# Week3_ARRAYS

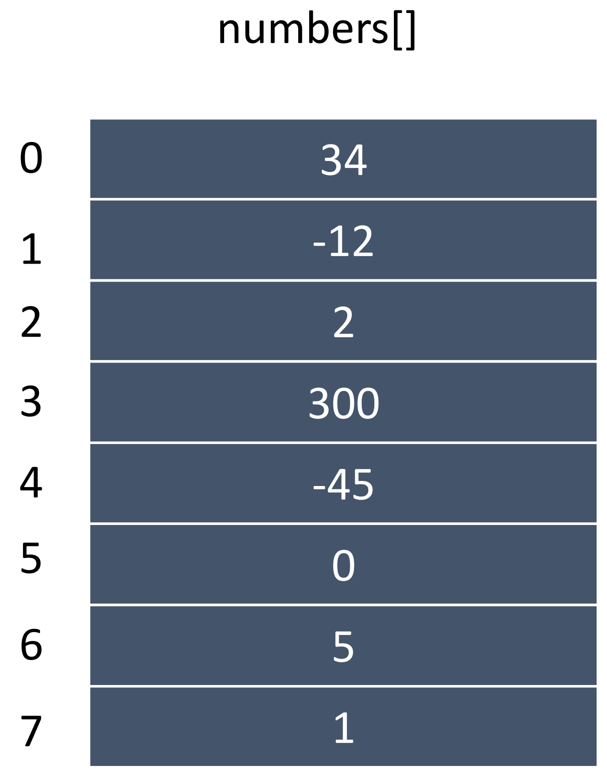
Dr. Ahmad Al Shami

August 2020

## **Arrays:** data structures

A collection of elements / values
each identified by an array index or key !!!

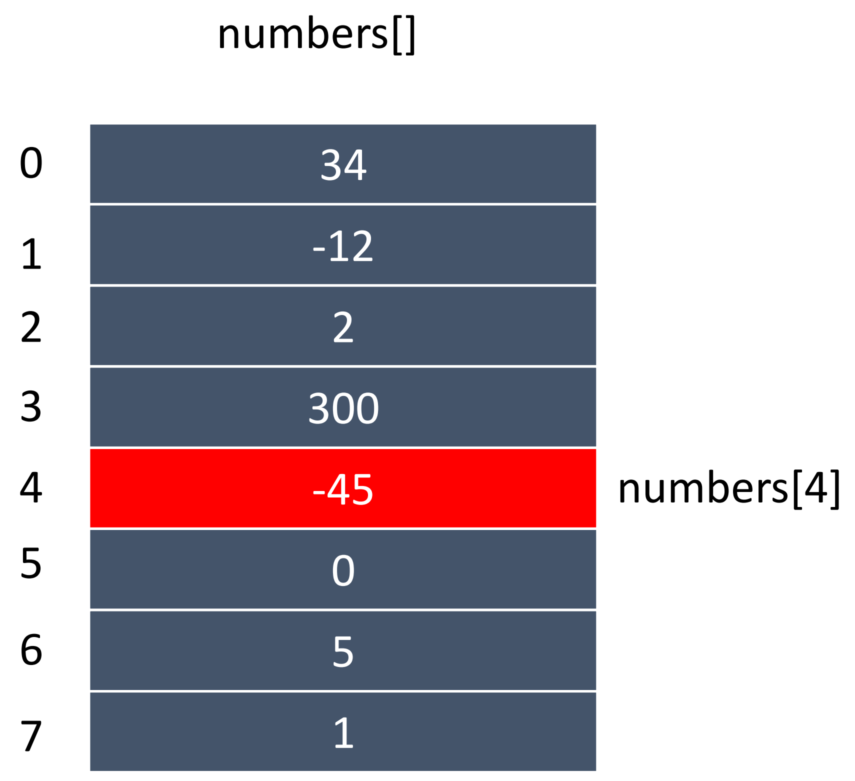
- index starts at zero
- because of the indexes: random access is possible



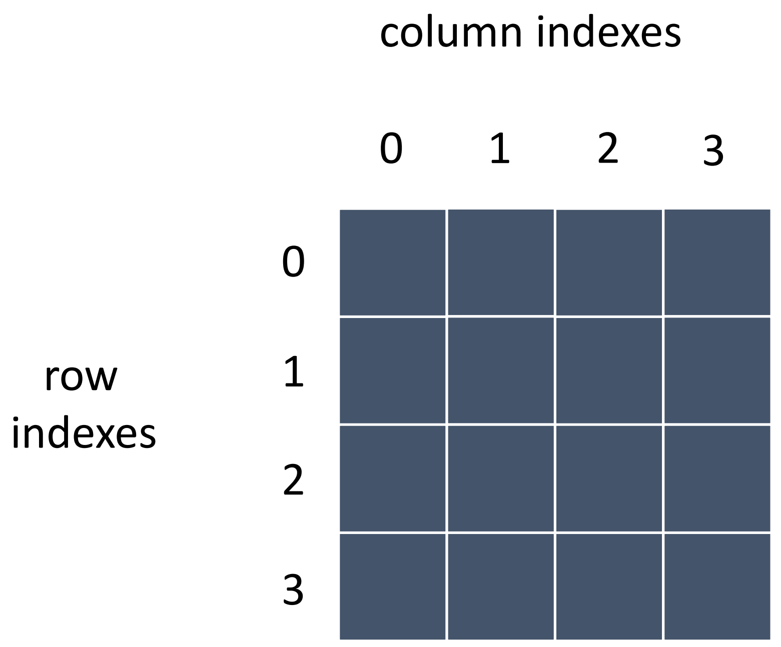
## **Arrays:** data structure

A collection of elements / values
each identified by an array index or key !!!

- index starts at zero
- because of the indexes: random access is possible



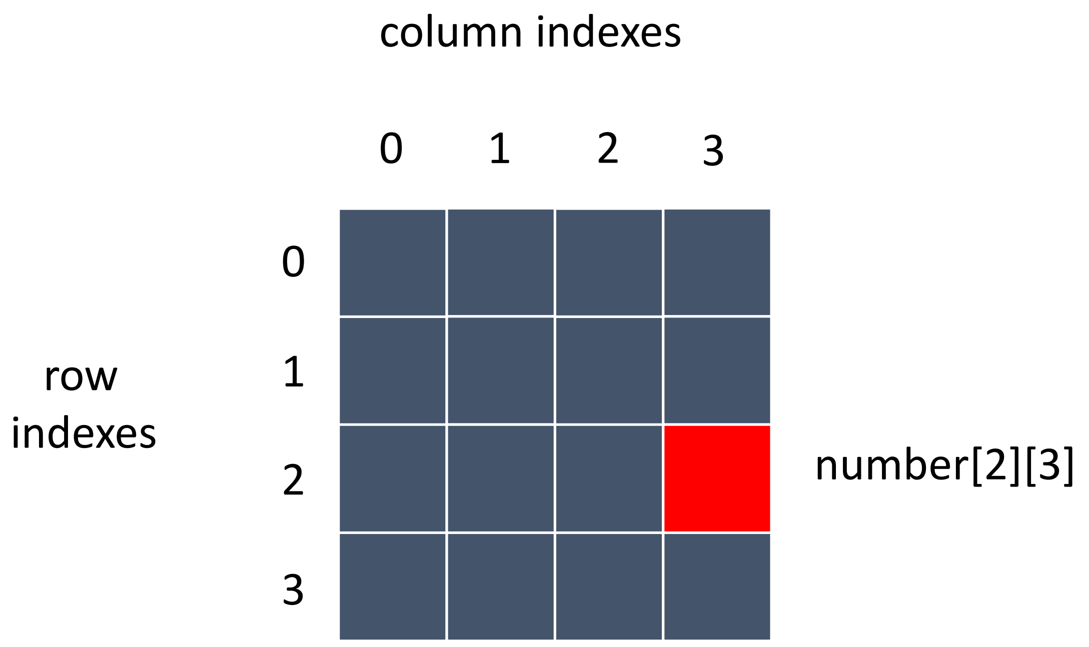
**Multidimensional arrays**: it can prove to be very important in mathematical related computations ( matrixes )



numbers[][]  two dimensional array

First paramter: row index
Second parameter: column index

**Multidimensional arrays**: it can prove to be very important in mathematical related computations ( matrixes )



numbers[][]  two dimensional array

First paramter: row index
Second parameter: column index

## Arrays

- Arrays are data structures in order to store items of the same type
- We use indices as keys !!!
- Arrays can have as many dimensions as we want: one or two dimensional arrays are quite popular
- For example: storing a matrix → two dimensional array
- Dynamic array: when the size of the array is changing dynamically
- Applications: lookup tables / hashtables, heaps

## Advantages

- We can use random access because of the keys: getItem(int index) will return the value with the given key very fast // **O(1)**
- Very easy to implement and to use
- Very fast data structure
- We should use arrays in applications when we want to add items over and over again and we want to take items with given indexes!!! ~ it will be fast

## Disadvantages

- We have to know the size of the array at compile-time: so it is not so dynamic data structure
- If it is full: we have to create a bigger array and have to copy the values one by one // reconstructing an array is **O(N)** operation
- It is not able to store items with different types

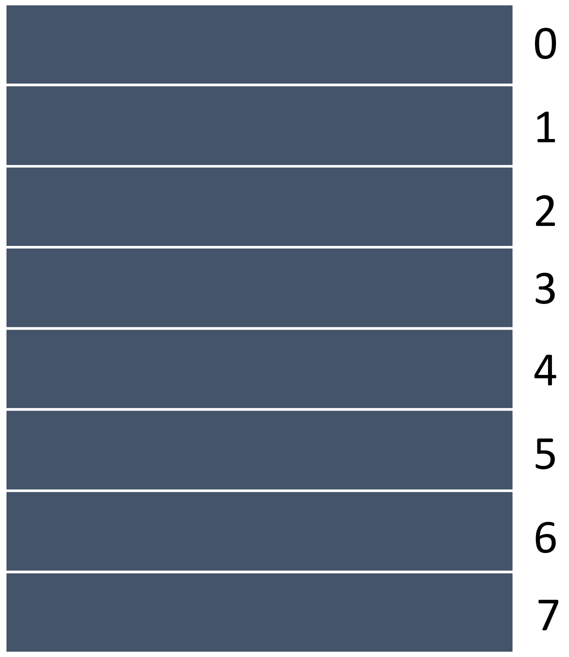
## Arrays operation: add

We can keep adding values to the end of the array as far as the array is not full

**Arrays operation**: add

We can keep adding values to the array as far as the array is not full

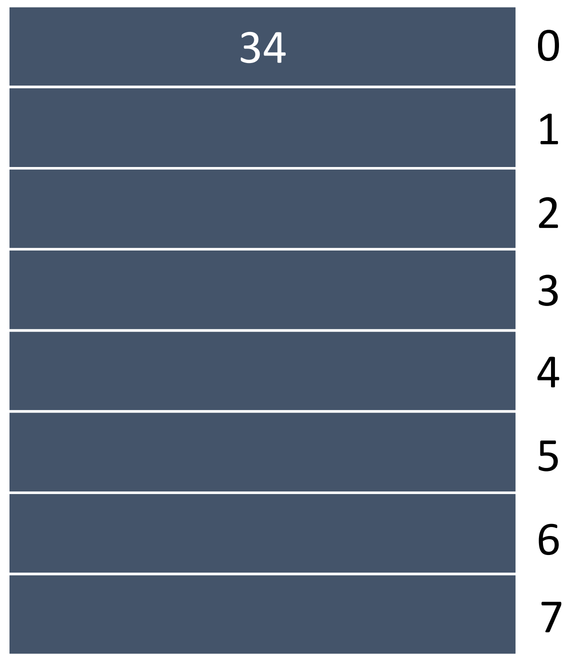
add(34)



**Arrays operation**: add

We can keep adding values to the array as far as the array is not full

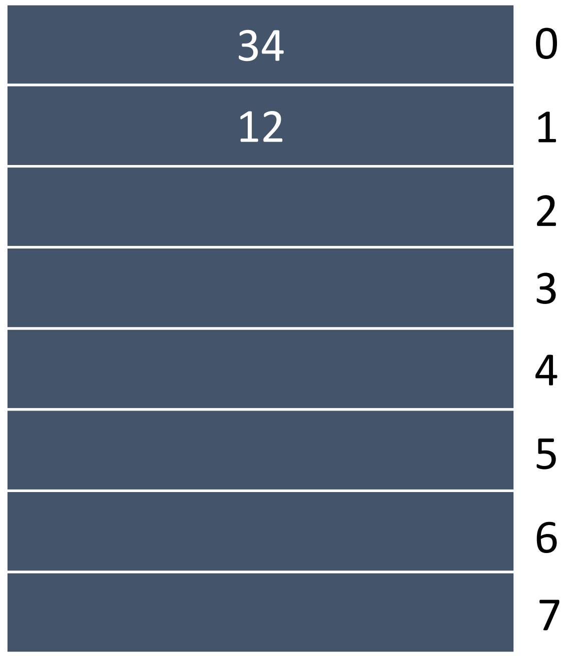
add(34)



**Arrays operation**: add

We can keep adding values to the array as far as the array is not full

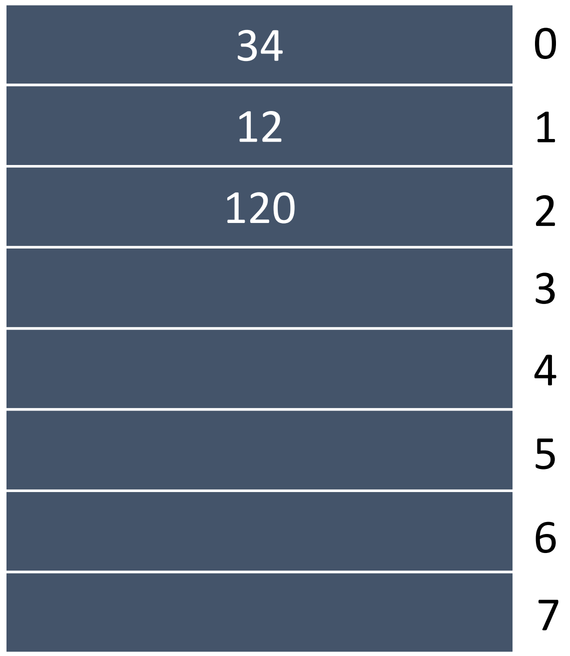
add(12)



## Arrays operation: add

We can keep adding values to the array as far as the array is not full

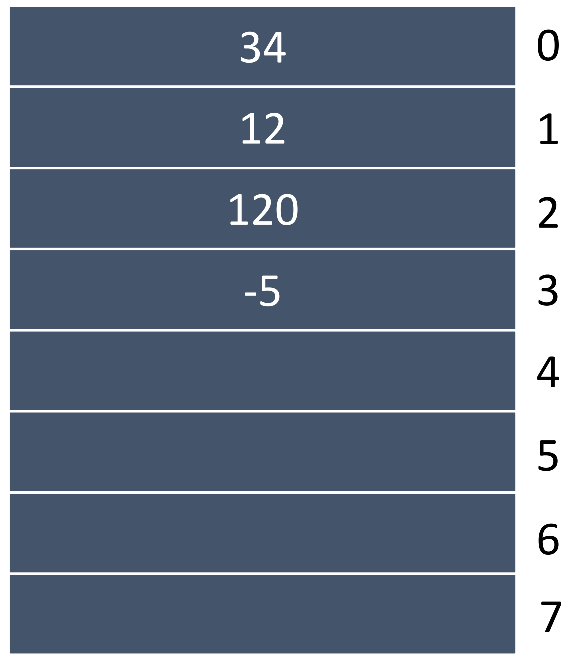
add(120)



## **Arrays operation**: add

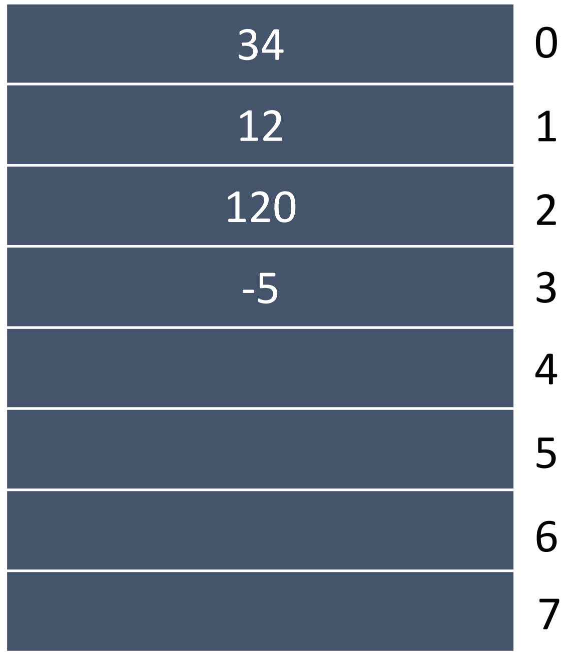
We can keep adding values to the array as far as the array is not full

add(-5)



## **Arrays operation**: add

We can keep adding values to the array as far as the array is not full

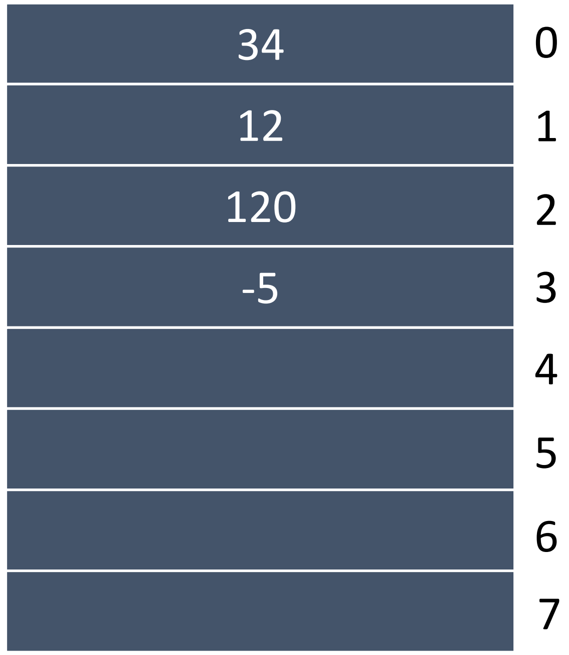


So: when adding new values to the list, we just have to insert it with the next index → very fast **O(1)** operation

## Arrays operation: insert item

We would like to insert a given value with a given index

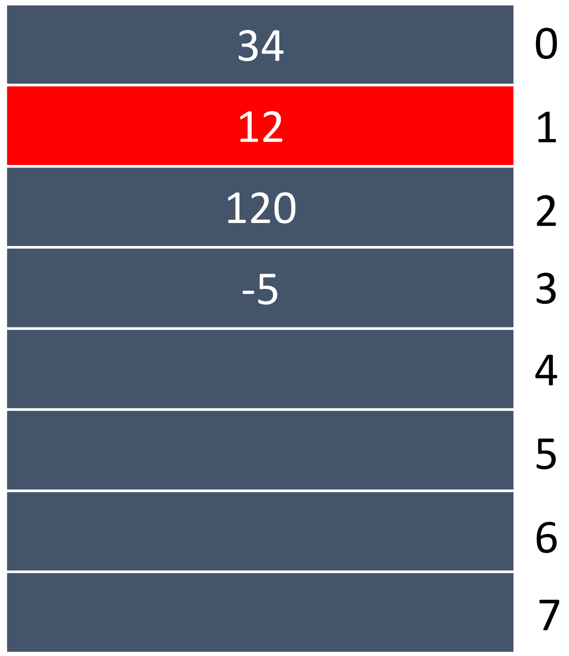
```
insert(23,1);
```



## **Arrays operation**: insert item

We would like to insert a given value with a given index

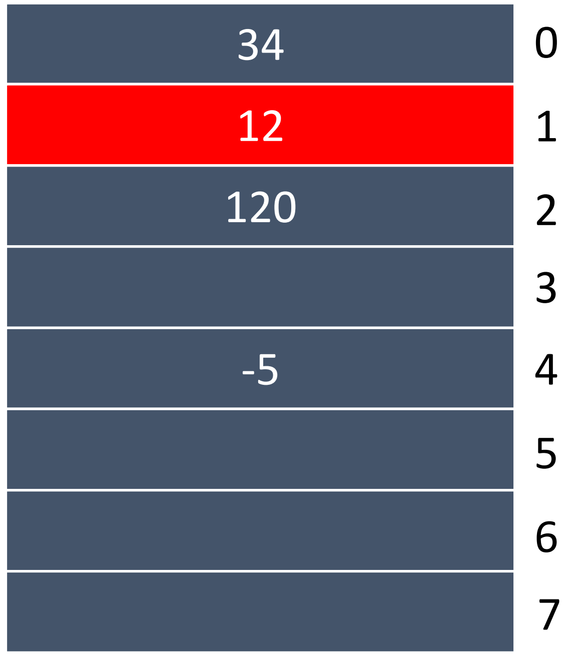
```
insert(23,1);
```



**Arrays operation**: insert item

We would like to insert a given value with a given index

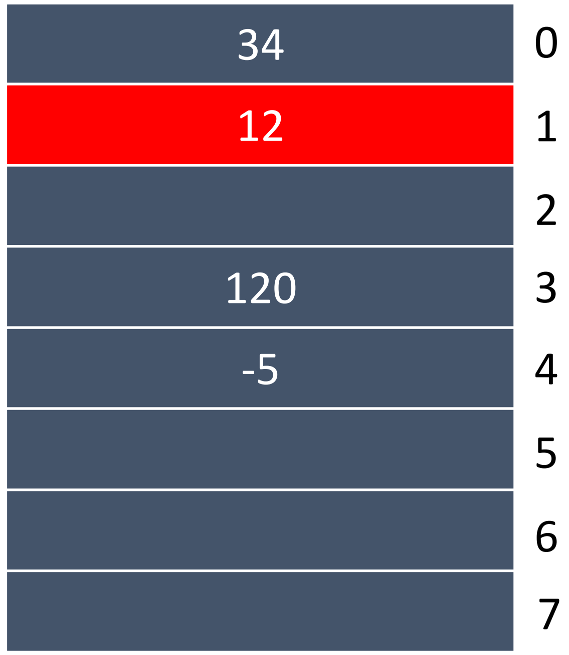
insert(23,1);



## **Arrays operation**: insert item

We would like to insert a given value with a given index

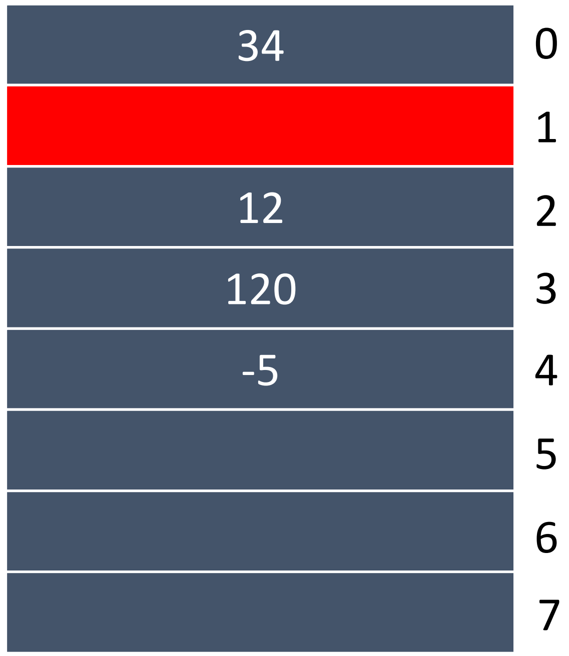
insert(23,1);



**Arrays operation**: insert item

We would like to insert a given value with a given index

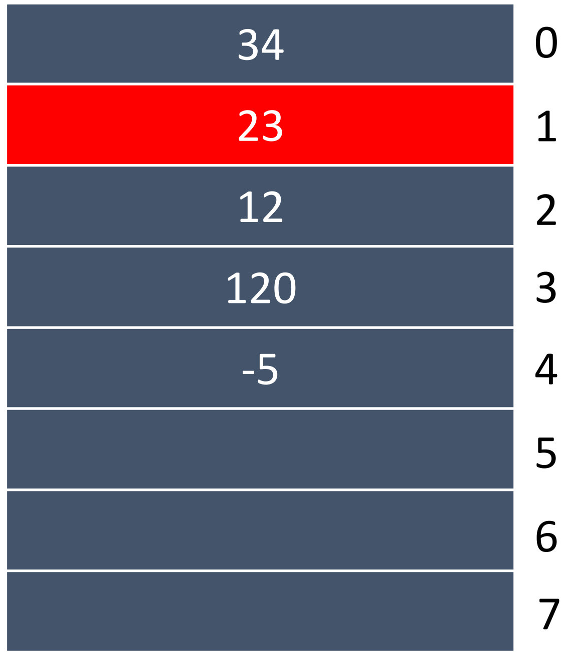
insert(23,1);



## **Arrays operation**: insert item

We would like to insert a given value with a given index

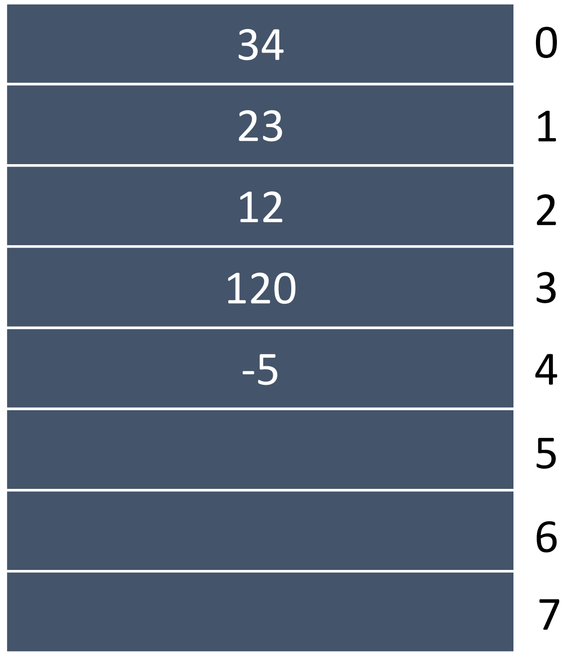
insert(23,1);



**Arrays operation**: insert item

We would like to insert a given value with a given index

insert(23,1);



So: it is a bit more problematic, sometime we have to shift lots of values in order to be able to insert the new one !!! ~ **O(N)** time complexity

**Arrays operation**: insert item

We would like to insert a given value with a given index

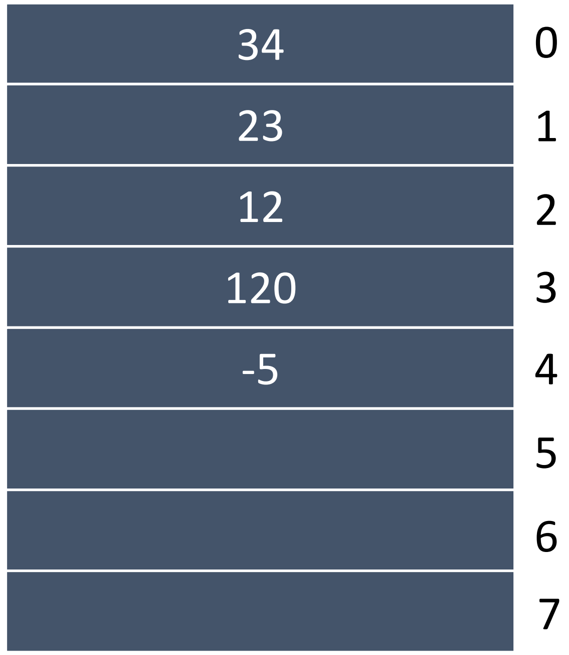
Add new item: **O(1) very fast!**

Insert item to a given index: **O(N) ----Why?**

**Because we have to shift the items in order to insert**
**At a certain index location.**

## Arrays operation: remove items

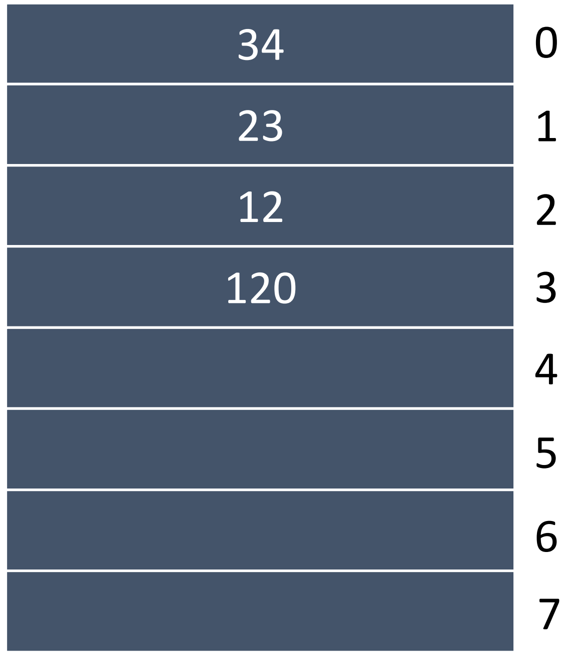
We would like to remove the last item, it is very simple, just remove it // **O(1)** time complexity



**Arrays operation**: remove items

We would like to remove the last item, it is very simple, just remove it // **O(1)** time complexity

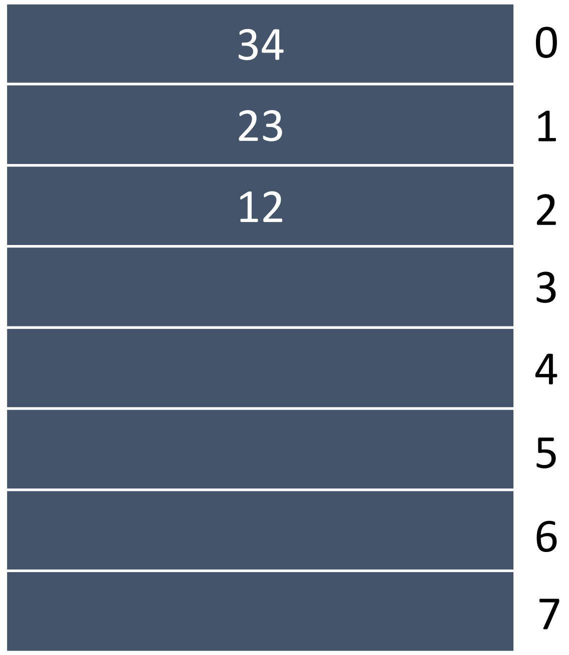
removeLast();



**<u>Arrays operation</u>**: remove items

We would like to remove the last item, it is very simple, just remove it // **O(1)** time complexity

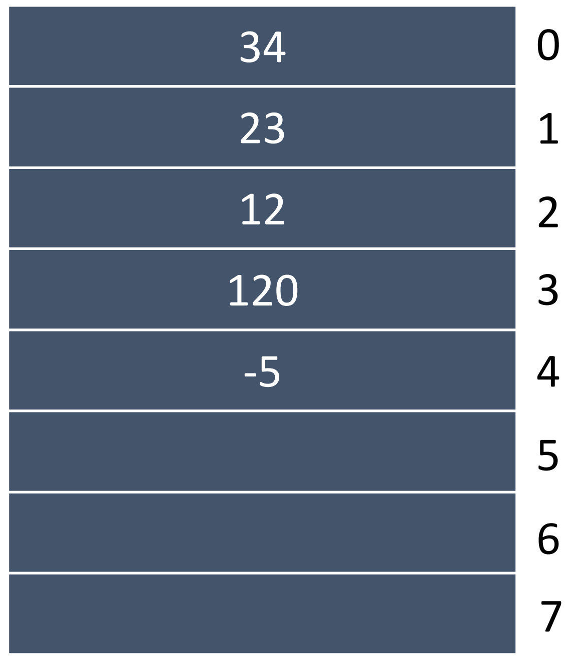
removeLast();



**Arrays operation**: remove items with given index

We would like to remove a value with a given index, it is not that simple, we may have to shift items

// **O(N)** time complexity

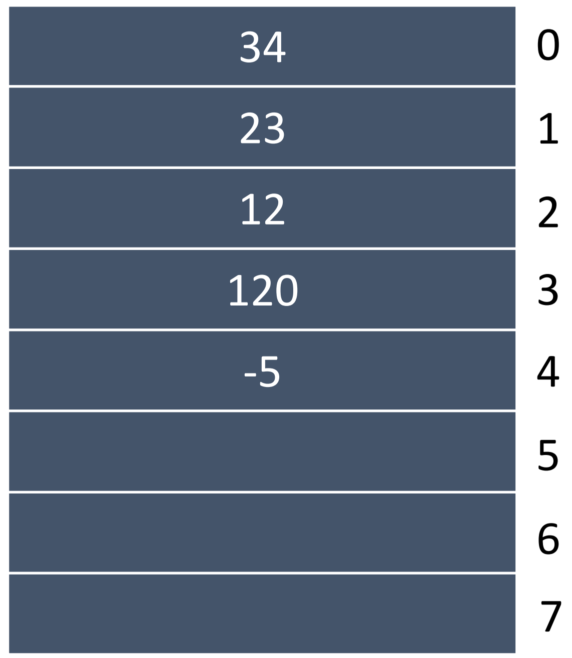


**Arrays operation**: remove items with given index

We would like to remove a value with a given index, it is not that simple, we may have to shift items

// **O(N)** time complexity

remove(1);

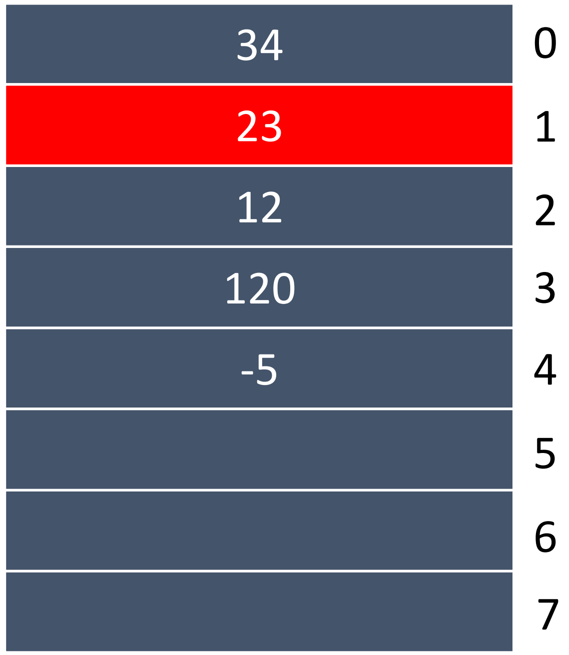


**Arrays operation**: remove items with given index

We would like to remove a value with a given index, it is not that simple, we may have to shift items

// **O(N)** time complexity

remove(1);

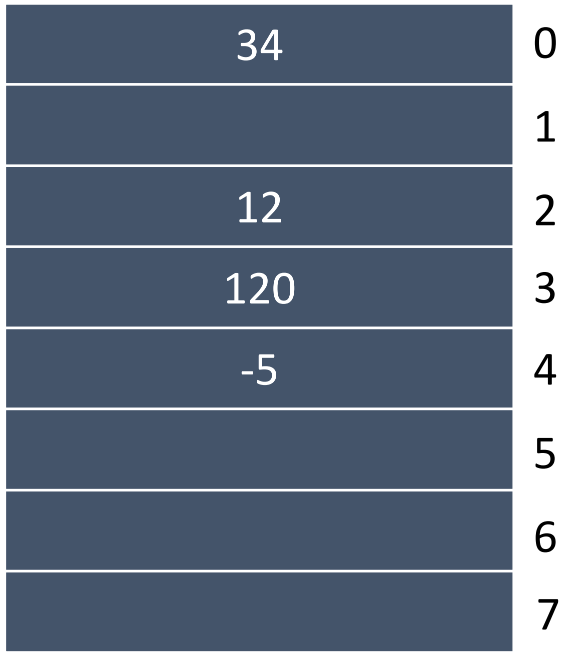


**Arrays operation**: remove items with given index

We would like to remove a value with a given index, it is not that simple, we may have to shift items

// **O(N)** time complexity

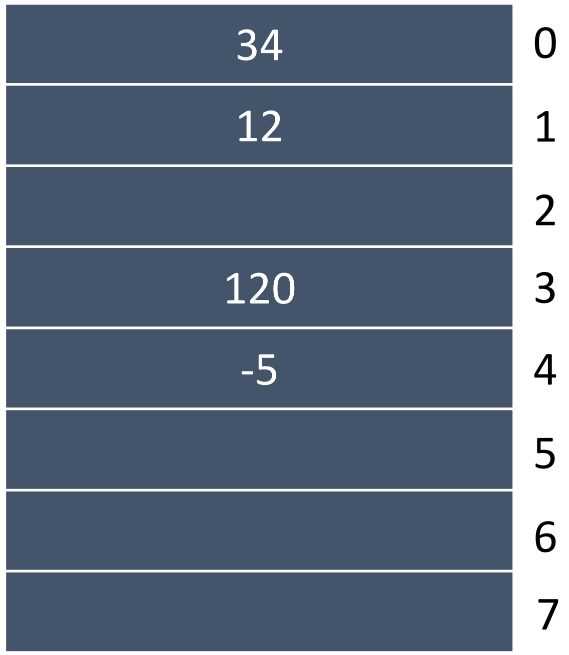
remove(1);



## **Arrays operation**: remove items with given index

We would like to remove a value with a given index, it is not that simple, we may have to shift items // **O(N)** time complexity

remove(1);

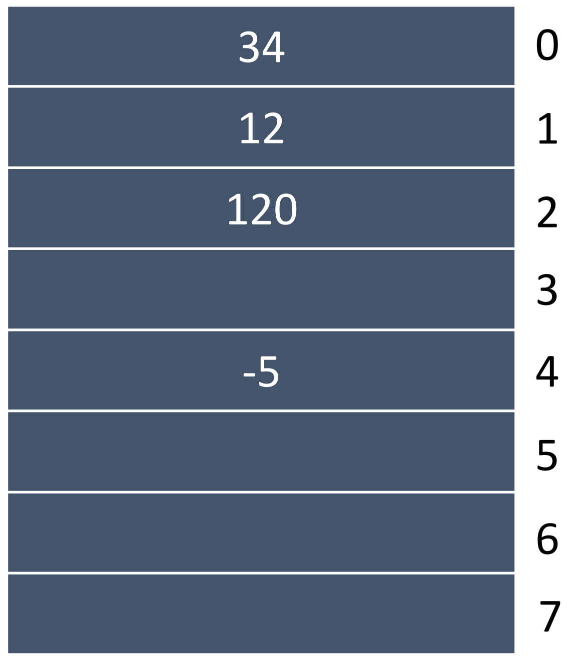


**Arrays operation**: remove items with given index

We would like to remove a value with a given index, it is not that simple, we may have to shift items

// **O(N)** time complexity

remove(1);

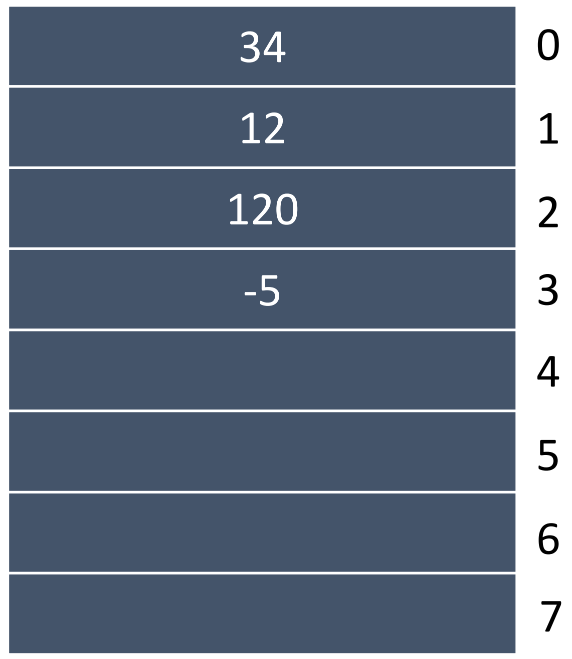


## **Arrays operation**: remove items with given index

We would like to remove a value with a given index, it is not that simple, we may have to shift items

// **O(N)** time complexity

remove(1);



So: overall complecity will be linear **O(N)**

**<u>Arrays operation</u>**: remove items with given index

We would like to remove a value with a given index, it is not that simple, we may have to shift items

// O(N) time complexity

Removing last item: **O(1)**

Removing f.e. middle item: **O(N)**