Алгоритмы и структуры данных

Линейные контейнеры.
ADT Стек, Очередь, Список



Практическое занятие **02**

2026–2027 учебный год

План

1. Реализация и применение ADT **Стек**, ADT **Очередь**
   - Правильная скобочная последовательность
   - Очередь на двух стеках и ассоциативные операции
2. Амортизационная оценка сложности • Метод **банкира**

Warm-Up

FAQ Контейнеры и упорядочивание

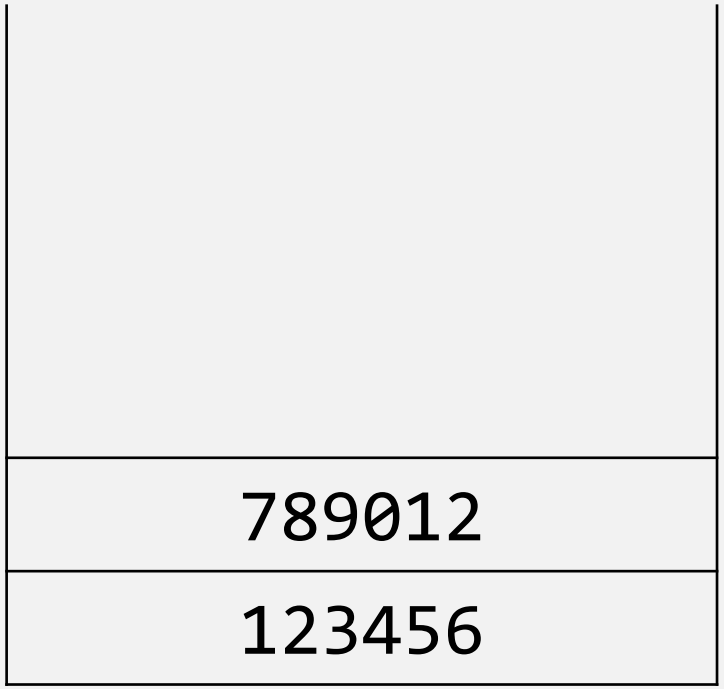
1. Выделите основные компоненты интерфейса контейнера
2. Что требуется для линейного упорядочивания объектов?
3. Чем отличается линейный порядок от частичного?
4. Какие операции эффективнее выполняются в связном списке, а какие — в обычном массиве?

ADT Стек

Last In First Out

Вспомним

Интерфейс ADT Стек



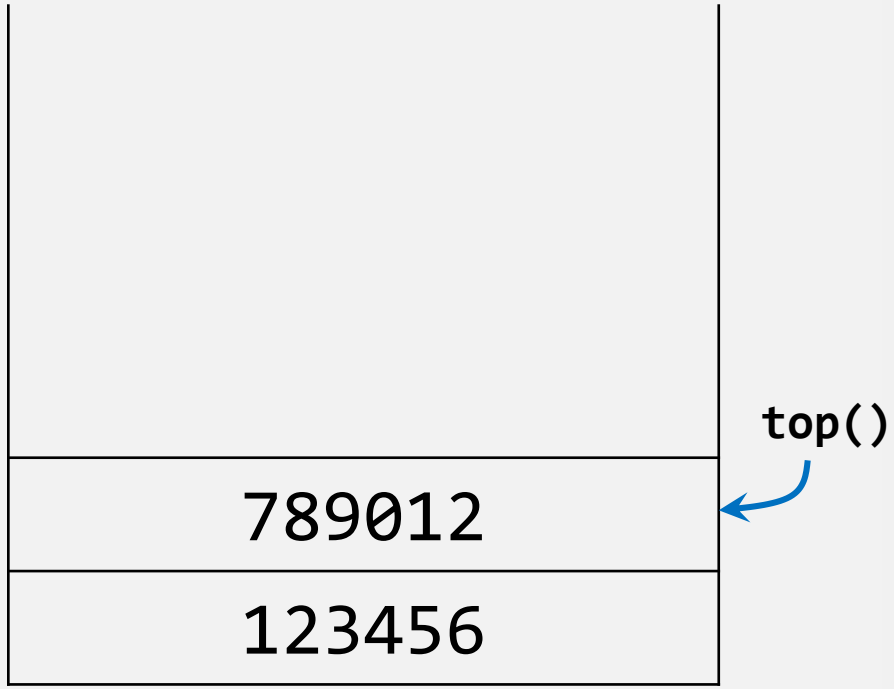
ADT_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

Вспомним

Интерфейс ADT Стек • top( )



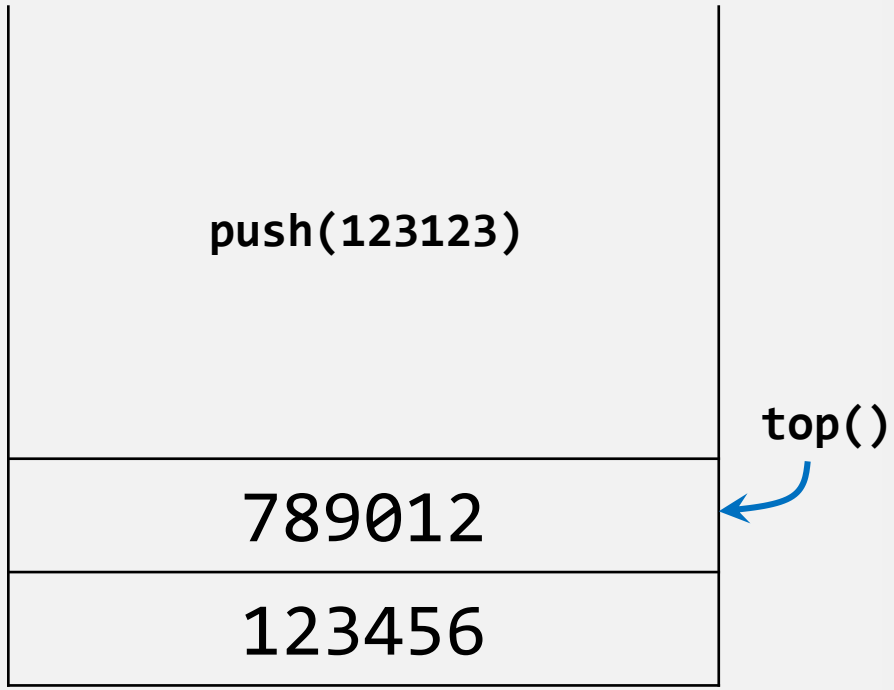
ADT_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

Вспомним

Интерфейс ADT Стек • push(T elem)



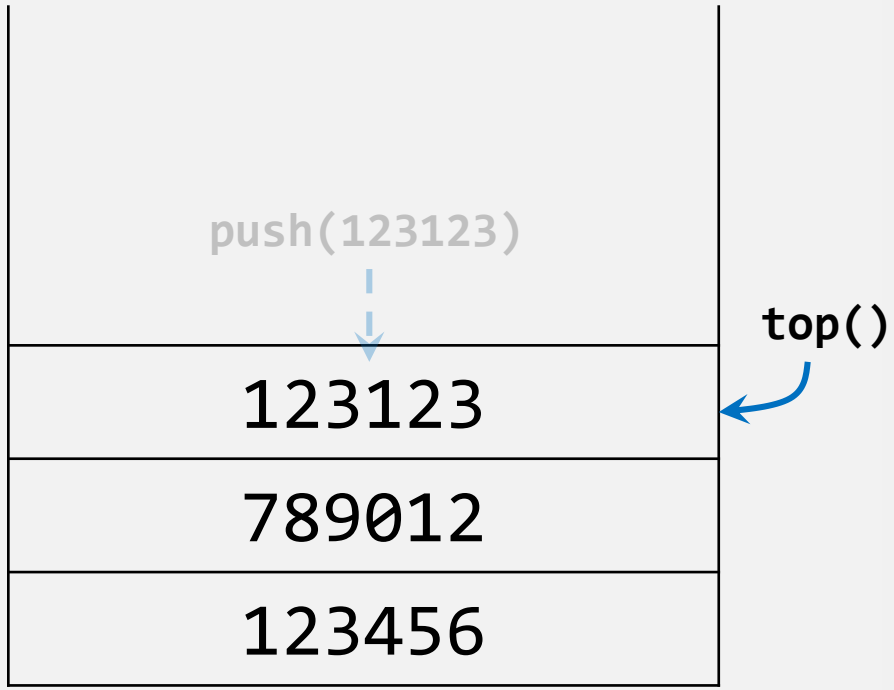
ADT_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

Вспомним

Интерфейс ADT Стек • push(T elem)



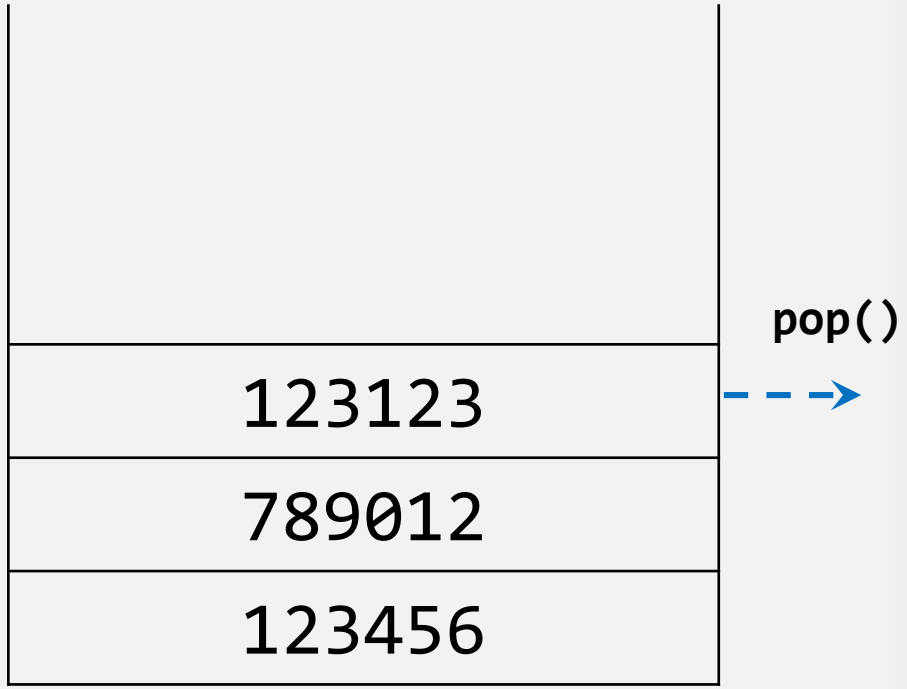
ADT_Stack.cpp

```cpp
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

Вспомним

Интерфейс ADT Стек • pop( )



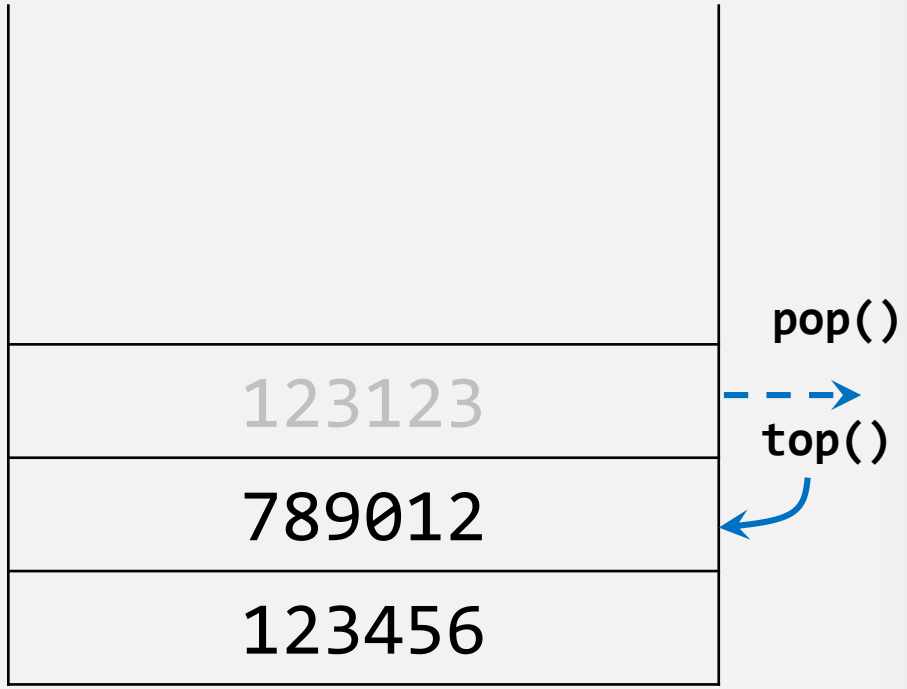
ADT_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

Вспомним

Интерфейс ADT Стек • pop( )



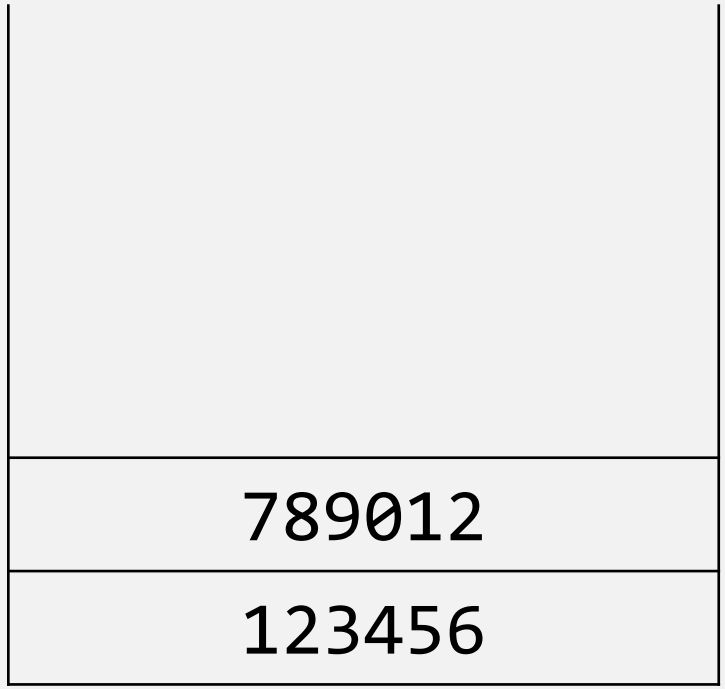
ADT_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

Вспомним

Интерфейс ADT Стек • исключения



Ошибка вставки в стек
`StackOverflow`

Ошибка удаления из стека
`StackUnderflow`

Ошибка чтения вершины
`StackIsEmpty`

Правильная расстановка скобок в выражении

Задача 1





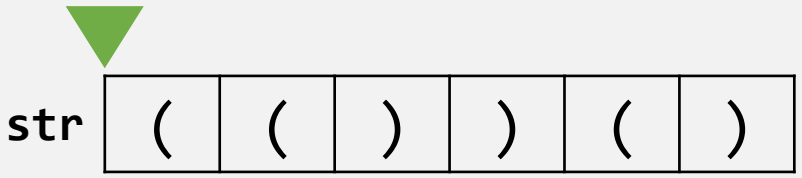
Правильная последовательность скобок

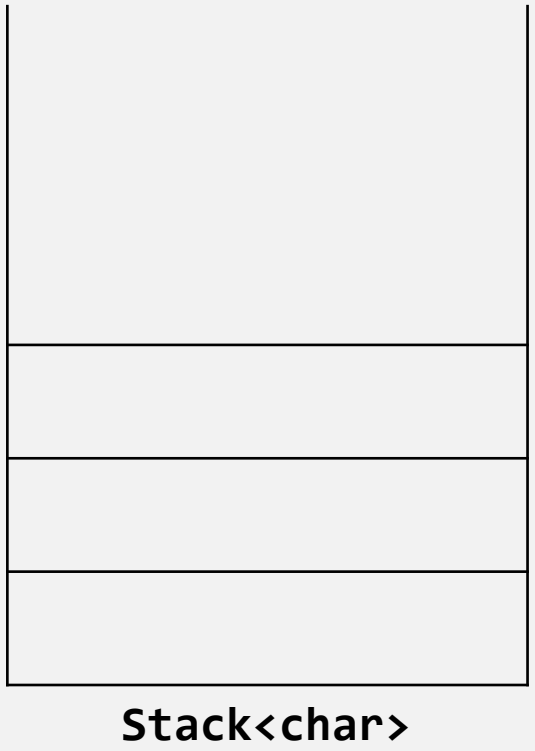
Вход: Строка **str**, состоящая из конечного числа открывающих **'('** и закрывающих **')'** скобок.

Выход: **YES**, если скобки расставлены верно;
NO — в противном случае.

Задача 1

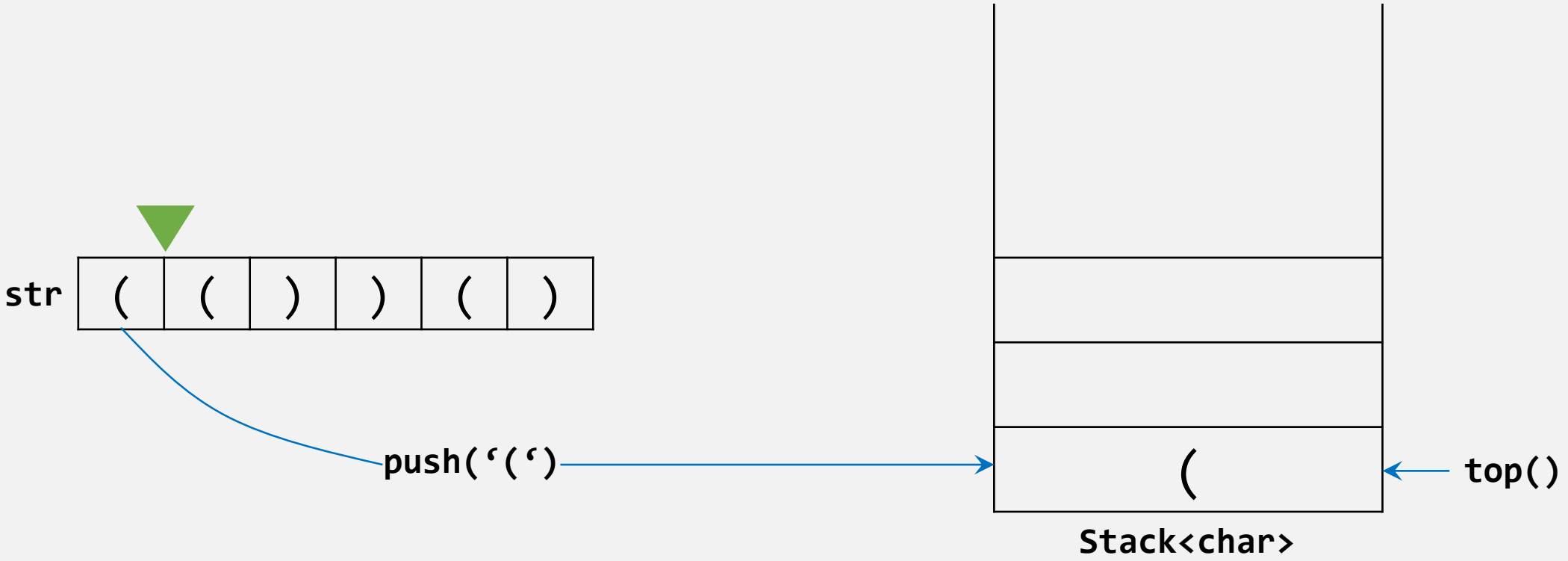
Правильная последовательность скобок





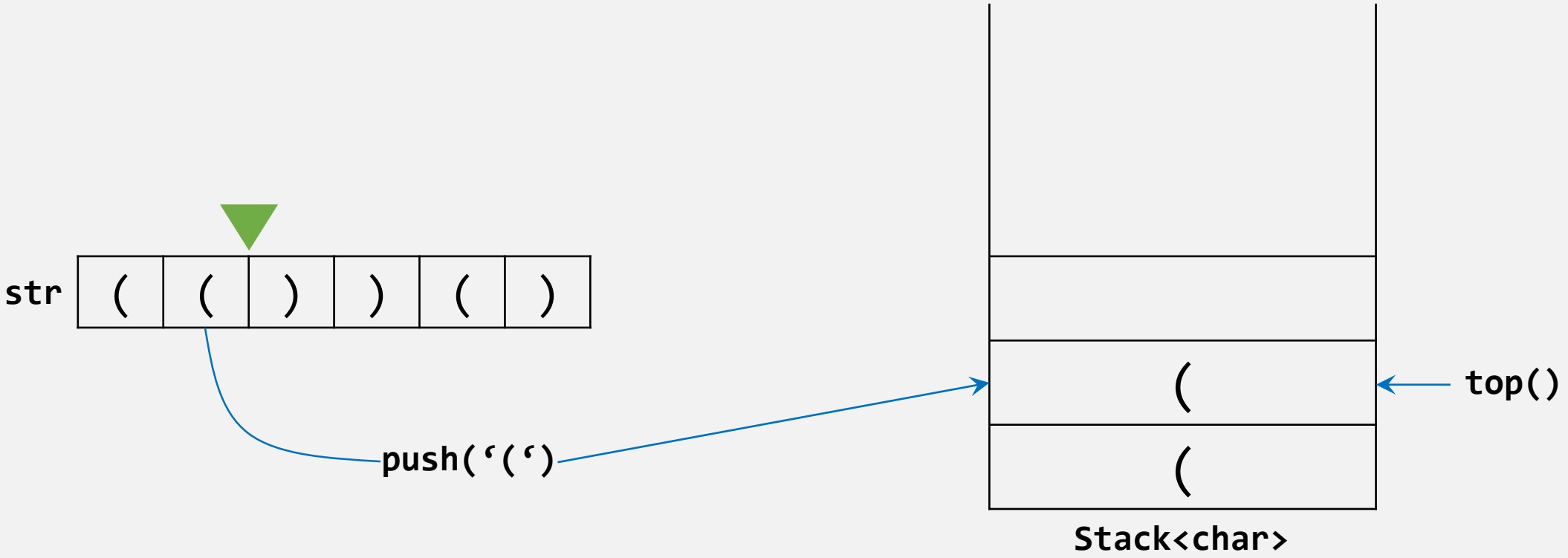
Задача 1

Правильная последовательность скобок



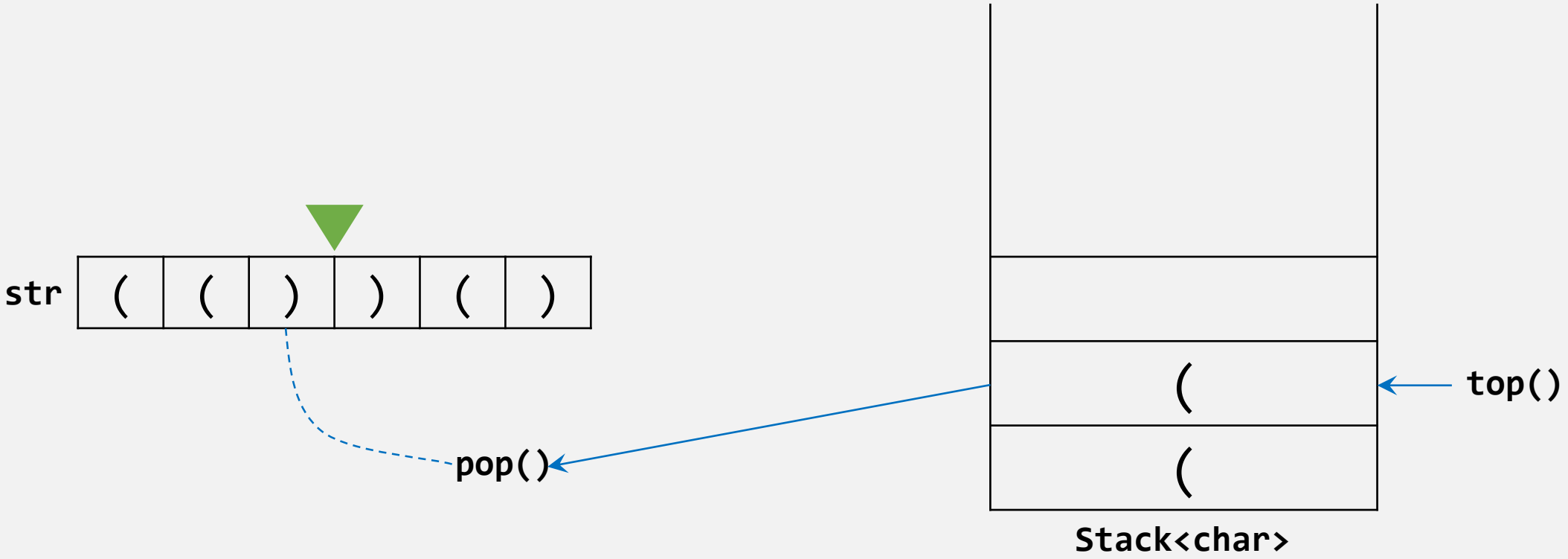
Задача 1

Правильная последовательность скобок



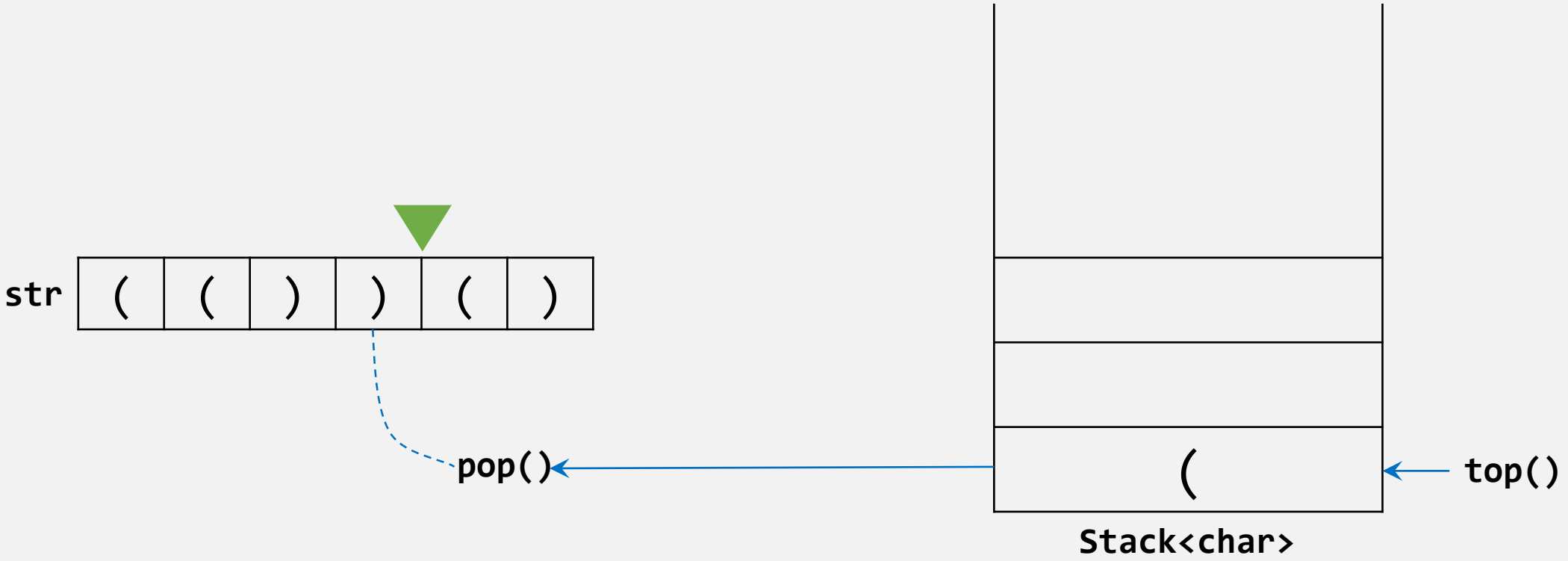
Задача 1

Правильная последовательность скобок



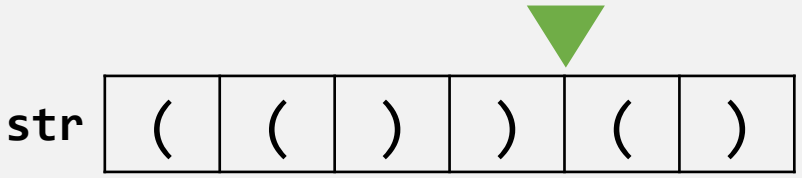
Задача 1

Правильная последовательность скобок

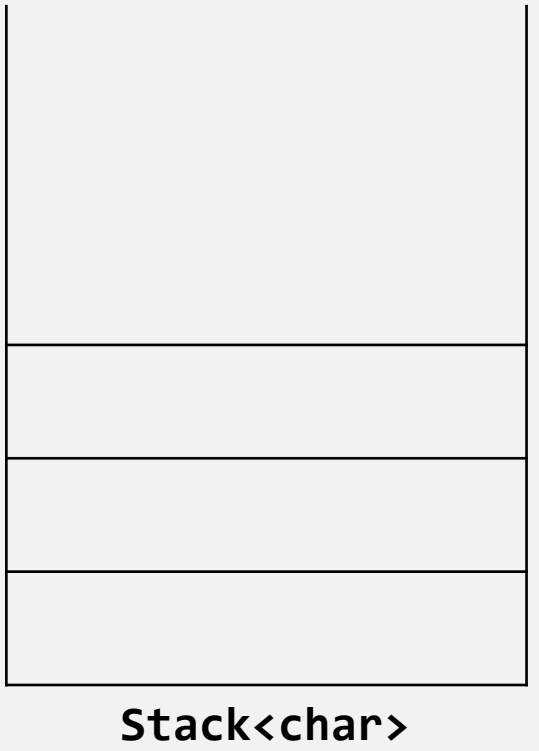


Задача 1

Правильная последовательность скобок



TO BE CONTINUED...



Задача 1



Правильная последовательность скобок

[Q] Когда скобки расставлены верно?

Задача 1





Правильная последовательность скобок

[Q] Когда скобки расставлены верно?

[A] Прочитана **вся входная строка** `str`
без попыток чтения из пустого стека, а стек пуст

Задача 1





Правильная последовательность скобок

[Q] Когда скобки расставлены верно?

[A] Прочитана **вся входная строка** `str`
без попыток чтения из пустого стека, а стек пуст

[Q] Как характеризуется сложность алгоритма?

Задача 1





Правильная последовательность скобок

[Q] Когда скобки расставлены верно?

[A] Прочитана **вся входная строка** `str`
без попыток чтения из пустого стека, а стек пуст

[Q] Как характеризуется сложность алгоритма?

[A] $\Theta(n)$ по времени и $\Theta(n)$ для стека (по памяти)
n — длина входной строки `str`

Задача 1





Правильная последовательность скобок

[Q] Когда скобки расставлены верно?

[A] Прочитана **вся входная строка** `str`
без попыток чтения из пустого стека, а стек пуст

[Q] Как характеризуется сложность алгоритма?

[A] $\Theta(n)$ по времени и $\Theta(n)$ для стека (по памяти)
n — длина входной строки `str`

можно реализовать и проще, но применение стека дает
наиболее универсальный способ решения этой задачи

Практические приложения ADT Стек

Организация рекурсивных вызовов

Операции Undo/Redo

Вычисление значения арифметического выражения в различных нотациях

Алгоритмы с возвратом backtracking

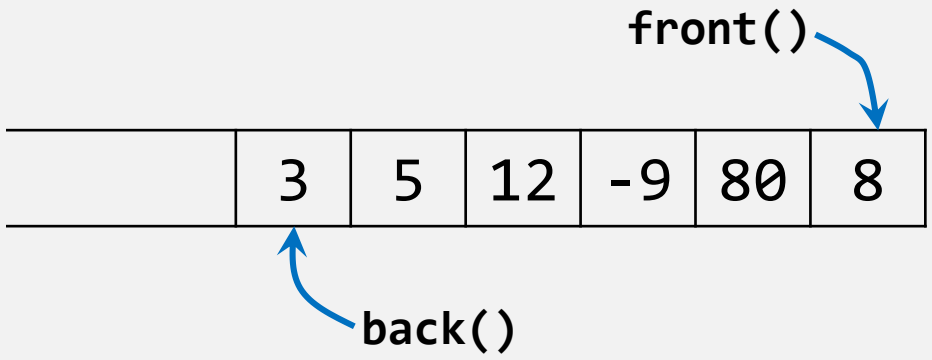
...

ADT Очередь

First In First Out

Вспомним

Интерфейс ADT Очередь

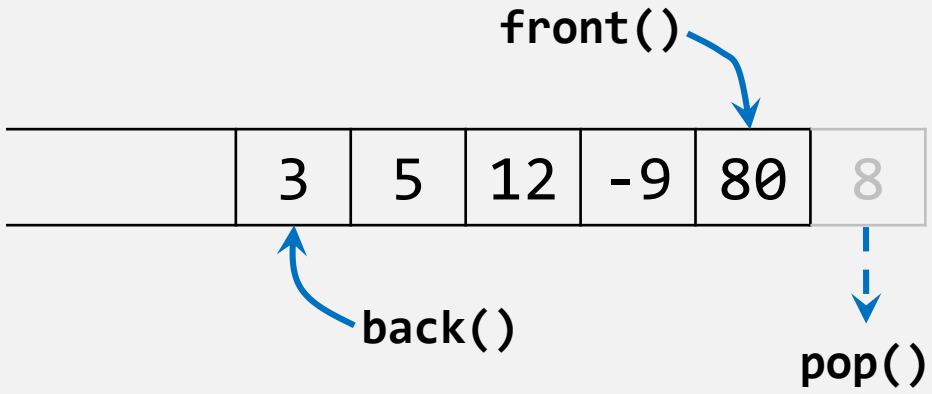


ADT_Queue.cpp

```cpp
template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

Вспомним

Интерфейс ADT Очередь • pop( )

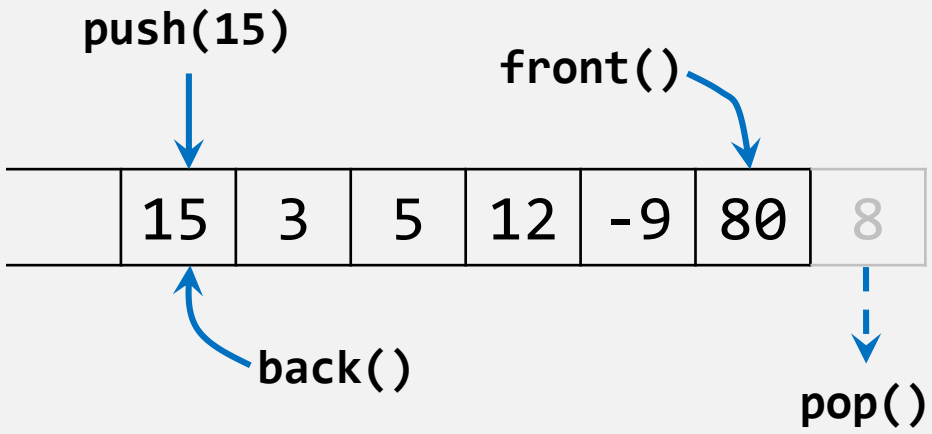


ADT_Queue.cpp

```cpp
template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

Вспомним

Интерфейс ADT Очередь • push(T elem)

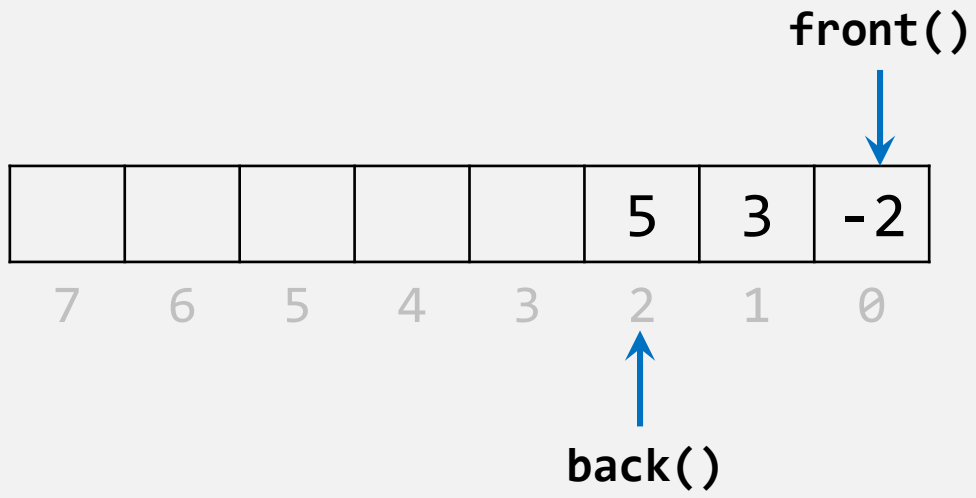


ADT_Queue.cpp

```cpp
template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

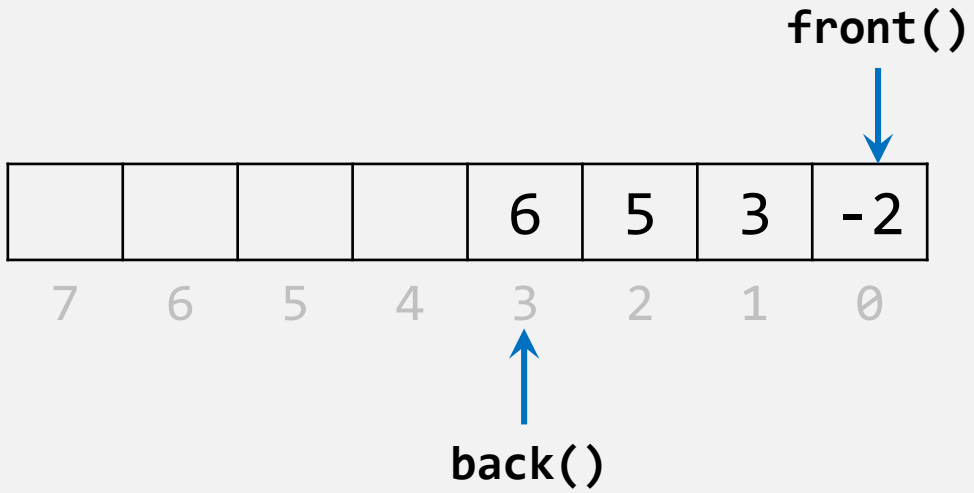
Задача 2

Реализация ADT Очередь на массиве



Задача 2

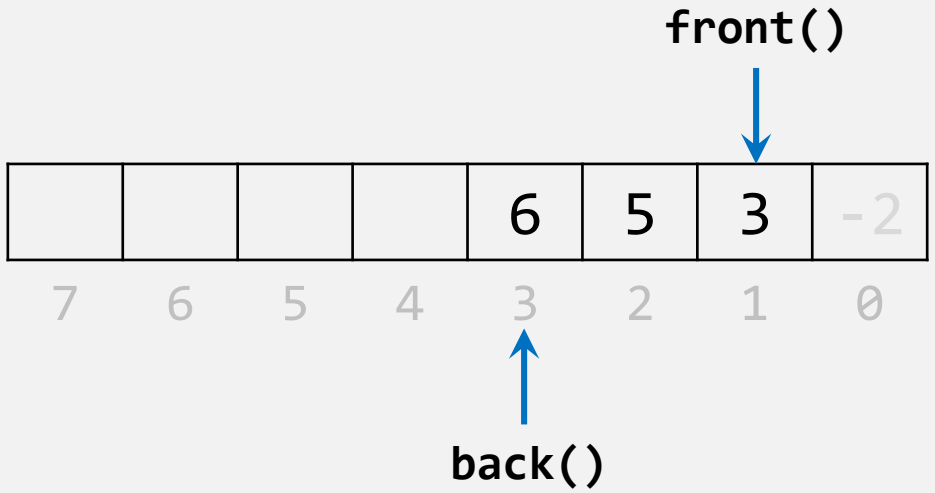
Реализация ADT Очередь на массиве




```
push(6)        back += 1
```

Задача 2

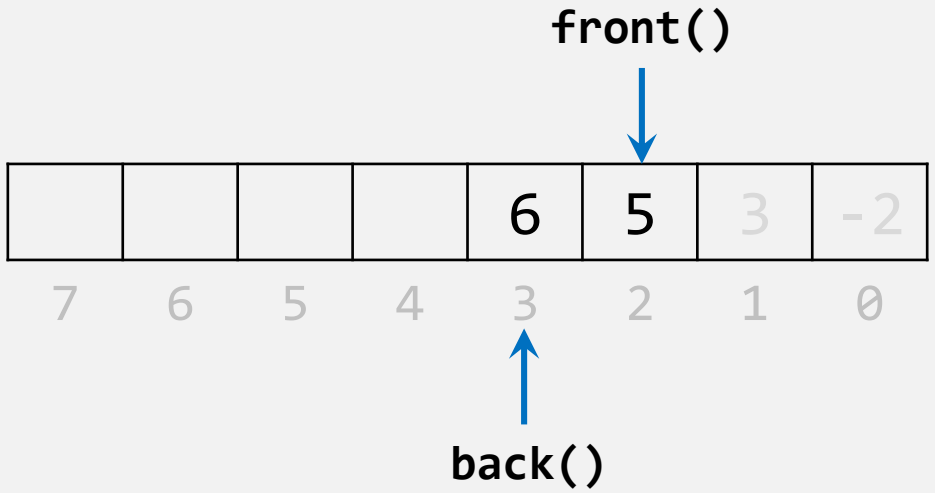
Реализация ADT Очередь на массиве




```
push(6)        back += 1
pop()          front += 1
```

Задача 2

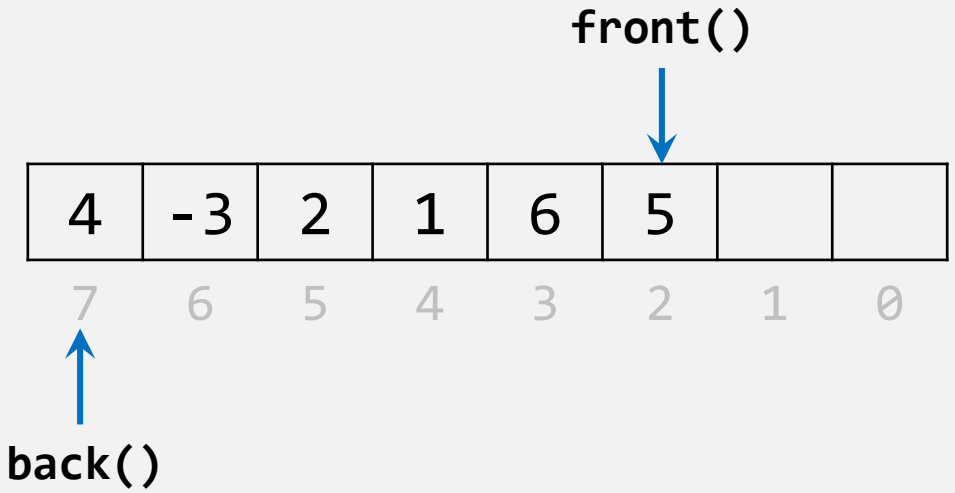
Реализация ADT Очередь на массиве




```
push(6)        back += 1
pop()          front += 1
pop()          front += 1
```

Задача 2

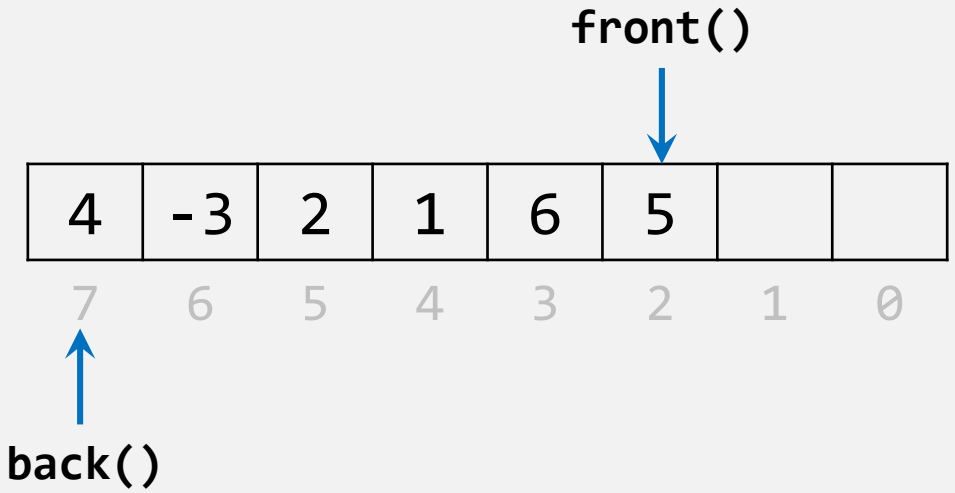
Реализация ADT Очередь на массиве




```
push(6)      back += 1
pop()        front += 1
pop()        front += 1
push(1)      back += 1
push(2)      back += 1
push(-3)     back += 1
push(4)      back += 1
```

Задача 2

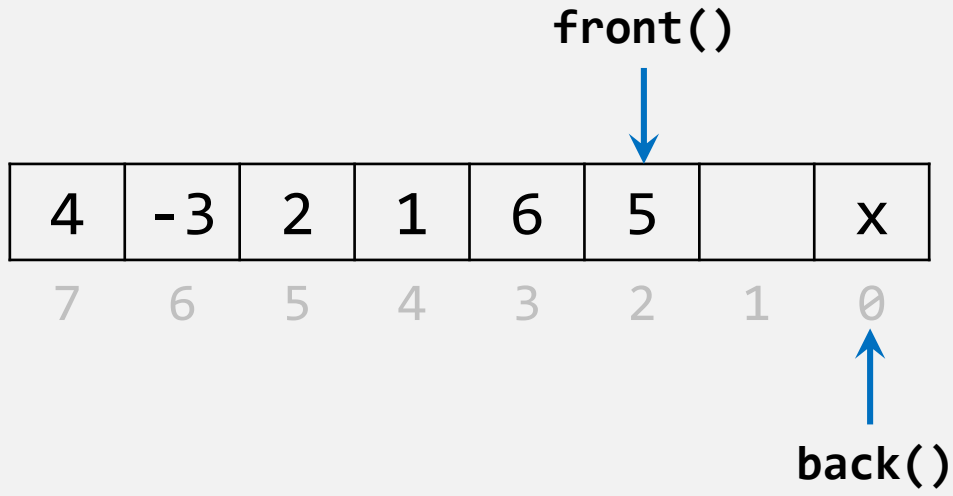
Реализация ADT Очередь на массиве



push(6)	back += 1
pop()	front += 1
pop()	front += 1
push(1)	back += 1
push(2)	back += 1
push(-3)	back += 1
push(4)	back += 1
push(x)	**???**

Задача 2

Реализация ADT Очередь на массиве



ЗАКОЛЬЦОВАННЫЙ МАССИВ

```
push(6)          back += 1
 pop()          front += 1
 pop()          front += 1
push(1)          back += 1
push(2)          back += 1
push(-3)         back += 1
push(4)          back += 1
push(x)    back = (back + 1) % n
```

Практические приложения ADT Очредь

Поиск в ширину (BFS) в графовых структурах данных

Выделение времени процессора

Асинхронная коммуникация

Сетевые протоколы

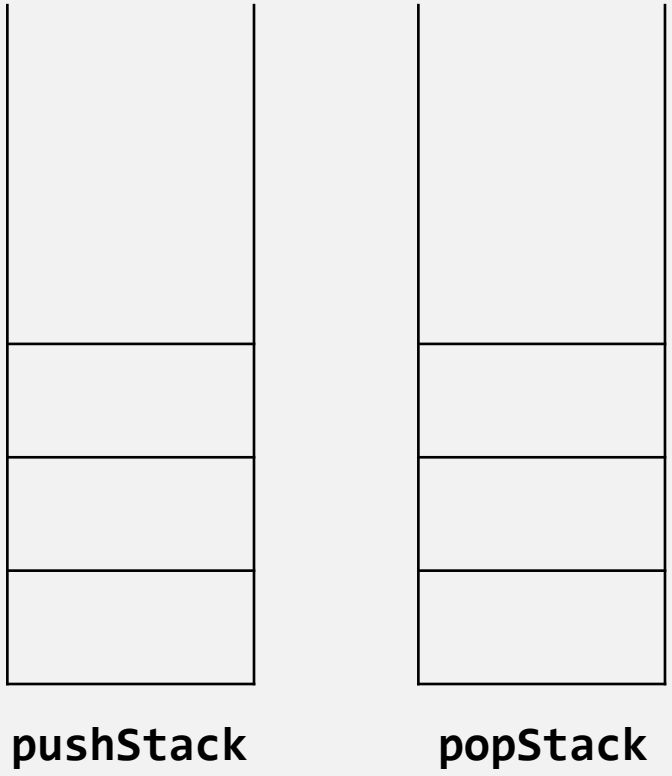
...

ADT Очередь

на двух стеках

Задача 3

Реализация ADT Очередь на двух стеках

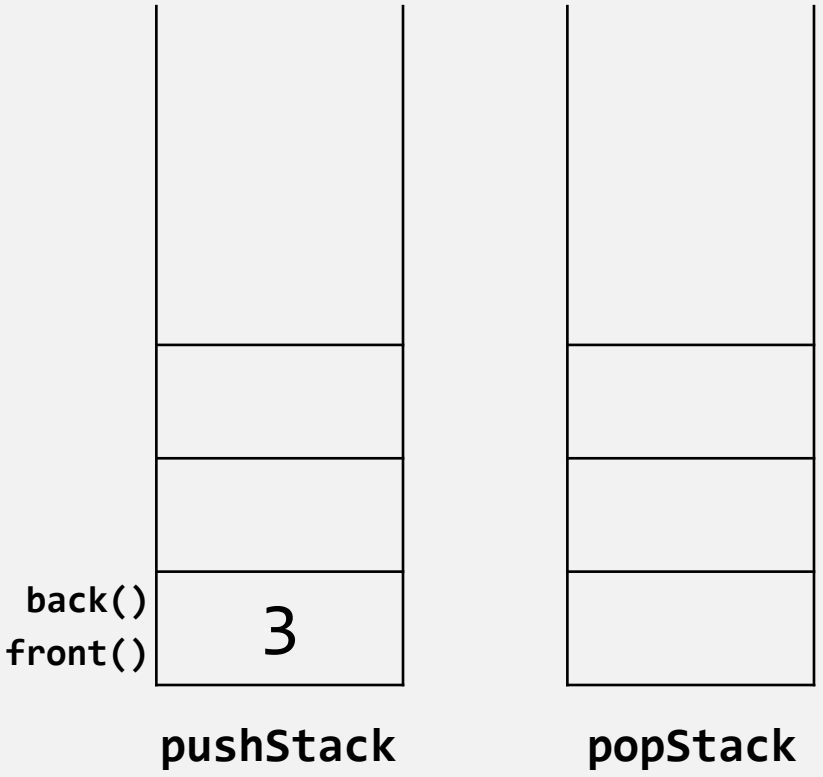


ADT_Queue.cpp

```cpp
template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    Stack<T> pushStack;
    Stack<T> popStack;
}
```

Задача 3

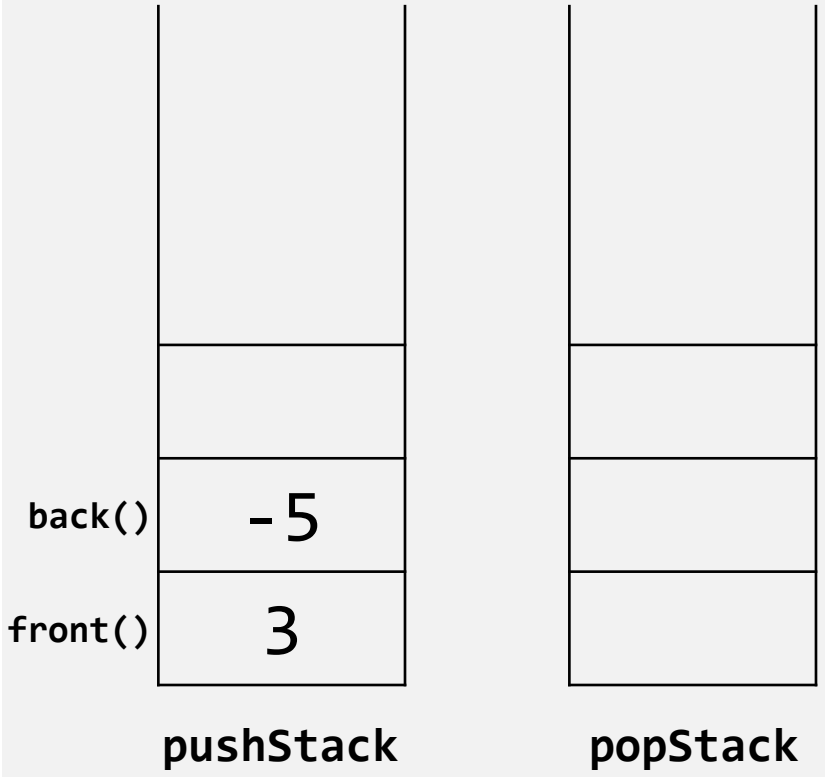
Реализация ADT Очередь на двух стеках




```
push(3)        pushStack.push(3)
```

Задача 3

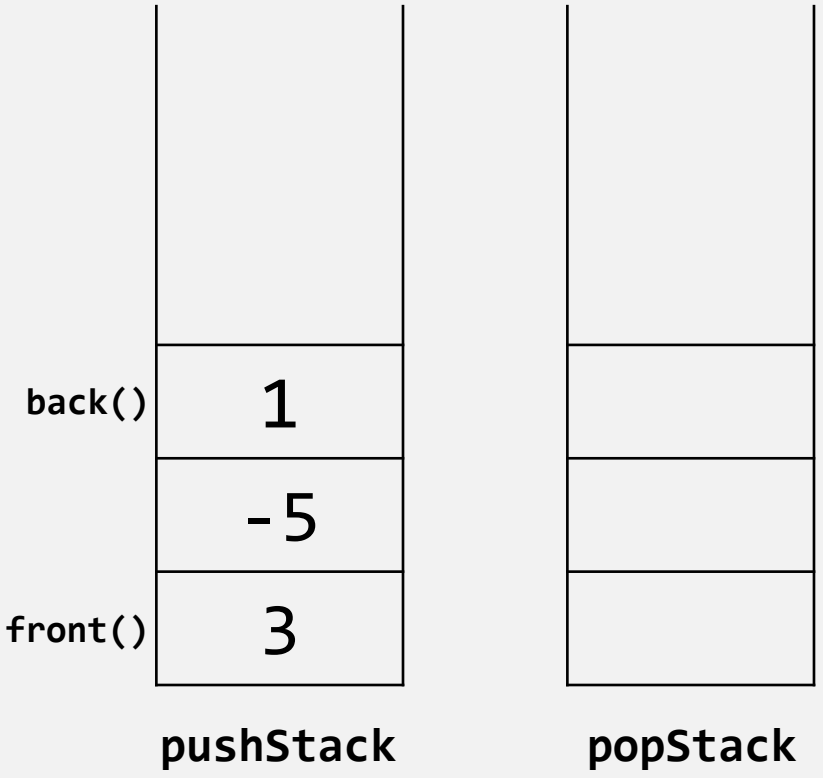
Реализация ADT Очередь на двух стеках



push(3)	**pushStack.**push(3)
push(-5)	**pushStack.**push(-5)

Задача 3

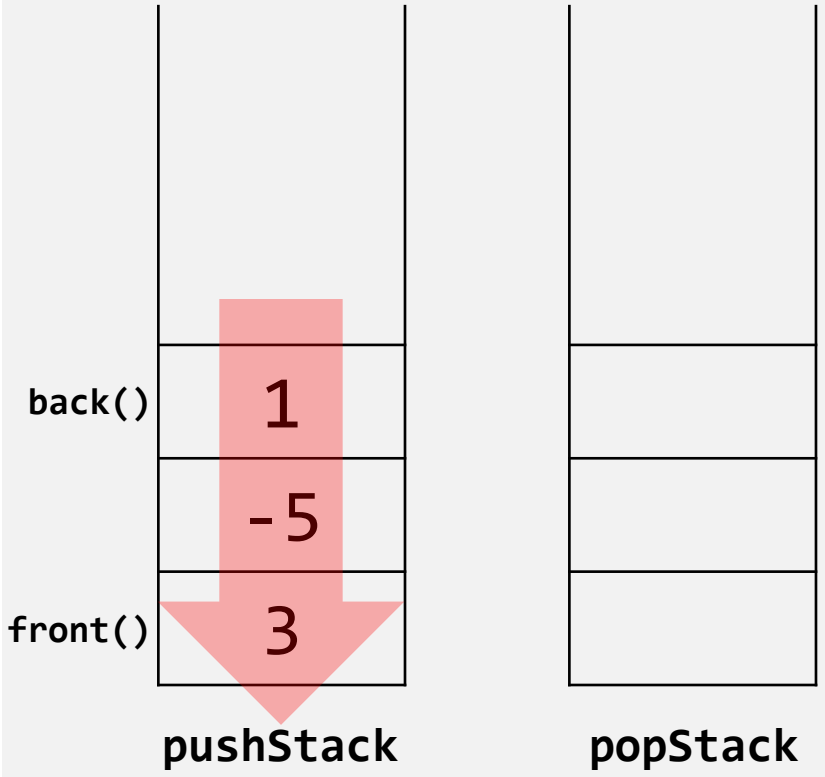
Реализация ADT Очередь на двух стеках



push(3)	**pushStack.**push(3)
push(-5)	**pushStack.**push(-5)
push(1)	**pushStack.**push(1)

Задача 3

Реализация ADT Очередь на двух стеках



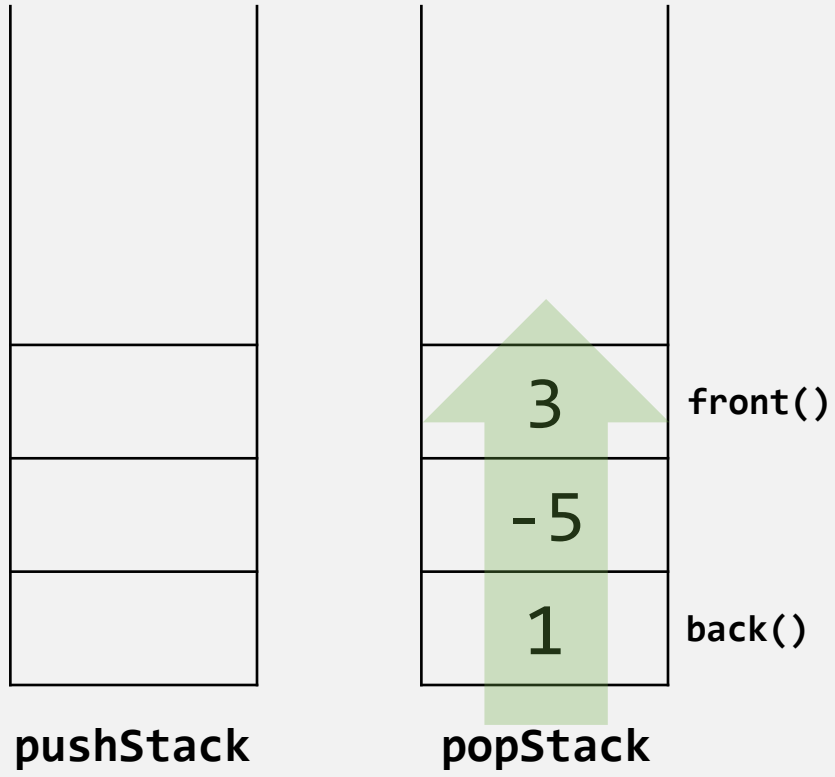
push(3)	**pushStack**.push(3)
push(-5)	**pushStack**.push(-5)
push(1)	**pushStack**.push(1)
pop()	

- в **popStack** нет элементов
- переложим элементы из **pushStack** в **popStack**

к слову, порядок элементов в **pushStack** не соответствует обычному порядку в очереди

Задача 3

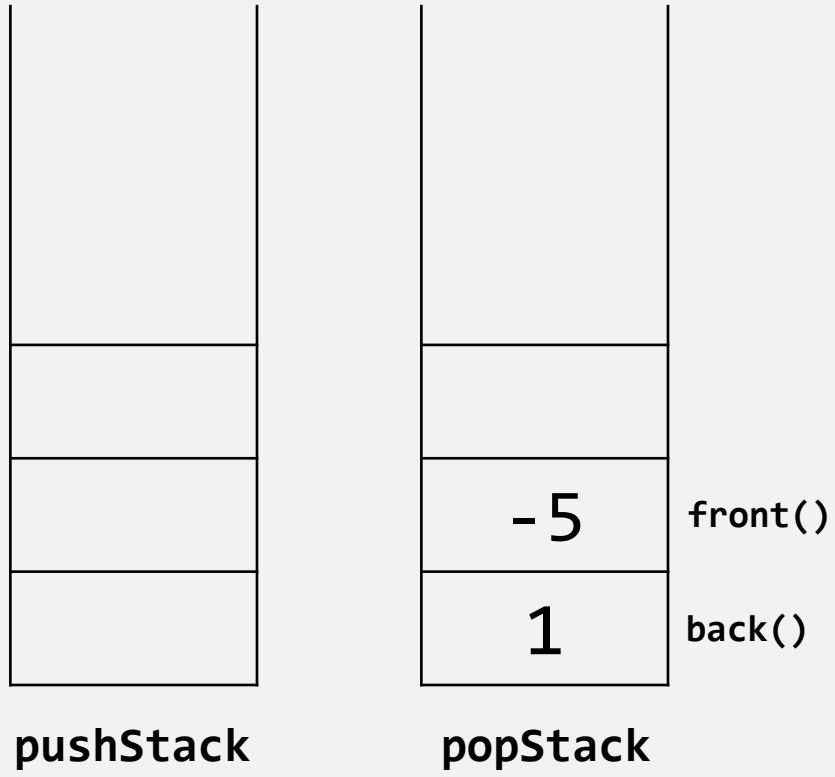
Реализация ADT Очередь на двух стеках



push(3)	**pushStack**.push(3)
push(-5)	**pushStack**.push(-5)
push(1)	**pushStack**.push(1)
pop()	**popStack**.push(**pushStack**.pop()) **popStack**.push(**pushStack**.pop()) **popStack**.push(**pushStack**.pop())

Задача 3

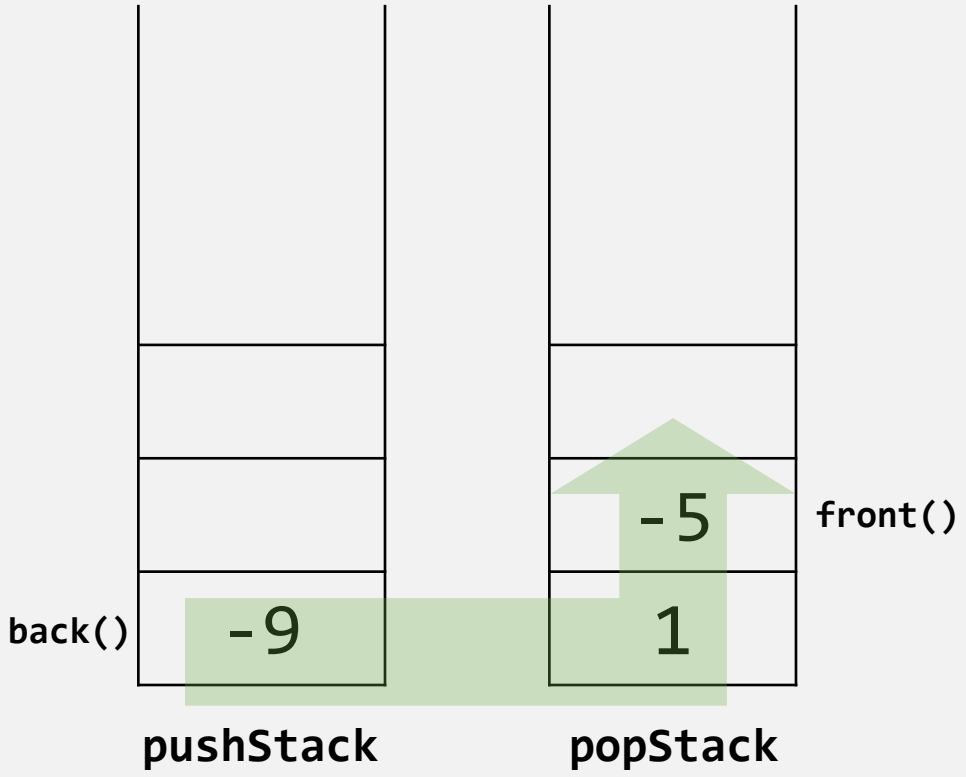
Реализация ADT Очередь на двух стеках



push(3)	**pushStack**.push(3)
push(-5)	**pushStack**.push(-5)
push(1)	**pushStack**.push(1)
pop()	**popStack**.push(**pushStack**.pop()) **popStack**.push(**pushStack**.pop()) **popStack**.push(**pushStack**.pop()) **popStack**.pop()

Задача 3

Реализация ADT Очередь на двух стеках



push(3)	**pushStack**.push(3)
push(-5)	**pushStack**.push(-5)
push(1)	**pushStack**.push(1)
pop()	**popStack**.push(**pushStack**.pop()) **popStack**.push(**pushStack**.pop()) **popStack**.push(**pushStack**.pop()) **popStack**.pop()
push(-9)	**pushStack**.push(-9)

Задача 3




Сложность ADT Очередь на двух стеках

		фактическая реализация	сложность	комментарий
Queue.push()		pushStack.push()	$\Theta(1)$	простое добавление объекта в стек
Queue.pop()	**!popStack.empty()**	popStack.pop()	$\Theta(1)$	простое удаление объекта из стека
	popStack.empty()	popStack.push(…) popStack.push(…) … popStack.push(…) popStack.pop()	$\Theta(S) + \Theta(1)$ S — размер стека для вставки	○ перекладывание из pushStack в popStack ○ удаление из стека

Задача 3




Сложность ADT Очередь на двух стеках

		фактическая реализация	сложность	комментарий
`Queue.push()`		`pushStack.push()`	$\Theta(1)$	простое добавление объекта в стек
`Queue.pop()`	`!popStack.empty()`	`popStack.pop()`	$\Theta(1)$	простое удаление объекта из стека

	`popStack.empty()`	`popStack.push(…)`
		`popStack.push(…)`
		`…`
		`popStack.push(…)`
		`popStack.pop()`
			$\Theta(S) + \Theta(1)$
			S — размер стека для вставки
				○ перекладывание из `pushStack` в `popStack`
				○ удаление из стека

для того, чтобы вычислить обобщенную оценку операции **`Queue.pop()`**, обратимся к способам **усреднения** временной сложности

средняя сложность $\bar{T}$

- выборкой из какого распределения являются входные данные?
- какова вероятность того, что массив уже будет отсортирован?

....

$\bar{T}$ — математическое ожидание временной сложности

[пример???]

средняя сложность $\bar{T}$

- выборкой из какого распределения являются входные данные?
- какова вероятность того, что массив уже будет отсортирован?

....

$\bar{T}$ — математическое ожидание временной сложности

[пример???]

амортизированная сложность $\dot{T}$

- предположения о входных данных не выдвигаются
- усреднение влияния долгих операций среди n операций
- амортизированная сложность:

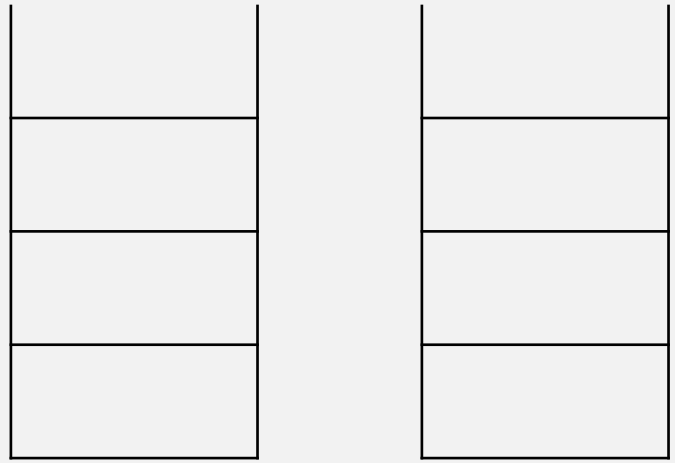
$\dot{c}(op) = t(op) +$ амортизация,
где op — некая операция

$\dot{T}_n = \sum_{i=1}^{n} \dot{c}(op_i)$ — суммарная сложность выполнения n операций

[пример далее...]

Новое

Метод банкира для оценки сложности



операция op	**амортизация**		$\dot{c}(op)$
	отложено	потрачено	

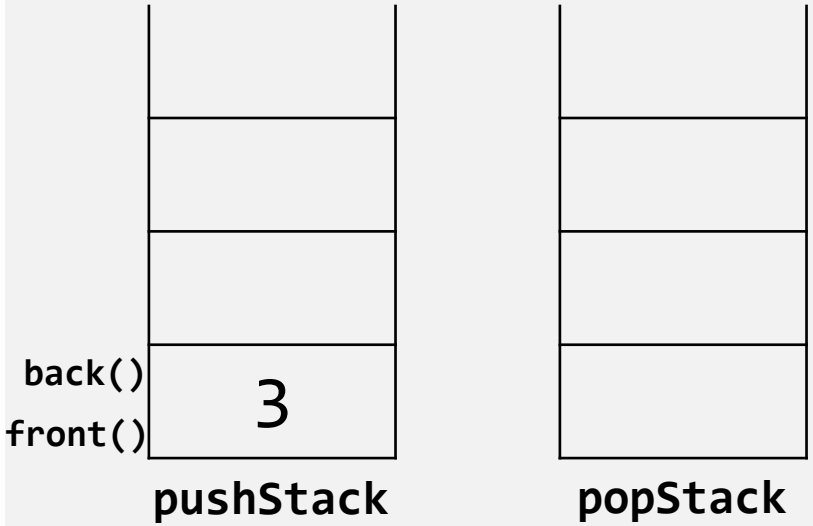
сложность стандартных операций со стеком:

$$t(\text{push}) = t(\text{pop}) = 1$$

$\dot{c}(op) = t(op) +$ амортизация

Новое

Метод банкира для оценки сложности



сложность стандартных операций со стеком:

$$t(\text{push}) = t(\text{pop}) = 1$$

$\dot{c}(op) = t(op) +$ амортизация

операция op	амортизация		$\dot{c}(op)$
	отложено	потрачено	
`push(3)`	1 1	—	$1 + 2 - 0 = 3$

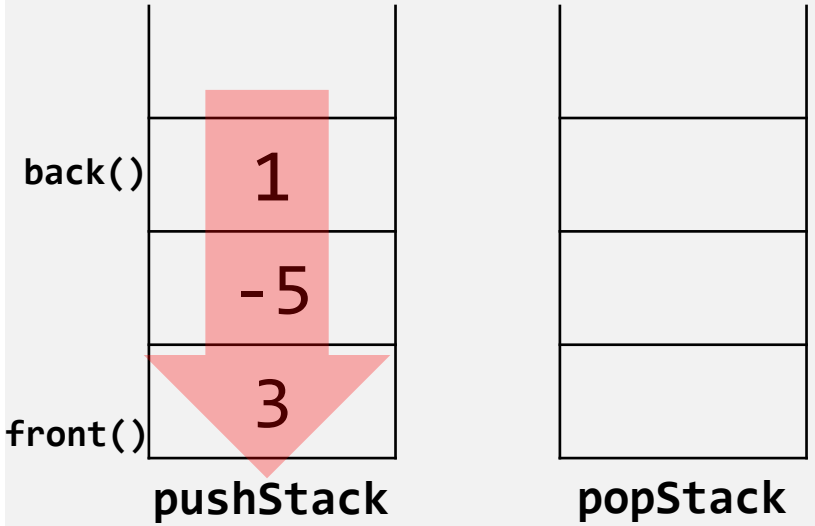
при выполнении стандартных операций откладываем ресурсы [амортизируем] для выполнения долгих операций потом...

- амортизируем операцию `push(T elem)`
  откладываем два ресурса 1 1

амортизация — разница между отложенными и потраченными ресурсами

Новое

Метод банкира для оценки сложности



операция op	амортизация		$\dot{c}(op)$
	отложено	потрачено	
push(3)	1 1	—	$1 + 2 - 0 = 3$
push(-5)	1 1	—	$1 + 2 - 0 = 3$
push(1)	1 1	—	$1 + 2 - 0 = 3$

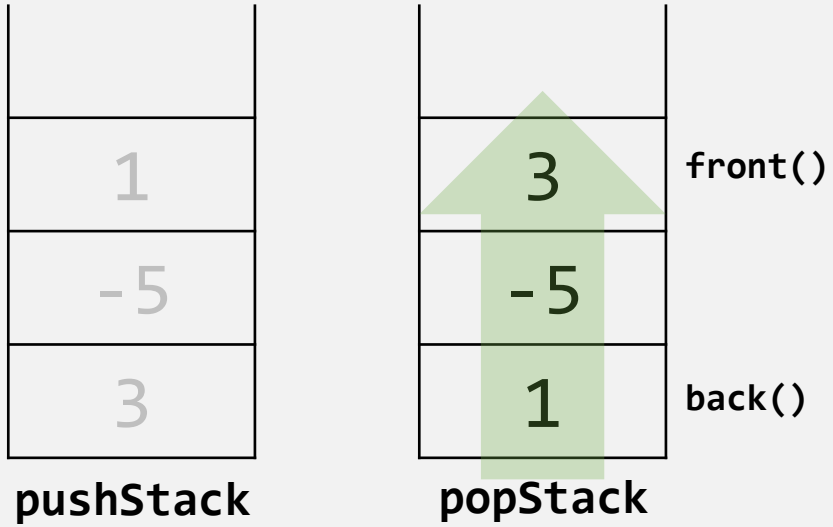
сложность стандартных операций со стеком:

$$t(\text{push}) = t(\text{pop}) = 1$$

$\dot{c}(op) = t(op) +$ амортизация

Новое

Метод банкира для оценки сложности



сложность стандартных операций со стеком:

$$t(\text{push}) = t(\text{pop}) = 1$$

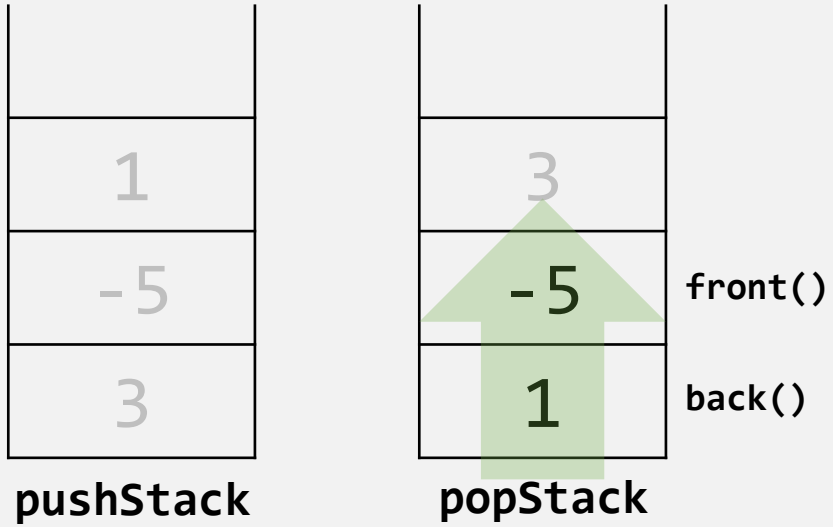
$\dot{c}(op) = t(op) +$ амортизация

операция op		амортизация: отложено	амортизация: потрачено	$\dot{c}(op)$
push(3)		1 1	—	$1 + 2 - 0 = 3$
push(-5)		1 1	—	$1 + 2 - 0 = 3$
push(1)		1 1	—	$1 + 2 - 0 = 3$
pop()	pushStack.pop()	—	1	
	popStack.push(1)	—	1	
	pushStack.pop()	—	1	
	popStack.push(-5)	—	1	
	pushStack.pop()	—	1	
	popStack.push(3)	—	1	

потратили отложенные ресурсы

Новое

Метод банкира для оценки сложности



сложность стандартных операций со стеком:

$$t(\text{push}) = t(\text{pop}) = 1$$

$\dot{c}(op) = t(op) +$ амортизация

операция op		амортизация: отложено	амортизация: потрачено	$\dot{c}(op)$
push(3)		1 1	—	$1 + 2 - 0 = 3$
push(-5)		1 1	—	$1 + 2 - 0 = 3$
push(1)		1 1	—	$1 + 2 - 0 = 3$
pop()	pushStack.pop()	—	1	$7 + 0 - 6 = 1$
	popStack.push(1)	—	1	
	pushStack.pop()	—	1	
	popStack.push(-5)	—	1	
	pushStack.pop()	—	1	
	popStack.push(3)	—	1	
	popStack.pop()	—	—	

Задача 3




Сложность ADT Очередь на двух стеках

		фактическая реализация	сложность	комментарий
Queue.push()		pushStack.push()	$\Theta(1)$	простое добавление объекта в стек
Queue.pop()	**!popStack.empty()**	popStack.pop()	$\Theta(1)$	простое удаление объекта из стека
	popStack.empty()	popStack.push(…) popStack.push(…) … popStack.push(…) popStack.pop()	$\Theta(S) + \Theta(1)$ S — размер стека для вставки	○ перекладывание из pushStack в popStack ○ удаление из стека

для того, чтобы вычислить обобщенную оценку операции **Queue.pop()**, обратимся к способам **усреднения** временной сложности

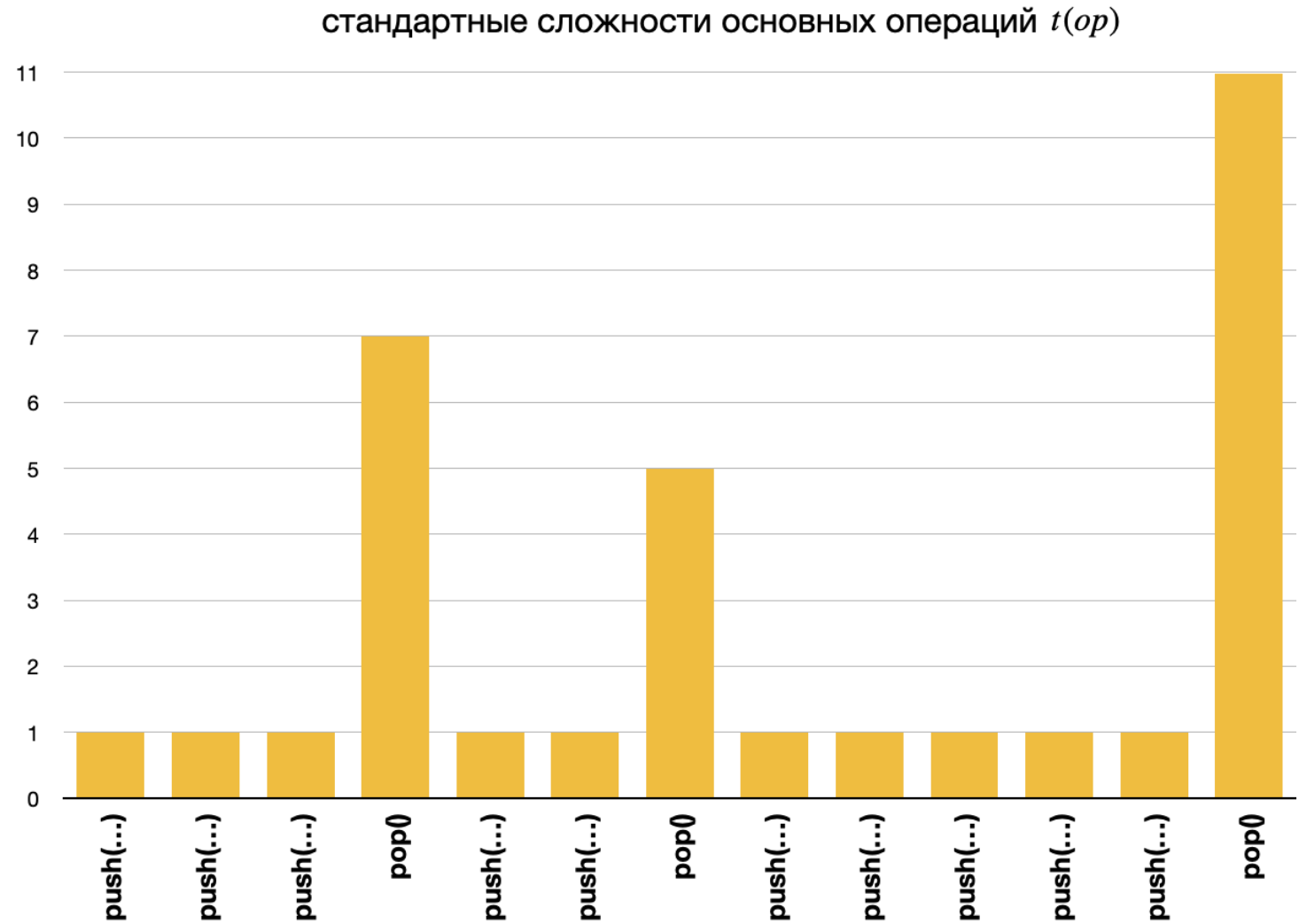
Задача 3

Сложность ADT Очередь на двух стеках

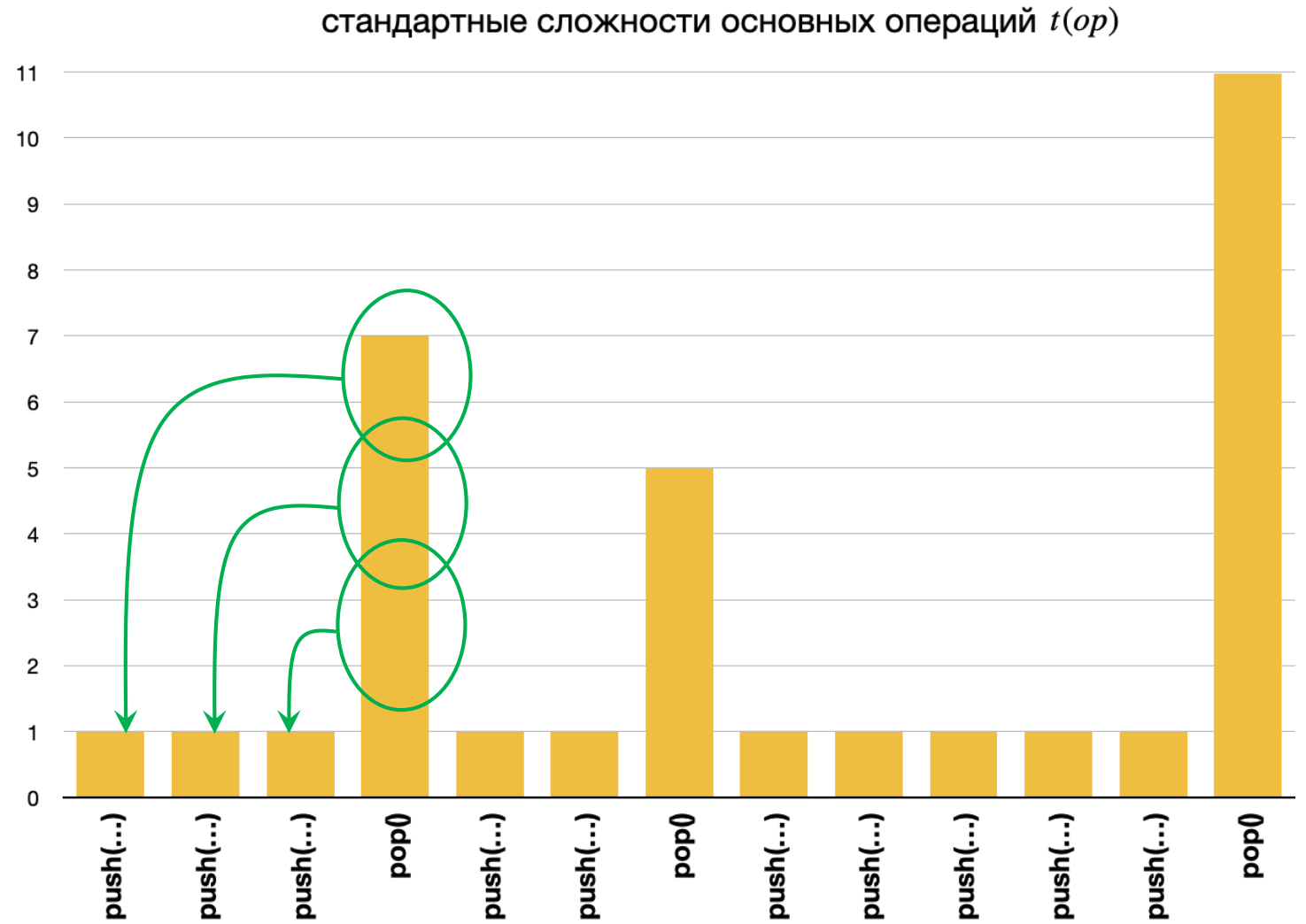
		фактическая реализация	сложность	комментарий
Queue.push()		pushStack.push()	$\Theta(1)$	простое добавление объекта в стек
Queue.pop()	**!popStack.empty()**	popStack.pop()	$\Theta(1)$	простое удаление объекта из стека
	popStack.empty()	popStack.push(…) popStack.push(…) … popStack.push(…) popStack.pop()	$\Theta(1)$ амортизированная	○ перекладывание из pushStack в popStack ○ удаление из стека

кстати говоря, каждый элемент переместится из pushStack в popStack не более одного раза

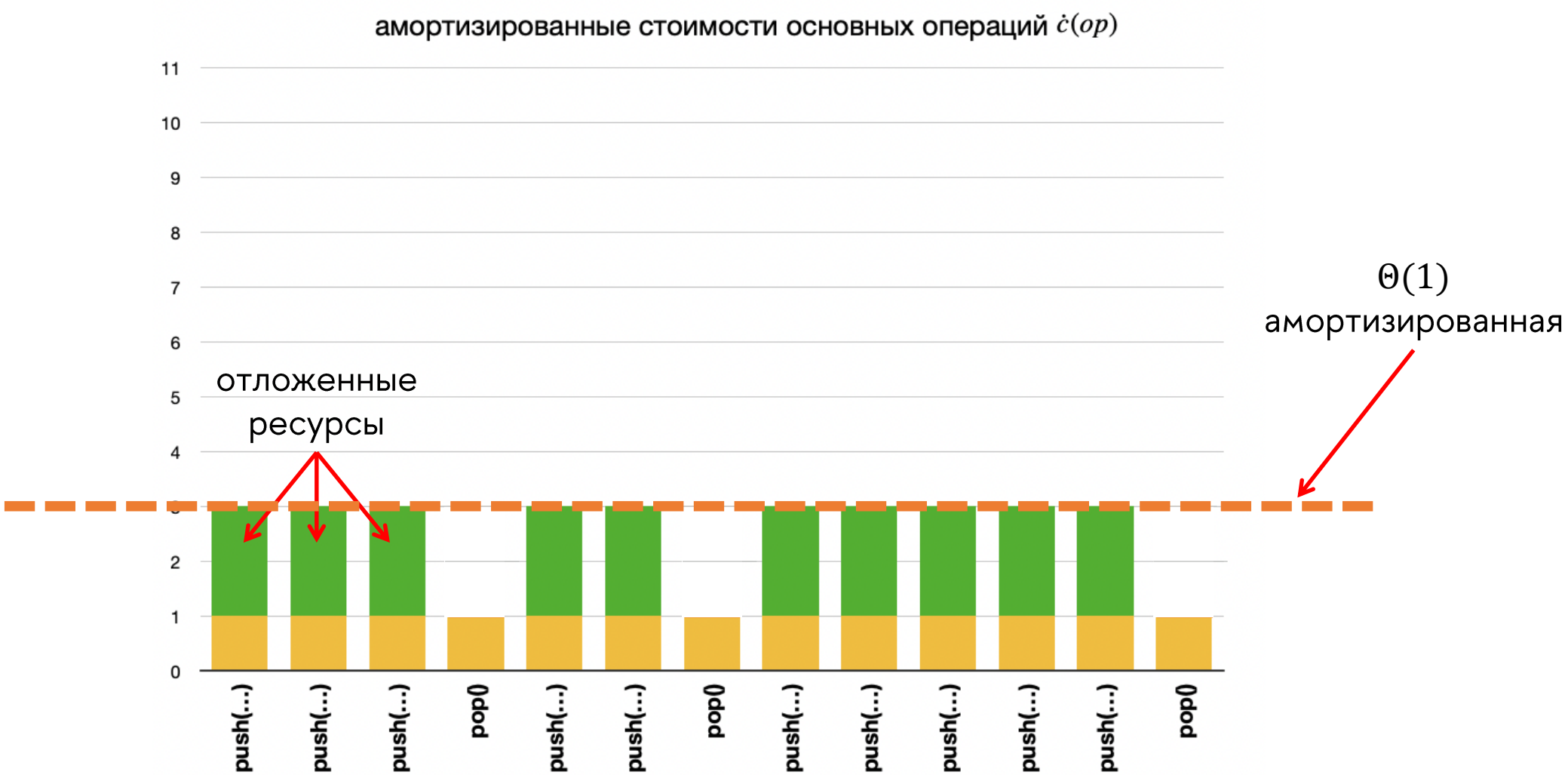
динамику сложности выполнения последовательности операций вставки и удаления с очередью на двух стеках можно также условно представить в виде гистограммы



при амортизированном подсчете, мы интуитивно
перераспределяем сложность долгих операций
на сложность предшествующих быстрых операций



при амортизированном подсчете, мы интуитивно
перераспределяем сложность долгих операций
на сложность предшествующих быстрых операций



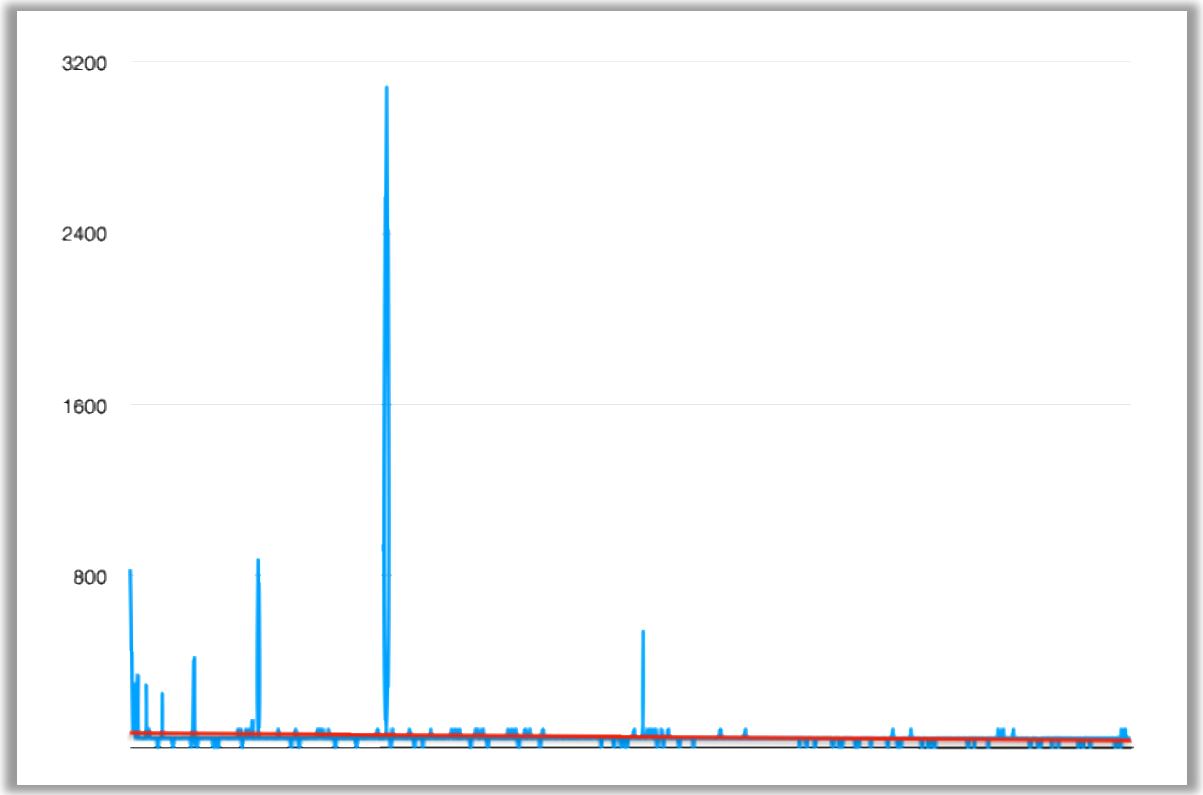
FAQ Амортизированная сложность

В каких случаях мы также можем столкнуться с ситуацией, когда перед выполнением затратных операций происходит множество быстрых?

FAQ Амортизированная сложность

В каких случаях мы также можем столкнуться с ситуацией, когда перед выполнением затратных операций происходит множество быстрых?

Массив, хеш-таблица [увеличение размера]
Сбалансированное дерево поиска [высота]
Система непересекающихся множеств [принадлежность]
stay tuned...



Реальное время (наносекунды) выполнения `push_back` в конец `std::vector`

- вставка 1000 элементов
- $\Theta(1)$ амортизированная

[Q] Что является причиной скачков времени выполнения?

ADT Стек/Очередь

ассоциативные операции

Задача 4





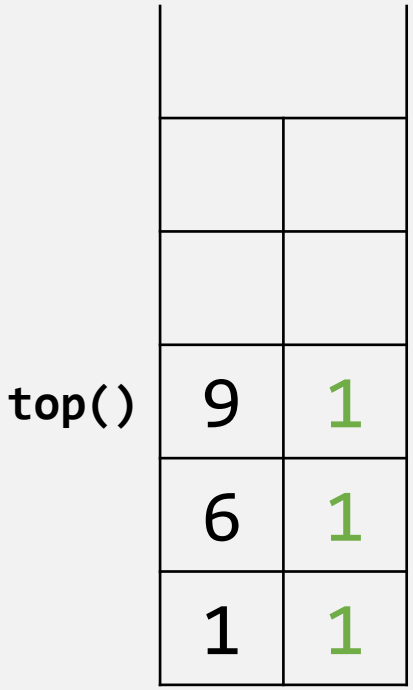
Стек с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- реализация стека с поддержкой хранения промежуточных минимумов для заданной последовательности целых чисел

Задача 4

Стек с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- реализация стека с поддержкой хранения промежуточных минимумов для заданной последовательности целых чисел



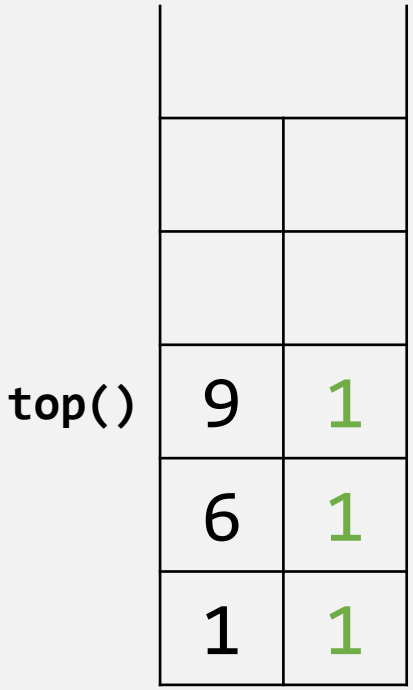
1	6	9	-8	3

```
stack.push(1, min(1, stack.top().second())
stack.push(6, min(6, stack.top().second())
stack.push(9, min(9, stack.top().second())
```

Задача 4

Стек с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- реализация стека с поддержкой хранения промежуточных минимумов для заданной последовательности целых чисел



1	6	9	-8	3

```
stack.push(1, min(1, stack.top().second())
stack.push(6, min(6, stack.top().second())
stack.push(9, min(9, stack.top().second())
```

на вершине стека хранится минимум среди всех введенных на данных момент значений исходной последовательности — $\Theta(1)$

Задача 5





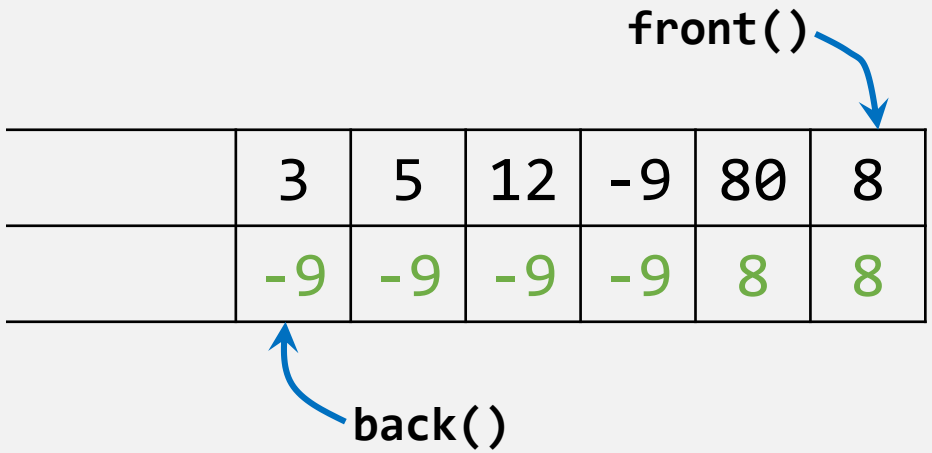
Очередь с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- стандартная реализации очереди не обладает такими же быстрыми возможностями вычисления `min`, как стек... ПОЧЕМУ?

Задача 5

Очередь с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- стандартная реализации очереди не обладает такими же быстрыми возможностями вычисления `min`, как стек... ПОЧЕМУ?

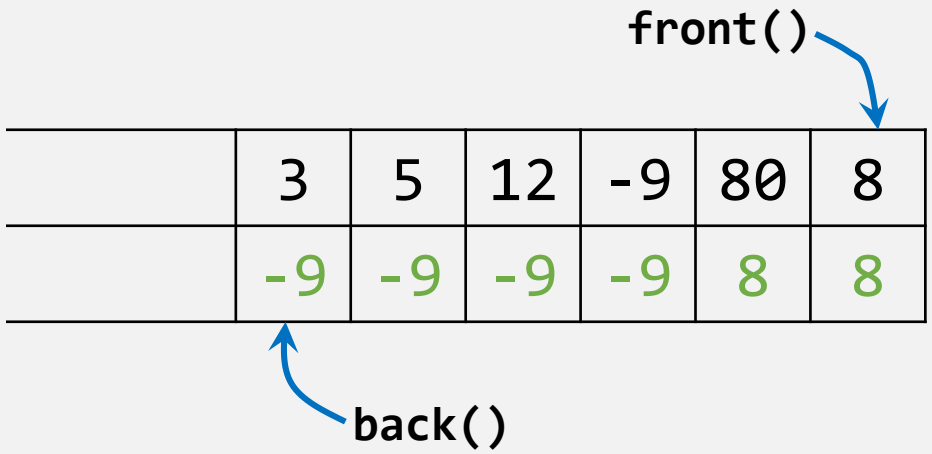


1. В голове очереди не будет находиться минимум из всех введенных на данный момент элементов
2. Операцию `pop()` использовать не получится — сложность вычисления минимума станет линейной $\Theta(k)$
3. Можно воспользоваться deque, но это означает возврат к применению стека...

Задача 5

Очередь с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- стандартная реализации очереди не обладает такими же быстрыми возможностями вычисления `min`, как стек... ПОЧЕМУ?



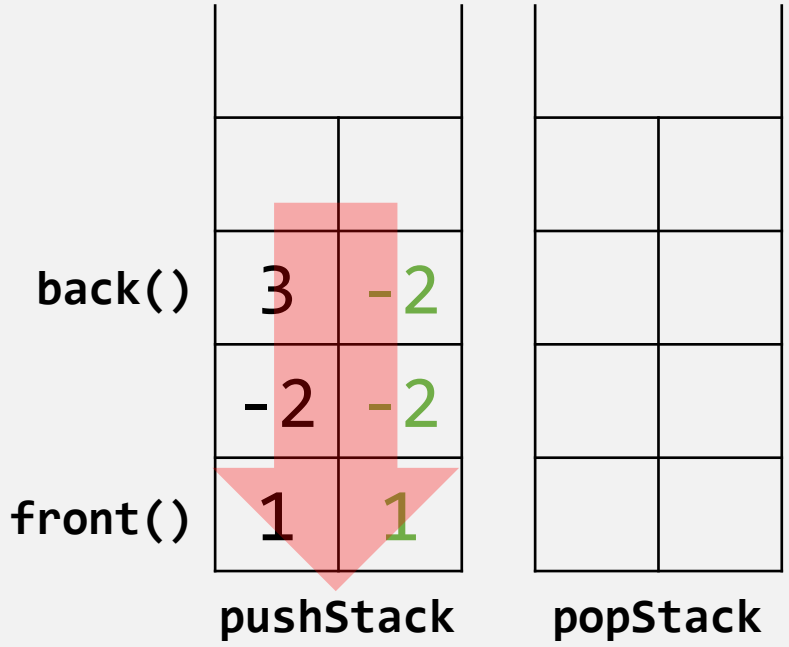
1. В голове очереди не будет находиться минимум из всех введенных на данный момент элементов
2. Операцию `pop()` использовать не получится — сложность вычисления минимума станет линейной $\Theta(k)$
3. Можно воспользоваться deque, но это означает возврат к применению стека...

- это не работает, поскольку `min` не является обратимой операцией

Задача 5

Очередь с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- стандартная реализации очереди не обладает такими же быстрыми возможностями вычисления `min`, как стек...



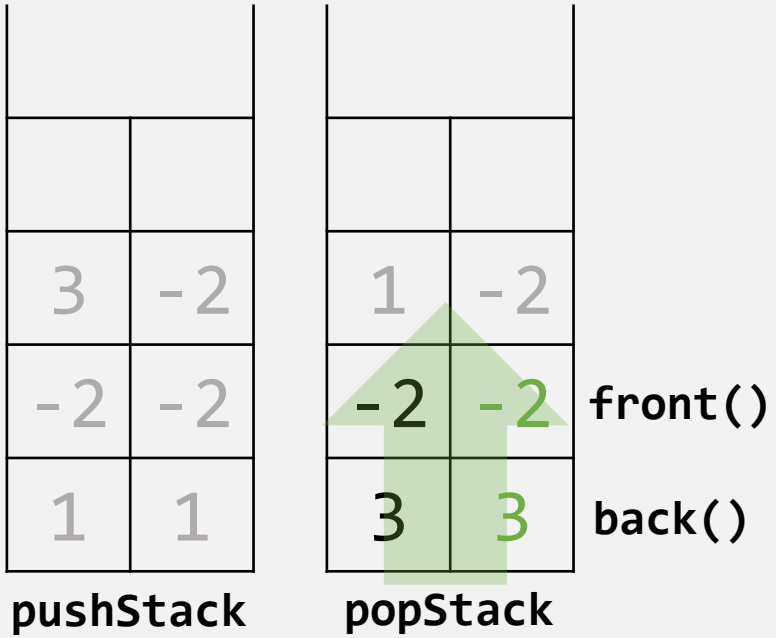
1	-2	3	5	0	-4

```
push(1)
push(-2)
push(3)
```

Задача 5

Очередь с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- стандартная реализации очереди не обладает такими же быстрыми возможностями вычисления `min`, как стек...



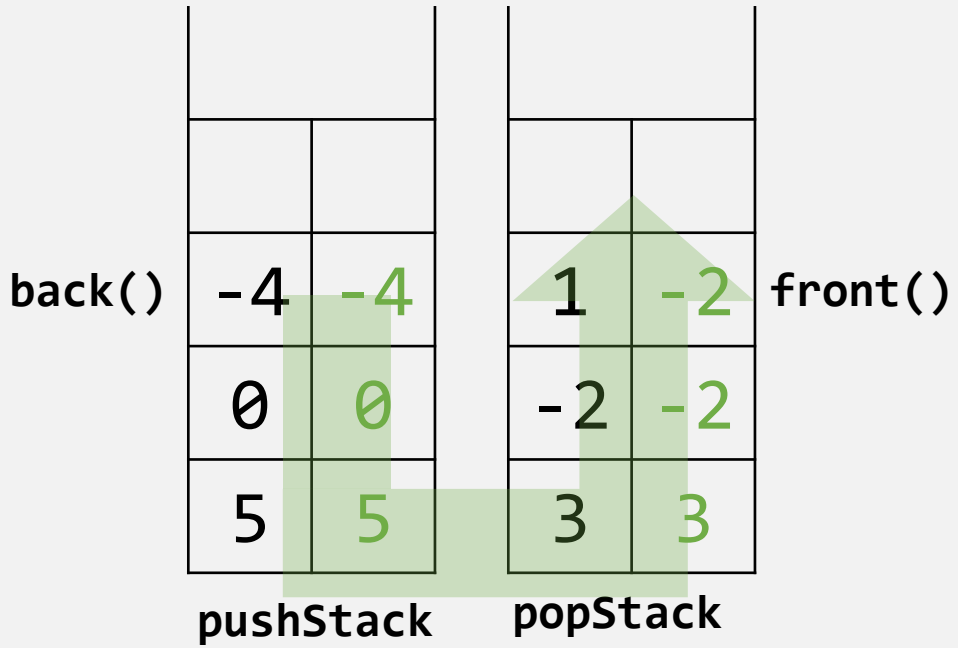
1	-2	3	5	0	-4

```
push(1)
push(-2)
push(3)
pop()
min = popStack.top().second()
```

Задача 5

Очередь с поддержкой минимумов

- ассоциативная операция: `min(a,min(b,c) = min(min(a,b),c)`
- стандартная реализации очереди не обладает такими же быстрыми возможностями вычисления `min`, как стек...



1	-2	3	5	0	-4

```
push(5)
push(0)
push(-4)
min = min(pushStack.top().second(),
          popStack.top().second())
```

Резюме

1. Различные подходы к реализации **линейных контейнеров** [ADT] на примере стека и очереди
   - правильные скобочные последовательности
   - поддержка ассоциативных операций
2. Амортизационный анализ сложности: **много быстрых** операций и **мало долгих**