LUT University

BM40A1500 DATA STRUCTURES AND ALGORITHMS

LISTS

2024

LISTS

- A finite, ordered sequence of data items known as elements.
  - "Ordered" here means that each element has a position in the list.
  - The list elements are not necessarily sorted by value.
- Lists of integers, lists of characters, lists of payroll records, lists of lists.
- The number of elements currently stored is called the length of the list.
- The beginning of the list is called the head, the end of the list is called the tail.
- To be consistent with standard array indexing, the first position on the list is denoted as 0.
  - If if there are n elements in the list, they are given positions 0 through $n-1$ as

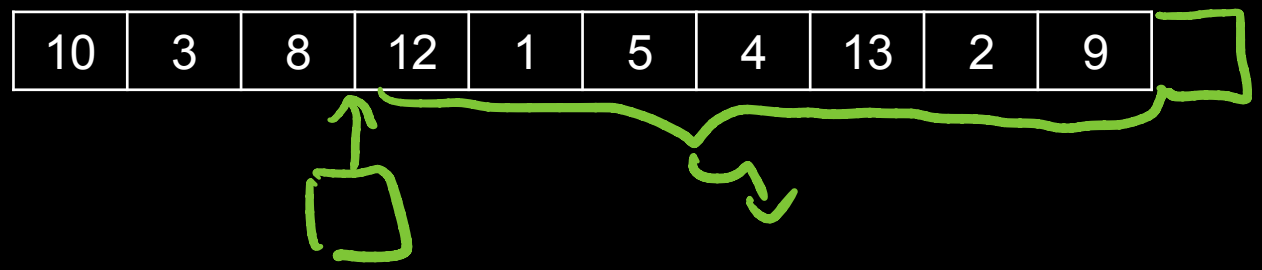
$$\langle a_0, a_1, \ldots, a_{n-1} \rangle$$

ABSTRACT DATA STRUCTURE (ADT) FOR A LIST

- A list ADT can be defined, for example, based on the concept of current position.
- Basic operations:
  - insert(it): Insert "it" at the current position.
  - append(it): Append "it" at the end of the list.
  - remove(): Remove and return the current element.
  - getValue(): Return the current element.
  - prev(): Move the current position one step left.
  - next(): Move the current position one step right.
  - moveToStart(): Set the current position to the start of the list.
  - moveToEnd(): Set the current position to the end of the list.
  - moveToPos(pos): Set the current position to "pos".
  - length(): Return the number of elements in the list.
  - currPos(): Return the position of the current element.
  - isAtEnd(): Return true if current position is at end of the list.
  - isEmpty(): Tell if the list is empty or not.
  - clear(): Remove all contents from the list.
- Also, other ways to define a list ADT.
- ADT does not define how the list is implemented.
- Two standard approaches to implementing lists:
  - array-based list
  - linked list

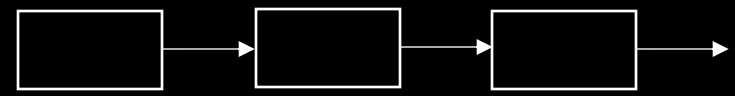
ARRAY-BASED LIST

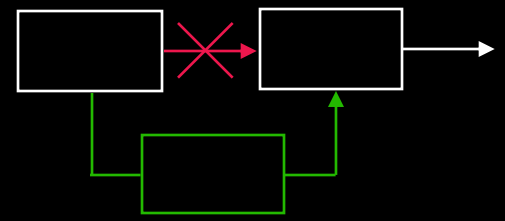
- List elements are stored in contiguous cells of the array.
- Accessing *n*th element or moving to certain position is very fast.
- Adding and removing an element from the middle of list is relatively slow.
  - Appending to the end of list can be done efficiently



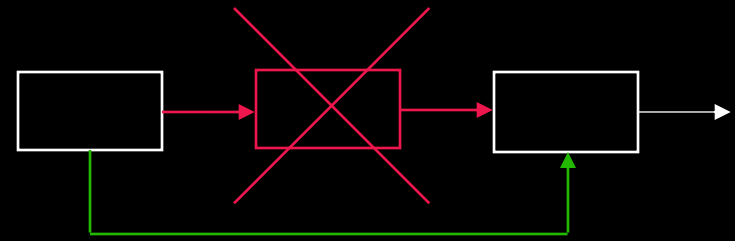
LINKED LIST

- Dynamic memory allocation: memory is allocated for new list elements as needed.
- Each node contains a link to the next node.
  - Nodes can be located anywhere in the memory.
- Adding and removing an element from the current position (even in the middle of list) is very fast.
- Accessing *n*th element in the middle of list is relatively slow.
  - No efficient way to implement the moveToPos operation.





$\Theta(1)$



$\Theta(n)$

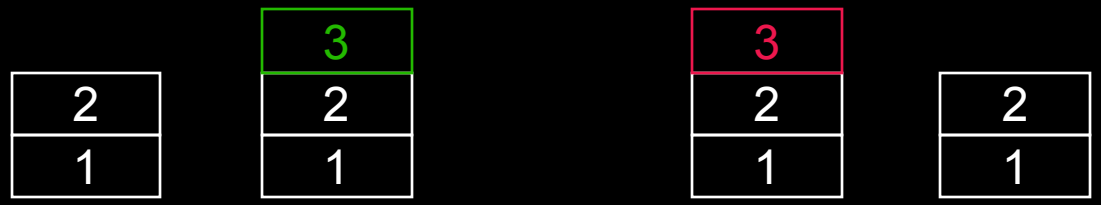
LINKED LIST

- Dynamic memory allocation: memory is allocated for new list elements as needed.
- Each node contains a link to the next node.
  - Nodes can be located anywhere in the memory.
- Adding and removing an element from the current position (even in the middle of list) is very fast.
- Accessing *n*th element in the middle of list is relatively slow.
  - No efficient way to implement the moveToPos operation.

```
class Node:
    def __init__(self, data, next):
        self.data = data
        self.next = next

class LinkedList:
    def __init__(self):
        self.tail = Node(None, None)
        self.head = Node(None, self.tail)
        self.len = 0

    def print(self):
        # ...

    def append(self, data):
        # ...

LinkedList()
L.append(1)
L.print()
```

STACK

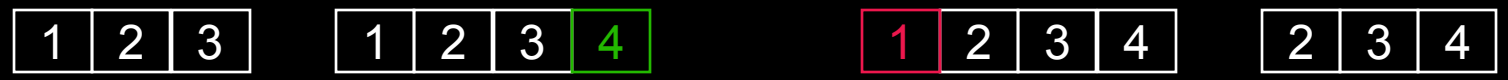
- A list-like structure in which elements may be inserted or removed from only on top.
- Last-In, First-Out (LIFO)
- When inserted, an element is said to be **pushed** onto the stack.
- When removed, an element is said to be **popped** from the stack.
- Can be implemented as array-based or linked stack.



QUEUE

- A list-like structure in which elements may be inserted only at the back and removed only from the front.
- First-In, First-Out (FIFO)
- **Enqueue** and **dequeue** operations.
- Can be implemented as array-based or linked ~~stack~~.

queue



- Neither stack or queue does provide noticeable benefits over lists, but they are useful concepts when designing algorithms.

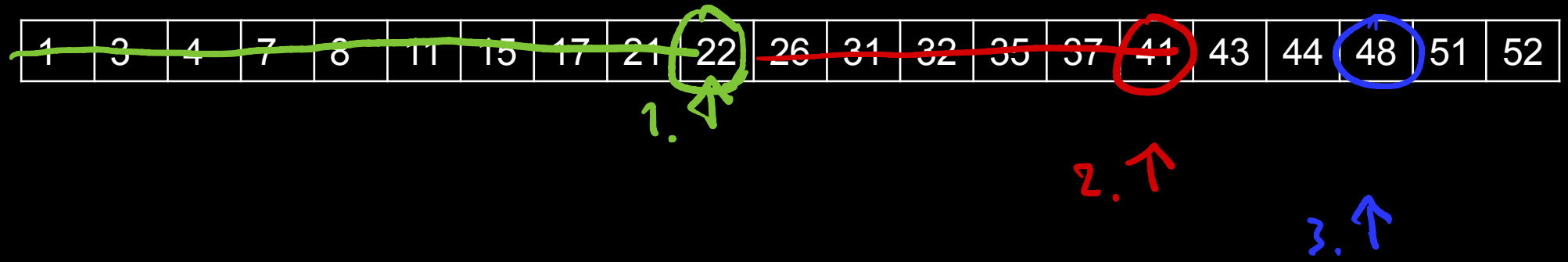
SEARCHING IN AN ARRAY

Find(48)

- Sequential search:

  - For an unsorted array, the sequential search is the best there is.

- Binary search:
  - Sorted array




LUT
University