SORTING ALGORITHMS

INSERTION SORT

Insertion sort

- It is a simple sorting algorithm that builds the final sorted array one item at a time
- It has quadratic running time **O(N^2)**
- On large datasets it is very inefficient but on arrays with **10-20** items it is quite good
- Simple implementation !!!
- It is more efficient than other quadratic running time sorting procedures such as bubble sort or selection sort
- Adaptive algorithm → speeds up when array is already substantially sorted
- Stable sort → preserve the order of the items with equal keys

Insertion sort

- In-place algorithm → does not need any additional memory
- It is an online algorithm → it can sort an array as it receives it for example downloading data from web
- Hybrid algorithms uses insertion sort if the subarray is small enough

  Insertion sort is faster for small subarrays than quicksort !!!
- Variant of insertion sort is shell sort
- Sometimes selection sort is better: they are very similar algorithms
- Insertion sort requires more writes because the inner loop can require shifting large sections of the sorted portion of the array
- In general, insertion sort will write to the array **O(n2)** times while selection sort will write only **O(n)** times
- For this reason selection sort may be preferable in cases where writing to memory is significantly more expensive than reading → for example flash

Pseudocode

```
insertionSort(array)

    for i=1 to length(array)
        j = i

        while j > 0 and array[j-1] > array[j]
            swap( array, j, j-1 )
            j = j – 1
        end
    end
end
```

Pseudocode

```
insertionSort(array)

    for i=1 to length(array)
        j = i

        while j > 0 and array[j-1] > array[j]
            swap( array, j, j-1 )
            j = j – 1
        end
    end
end
```

We iterate through the array sequentially !!!

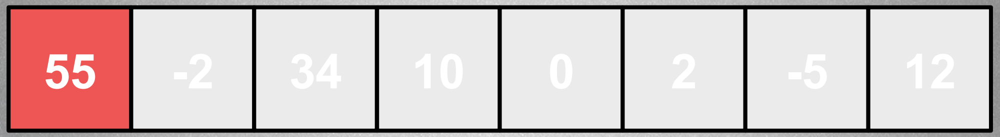
Pseudocode

```
insertionSort(array)

    for i=1 to length(array)
        j = i

        while j > 0 and array[j-1] > array[j]
            swap( array, j, j-1 )
            j = j – 1
        end
    end
end
```

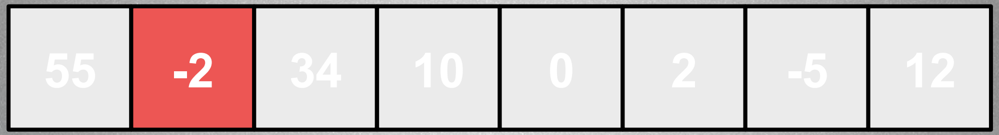
While the previous item is greater than the given one, we keep swapping them !!!

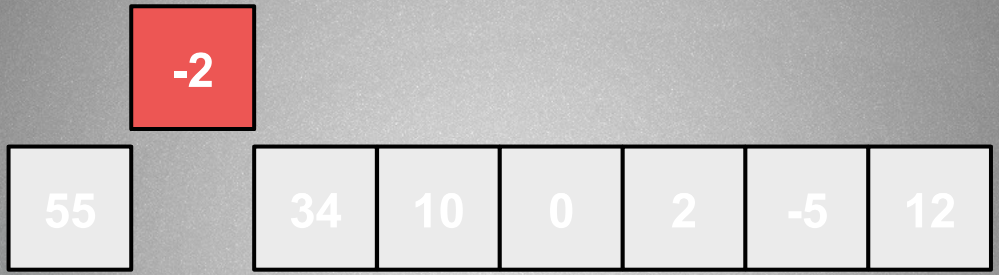
~ thats why there are so many shifts in insertion sort

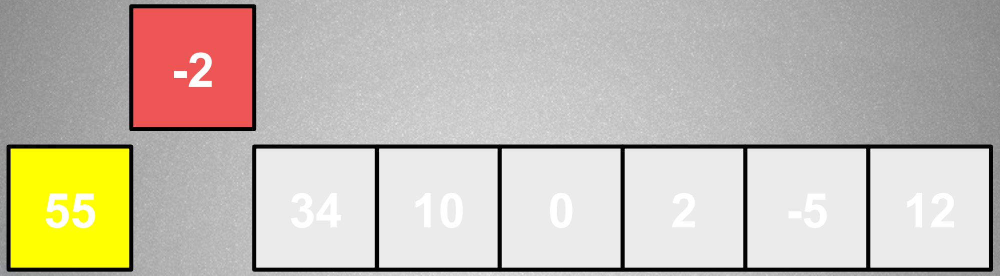
55	-2	34	10	0	2	-5	12

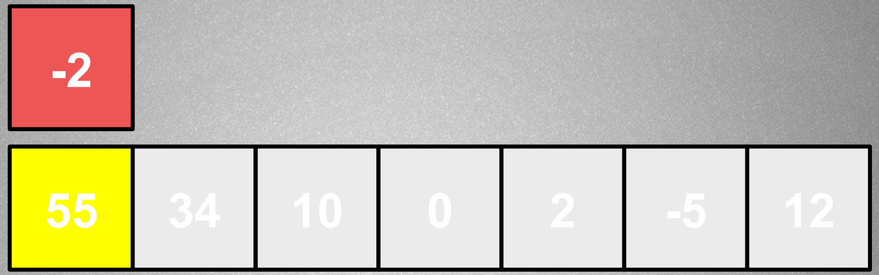

55
-2
34
10
0
2
-5
12

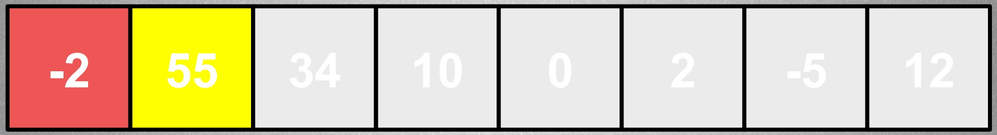
55	-2	34	10	0	2	-5	12


55
-2
34
10
0
2
-5
12

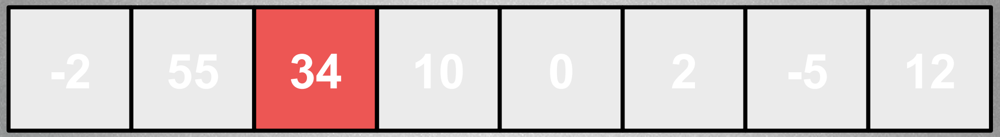

-2
55
34
10
0
2
-5
12

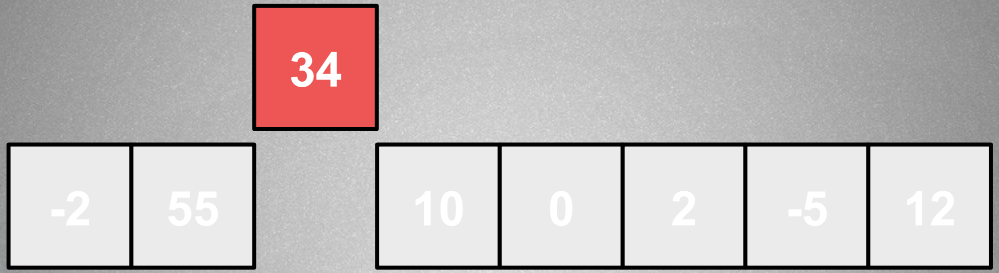

-2
55
34
10
0
2
-5
12

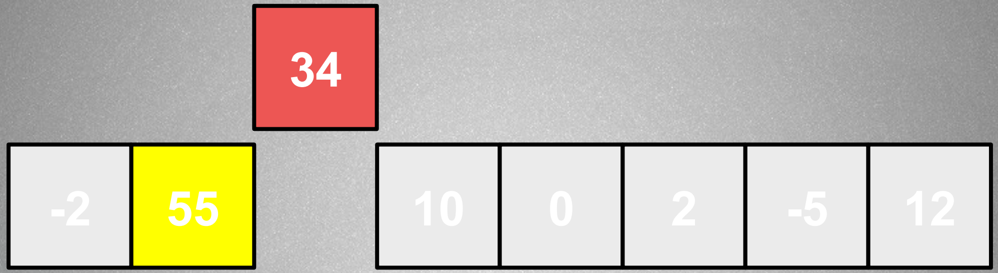

-2
55
34
10
0
2
-5
12

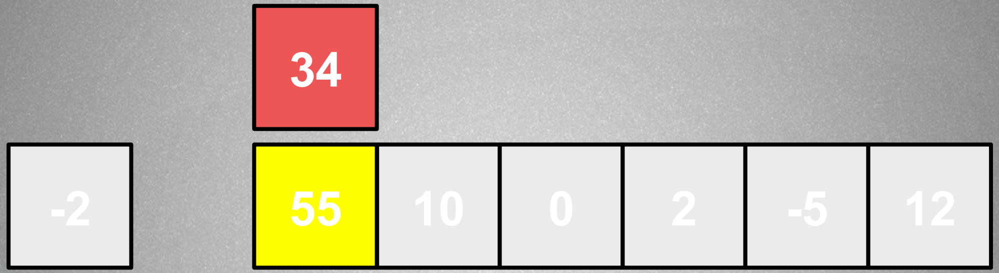

-2
55
34
10
0
2
-5
12

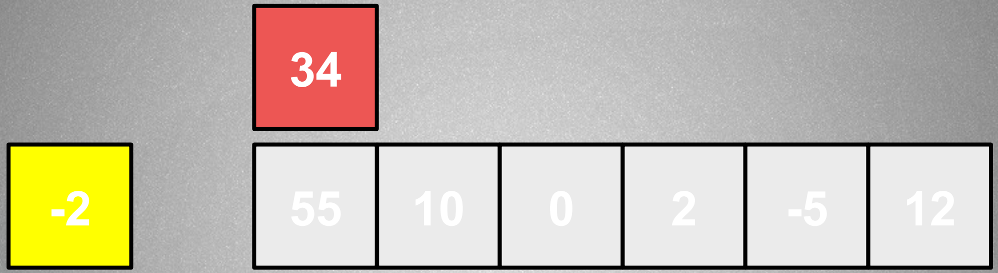
-2	55	34	10	0	2	-5	12

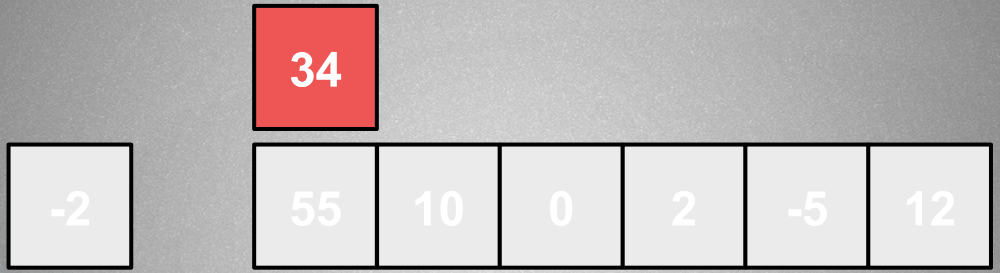

-2
55
34
10
0
2
-5
12

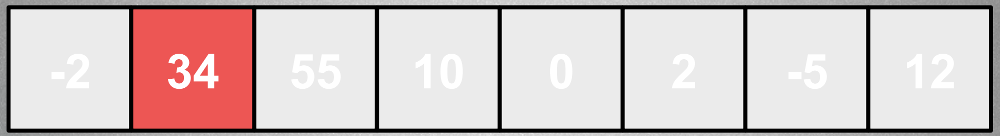

34
-2
55
10
0
2
-5
12


34
-2
55
10
0
2
-5
12


34
-2
55
10
0
2
-5
12

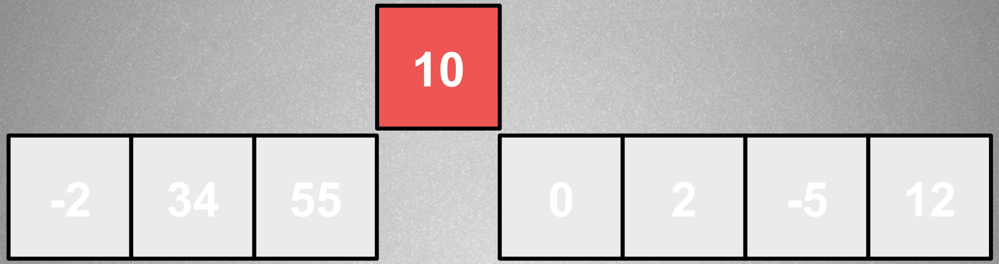

34
-2
55
10
0
2
-5
12

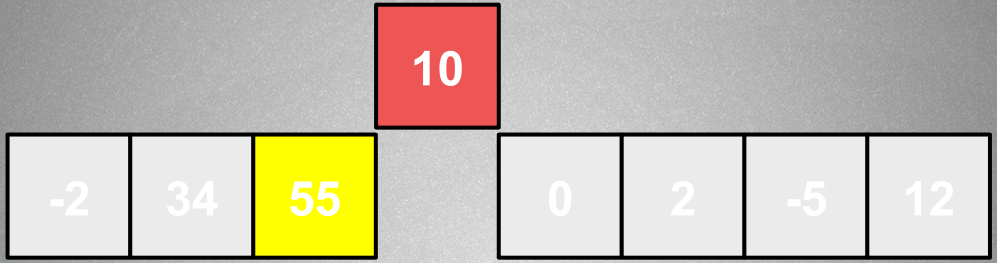

34
-2
55
10
0
2
-5
12

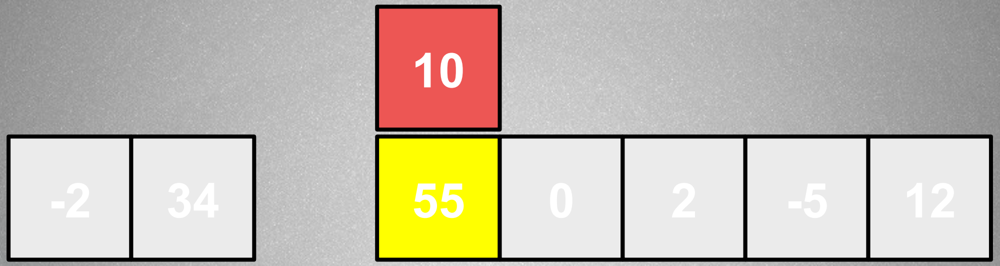

-2
34
55
10
0
2
-5
12

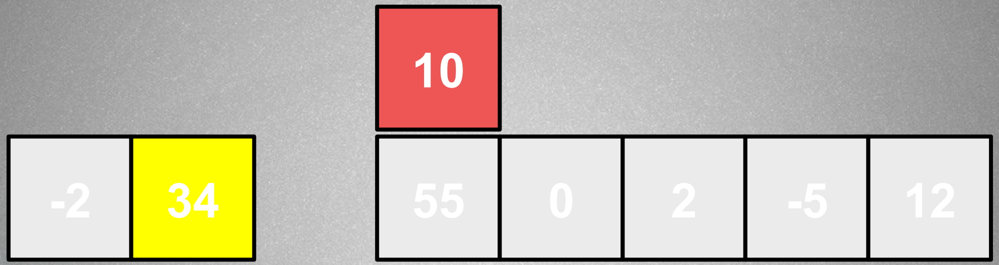
-2	34	55	10	0	2	-5	12

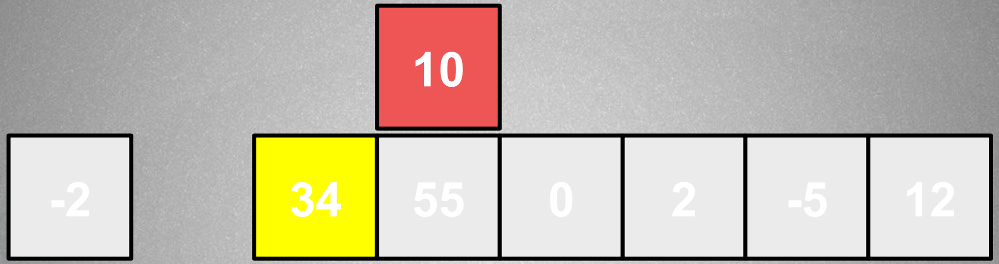
-2	34	55	10	0	2	-5	12

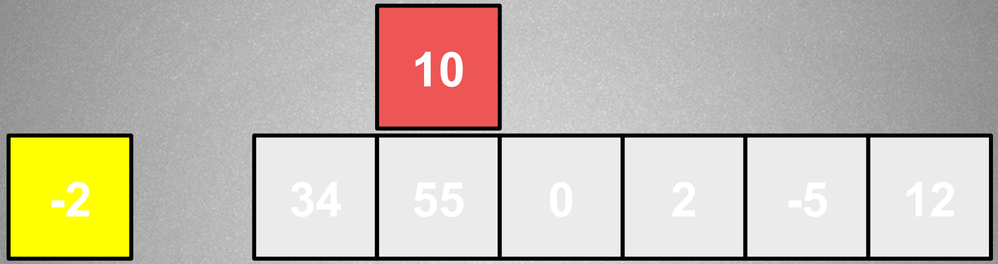

10
-2
34
55
0
2
-5
12

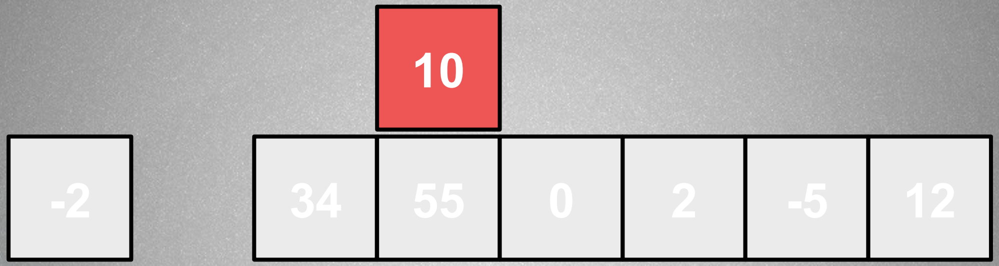

10
-2
34
55
0
2
-5
12

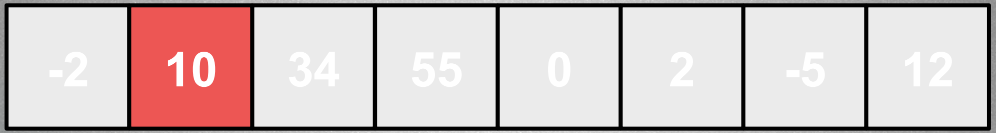

10
-2
34
55
0
2
-5
12


10
-2
34
55
0
2
-5
12


10
-2
34
55
0
2
-5
12

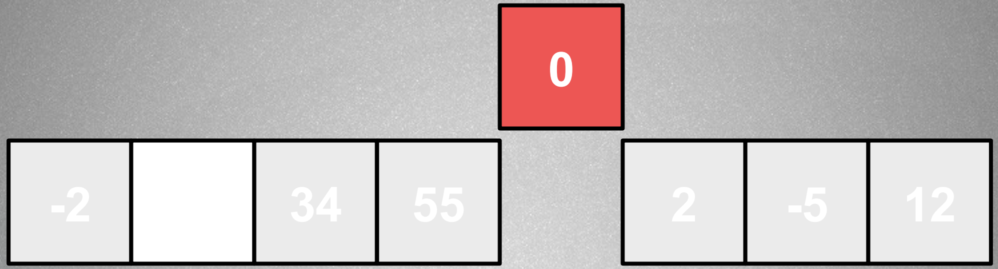

10
-2
34
55
0
2
-5
12

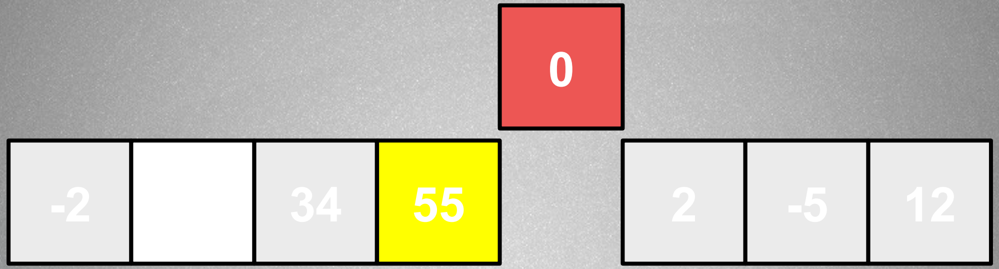

10
-2
34
55
0
2
-5
12

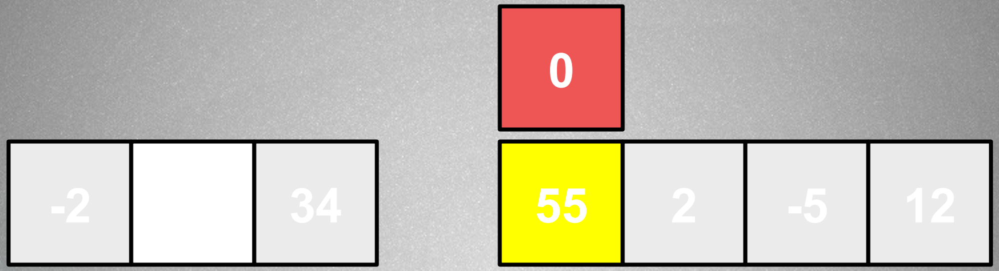

-2
10
34
55
0
2
-5
12

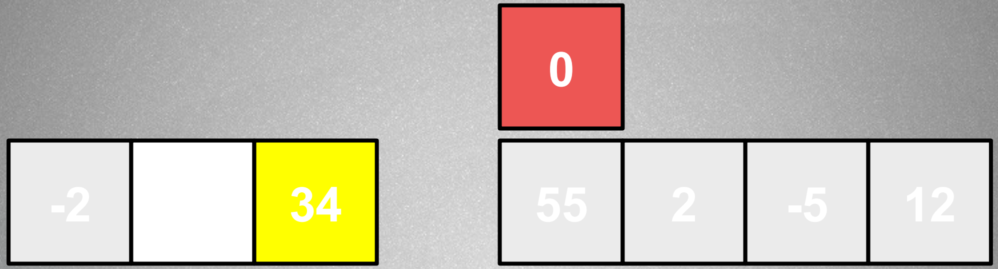
-2		34	55	0	2	-5	12

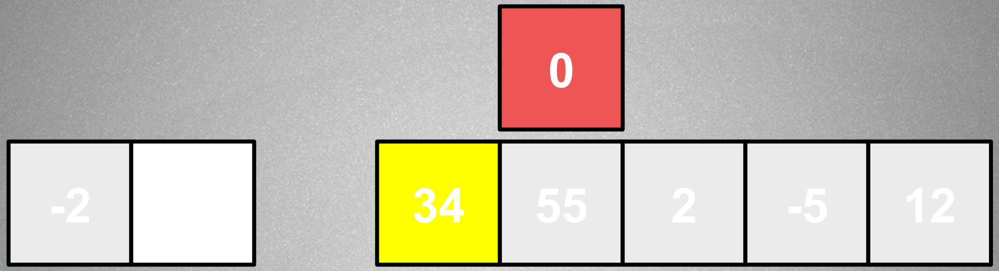
-2		34	55	0	2	-5	12

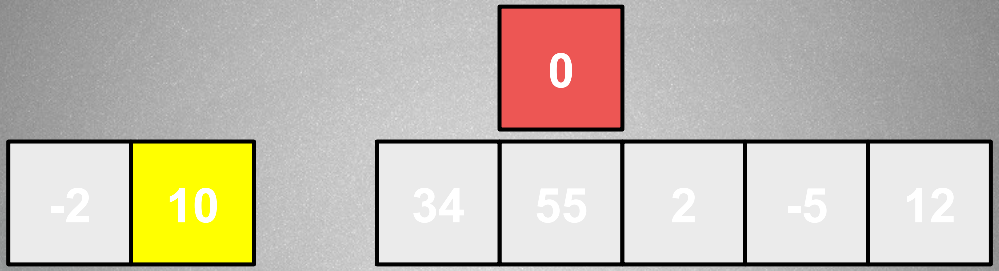

0
-2
34
55
2
-5
12

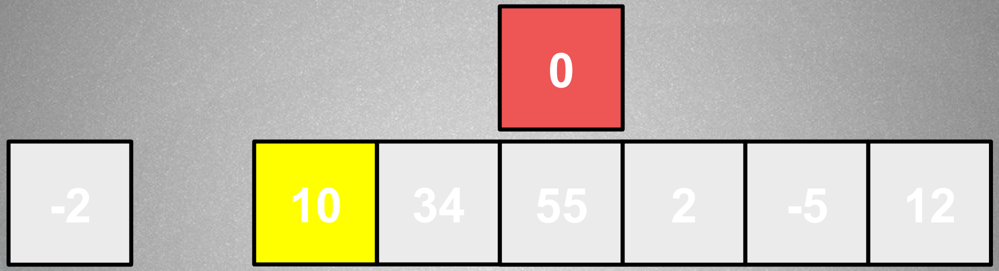

0
-2
34
55
2
-5
12

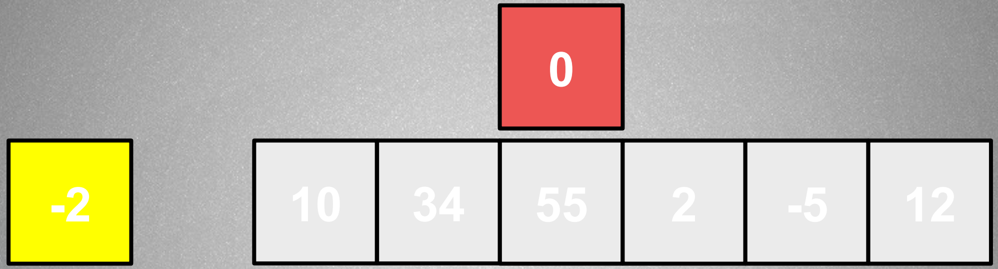

0
-2
34
55
2
-5
12

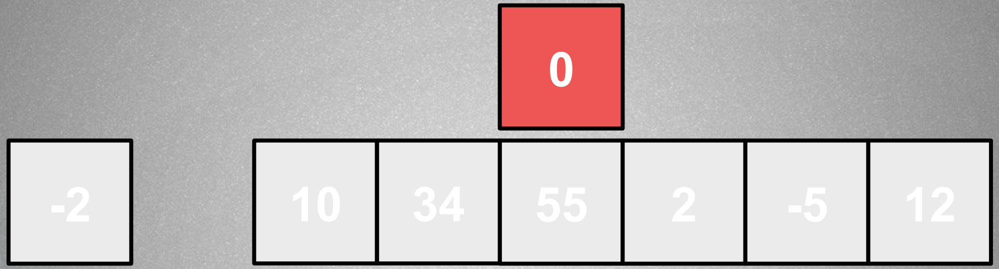

0
-2
34
55
2
-5
12

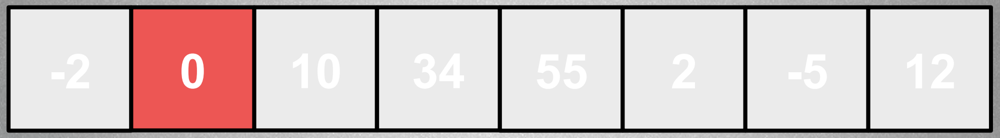

0
-2
34
55
2
-5
12


0
-2
10
34
55
2
-5
12

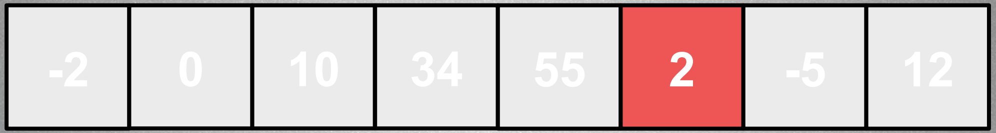

0
-2
10
34
55
2
-5
12

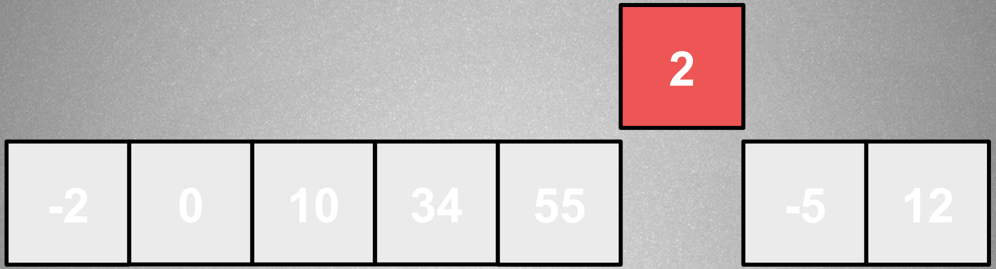

0
-2
10
34
55
2
-5
12

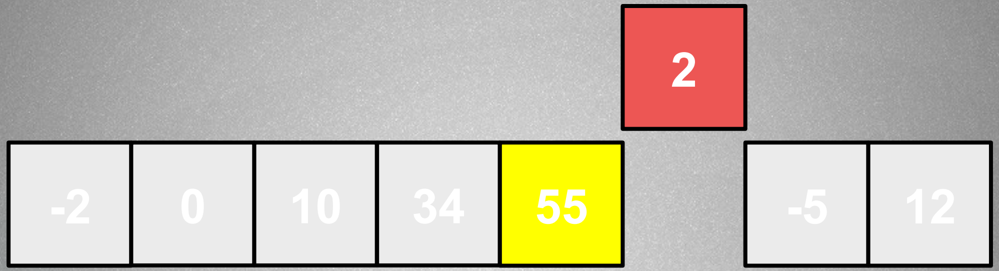

0
-2
10
34
55
2
-5
12

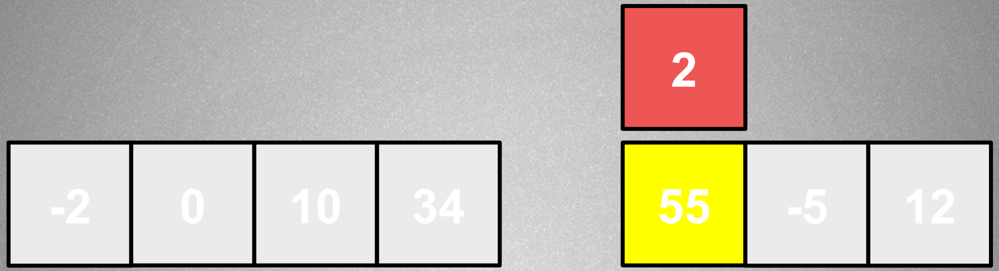

-2
0
10
34
55
2
-5
12

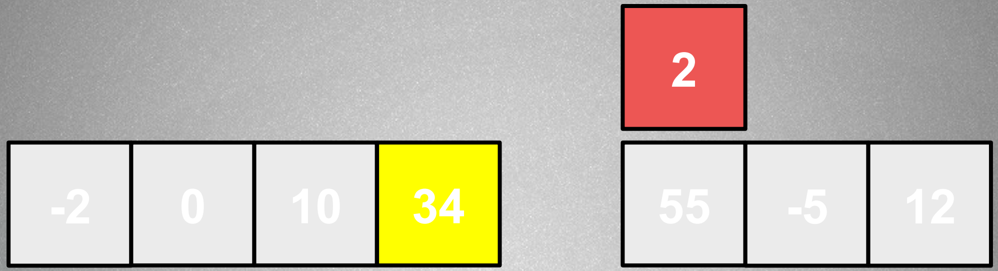
-2	0	10	34	55	2	-5	12

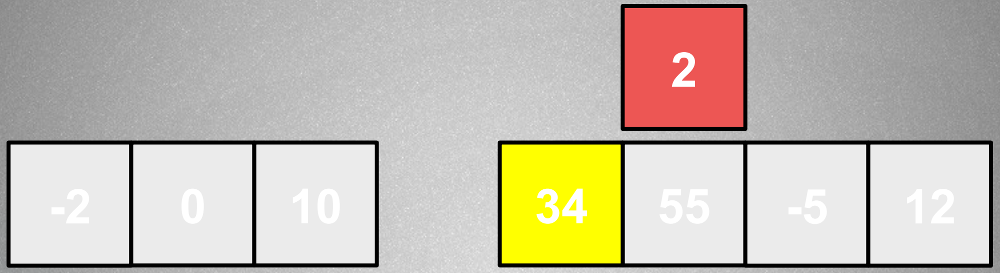

-2
0
10
34
55
2
-5
12

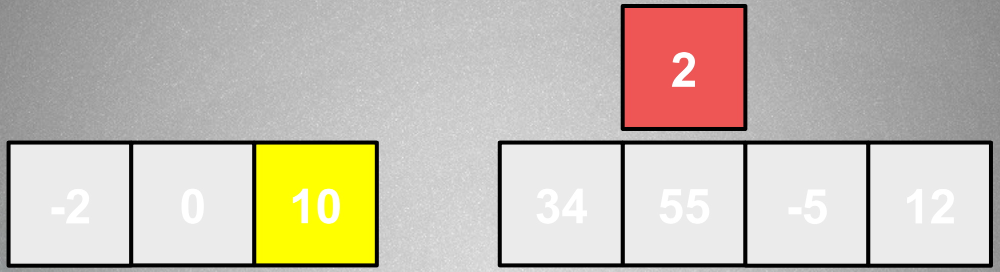

2
-2
0
10
34
55
-5
12

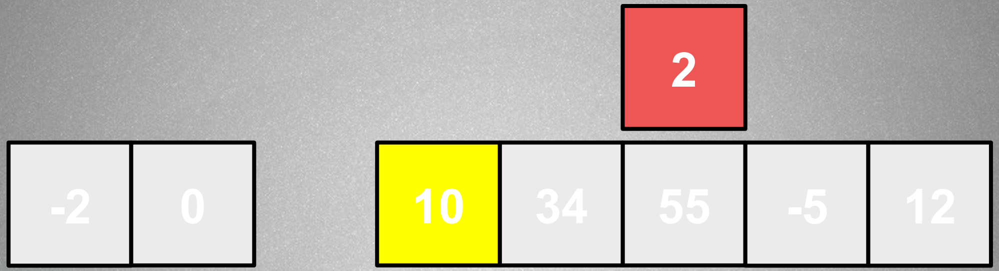

2
-2
0
10
34
55
-5
12

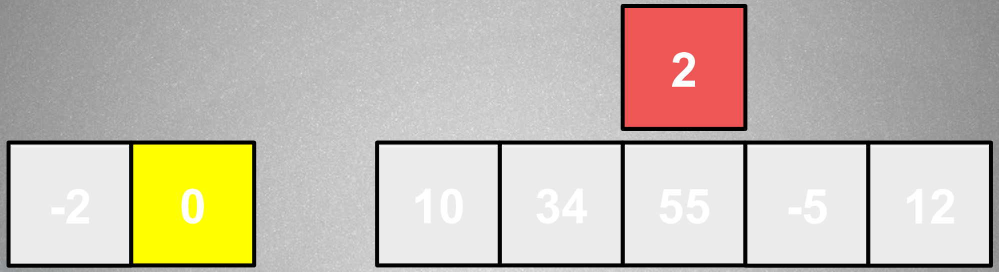

2
-2
0
10
34
55
-5
12

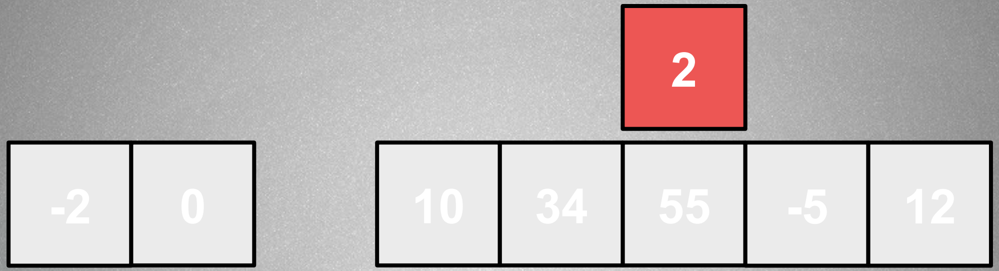

2
-2
0
10
34
55
-5
12

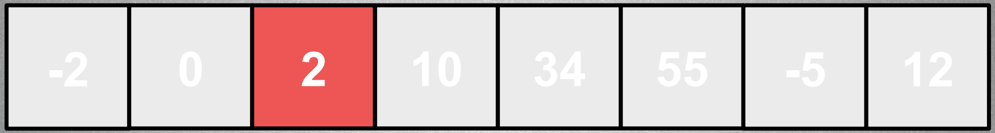

2
-2
0
10
34
55
-5
12


2
-2
0
10
34
55
-5
12

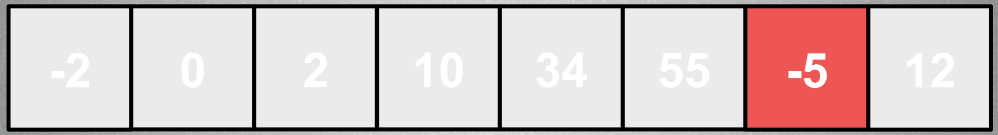

2
-2
0
10
34
55
-5
12

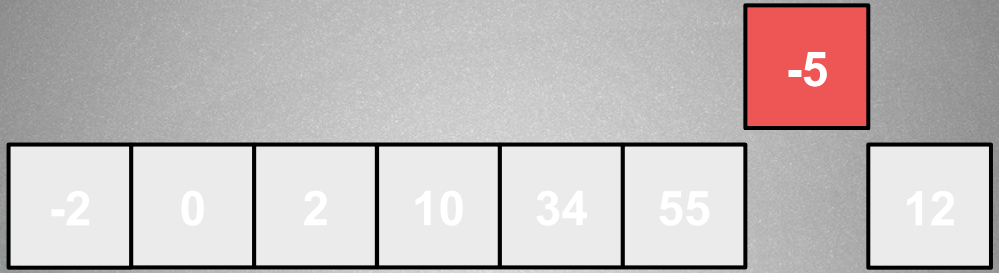

2
-2
0
10
34
55
-5
12

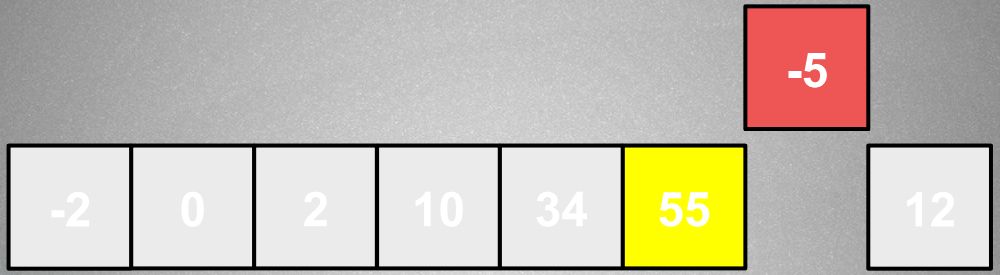

2
-2
0
10
34
55
-5
12

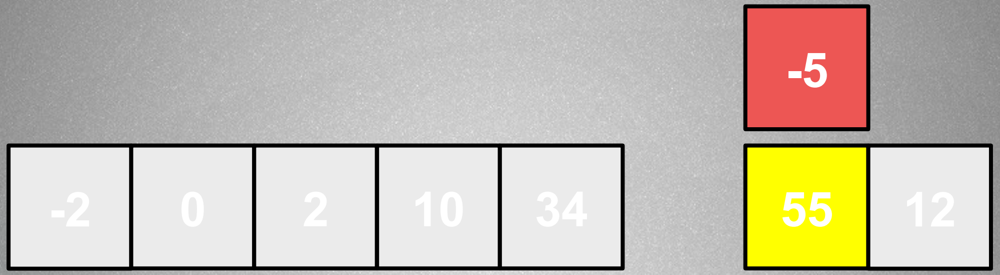

-2
0
2
10
34
55
-5
12

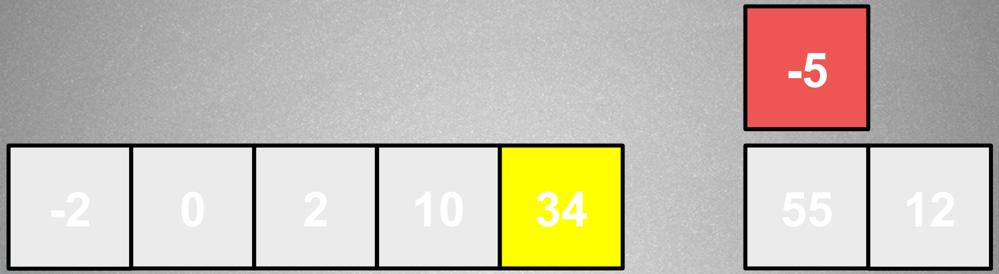
-2	0	2	10	34	55	-5	12

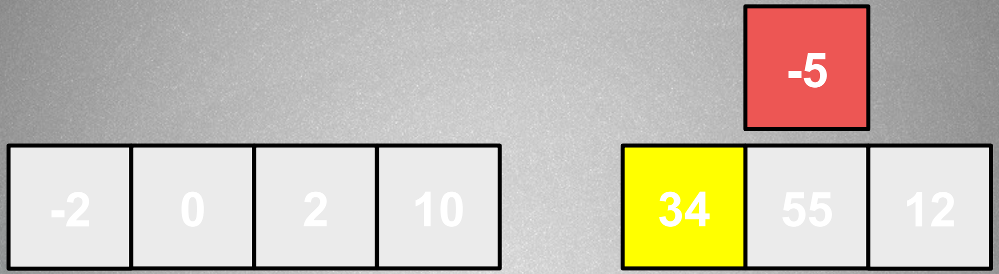

-2
0
2
10
34
55
-5
12

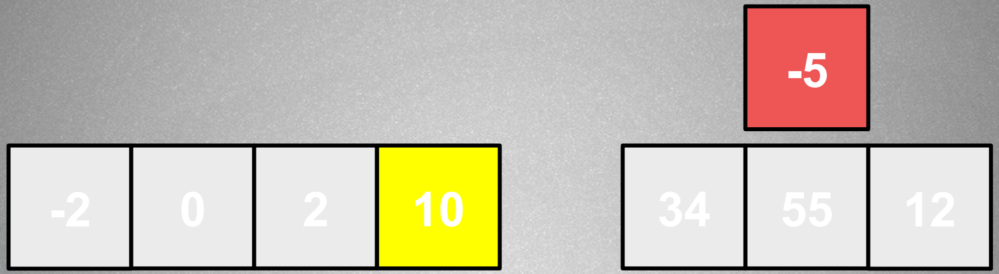

-5
-2
0
2
10
34
55
12

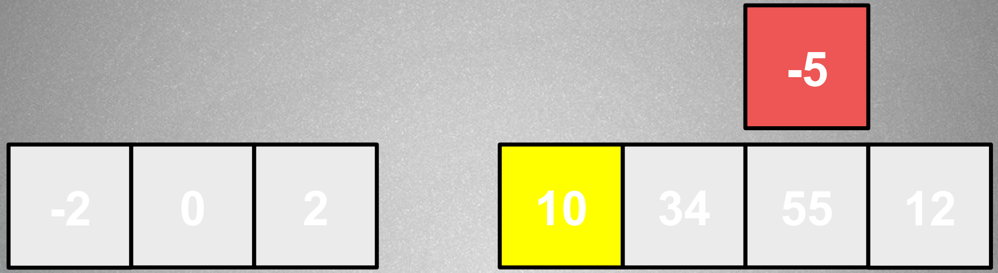

-5
-2
0
2
10
34
55
12

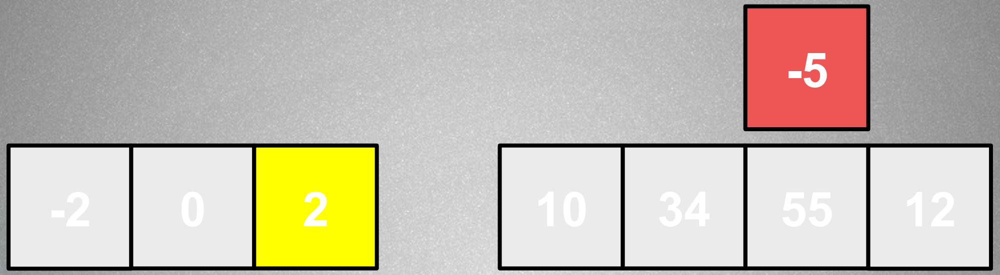

-5
-2
0
2
10
34
55
12

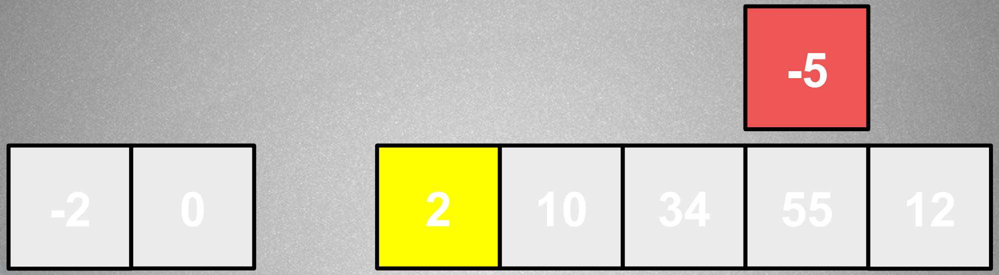

-5
-2
0
2
10
34
55
12

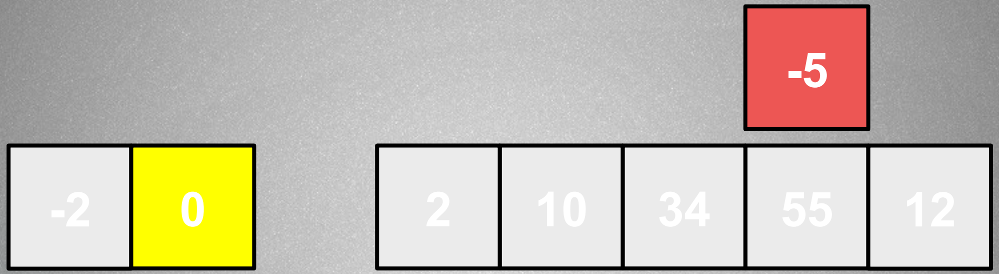

-5
-2
0
2
10
34
55
12

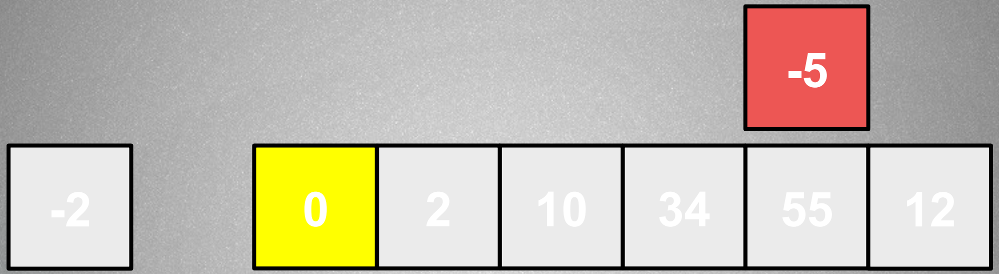

-5
-2
0
2
10
34
55
12

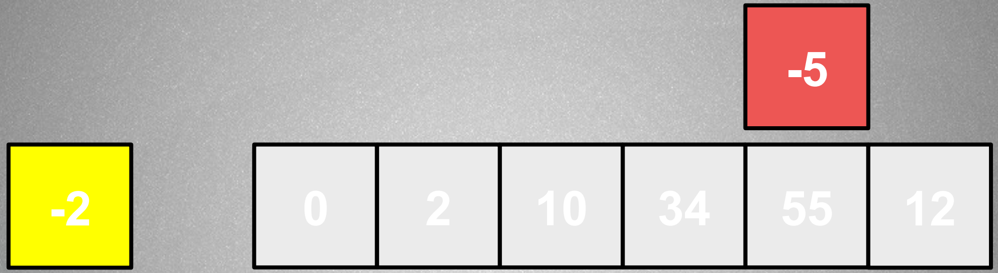

-5
-2
0
2
10
34
55
12

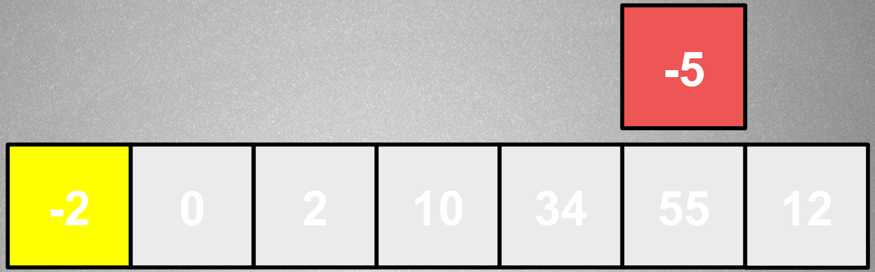

-5
-2
0
2
10
34
55
12

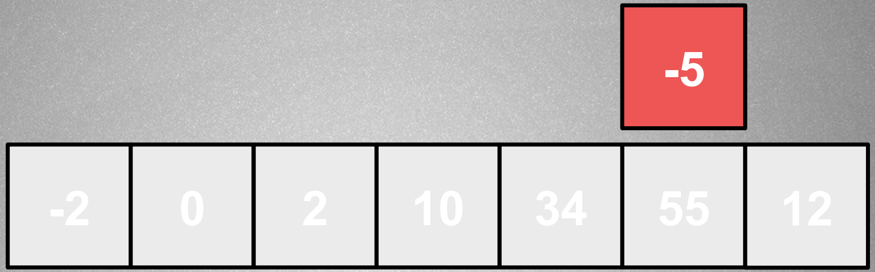

-5
-2
0
2
10
34
55
12

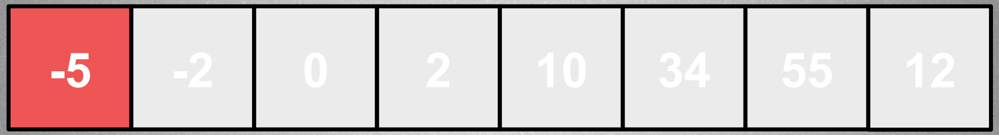

-5
-2
0
2
10
34
55
12


-5
-2
0
2
10
34
55
12

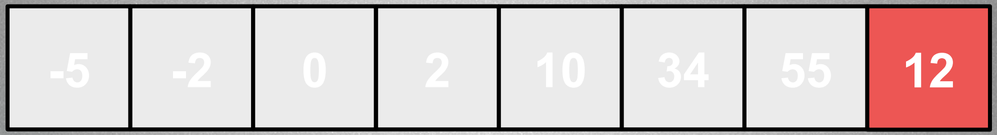

-5
-2
0
2
10
34
55
12

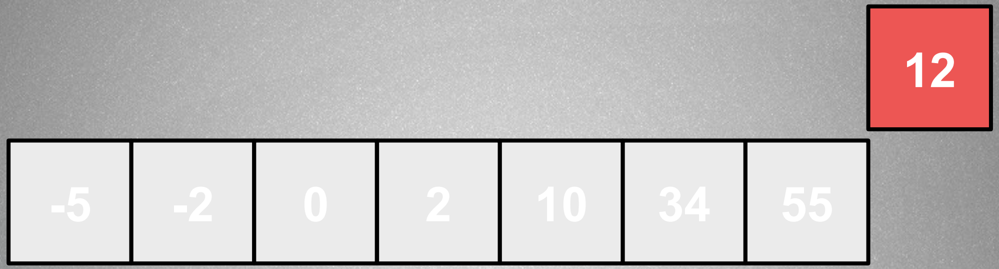

-5
-2
0
2
10
34
55
12

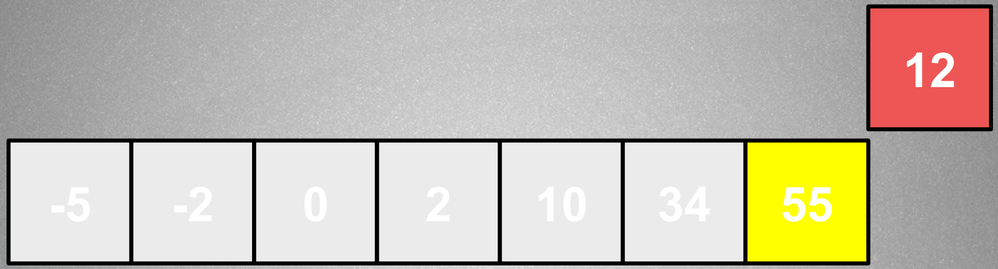

-5
-2
0
2
10
34
55
12

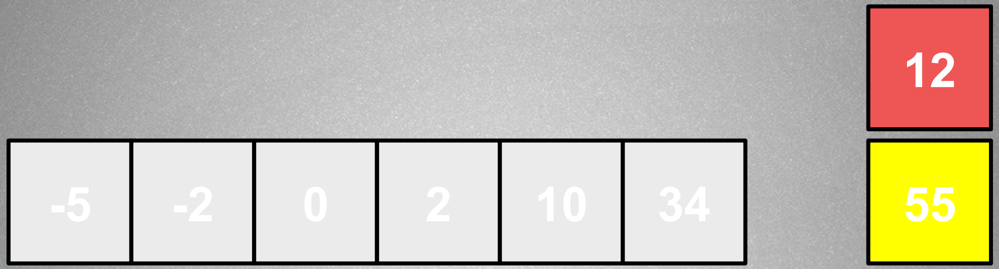

-5
-2
0
2
10
34
55
12

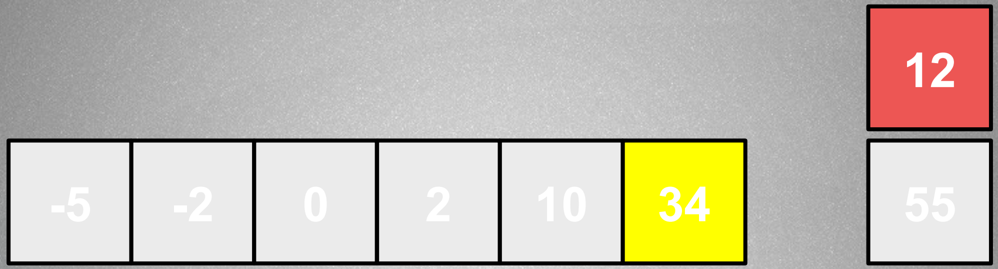
-5	-2	0	2	10	34	55	12

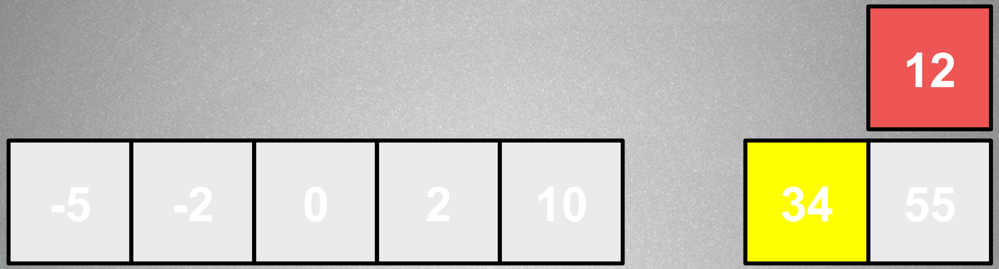

-5
-2
0
2
10
34
55
12

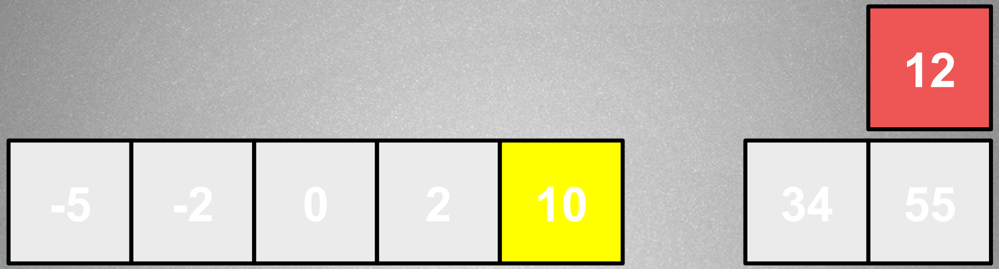

12
-5
-2
0
2
10
34
55

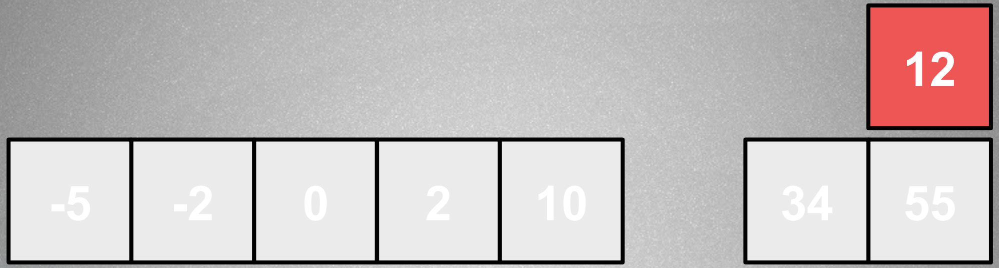

12
-5
-2
0
2
10
34
55

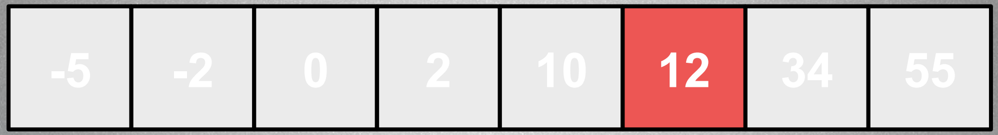

-5
-2
0
2
10
34
12
55

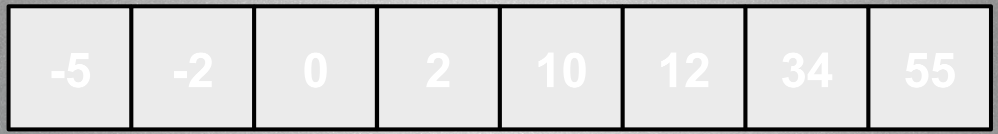

-5
-2
0
2
10
34
12
55

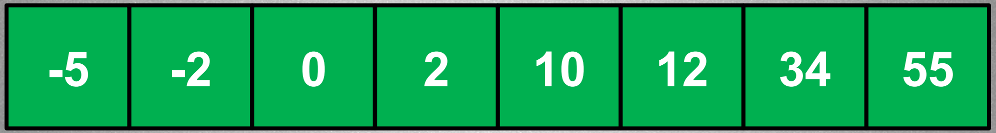

12
-5
-2
0
2
10
34
55


12
-5
-2
0
2
10
34
55


12
-5
-2
0
2
10
34
55


-5
-2
0
2
10
12
34
55


-5
-2
0
2
10
12
34
55


-5
-2
0
2
10
12
34
55