Algorithms & Data Structures I: 2_Stack

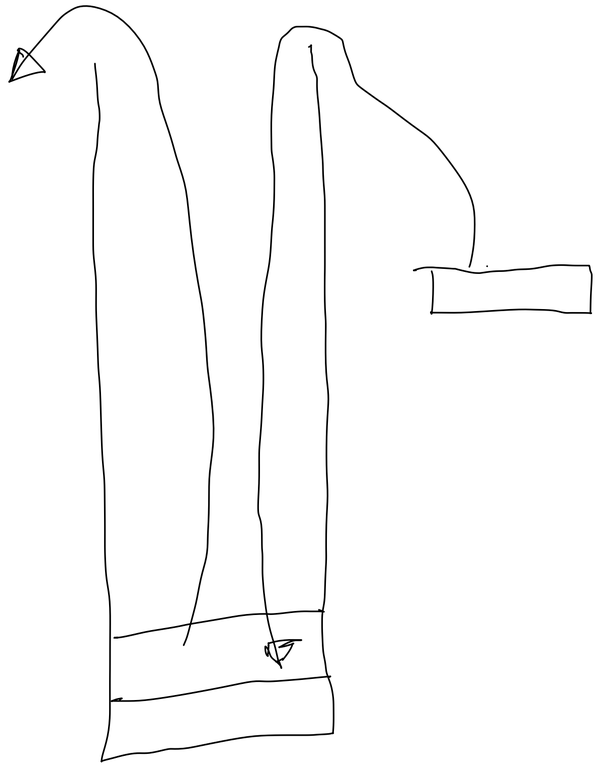
Mikhail Anukhin

Today's Topics

- What a stack is
- Stack applications
- Implementation
  - Using an array
  - Using a linked list
- Keeping minimum in Stack
- Correct brackets sequence

What a stack is

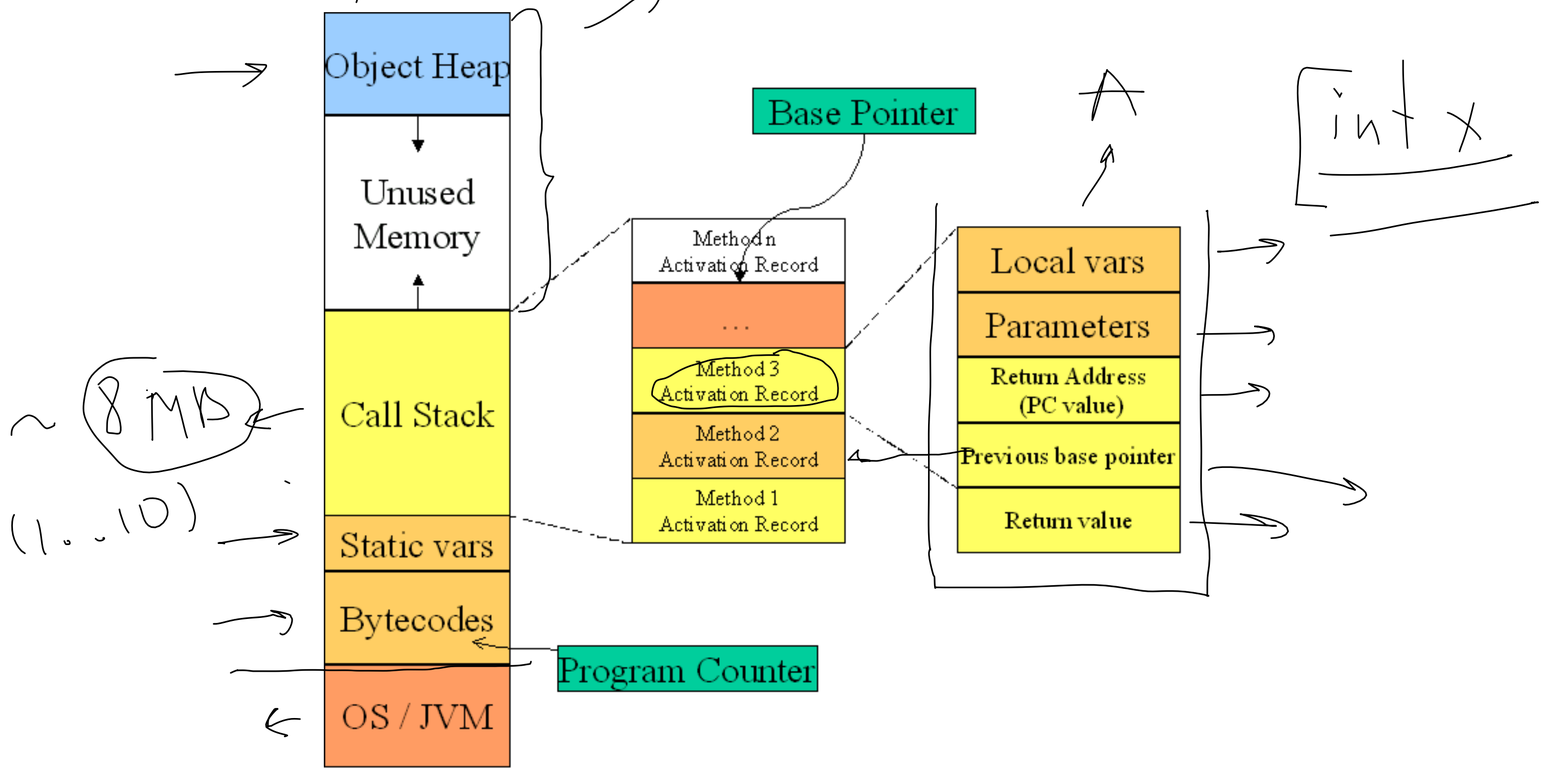
- Push(x)
  - add an element to the top
- Pop()
  - remove last added element from stack and return it
- LIFO
  - **L**ast **I**n **F**irst **O**ut



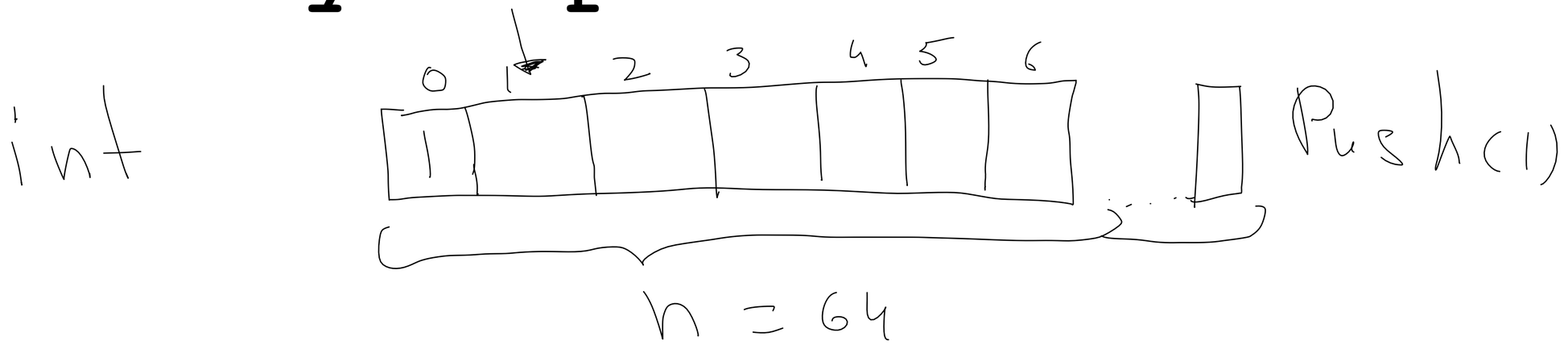
Applications

- Operation Systems
  - Memory management
  - Function Call Stack
- Calculators
  - Expression evaluation
  - Parenthesis matching

Call Stack

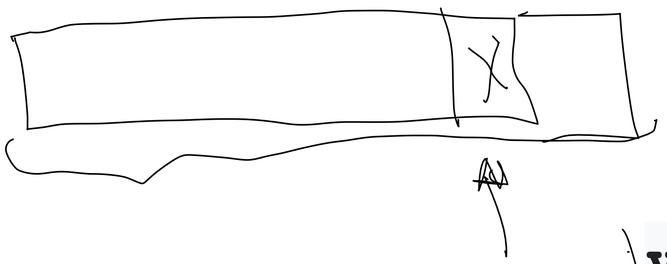

Object Heap
Unused Memory
Call Stack
Static vars
Bytecodes
OS / JVM
Program Counter
Base Pointer
Method n Activation Record
...
Method 3 Activation Record
Method 2 Activation Record
Method 1 Activation Record
Local vars
Parameters
Return Address (PC value)
Previous base pointer
Return value

Array Implementation Idea



- capacity = 64
- head = 0

Array Implementation.Push



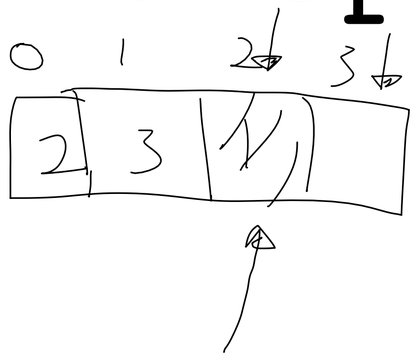

```
void Push(x):
  if head == capacity:
    ExtendStack()
  else:
    array[head] = x
    head = head + 1
```

head = 2

Push(4):

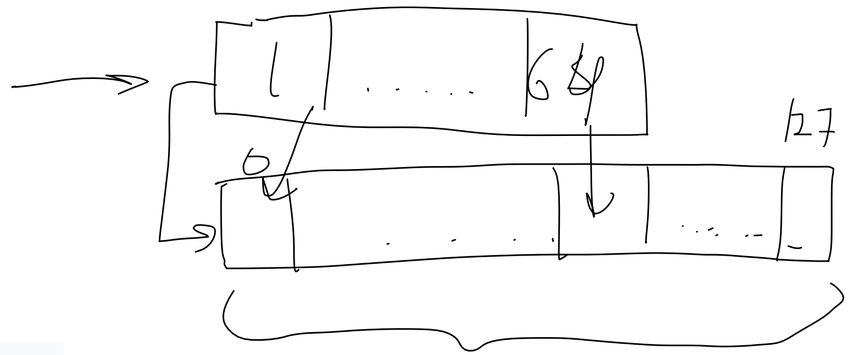
head = 3

Array Implementation.Pop




```
type Pop(x):
  if head == 0:
    error “underflow”
  else:
    head = head - 1
    return array[head]
```

Array Implementation.Extend



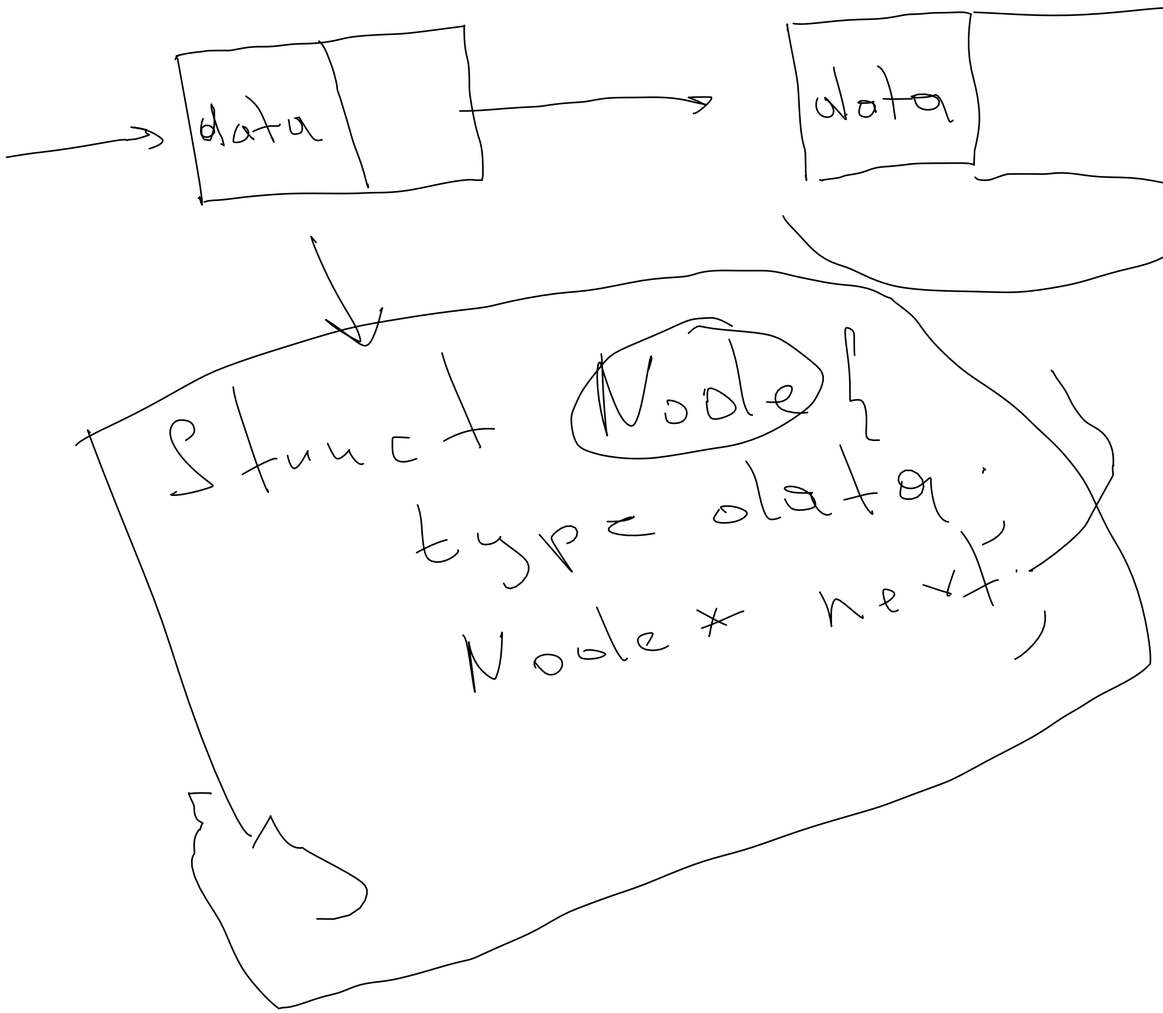
```
void ExtendStack(x):
  capacity = capacity * 2
  allocate new_array(capacity)
  copy array into new_array
  free(array)
```

array = new_array

O(1) → allocates on...

O(n) + O(n) = O(n)

Linked List Implementation Idea

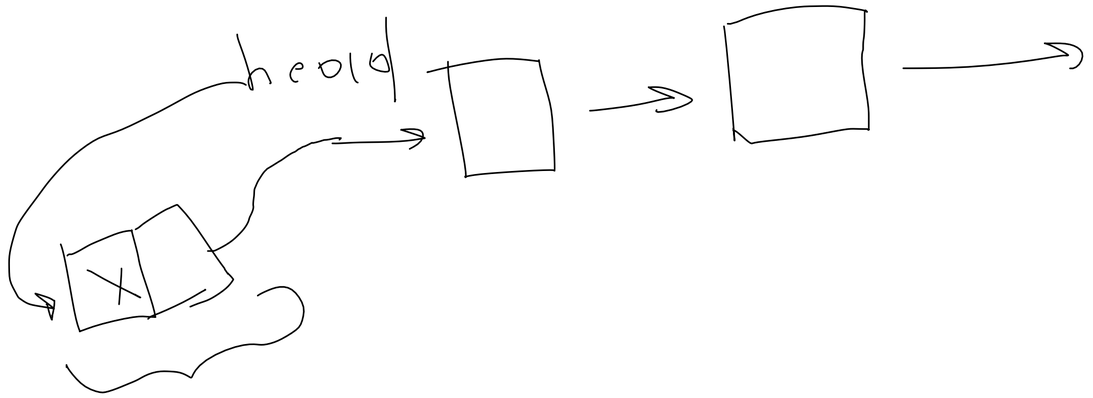


Fields:

- head = null
- size = 0

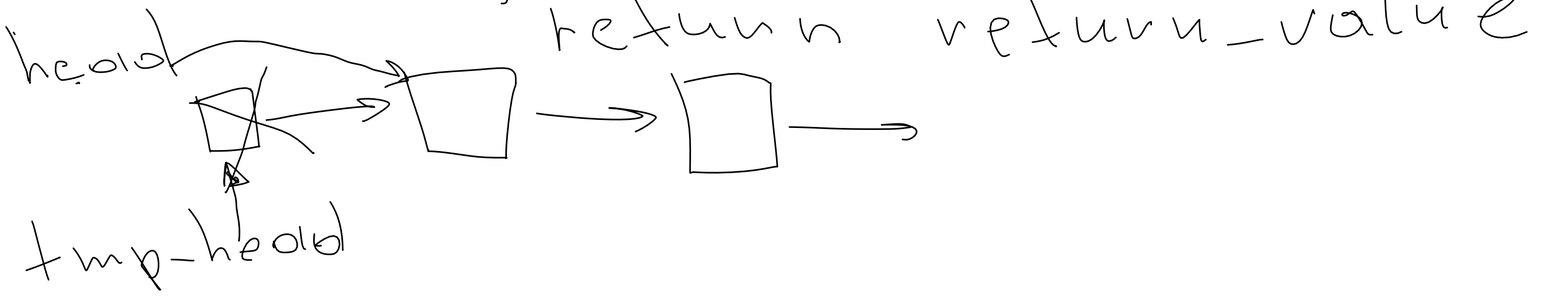
Linked List Implementation.Push

```
void Push(x):
  allocate new_head
  new_head.value = head.value
  new_head.next = head
  head = new_head
  size += 1
```



Linked List Implementation.Pop

```
type Pop(x):
  if head == null_ptr:
    error “underflow”
  else:
    return_value = head.value
    tmp_head = head
    head = head.next
    free(tmp_head)
    return return_value
```



Pros & Cons

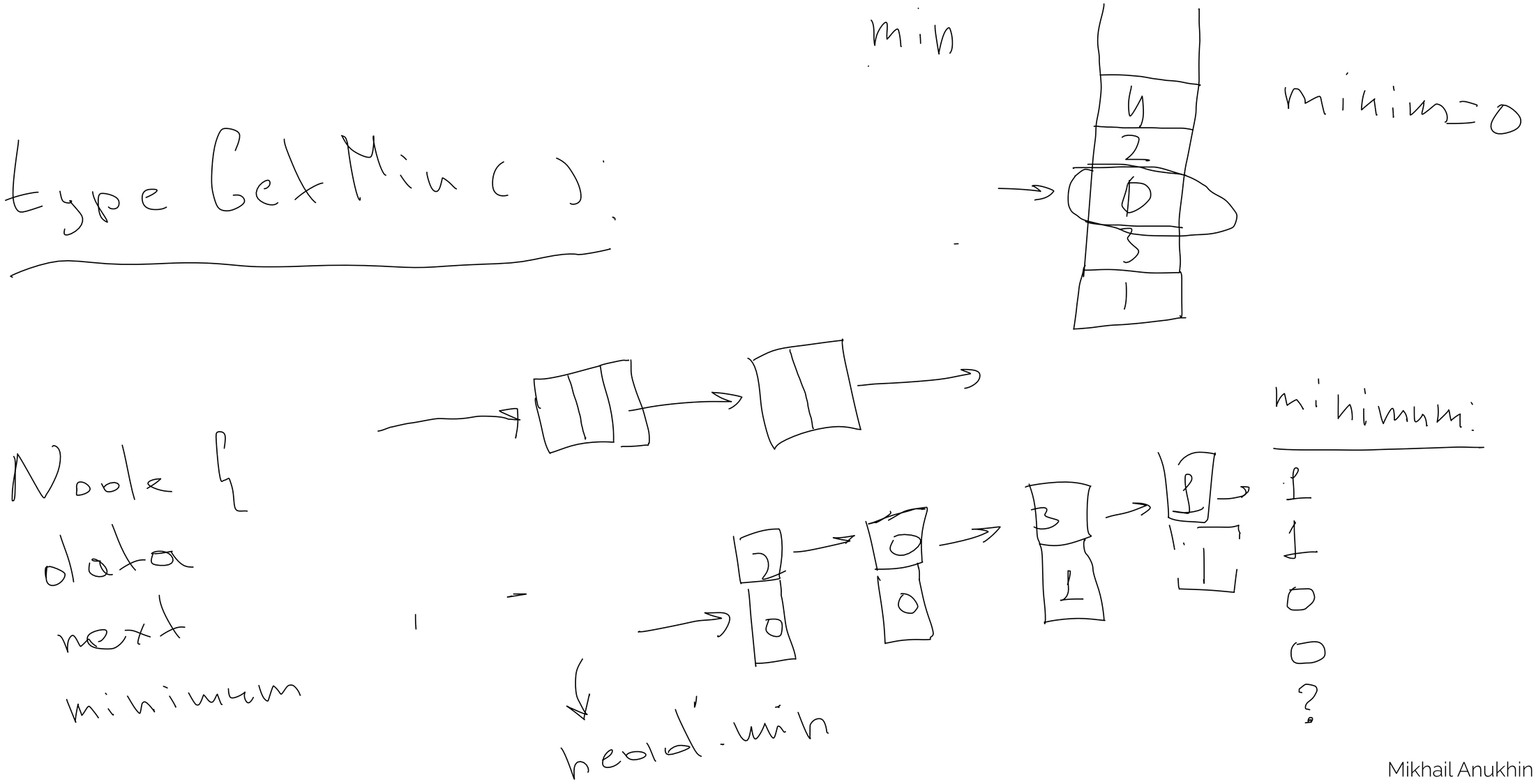
- Array
  Time Complexity
  Push : O(n)
  Pop : O(1)

  → Space Complexity:
  O(n)

- Linked List:
  Time complexity
  Push : O(1)
  Pop : O(1)

  Space Complexity:
  O(n)

Keeping minimum in Stack



Mikhail Anukhin

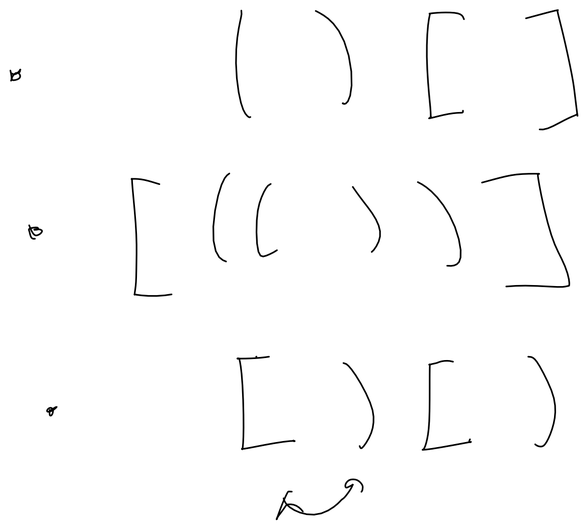
Correct brackets sequence.Definition

Alphabet = { '(', '[', '{', '}', ']', ')' }

Sequence S is correct if:

- empty sequence is correct: $S = \emptyset$
- $S \rightarrow (S)$
  $[S]$
  $\{S\}$
- $S \rightarrow S_1 S_2$

Correct brackets sequence.Example

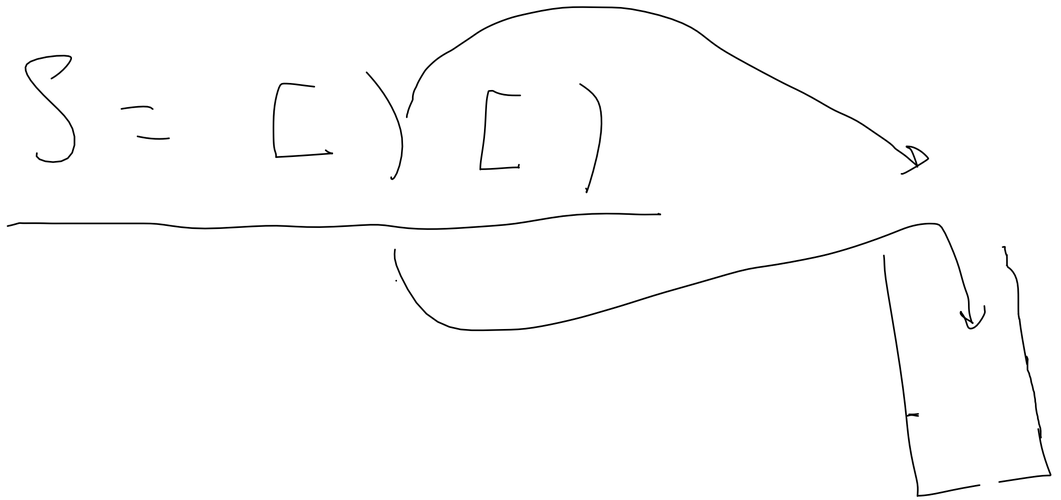


Mikhail Anukhin

Correct brackets sequence.Problem

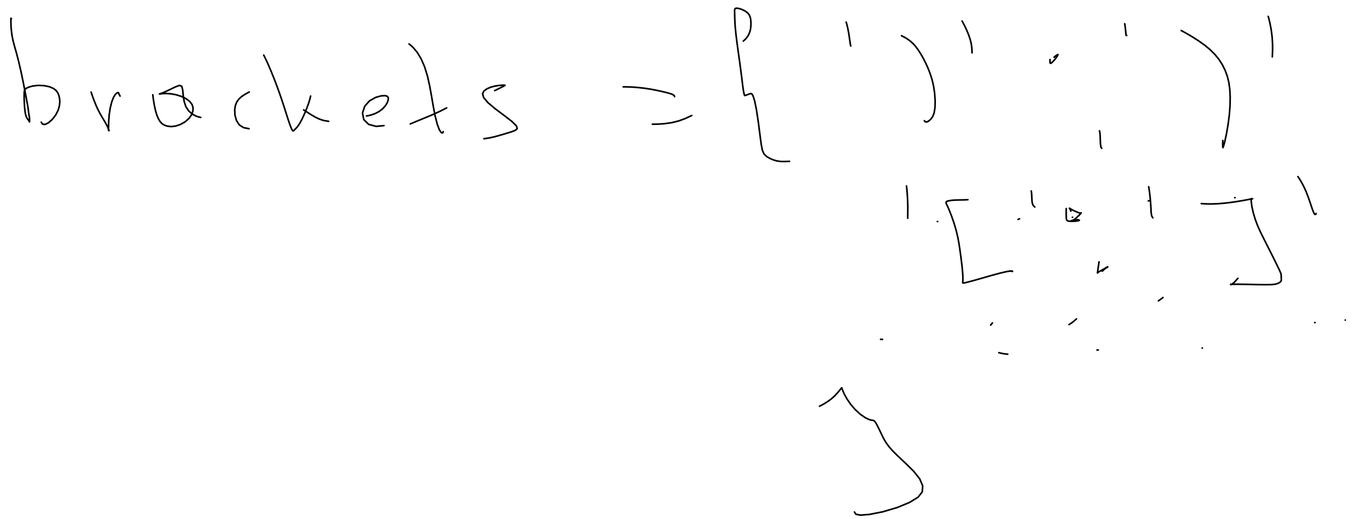
Problem: Check if Sequence S is correct

['(', ')']
['[', ']']



Mikhail Anukhin

Correct brackets sequence.Algorithm

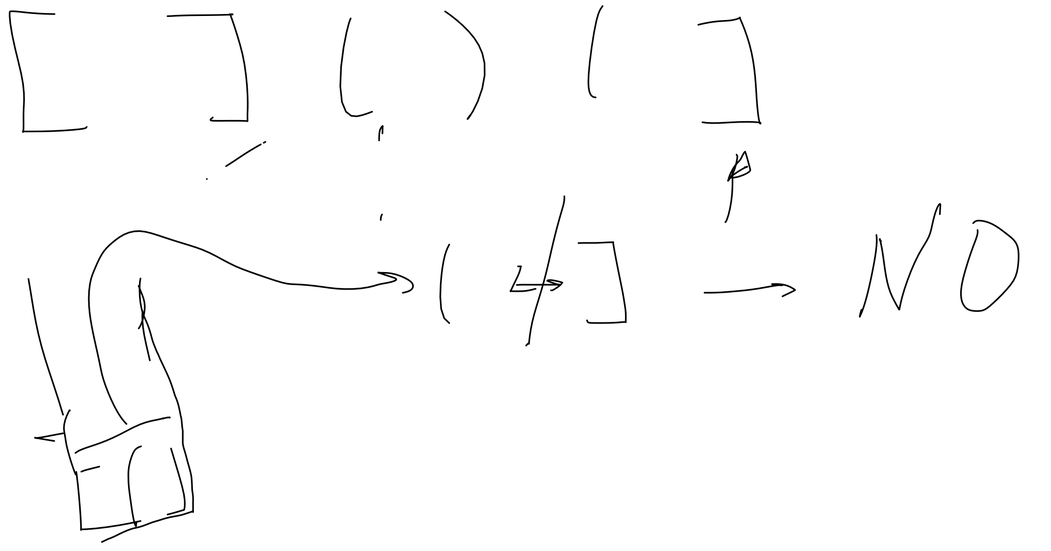


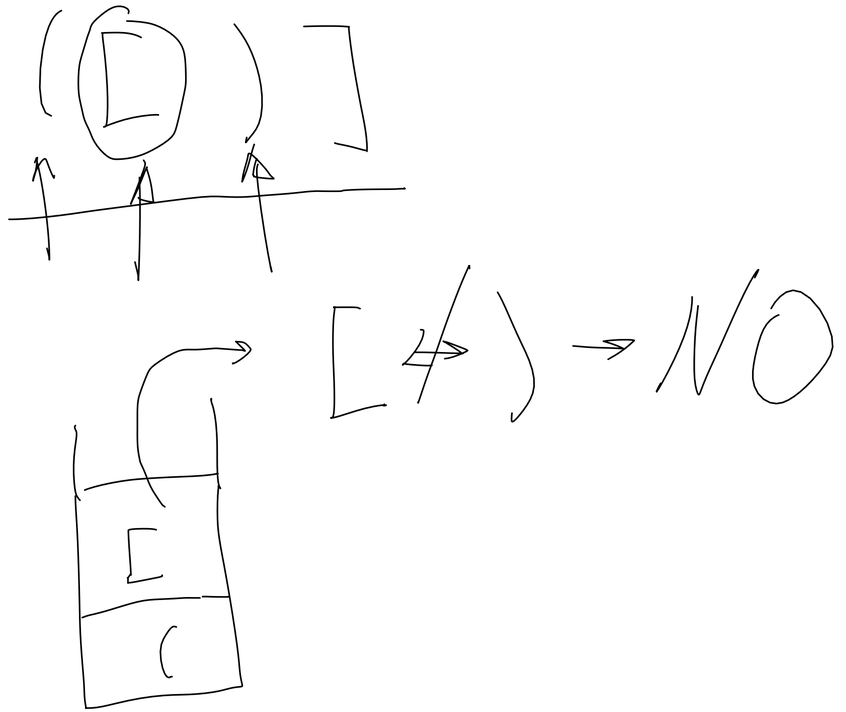
Mikhail Anukhin

Correct brackets sequence.Pseudocode

```
for symbol in S:
    if (isClosedBracket(symbol)){
        if (stack.empty()){
            return NO
        }else{
            last_element= stack.pop()
            ClassOf(last_element) != ClassOF(symbol)
                return false
```

```
        }
    }else{
            stack.push(symbol);
        }
    }
    return stack.empty();
```


NO


NO

Your questions!