Abstract Data Types

Specification:

- Interface
- Behavior

Implementation:

- Algorithm
- State

Abstract Data Types

Bag (of integers)

Interface:

```
void add(int x)
int remove()
boolean isEmpty()
```

Behavior:

- add(x) : adds an item to the bag
- remove() : removes an arbitrary item from the bag

Abstract Data Types

List

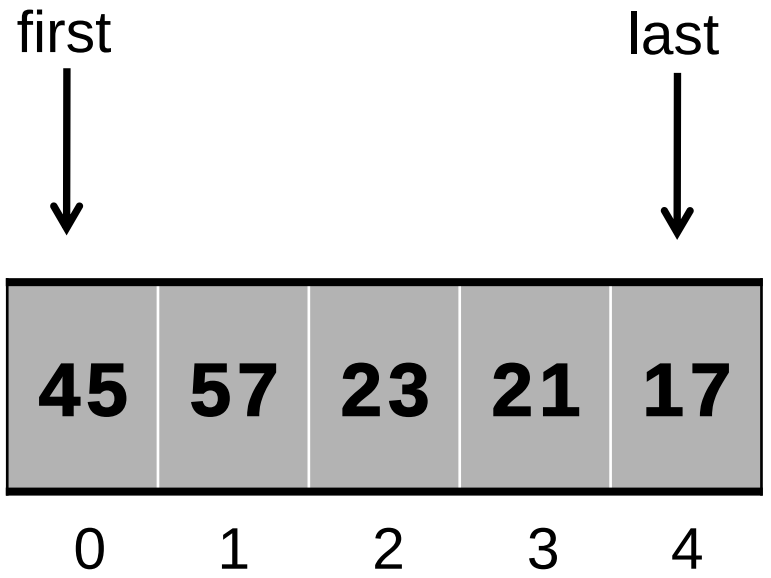
Interface:

Abstract Data Types

List

Interface:

```
void append(int x)
void prepend(int x)
void put(int x, int slot)
void remove(int x)
int getFirst()
int getLast()
int get(int slot)
boolean isEmpty()
```

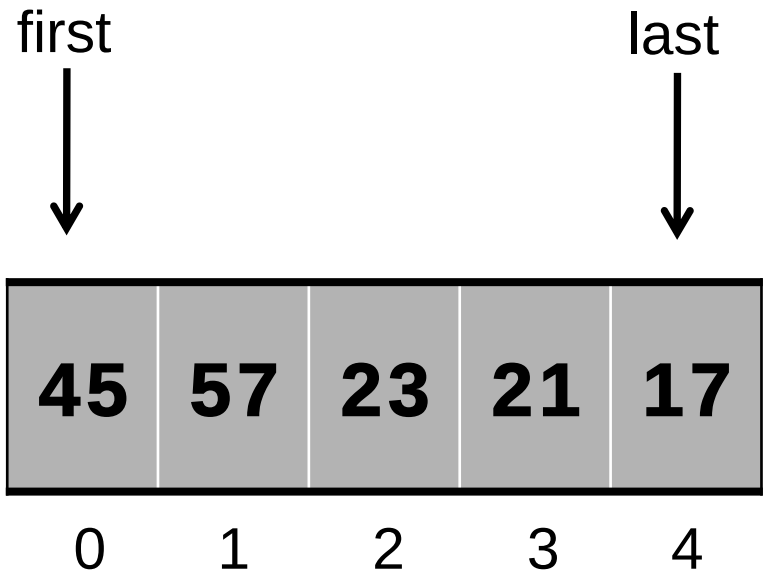


Abstract Data Types

interface java.util.List

Interface:

```
void add(int x)
void addAll(Collection c)
void clear()
void contains(int x)
void isEmpty()
int remove()
int set()
```

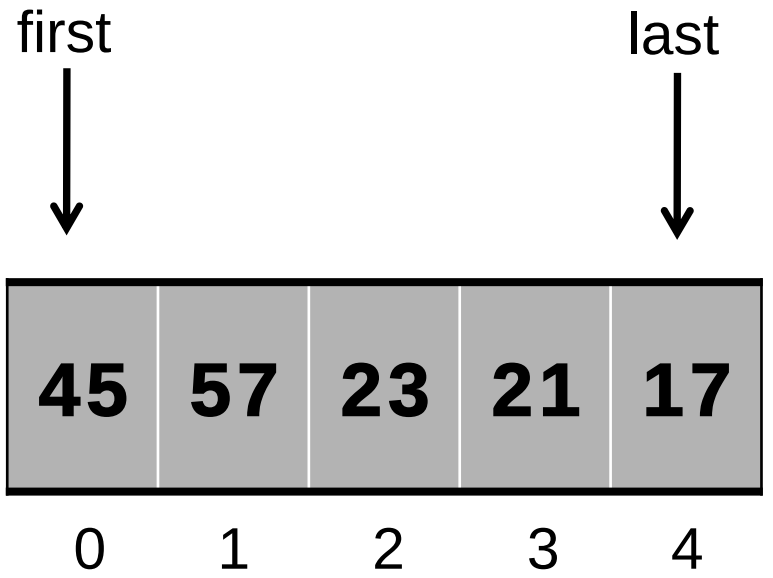


Abstract Data Types

interface java.util.List

Java implementations:

- `java.util.ArrayList`
- `java.util.Vector`
- `java.util.LinkedList`



Abstract Data Types

Stack

Interface:

- void push(element x)
- element pop()

Behavior: (LIFO: last-in, first-out)

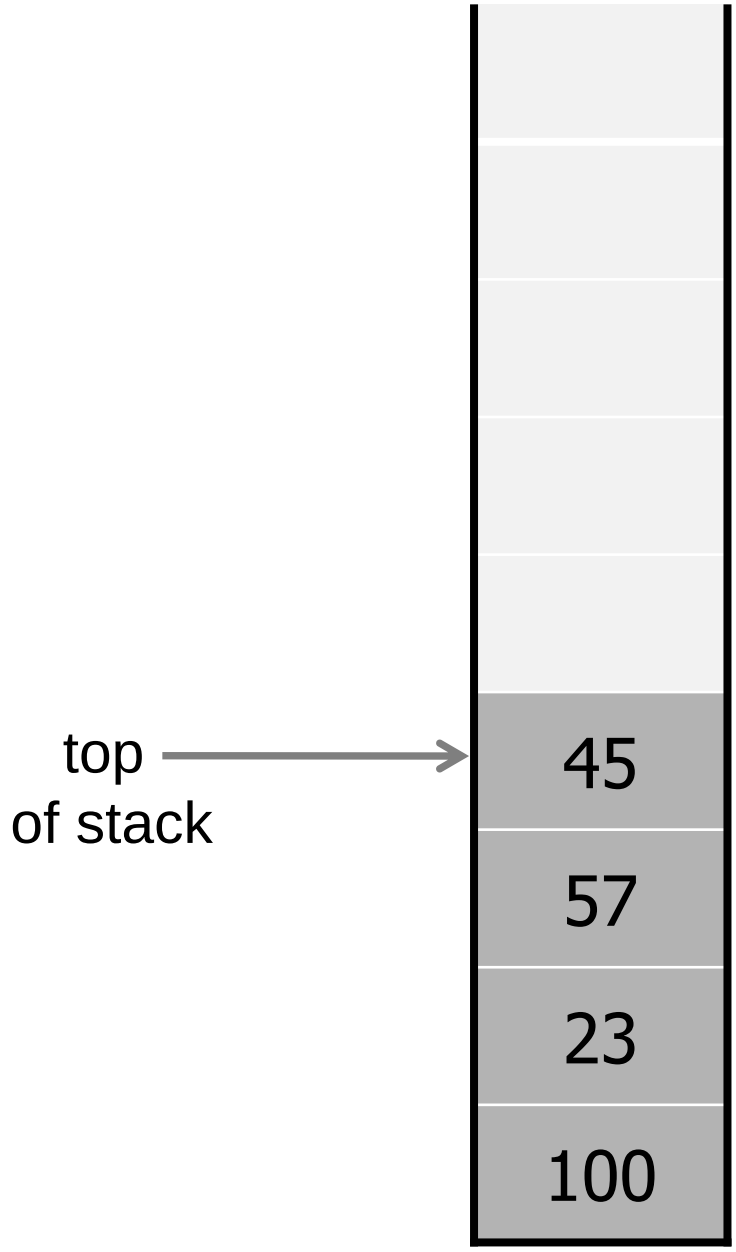
- push(x) : adds element x to the stack
- pop() : removes the mostly recently added element and returns it

Abstract Data Types

Stack

Interface:

- void push(element x)
- element pop()
- empty()

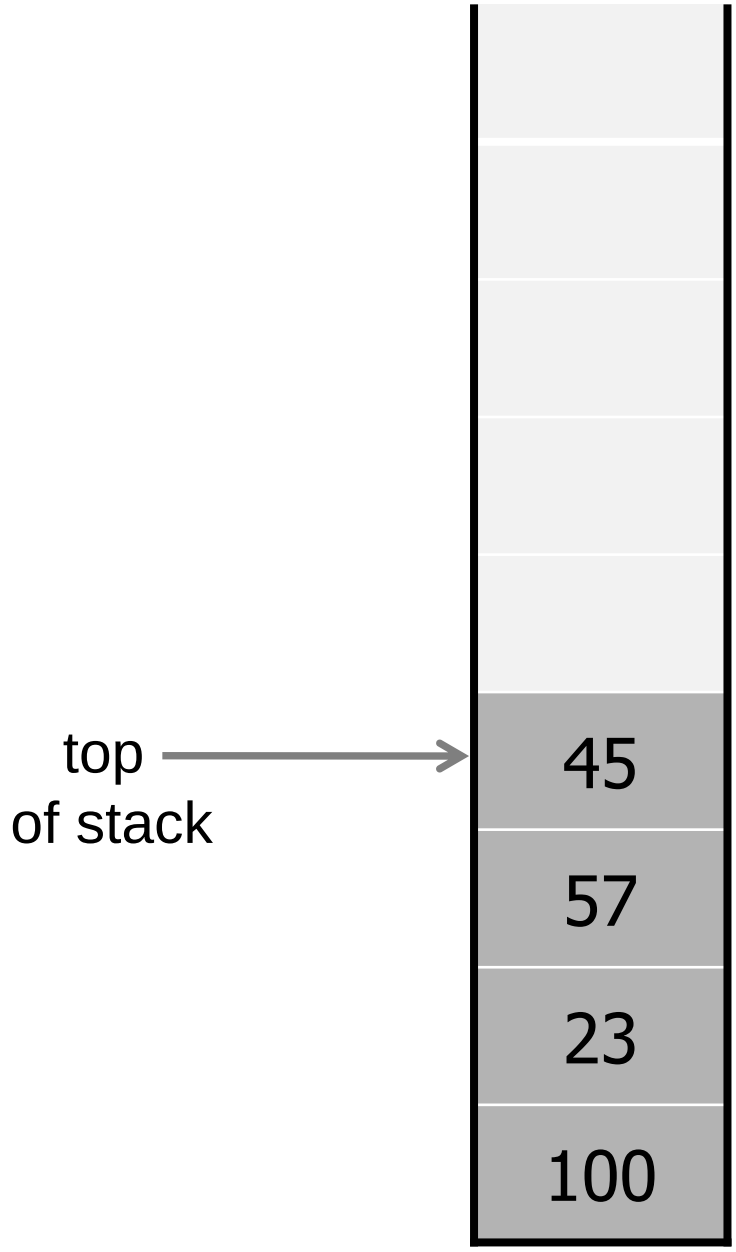


Abstract Data Types

Stack

Execution:

- push(77)

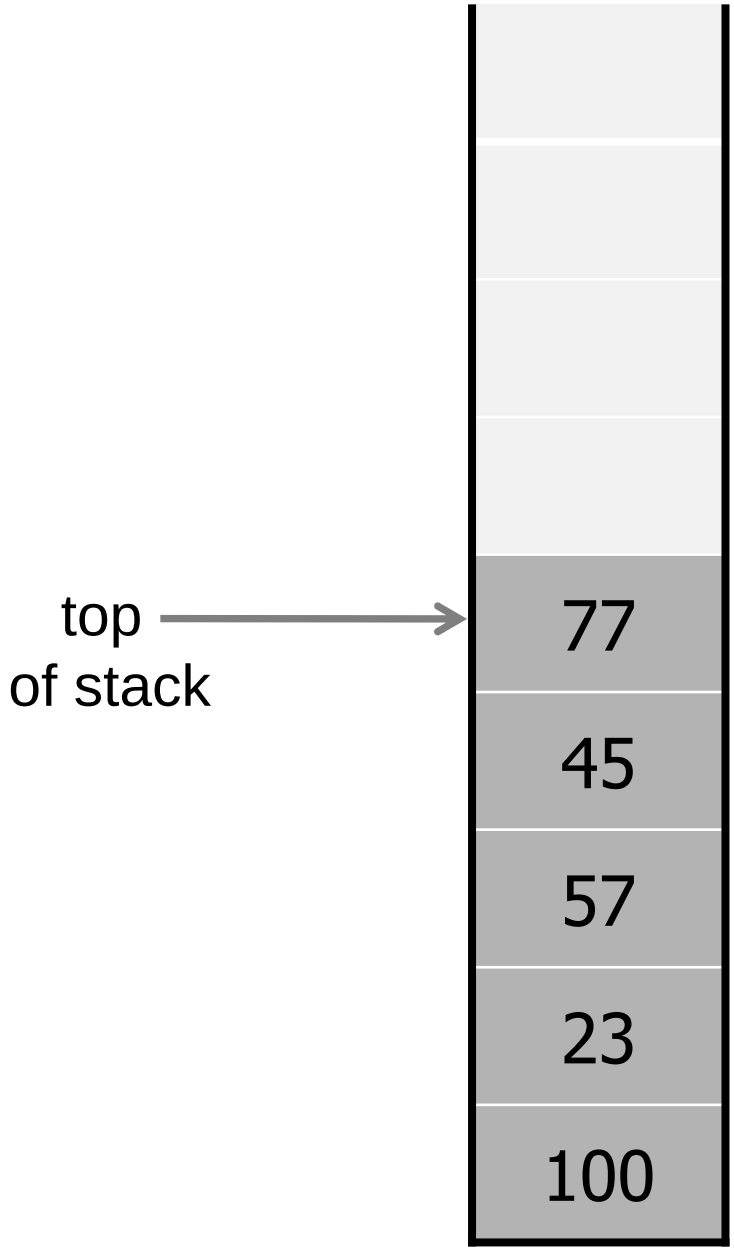


Abstract Data Types

Stack

Execution:

- push(77)

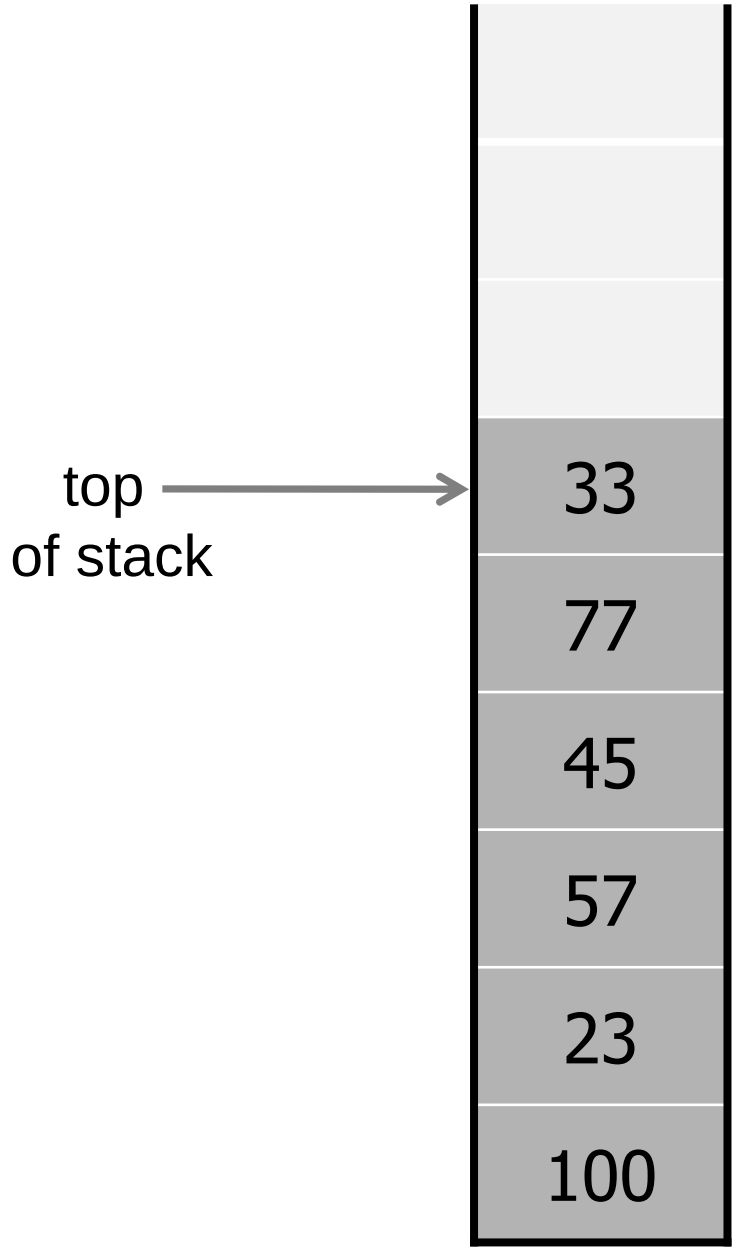


Abstract Data Types

Stack

Execution:

- push(77)
- push(33)

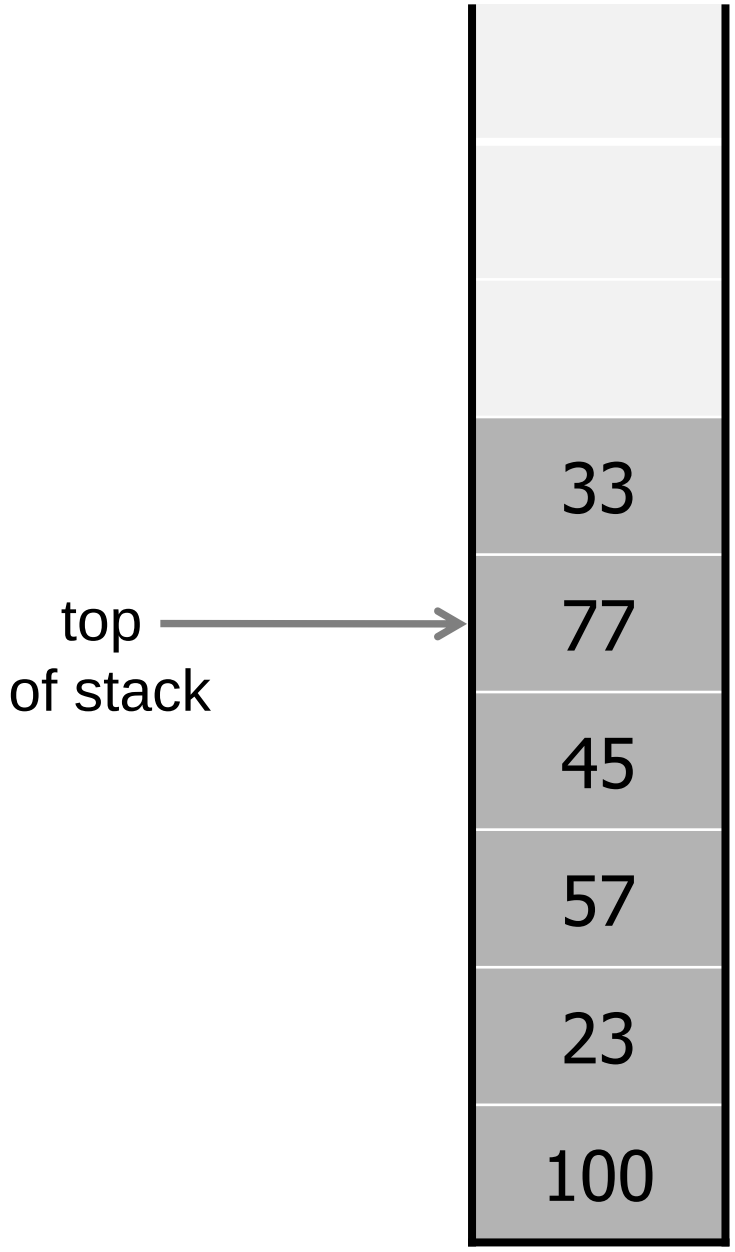


Abstract Data Types

Stack

Execution:

- push(77)
- push(33)
- pop() → ??

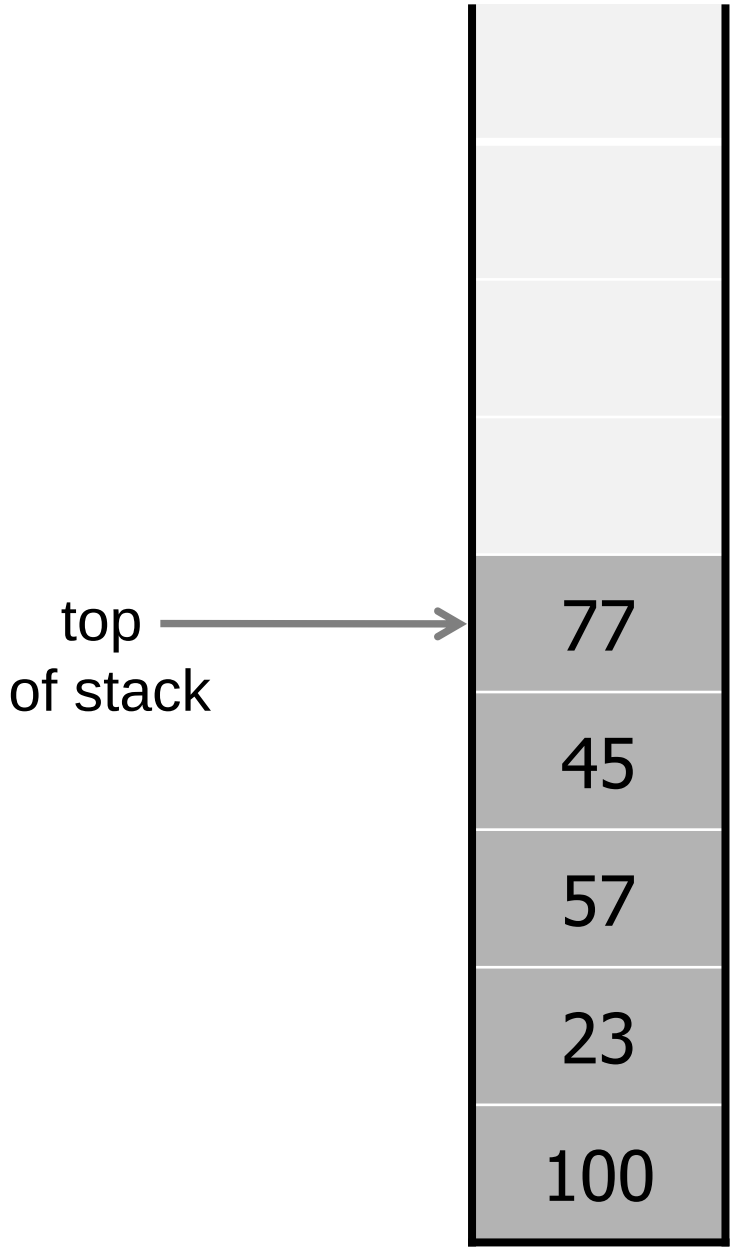


Abstract Data Types

Stack

Execution:

- push(77)
- push(33)
- pop() $\rightarrow$ 33

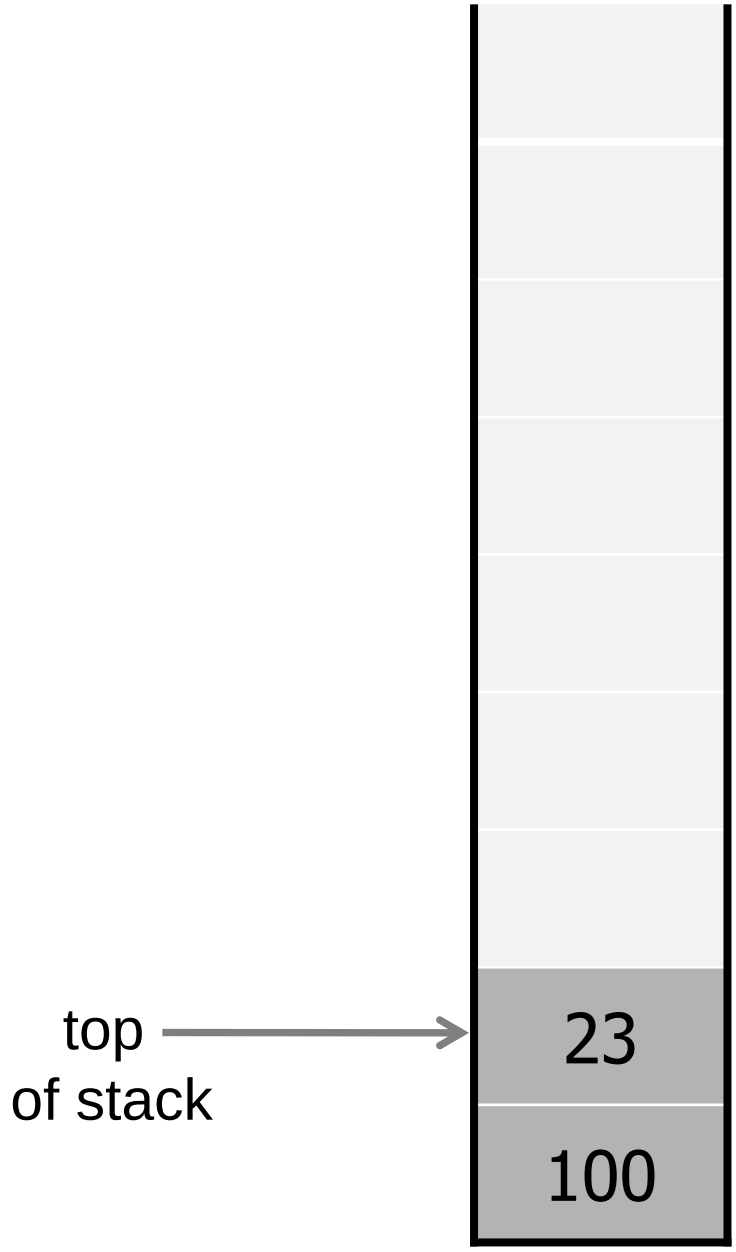


Abstract Data Types

Stack

Execution:

- push(77)
- push(33)
- pop() → 33
- pop() → 77
- pop() → 45
- pop() → 57

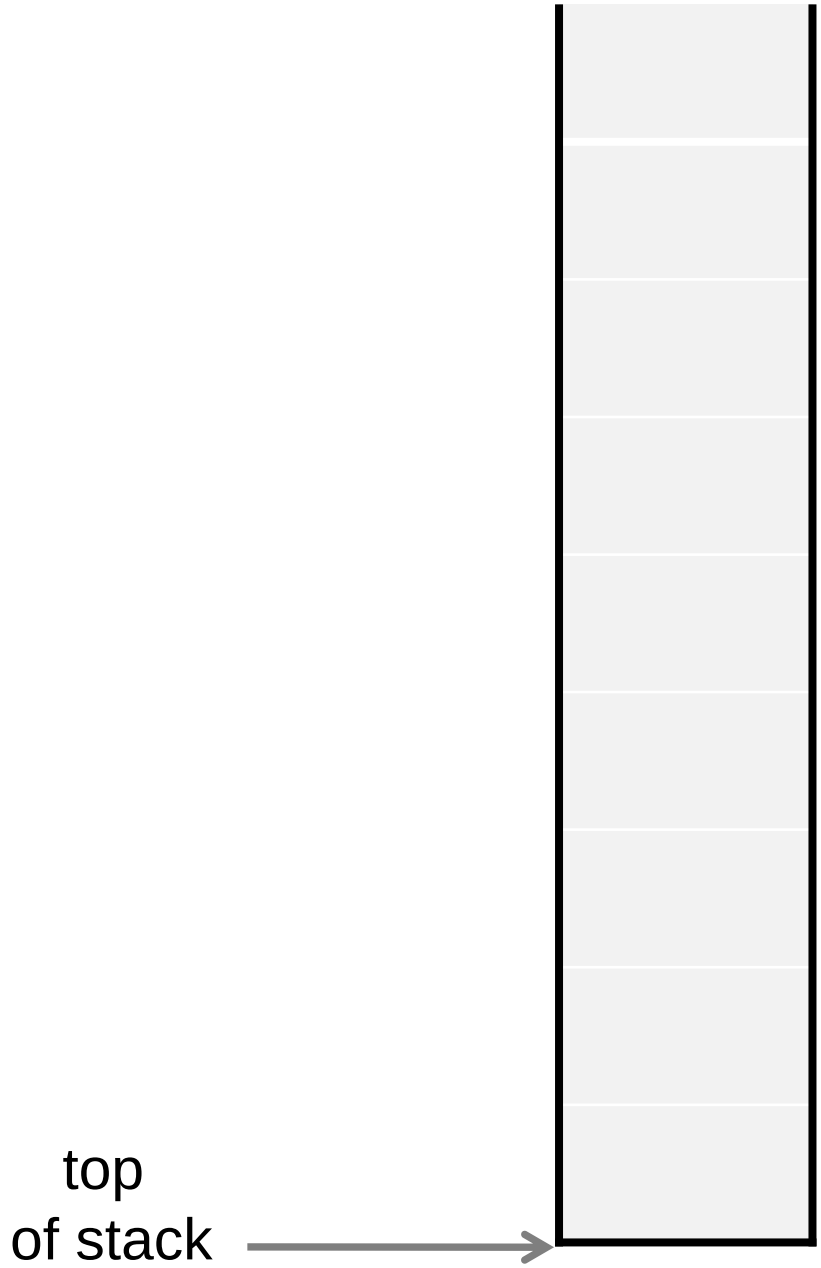


Abstract Data Types

Stack

Execution:

- pop() → 23
- pop() → 100

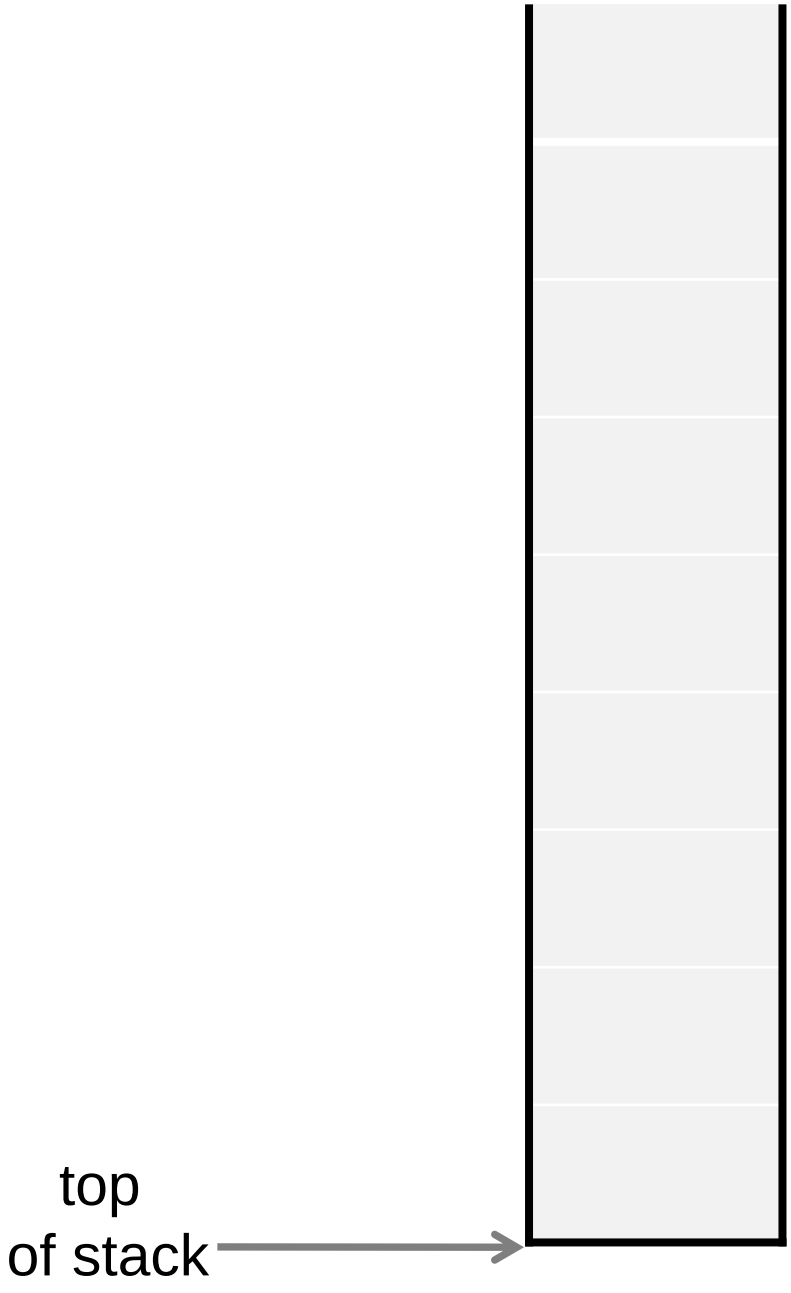


Abstract Data Types

Stack

Execution:

- pop() → 23
- pop() → 100
- pop() → ??



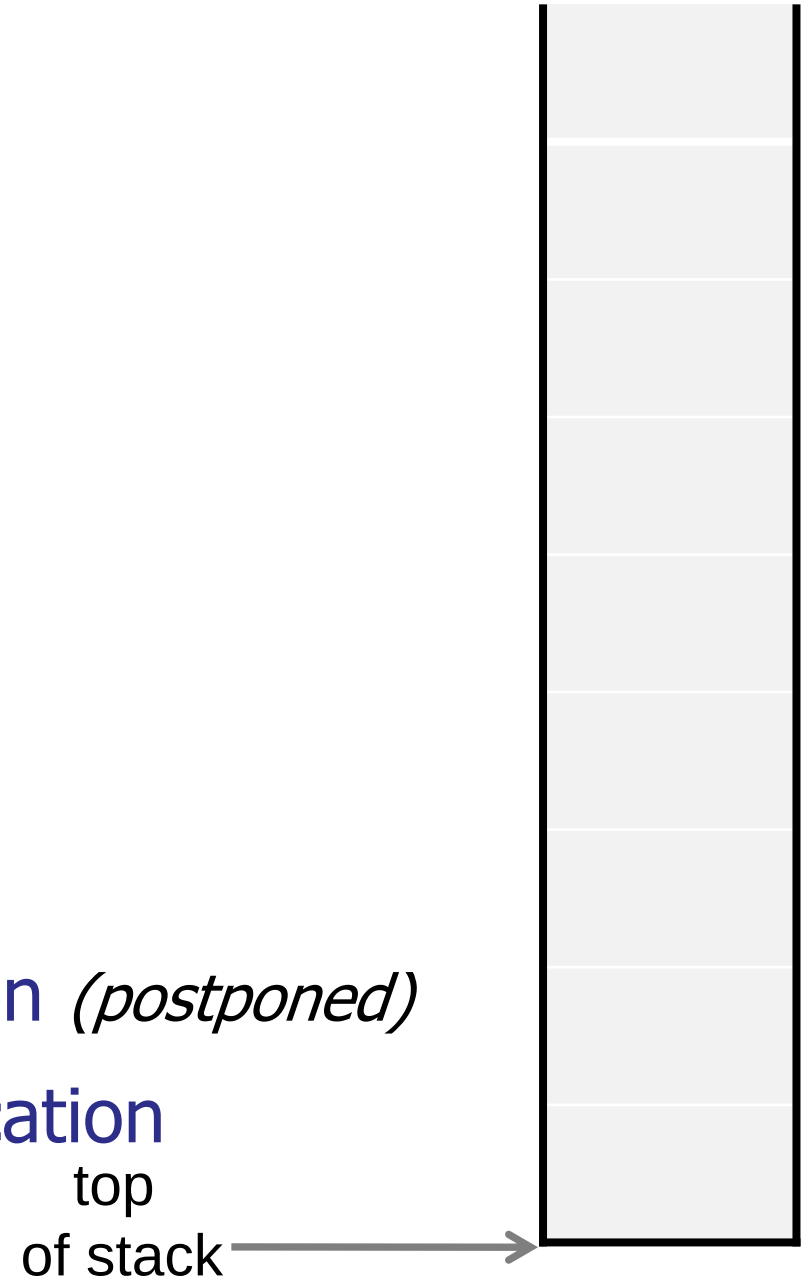
Abstract Data Types

Stack

Execution:

- pop() → 23
- pop() → 100
- pop() → ??

- Error!
  - Option 1: throw exception *(postponed)*
  - Option 2: modify specification

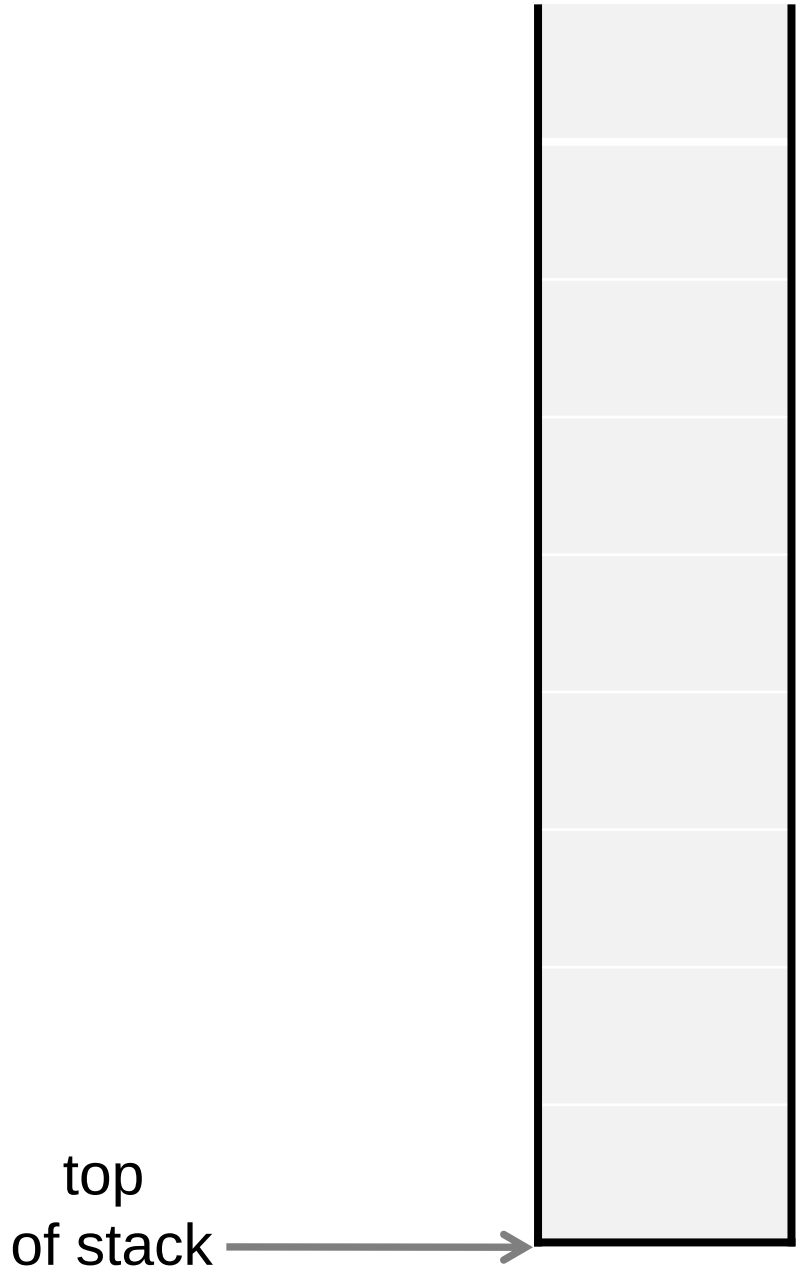


Abstract Data Types

Stack

Execution:

- pop() → 23
- pop() → 100
- empty() → true



Abstract Data Types

Queue

Interface:

- void enqueue(element x)
- element dequeue()

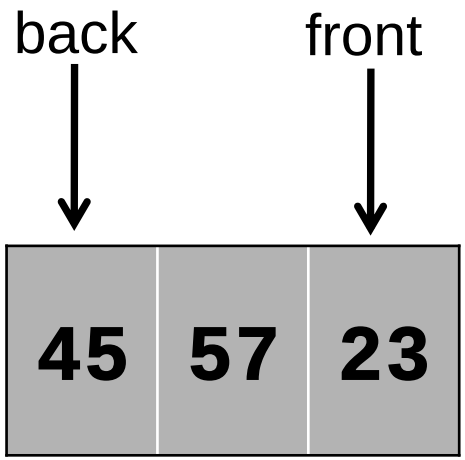
Behavior: (FIFO: first-in, first-out)

- enqueue(x) : adds element x to the front of the queue
- dequeue() : removes and returns element at the end of the queue

Abstract Data Types

Queue

Execution:

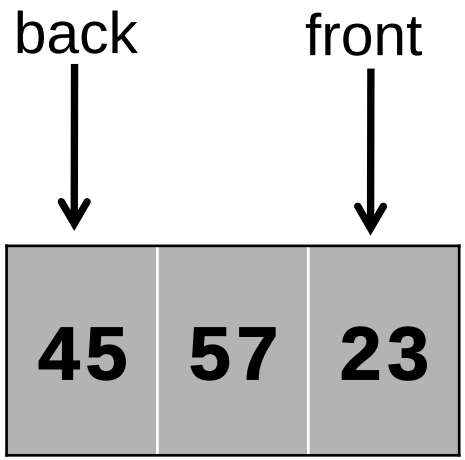


Abstract Data Types

Queue

Execution:

- enqueue(7)

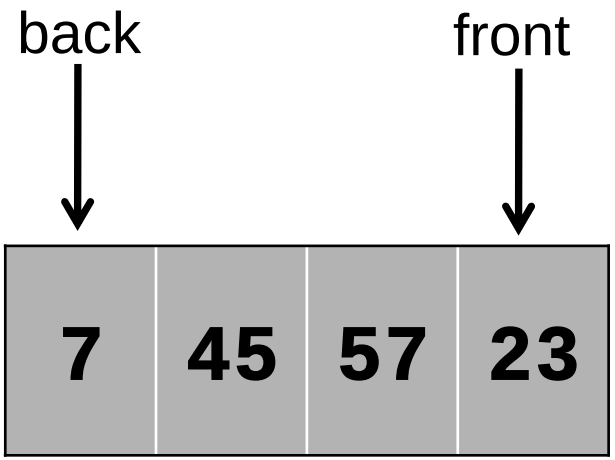


Abstract Data Types

Queue

Execution:

- enqueue(7)

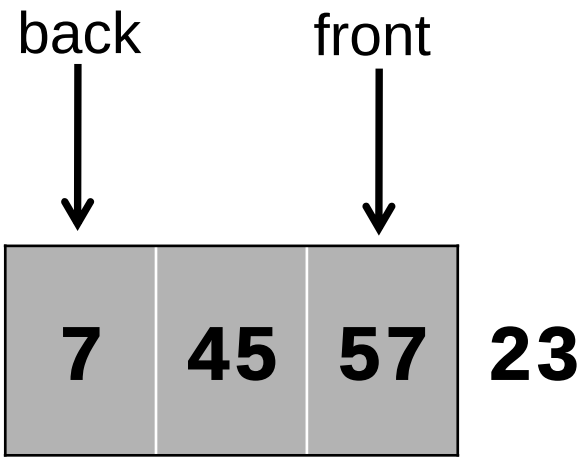


Abstract Data Types

Queue

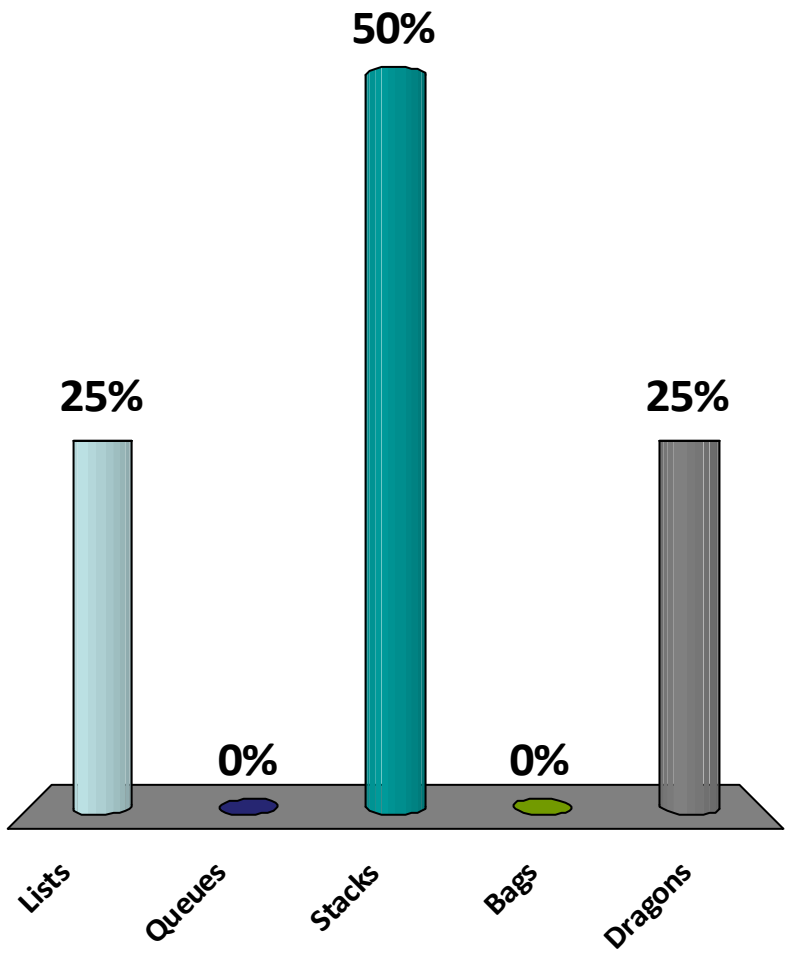
Execution:

- enqueue(7)
- dequeue() $\rightarrow$ 23



Which abstract data type appears most frequently in practice?

a. Lists
b. Queues
✓ c. Stacks
d. Bags
e. Dragons



Sorting with Stacks

Is it always possible to insert "pop" commands to make the output sorted?

Example:

6 5 4 3 2 1 ➔ 6 5 4 3 2 1 – – – – – –

Sorting with Stacks

Is it always possible to insert "pop" commands to make the output sorted?

Example:

6 5 4 3 2 1 ➔ 6 5 4 3 2 1 – – – – – –

1 2 3 4 5 6 ➔ 1 – 2 – 3 – 4 – 5 – 6

Sorting with Stacks

Is it always possible to insert "pop" commands to make the output sorted?

Example:

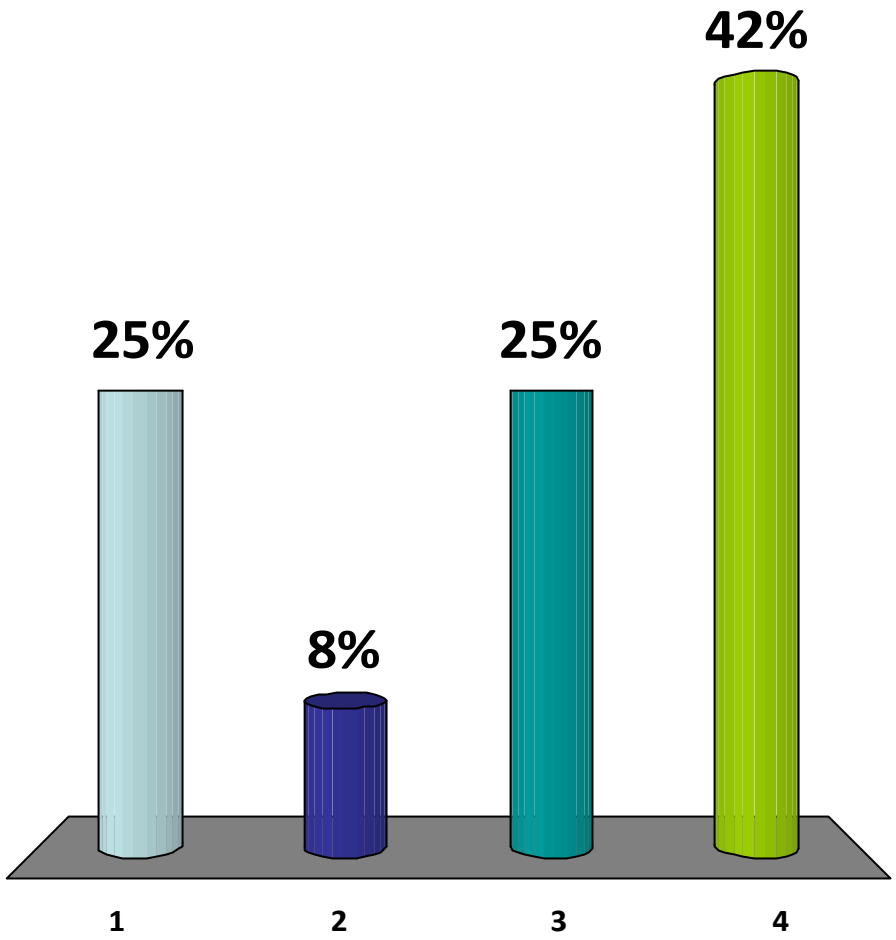
6 5 4 3 2 1 ➔ 6 5 4 3 2 1 – – – – – –

1 2 3 4 5 6 ➔ 1 – 2 – 3 – 4 – 5 – 6 –

4 1 3 2 6 5 ➔ 4 1 –3 2 – – –6 5 – –

Is it always possible to insert pop() commands to make the output sorted?

1. Yes
2. ✓ No
3. I have no idea
4. How does a stack work?



Sorting with Stacks

Challenge:

Device an algorithm that can determine how to sort a sequence with a stack, if it is possible (and fails if it is impossible).