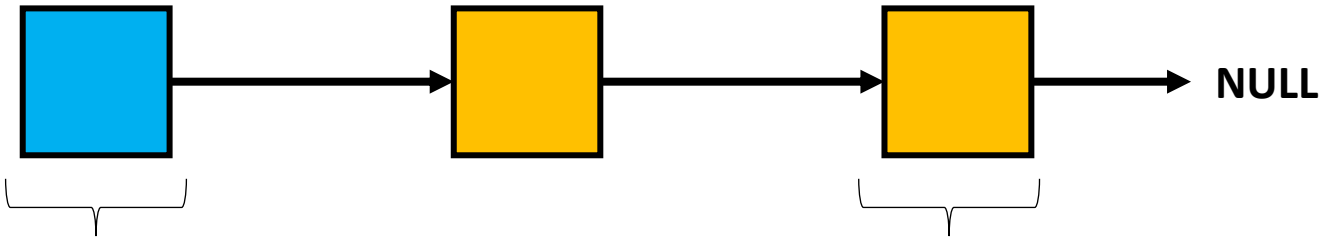
Linked List Data Structure
(Algorithms and Data Structures)

Linked Lists

- it is another **data structure** – so the aim is to be able to store items efficiently (*insertion* and *removal* operations)
- arrays have a huge disadvantage: there may be „holes” in the data structure and we have to shift a lot of items
- this problem can be eliminated by **linked lists**

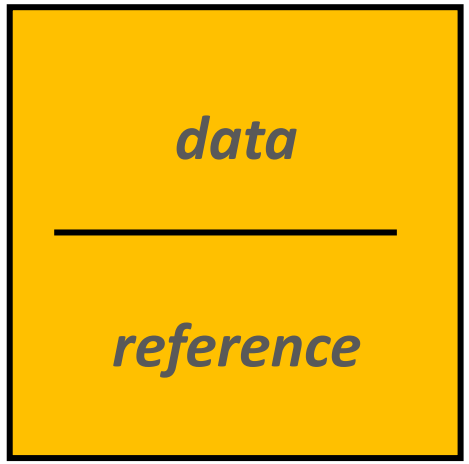


we have access to the first node of the linked list (head node)

- other items can be accessed starting with this node -

*last node of the linked list is pointing to a **NULL***

Linked Lists



- every node stores the **data** itself and a **reference** the next node in the linked list data structure
- this is why **linked lists** need more memory than arrays
- it has an advantage – there can not be „holes” in the data structure so **there is no need for shifting items**

Linked Lists

- easy data structures and easy to implement them
- the items are not stores next to each other in the memory – **so there is no random indexing**
- We can implement more complex data structures and abstract data types such as *stacks* and *queues*

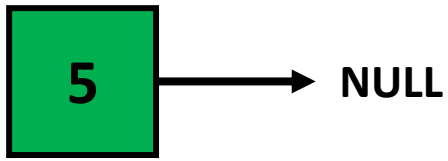
FINDING ARBITRARY ITEM IN THE LINKED LIST STILL HAS O(N) LINEAR RUNNING TIME

Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time

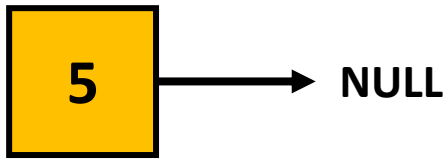
Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time



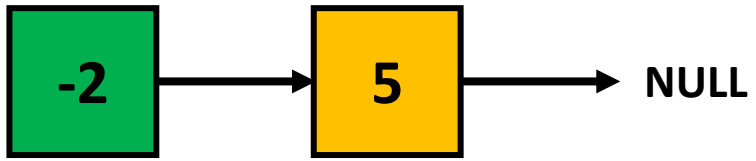
Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time



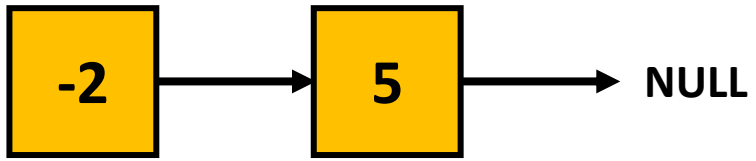
Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time



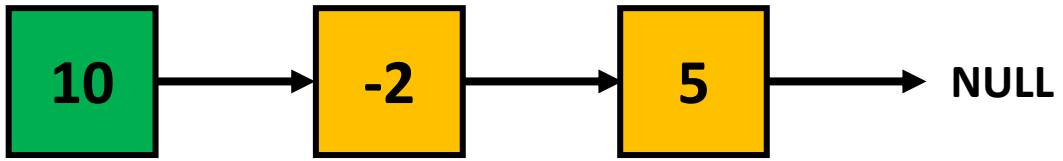
Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time



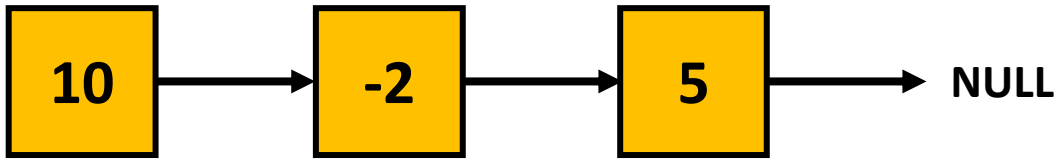
Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time



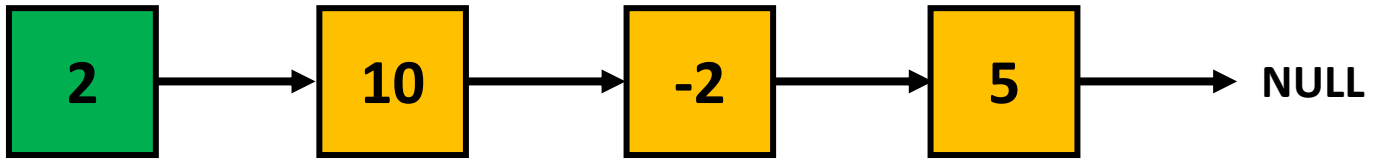
Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time



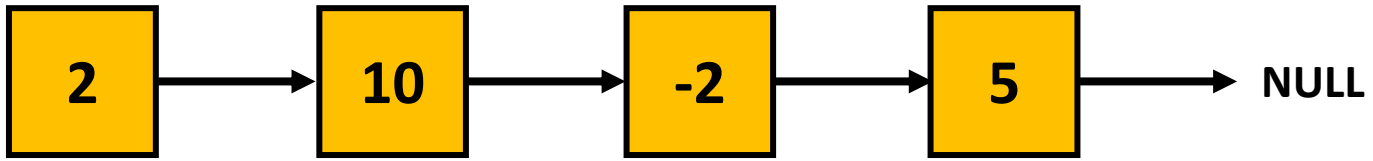
Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time



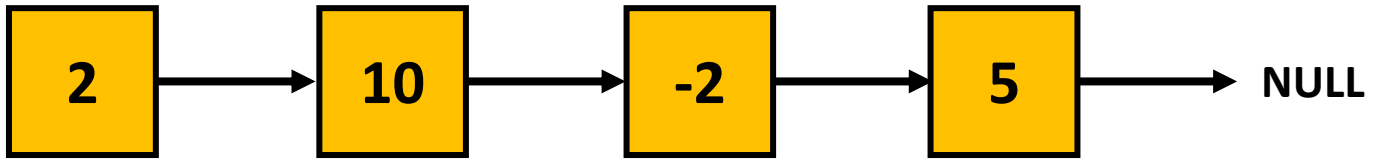
Linked Lists

Huge advantage of linked lists is that we can insert items at the beginning of the data structure fast – **O(1)** running time



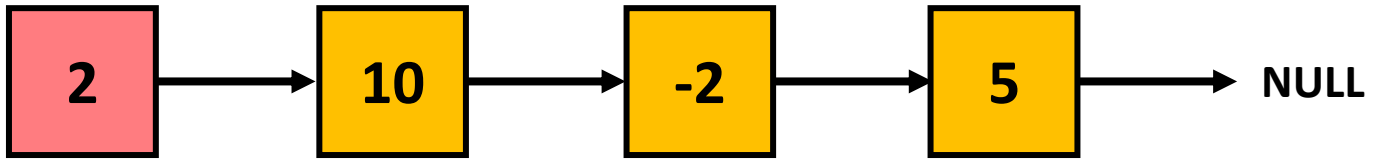
Linked Lists

Huge advantage of linked lists is that we can remove items at the beginning of the data structure fast – **O(1)** running time



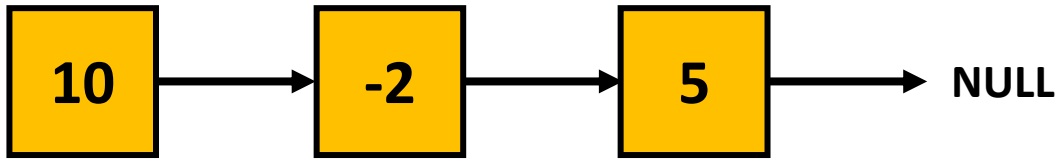
Linked Lists

Huge advantage of linked lists is that we can remove items at the beginning of the data structure fast – **O(1)** running time



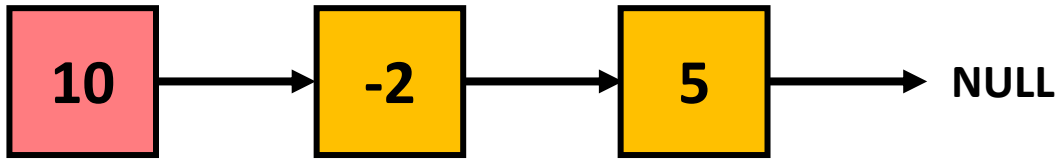
Linked Lists

Huge advantage of linked lists is that we can remove items at the beginning of the data structure fast – **O(1)** running time



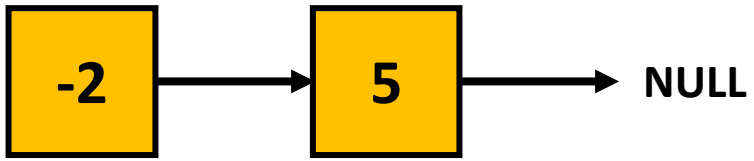
Linked Lists

Huge advantage of linked lists is that we can remove items at the beginning of the data structure fast – **O(1)** running time



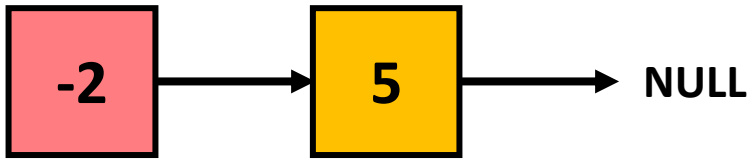
Linked Lists

Huge advantage of linked lists is that we can remove items at the beginning of the data structure fast – **O(1)** running time



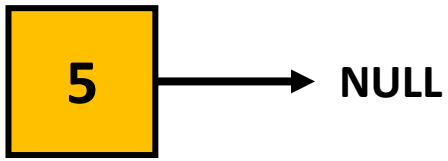
Linked Lists

Huge advantage of linked lists is that we can remove items at the beginning of the data structure fast – **O(1)** running time



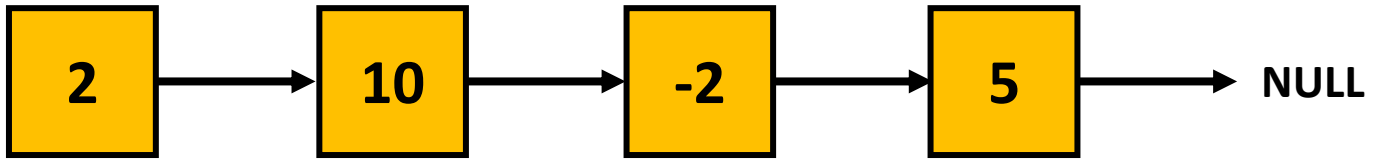
Linked Lists

Huge advantage of linked lists is that we can remove items at the beginning of the data structure fast – **O(1)** running time



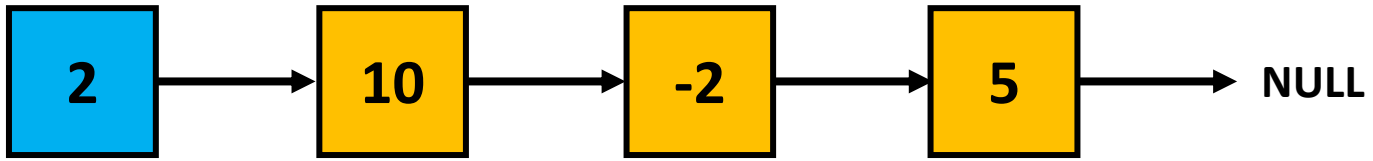
Linked Lists

Huge disadvantage of linked lists is that it is a slow operation to insert items at the end – **O(N)** running time



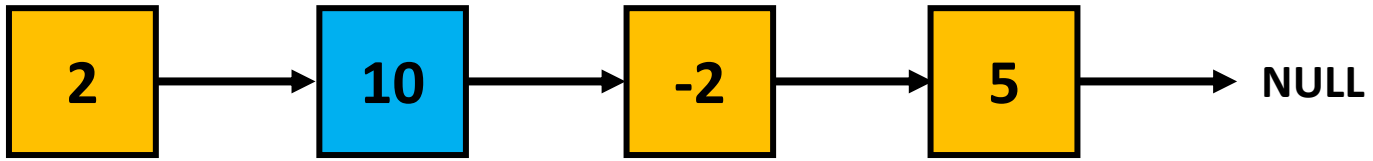
Linked Lists

Huge disadvantage of linked lists is that it is a slow operation to insert items at the end – **O(N)** running time



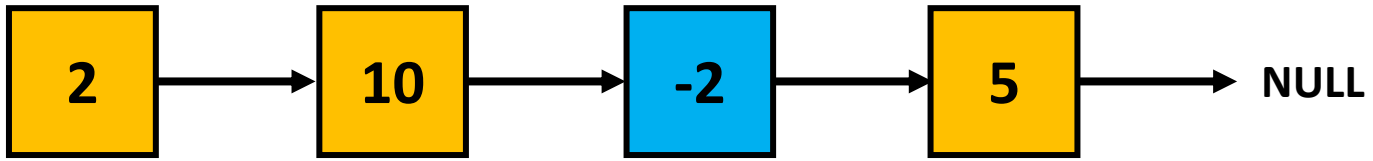
Linked Lists

Huge disadvantage of linked lists is that it is a slow operation to insert items at the end – **O(N)** running time



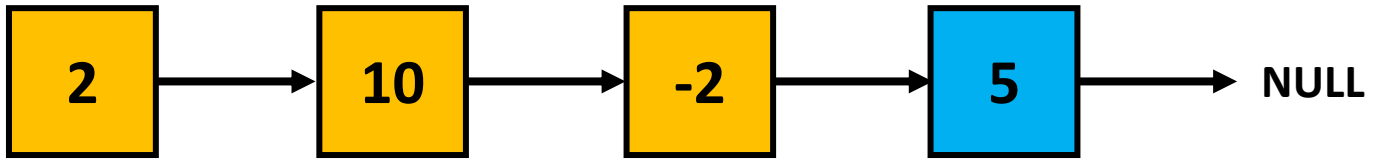
Linked Lists

Huge disadvantage of linked lists is that it is a slow operation to insert items at the end – **O(N)** running time



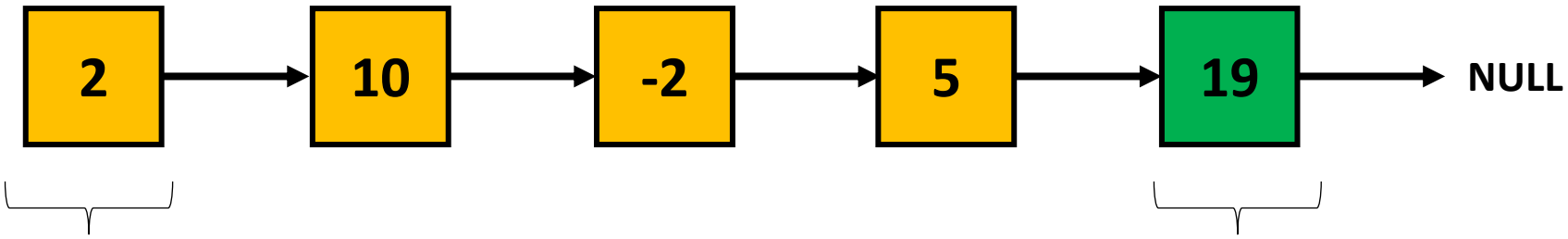
Linked Lists

Huge disadvantage of linked lists is that it is a slow operation to insert items at the end – **O(N)** running time



Linked Lists

Huge disadvantage of linked lists is that it is a slow operation to insert items at the end – **O(N)** running time



we can access the head node exclusively - so we have to start here always -

we have to find the item we want to remove (linear search) in ***O(N)*** *running time*

Linked Lists Operations

Manipulation the first item (insertion or removal):

O(1) running time – this is why we like linked lists

Manipulating arbitrary item

O(N) running time – if we have to do several of these operations then linked list is not the best option possible !!!

Linked Lists Advantages

- linked lists are **dynamic data structures**: they can acquire memory at *run-time* by inserting new nodes
- no need for resizing the data structures – as we have seen with arrays
- we can grow the data structure organically – not a problem if we do not know the size at *compile-time*
- manipulating the first item is fast – **O(1)** running time
- can store different sized items – arrays assume the items have the exact same size

Linked Lists Disadvantages

- need **more memory** because of the references
- there is no random access - we can only access the first node (*head node*) of the linked list
- can not go backwards – how to get the previous node?
- **not predictable** – the running time of the application relies heavily on the operations the users do
- still have not solved the main issue – how to search for arbitrary items faster than **O(N)** linear running time?