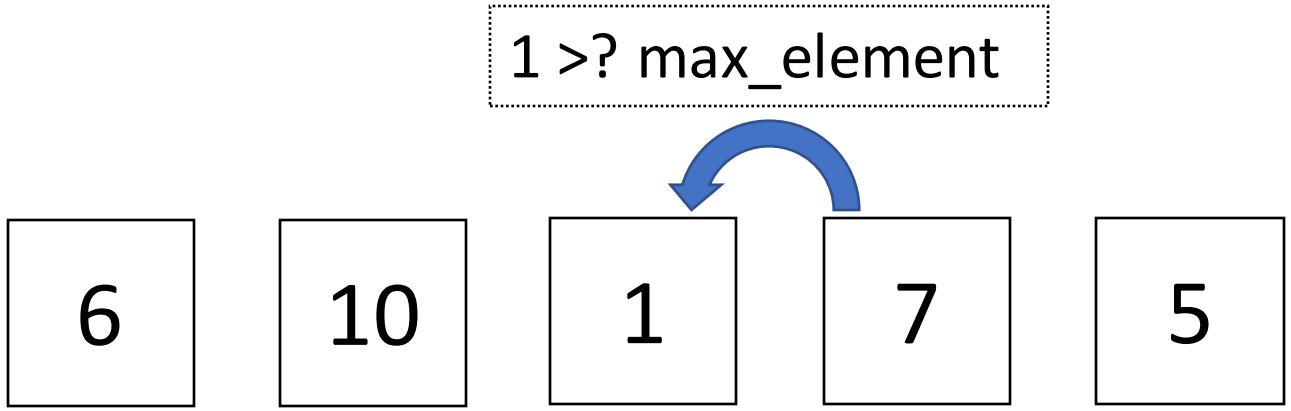
A solution with O(n) complexity:



max_element = 7
max_difference = 0

Solution 2

A solution with O(n) complexity:

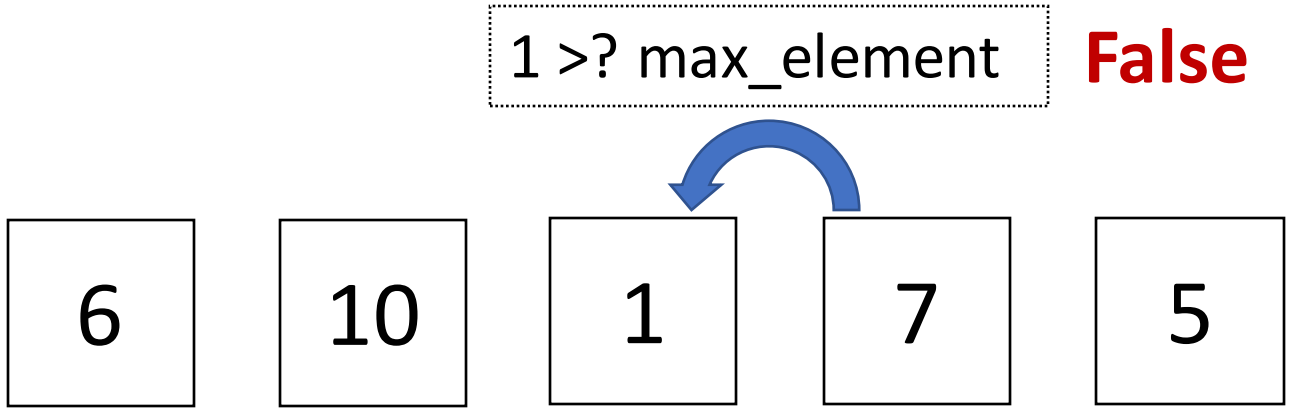


max_element = 7

max_difference = 0

Solution 2

A solution with O(n) complexity:

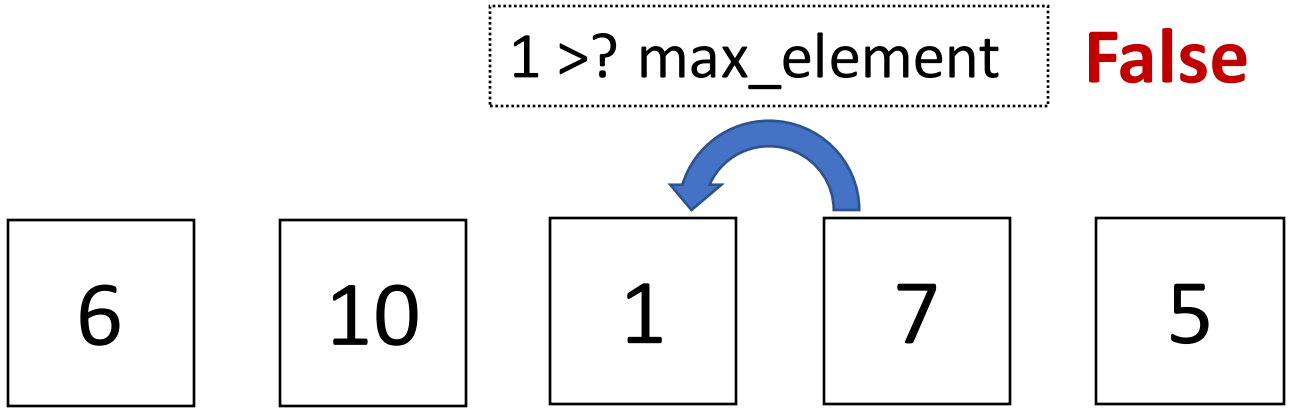


max_element = 7

max_difference = 0

Solution 2

A solution with O(n) complexity:



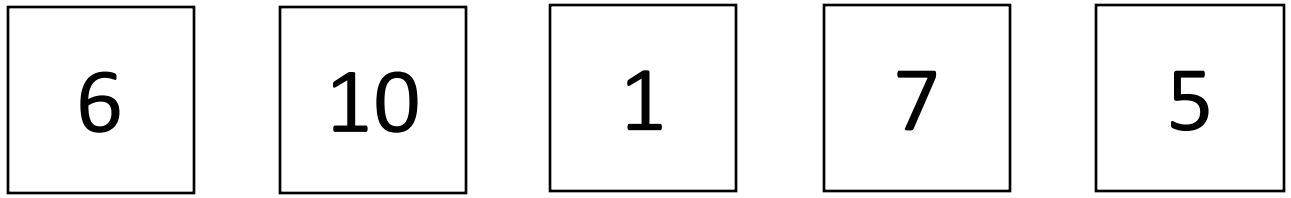
max_element = 7

max_difference = max(max_difference, max_element-current_element)

= max(0, 7 – 1) = max(0, 6) = 6

Solution 2

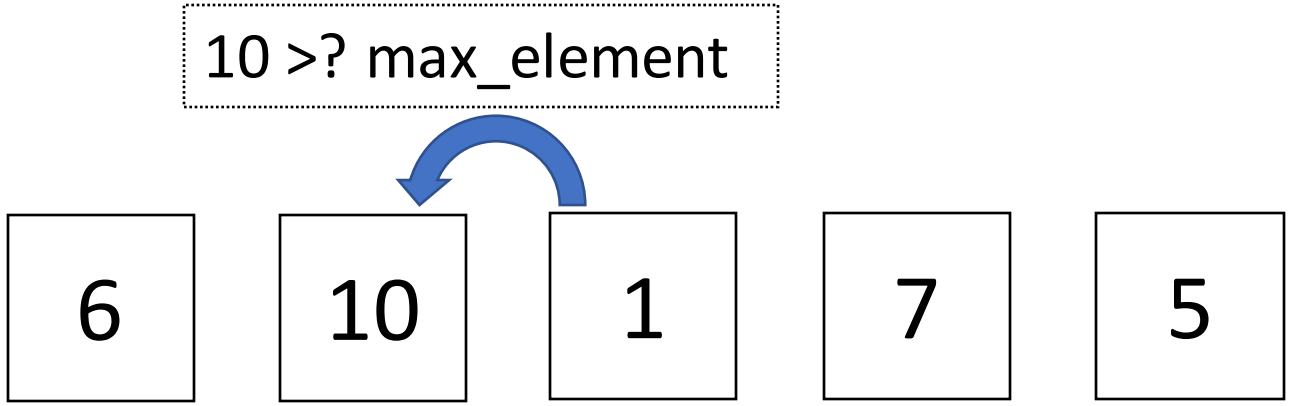
A solution with O(n) complexity:



max_element = 7
max_difference = 6

Solution 2

A solution with O(n) complexity:

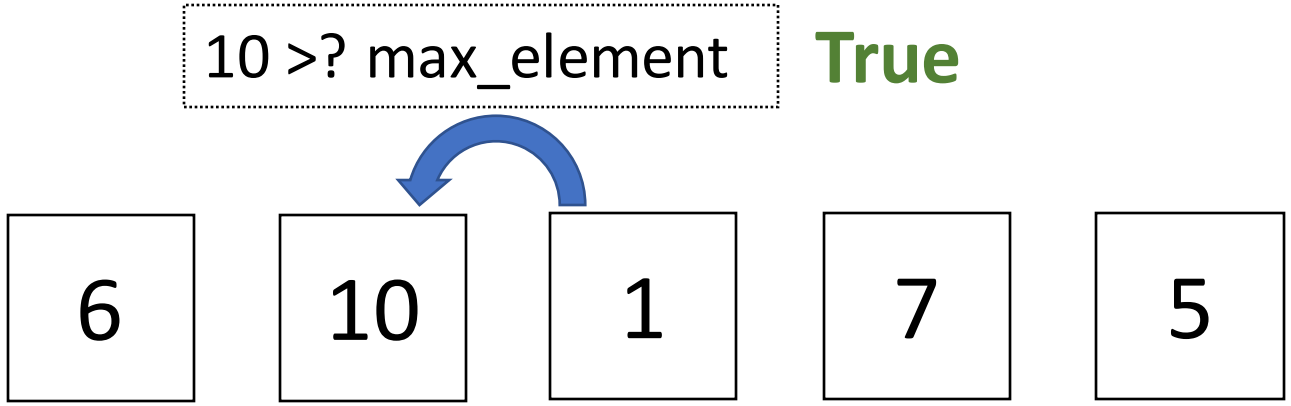


max_element = 7

max_difference = 6

Solution 2

A solution with O(n) complexity:

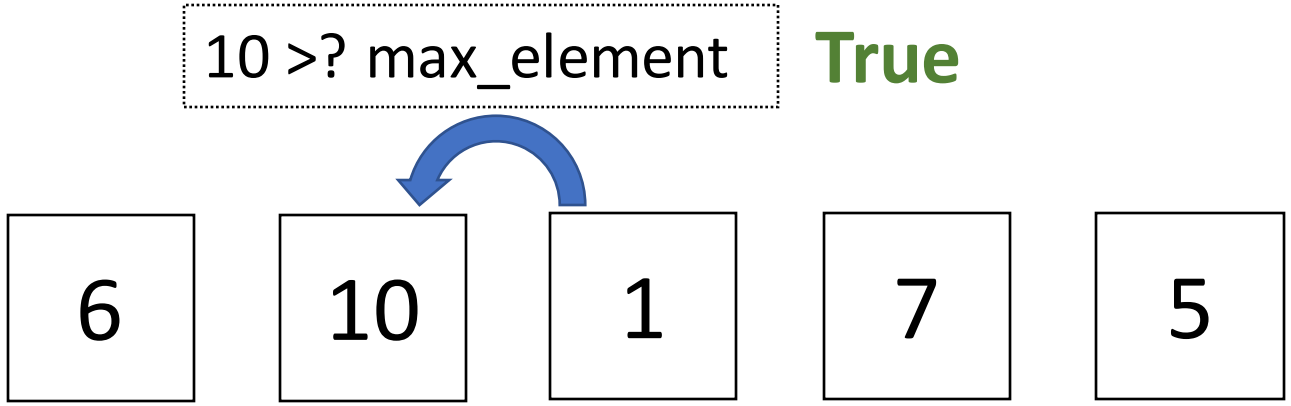


max_element = 7

max_difference = 6

Solution 2

A solution with O(n) complexity:

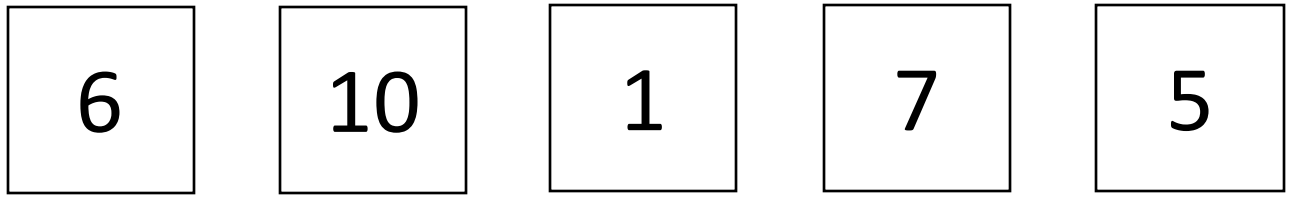


max_element = current_element = 10

max_difference = 6

Solution 2

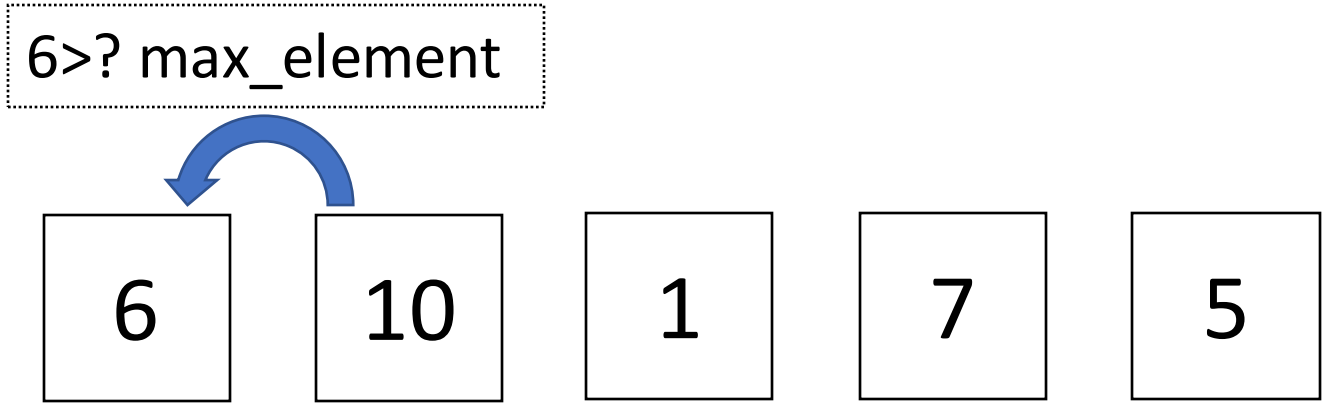
A solution with O(n) complexity:



max_element = 10
max_difference = 6

Solution 2

A solution with O(n) complexity:



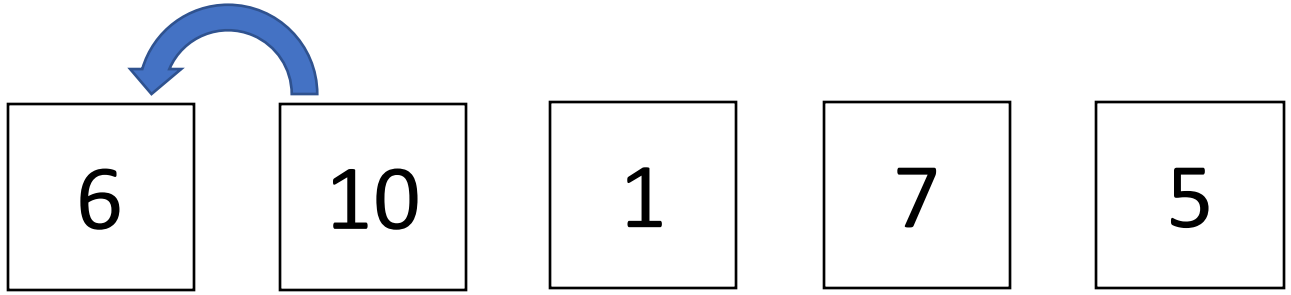
max_element = 10

max_difference = 6

Solution 2

A solution with O(n) complexity:

6>? max_element **False**



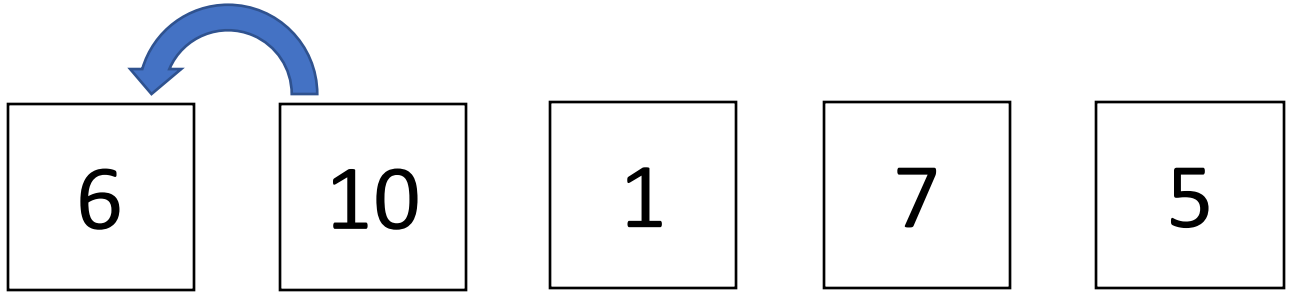
max_element = 10

max_difference = 6

Solution 2

A solution with O(n) complexity:

6>? max_element **False**



max_element = 10

max_difference = max(max_difference, max_element-current_element)

= max(6, 10 – 6) = max(6, 4) = 6