

Мини-задача #18 (2 балла)

Реализовать алгоритм сортировочной станции для преобразования выражения в инфиксной нотации к обратной польской записи.

Приоритеты и ассоциативность операторов взять отсюда:

https://en.cppreference.com/w/c/language/operator_precedence

В алгоритме поддержать:

1. Базовые арифметические, битовые и логические операторы
2. Подвыражения в скобках

Мини-задача #19 (2 балла)

Пусть есть k отсортированных связанных списков.

Написать программу, строящую один отсортированный связный список из всех элементов.

<https://leetcode.com/problems/merge-k-sorted-lists/>

Для решения используйте очередь с приоритетами.

Альтернативные решения приветствуются, но не отменяют необходимость продемонстрировать решение через priority queue.

Мини-задача #20 (1 балл)

Реализовать стек, который бы поддерживал одну дополнительную операцию:

`get_min()` -> T: возвращение минимального элемента в стеке
(без изменения стека)

Новая операция (как и все старые) должна работать за $O(1)$

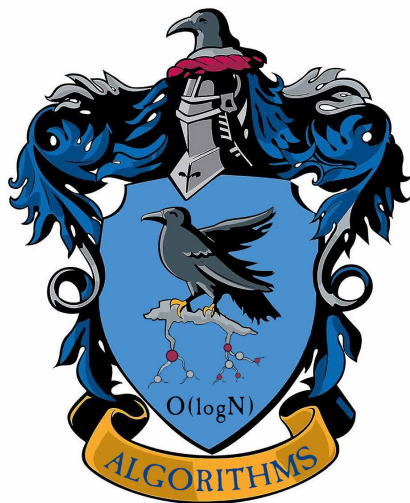
<https://leetcode.com/problems/min-stack/>

Алгоритмы и структуры данных



Алгоритмы и структуры данных

Стеки, очереди, пирамиды



Задача

Реализовать **хранилище** однотипных упорядоченных данных (например, чисел), которое бы поддерживало следующие **запросы**:

1. Доступ к i -ому элементу
2. Добавление элемента в "конец"
3. Удаление последнего элемента

Здесь задан **контракт**: множество значений
и операций с их семантикой

Задача

Реализовать **хранилище** однотипных упорядоченных данных (например, чисел), которое бы поддерживало следующие **запросы**:

1. Доступ к i -ому элементу
2. Добавление элемента в "конец"
3. Удаление последнего элемента

Здесь задан контракт: множество значений
и операций с их семантикой

Задача

Реализовать хранилище **однотипных упорядоченных данных (например, чисел)**, которое бы поддерживало следующие запросы:

1. Доступ к i -ому элементу
2. Добавление элемента в "конец"
3. Удаление последнего элемента

Здесь задан контракт: множество значений
и операций с их семантикой

Задача

Реализовать хранилище однотипных упорядоченных данных (например, чисел), которое бы поддерживало следующие запросы:

1. Доступ к i -ому элементу
2. Добавление элемента в "конец"
3. Удаление последнего элемента

Здесь задан **контракт**: множество значений
и операций с их семантикой

Задача

Реализовать хранилище однотипных упорядоченных данных (например, чисел), которое бы поддерживало следующие запросы:

1. Доступ к i -ому элементу
2. Добавление элемента в "конец"
3. Удаление последнего элемента

Про **реализацию** здесь не уточняется,
только описание, как этим пользоваться

Здесь задан **контракт**: множество значений
и операций с их семантикой

Задача

Реализовать хранилище однотипных упорядоченных данных (например, чисел), которое бы поддерживало следующие запросы:

1. Доступ к i -ому элементу
2. Добавление элемента в "конец"
3. Удаление последнего элемента

Про **реализацию** здесь не уточняется,
только описание, как этим пользоваться

Так что реализация может быть разная:
динамический массив, список и т.д.

Здесь задан **контракт**: множество значений
и операций с их семантикой

Задача

Реализовать хранилище однотипных упорядоченных данных (например, чисел), которое бы поддерживало следующие запросы:

1. Доступ к i -ому элементу
2. Добавление элемента в "конец"
3. Удаление последнего элемента

Про **реализацию** здесь не уточняется,
только описание, как этим пользоваться

Такое описание будем называть
абстрактным типом данных.

Так что реализация может быть разная:
динамический массив, список и т.д.

Решение

Храним элементы в виде связного списка

1. Доступ к i -ому элементу - линейный проход по списку за $O(N)$
2. Добавление элемента в "конец" - вставка после хвостового элемента за $O(1)$
3. Удаление последнего элемента - удаление хвостового элемента за $O(1)$

Решение

Храним элементы в виде связного списка

1. Доступ к i -ому элементу - линейный проход по списку за $O(N)$
2. Добавление элемента в "конец" - вставка после хвостового элемента за $O(1)$
3. Удаление последнего элемента - удаление хвостового элемента за $O(1)$

Решение

Храним элементы в виде связного списка

1. Доступ к i -ому элементу - линейный проход по списку за $O(N)$
2. Добавление элемента в "конец" - вставка после хвостового элемента за $O(1)$
3. Удаление последнего элемента - удаление хвостового элемента за $O(1)$

Будем называть хранилище элементов, с описанным способом их хранения и реализацией операций – **структурой данных**

Здесь уже полноценная **реализация**, которая удовлетворяет **контракту**

Храним элементы в виде связного списка

1. Доступ к i -ому элементу – линейный проход по списку за $O(N)$
2. Добавление элемента в "конец" – вставка после хвостового элемента за $O(1)$
3. Удаление последнего элемента – удаление хвостового элемента за $O(1)$

Абстрактный тип данных VS структура данных

Абстрактный тип данных:

1. Множество значений
2. Семантика операций

Абстрактный тип данных VS структура данных

Абстрактный тип данных:

1. Множество значений
2. Семантика операций

Структура данных:

1. Способ хранения значений
2. Реализация операций

Абстрактный тип данных VS структура данных

Абстрактный тип данных:

1. Множество значений
2. Семантика операций

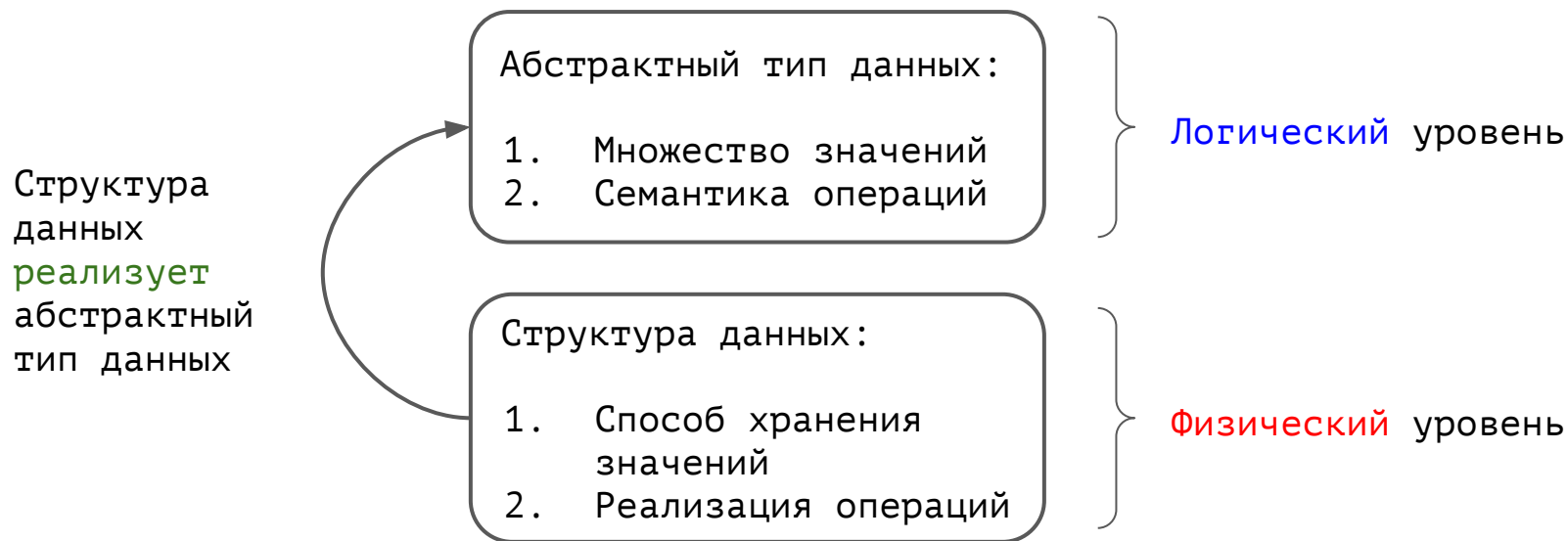
} Логический уровень

Структура данных:

1. Способ хранения значений
2. Реализация операций

} Физический уровень

Абстрактный тип данных VS структура данных



Абстрактный тип данных: стек



Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Абстрактный тип данных: стек

Множество значений: элементы заданного типа T
(можно и разных типов, не принципиально)

Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)` – добавление нового элемента

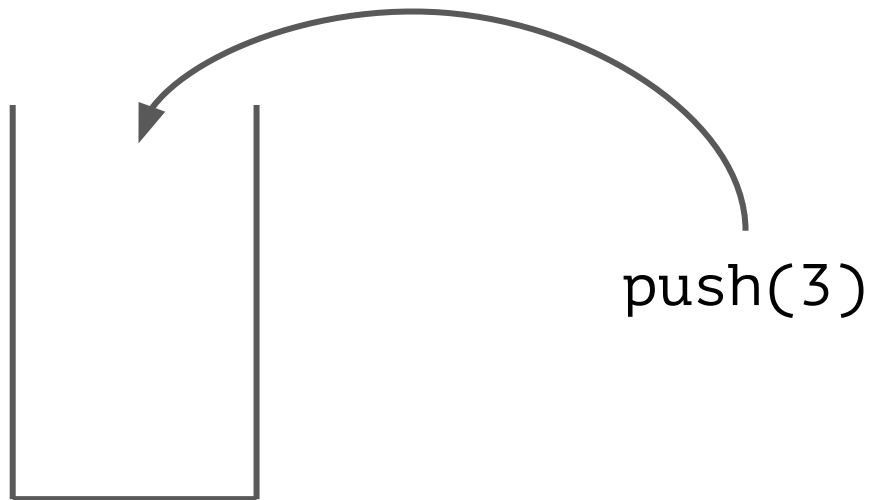
Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

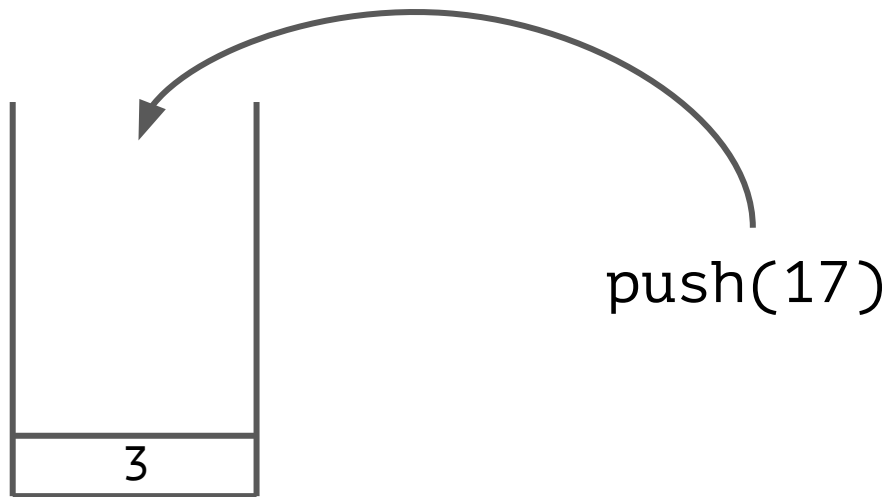
Операции:

1. `push(value: T)` – добавление нового элемента
2. `pop() -> T` – удаление **последнего** добавленного элемента и возвращение его значения

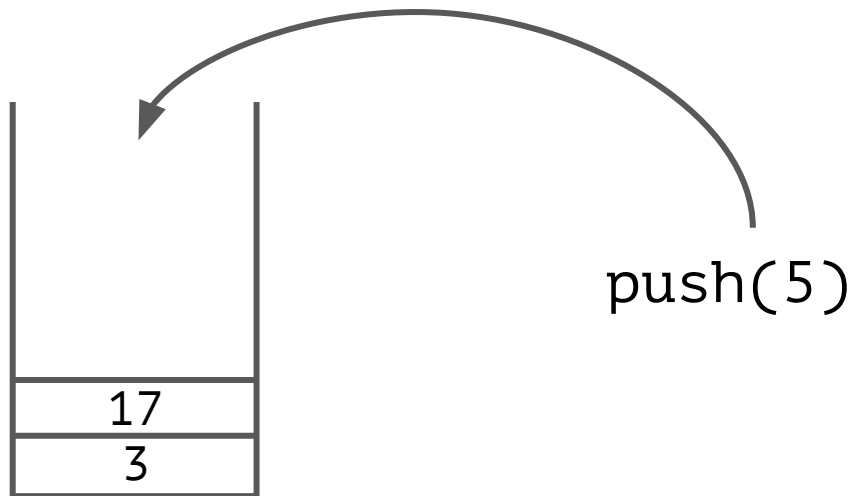
Абстрактный тип данных: стек



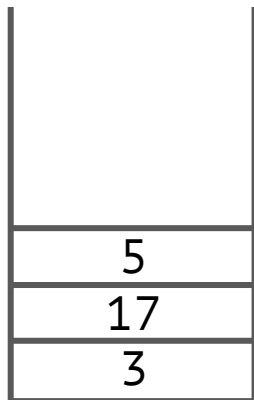
Абстрактный тип данных: стек



Абстрактный тип данных: стек

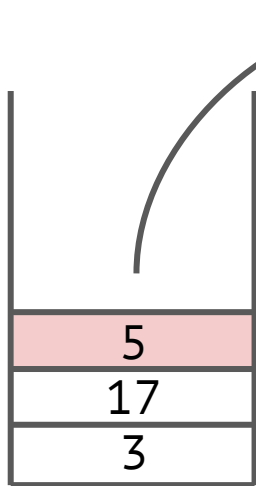


Абстрактный тип данных: стек



Абстрактный тип данных: стек

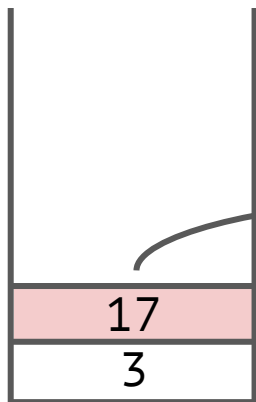
еще называется LIFO -
Last In, First Out



```
a = pop() // a <- 5
```

Абстрактный тип данных: стек

еще называется LIFO -
Last In, First Out



```
a = pop() // a <- 5  
b = pop() // b <- 17
```

Абстрактный тип данных: стек

еще называется LIFO -
Last In, First Out



```
a = pop() // a <- 5  
b = pop() // b <- 17
```


Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)` – добавление нового элемента
2. `pop() -> T` – удаление **последнего** добавленного элемента и возвращение его значения

Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)` – добавление нового элемента
2. `pop() -> T` – удаление **последнего** добавленного элемента и возвращение его значения
3. `peek() -> T` – получение **последнего** добавленного элемента и без его удаления

Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)` – добавление нового элемента
2. `pop() -> T` – удаление **последнего** добавленного элемента и возвращение его значения
3. `peek() -> T` – получение **последнего** добавленного элемента и без его удаления
4. `empty() -> bool` – проверка на пустоту

Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)`
2. `pop() -> T`
3. `peek() -> T`
4. `empty() -> bool`

Как реализовать?

Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)`
2. `pop() -> T`
3. `peek() -> T`
4. `empty() -> bool`

Как реализовать?

Динамические массивы или связанные списки
(см. предыдущую лекцию)

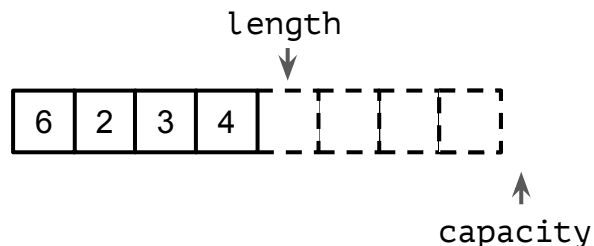


Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)`
2. `pop() -> T`
3. `peek() -> T`
4. `empty() -> bool`



Как реализовать?

Динамические массивы или связанные списки
(см. предыдущую лекцию)

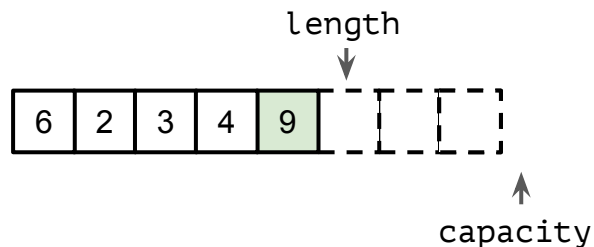


Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. **push**(value: T)
2. **pop**() $\rightarrow T$
3. **peek**() $\rightarrow T$
4. **empty**() $\rightarrow \text{bool}$



Как реализовать?

Динамические массивы или связанные списки
(см. предыдущую лекцию)

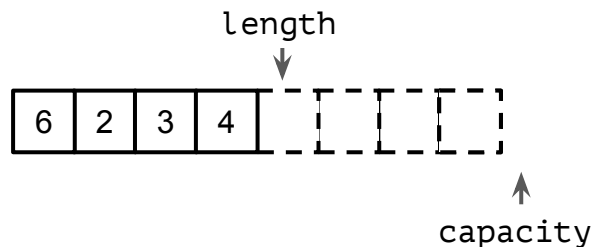


Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)`
2. `pop()` $\rightarrow T$
3. `peek()` $\rightarrow T$
4. `empty()` $\rightarrow \text{bool}$



Как реализовать?

Динамические массивы или связанные списки
(см. предыдущую лекцию)

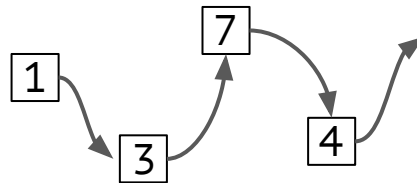


Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)`
2. `pop() -> T`
3. `peek() -> T`
4. `empty() -> bool`



Как реализовать?

Динамические массивы или **связные списки**
(см. предыдущую лекцию)

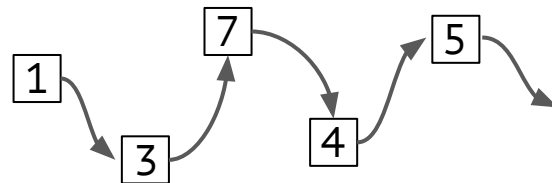


Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. **push**(value: T)
2. **pop**() $\rightarrow T$
3. **peek**() $\rightarrow T$
4. **empty**() $\rightarrow \text{bool}$



Как реализовать?

Динамические массивы или **связные списки**
(см. предыдущую лекцию)

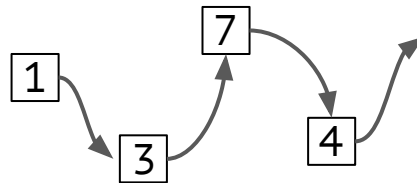


Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)`
2. `pop()` $\rightarrow T$
3. `peek()` $\rightarrow T$
4. `empty()` $\rightarrow \text{bool}$



Как реализовать?

Динамические массивы или **связные списки**
(см. предыдущую лекцию)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

<https://leetcode.com/problems/valid-parentheses/>

Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

[test{312(([str]))asd}eee] ✓

[test(abc]asd) ✗

<https://leetcode.com/problems/valid-parentheses/>

Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

[test(abc]asd)

Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

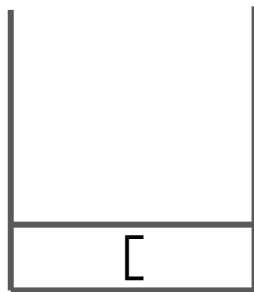
[test(abc]asd)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

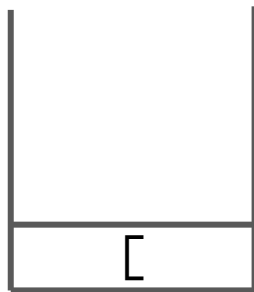
[test(abc]asd)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

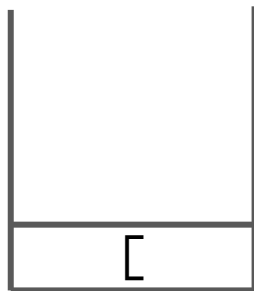
[test(abc]asd)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

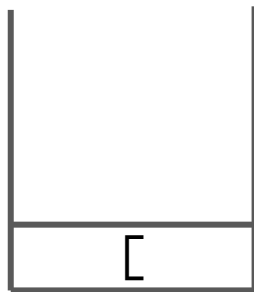
[t**est**(abc]asd)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

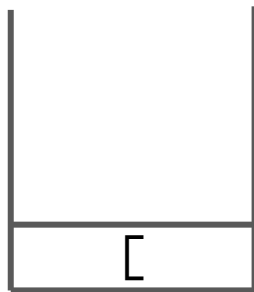
[te**st**(abc]asd)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

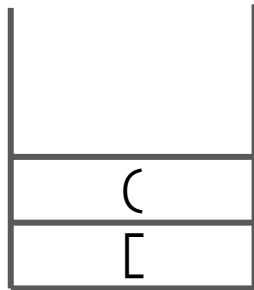
[test^t(abc]asd)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

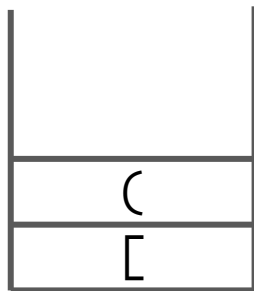
[test(abc]asd)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

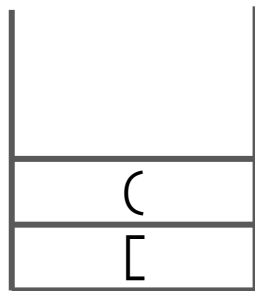
[test(abc]asd)



Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

[test(abc]asd)

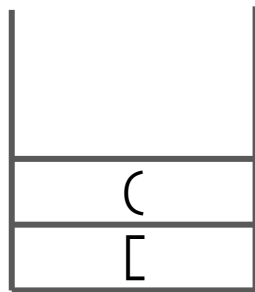


1. v = pop()

Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

[test(abc]asd)

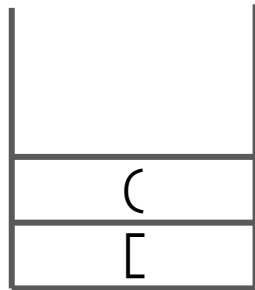


1. `v = pop()`
2. `is_same_type(v, curr)?`
3. `if not => invalid`

Стек: применения

Задача: написать функцию для проверки корректности расстановки скобок трех типов (), [], { } в заданной строке

[test(abc]asd)



1. `v = pop()`
2. `is_same_type(v, curr)?`
3. `if not => invalid`

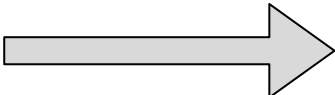
В конце проверяем, что стек **пустой**

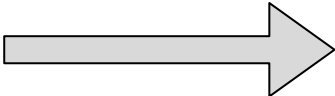
Стек: применения

Обратная польская запись (выражения) —
аргументы перед знаком действия

Стек: применения

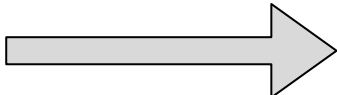
Обратная польская запись (выражения) –
аргументы перед знаком действия

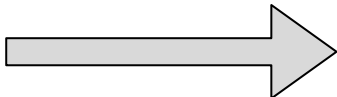
3 + 14  3 14 +

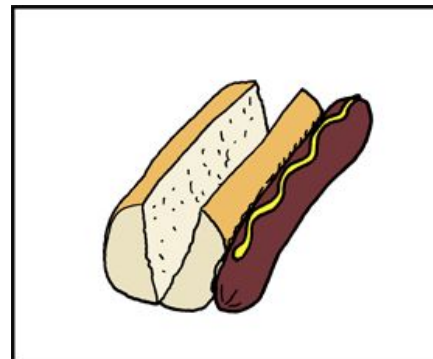
1 - 5*3  1 5 3 * -

Стек: применения

Обратная польская запись (выражения) –
аргументы перед знаком действия

$3 + 14$  $3 \ 14 \ +$

$1 - 5 * 3$  $1 \ 5 \ 3 \ * \ -$



REVERSE POLISH SAUSAGE

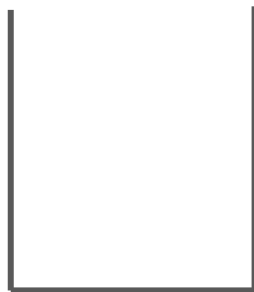
Стек: применения

Задача: вычислить значение выражения в обратной польской записи

Стек: применения

Задача: вычислить значение выражения в обратной польской записи

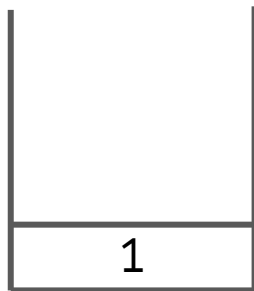
1 5 3 * -



Стек: применения

Задача: вычислить значение выражения в обратной польской записи

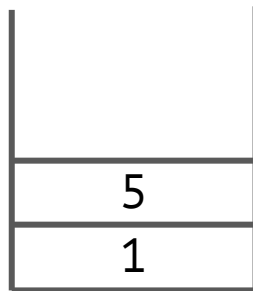
1 5 3 * -



Стек: применения

Задача: вычислить значение выражения в обратной польской записи

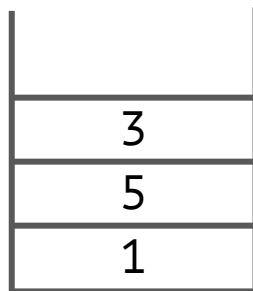
1 5 3 * -



Стек: применения

Задача: вычислить значение выражения в обратной польской записи

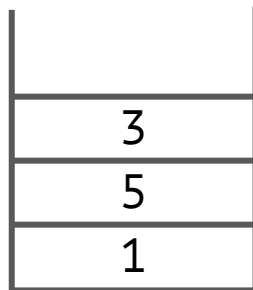
1 5 3 * -



Стек: применения

Задача: вычислить значение выражения в обратной польской записи

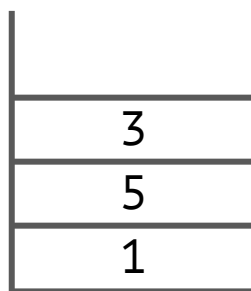
1 5 3 * -



Стек: применения

Задача: вычислить значение выражения в обратной польской записи

1 5 3 * -

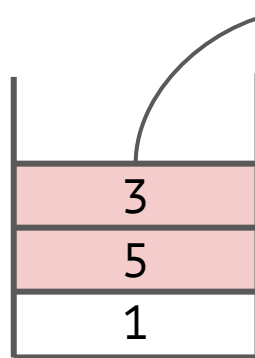


1. `a = pop()`
2. `b = pop()`
3. `push(oper(a, b))`

Стек: применения

Задача: вычислить значение выражения в обратной польской записи

1 5 3 * -

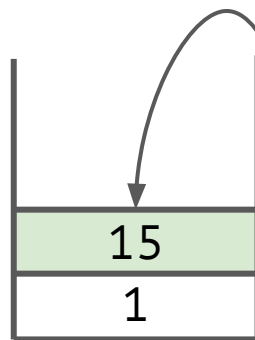


```
1. a = pop()
2. b = pop()
3. push(oper(a, b))
```

Стек: применения

Задача: вычислить значение выражения в обратной польской записи

1 5 3 * -

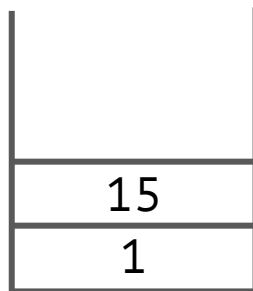


1. `a = pop()`
2. `b = pop()`
3. `push(oper(a, b))`

Стек: применения

Задача: вычислить значение выражения в обратной польской записи

1 5 3 * -

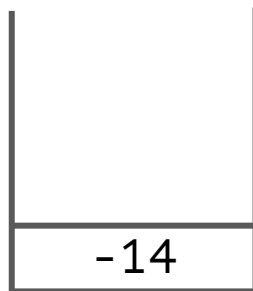


1. `a = pop()`
2. `b = pop()`
3. `push(oper(a, b))`

Стек: применения

Задача: вычислить значение выражения в обратной польской записи

1 5 3 * -

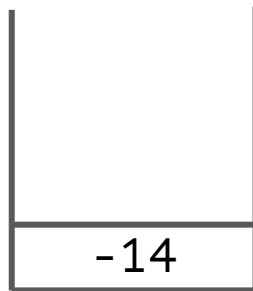


1. `a = pop()`
2. `b = pop()`
3. `push(oper(a, b))`

Стек: применения

Задача: вычислить значение выражения в обратной польской записи

1 5 3 * -



1. `a = pop()`
2. `b = pop()`
3. `push(oper(a, b))`

Ответ - последний
элемент в стеке

Стек: применения

Задача: вычислить значение выражения в обратной польской записи

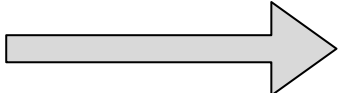
<https://leetcode.com/problems/evaluate-reverse-polish-notation/>

Стек: применения

Задача: вычислить значение выражения в обратной польской записи

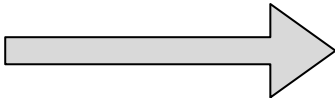
<https://leetcode.com/problems/evaluate-reverse-polish-notation/>

Задача: преобразовать выражение в инфиксной нотации к виду обратной польской записи

1 - 5*3  1 5 3 * -

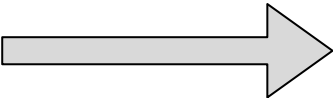
Стек: применения

Задача: преобразовать выражение в инфиксной нотации к виду обратной польской записи

1 - 5*3  1 5 3 * -

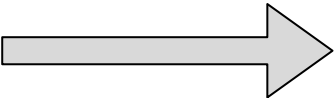
Стек: применения

Задача: преобразовать выражение в инфиксной нотации к виду обратной польской записи

1 - 5 ^ 3 * 2  1 5 3 ^ 2 * -

Стек: применения

Задача: преобразовать выражение в инфиксной нотации к виду обратной польской записи

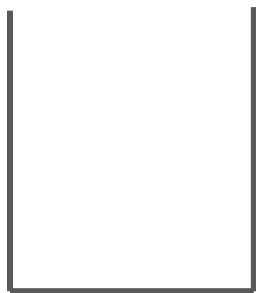
1 - 5 ^ 3 * 2  1 5 3 ^ 2 * -

^ приоритетнее *

* приоритетнее -

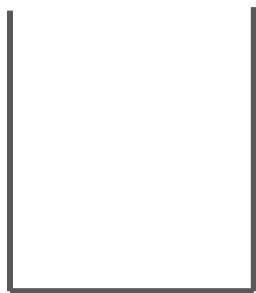
Стек: применения

1 - 5 ^ 3 * 2 



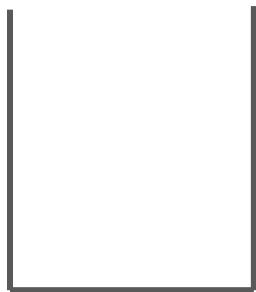
Стек: применения

1 - 5 ^ 3 * 2 



Стек: применения

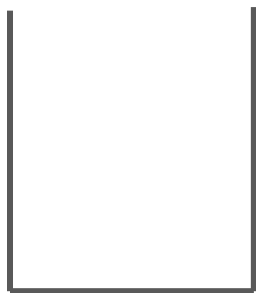
1 - 5 ^ 3 * 2  **1**



Стек: применения

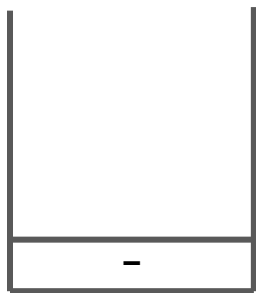
1 - 5 ^ 3 * 2  1



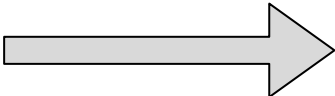



Стек: применения

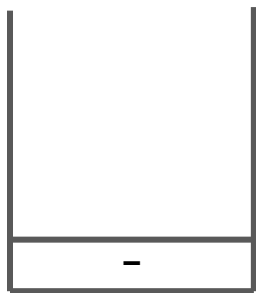
1 - 5 ^ 3 * 2  1



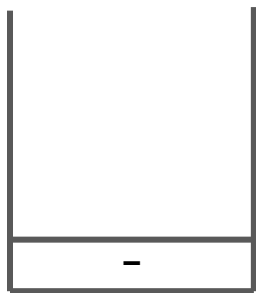
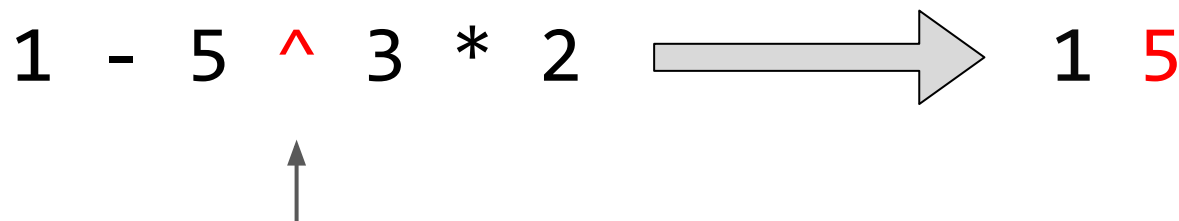
Стек: применения

1 - 5 ^ 3 * 2  1 5

 ↑

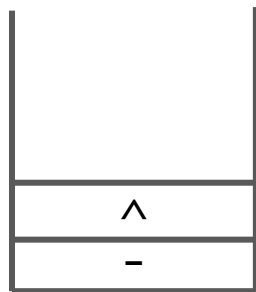
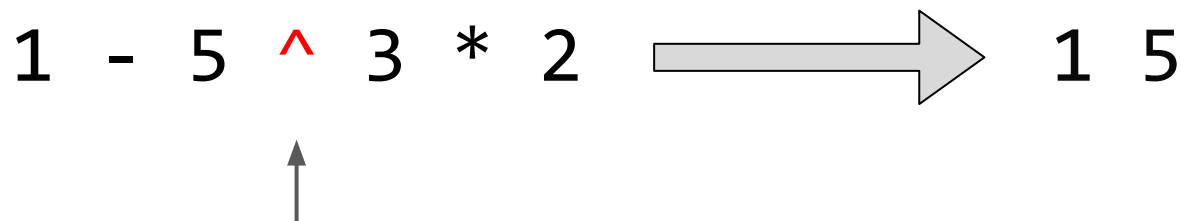


Стек: применения



Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

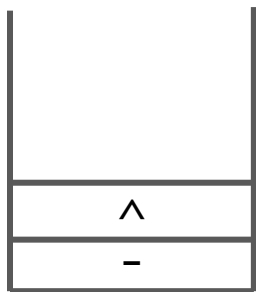
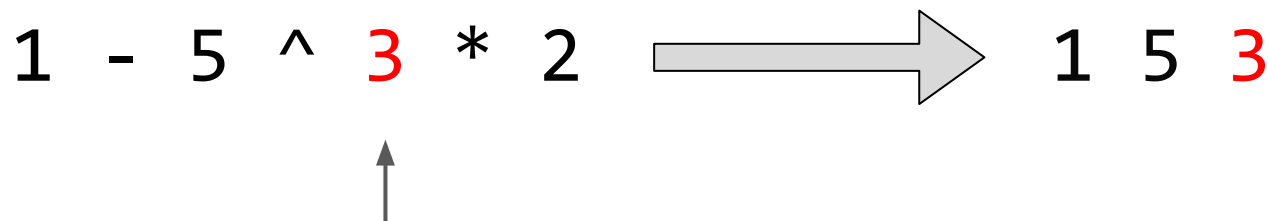
Стек: применения



Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

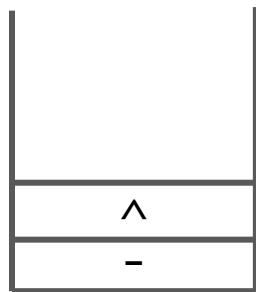
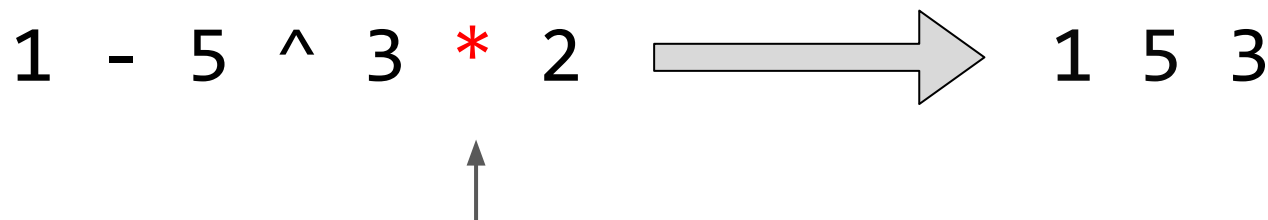
Стек: применения



Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

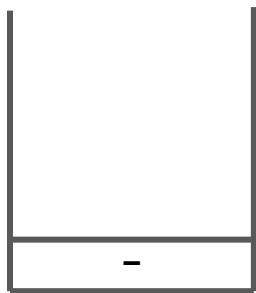
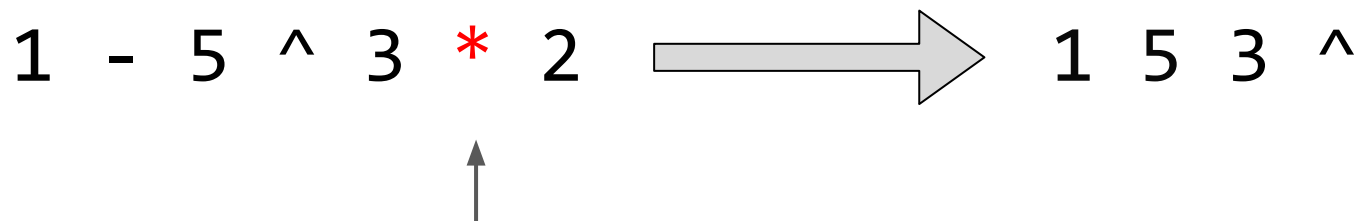
Стек: применения



Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

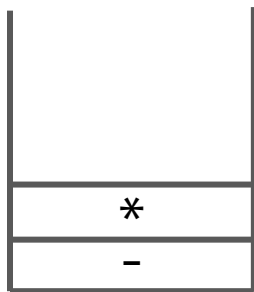
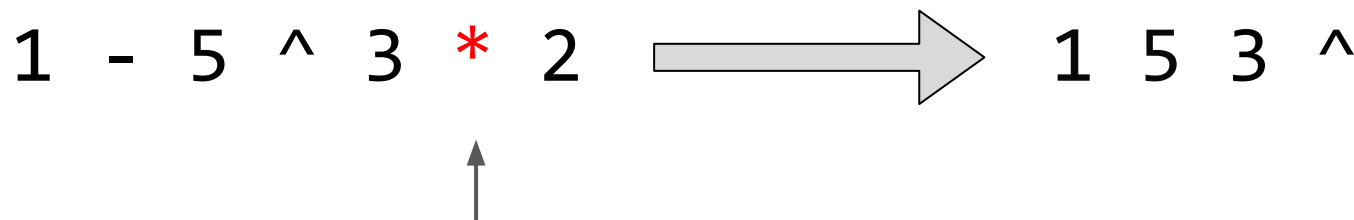
Стек: применения



Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

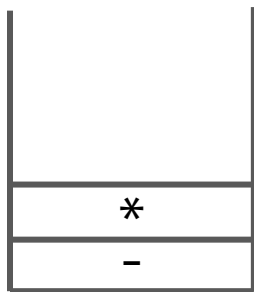
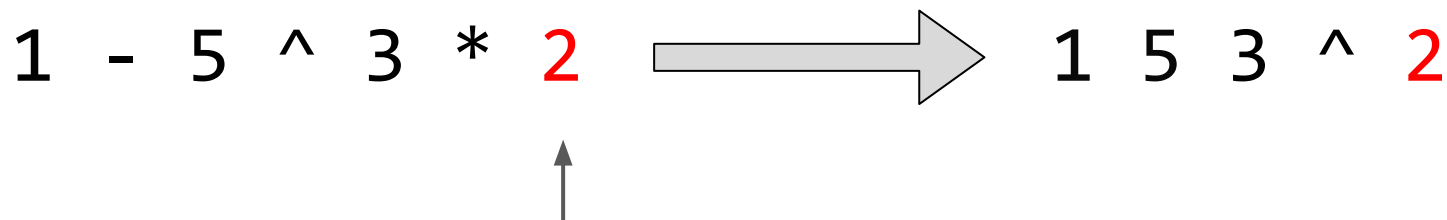
Стек: применения



Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

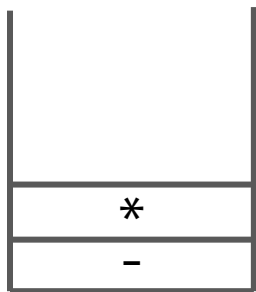
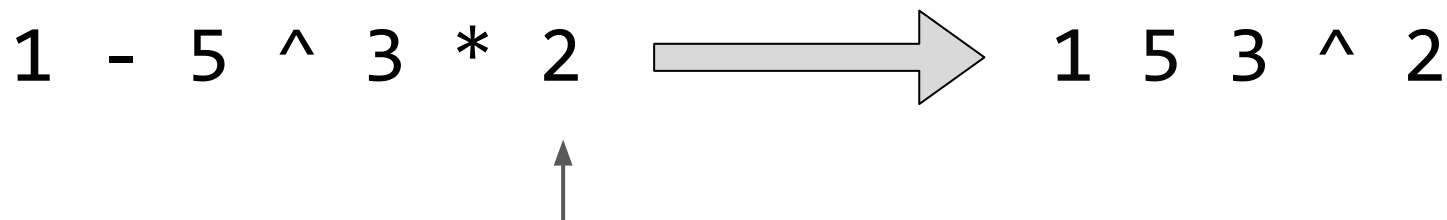
Стек: применения



Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

Стек: применения

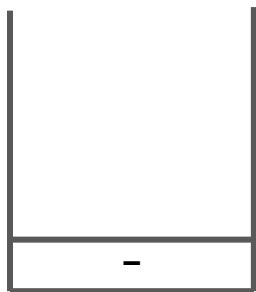


Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

В конце выталкиваем все оставшиеся операции из стека

Стек: применения

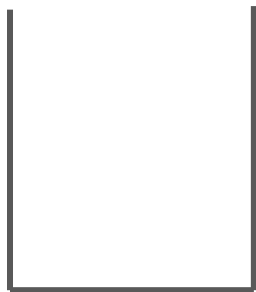
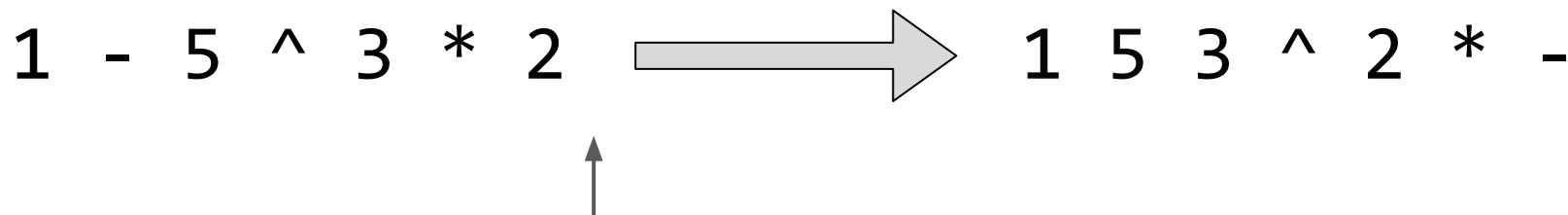


Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

В конце выталкиваем все оставшиеся операции из стека

Стек: применения

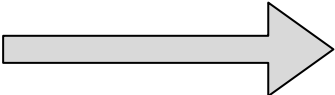


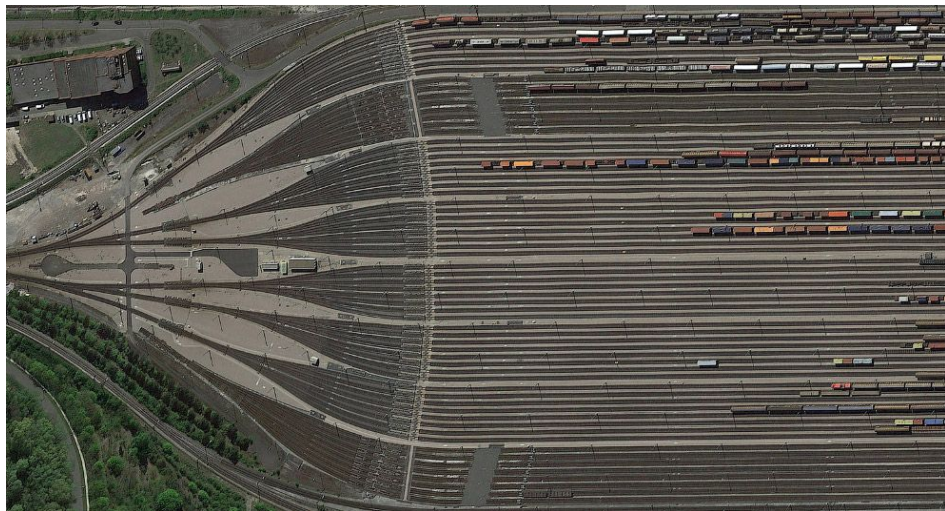
Пока операция на вершине стека **приоритетнее** или же с таким же приоритетом, но при этом текущая операция **левоассоциативная**, выталкиваем значение из стека в результирующую строку

После этого самую текущую операцию добавляем в стек

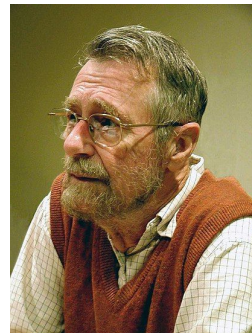
В конце выталкиваем все оставшиеся операции из стека

Стек: применения

1 - 5 ^ 3 * 2  1 5 3 ^ 2 * -



Алгоритм сортировочной станции
Автор: Эдсгер Дейкстра



Мини-задача #18 (2 балла)

Реализовать алгоритм сортировочной станции для преобразования выражения в инфиксной нотации к обратной польской записи.

Приоритеты и ассоциативность операторов взять отсюда:

https://en.cppreference.com/w/c/language/operator_precedence

В алгоритме поддержать:

1. Базовые арифметические, битовые и логические операторы
2. Подвыражения в скобках

Абстрактный тип данных: очередь



Абстрактный тип данных: очередь

Множество значений: элементы заданного типа T

Операции:

1. `enqueue(value: T)` – добавление нового элемента
2. `dequeue() -> T` – удаление **первого** добавленного элемента и возвращение его значения

Абстрактный тип данных: очередь

Множество значений: элементы заданного типа T

Операции:

1. `enqueue(value: T)` – добавление нового элемента
2. `dequeue() -> T` – удаление **первого** добавленного элемента и возвращение его значения

еще называется FIFO –
First In, First Out

Абстрактный тип данных: очередь

Множество значений: элементы заданного типа T

Операции:

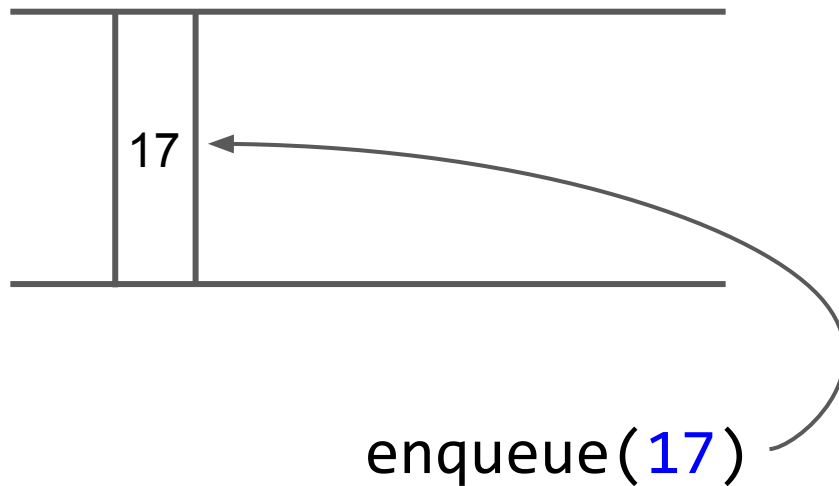
1. `enqueue(value: T)` – добавление нового элемента
2. `dequeue() -> T` – удаление **первого** добавленного элемента и возвращение его значения
3. `empty() -> bool`
4. ...

Абстрактный тип данных: очередь

First In, First Out

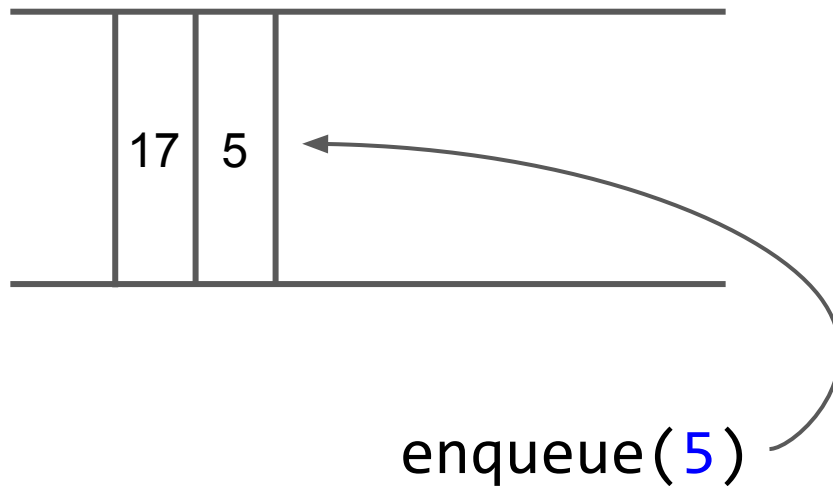
Абстрактный тип данных: очередь

First In, First Out



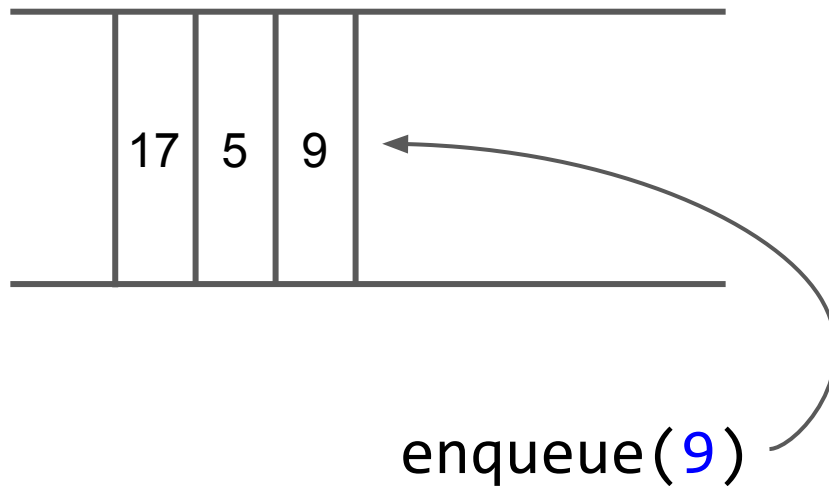
Абстрактный тип данных: очередь

First In, First Out



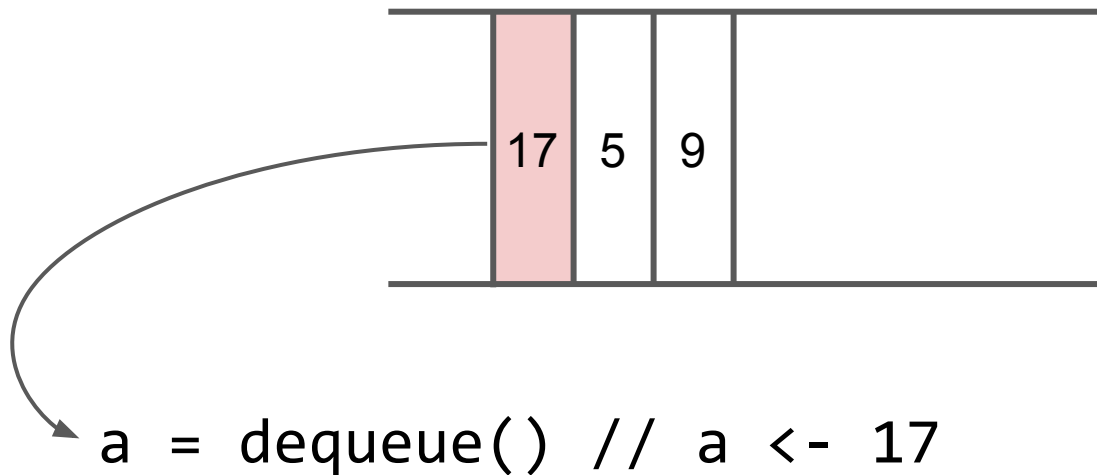
Абстрактный тип данных: очередь

First In, First Out



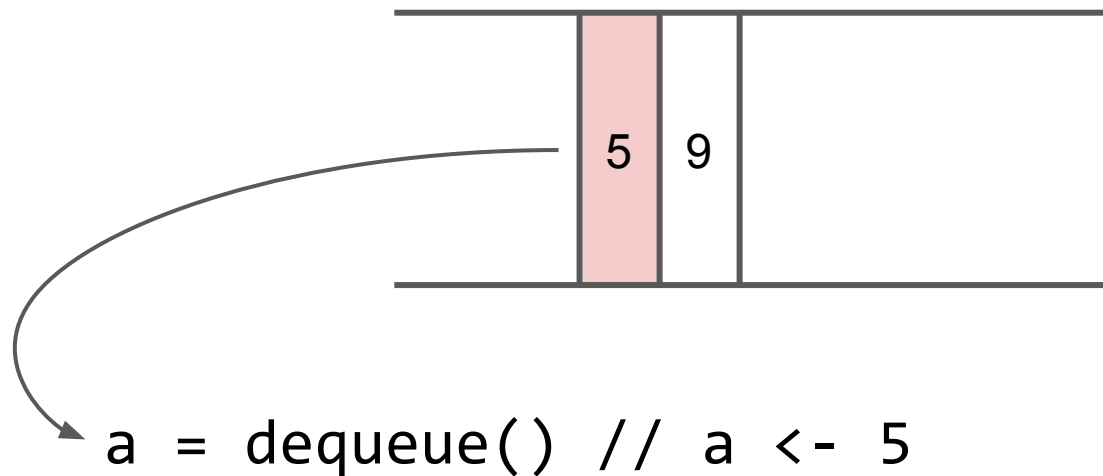
Абстрактный тип данных: очередь

First In, First Out



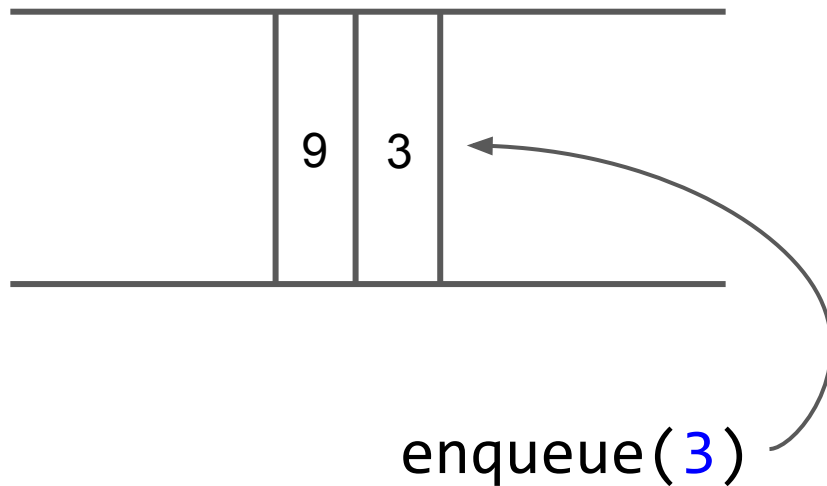
Абстрактный тип данных: очередь

First In, First Out



Абстрактный тип данных: очередь

First In, First Out



Абстрактный тип данных: очередь с приоритетами



Абстрактный тип данных: очередь с приоритетами

Множество значений: пары `<priority: int, value: T>`

Абстрактный тип данных: очередь с приоритетами

Множество значений: пары `<priority: int, value: T>`

Операции:

1. `insert(priority, value)` - добавить в очередь задание с указанным приоритетом

Абстрактный тип данных: очередь с приоритетами

Множество значений: пары `<priority: int, value: T>`

Операции:

1. `insert(priority, value)` - добавить в очередь задание с указанным приоритетом
2. `peek_max()` -> T - взять элемент с максимальным* приоритетом (без изменения очереди)

Абстрактный тип данных: очередь с приоритетами

Множество значений: пары `<priority: int, value: T>`

Операции:

1. `insert(priority, value)` - добавить в очередь задание с указанным приоритетом
2. `peek_max()` -> `T` - взять элемент с максимальным* приоритетом (без изменения очереди)

*Если есть несколько элементов с одинаковым приоритетом, выбираем **любой**

Абстрактный тип данных: очередь с приоритетами

Множество значений: пары `<priority: int, value: T>`

Операции:

1. `insert(priority, value)` - добавить в очередь задание с указанным приоритетом
2. `peek_max()` $\rightarrow T$ - взять элемент с максимальным* приоритетом (без изменения очереди)
3. `extract_max()` $\rightarrow T$ - взять элемент с максимальным* приоритетом (удалив его из очереди)

Абстрактный тип данных: очередь с приоритетами

Множество значений: пары `<priority: int, value: T>`

Операции:

1. `insert(priority, value)` - добавить в очередь задание с указанным приоритетом
2. `peek_max()` $\rightarrow T$ - взять элемент с максимальным* приоритетом (без изменения очереди)
3. `extract_max()` $\rightarrow T$ - взять элемент с максимальным* приоритетом (удалив его из очереди)

Как реализовывать будем?

Структура данных: невозрастающая пирамида



Структура данных: невозрастающая пирамида

Массив A размерности N называется невозрастающей пирамидой*, если:

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Структура данных: невозрастающая пирамида

Массив A размерности N называется невозрастающей пирамидой*, если:

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

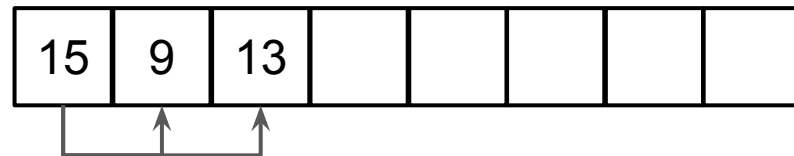
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Другие названия: бинарная куча, `binary heap`

Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

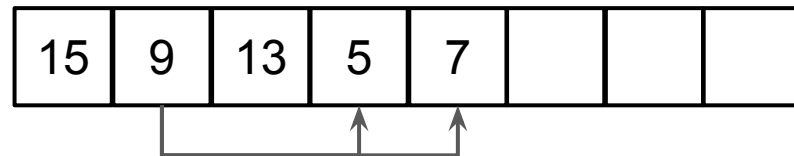
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$



Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

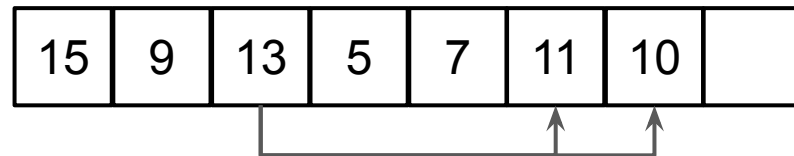
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$



Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

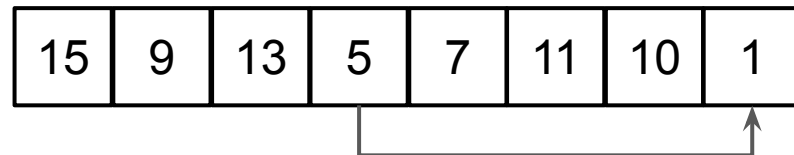
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$



Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

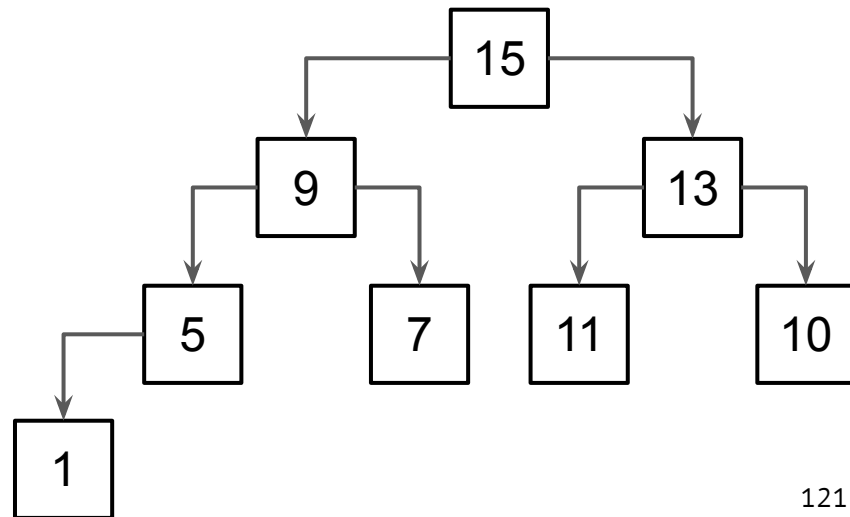
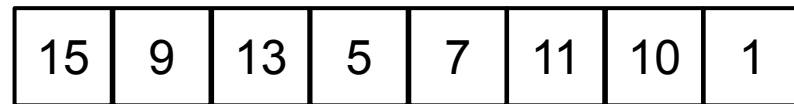
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$



Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

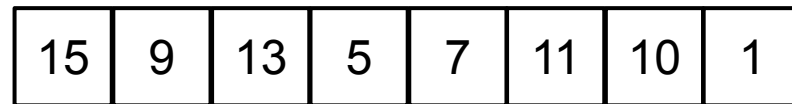
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$



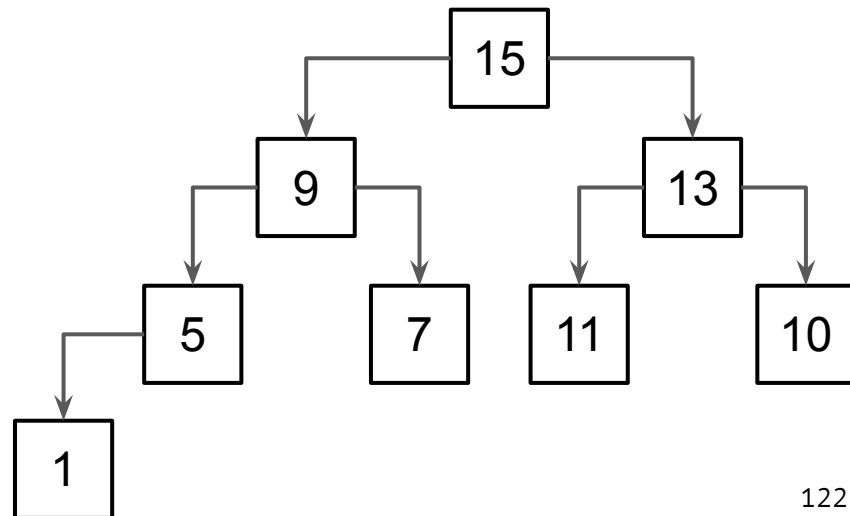
Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$



Что можете сказать
про такой массив?



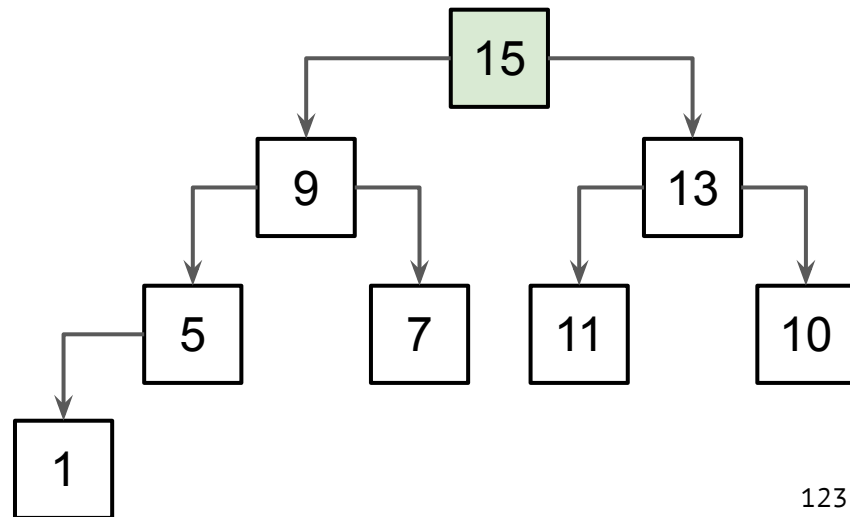
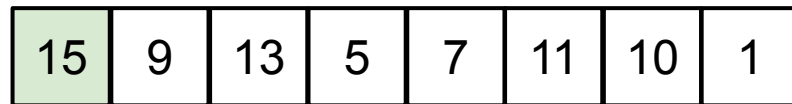
Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Что можете сказать
про такой массив?

Мы всегда знаем, где
его **максимальный**
элемент!



Структура данных: невозрастающая пирамида

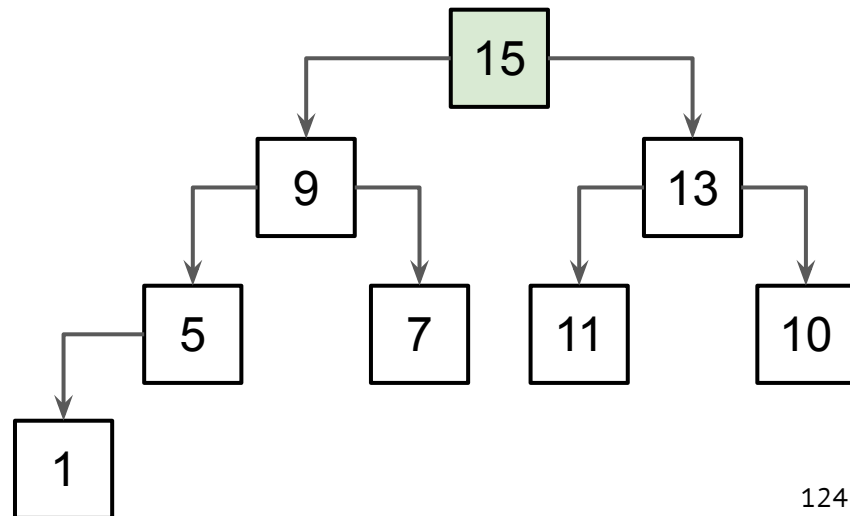
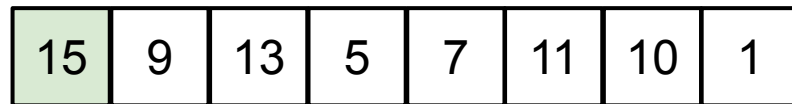
$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Что можете сказать
про такой массив?

Мы всегда знаем, где
его **максимальный**
элемент!

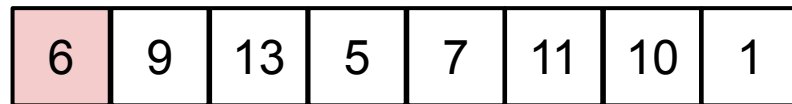
$A[0]$ - максимальный



Структура данных: невозрастающая пирамида

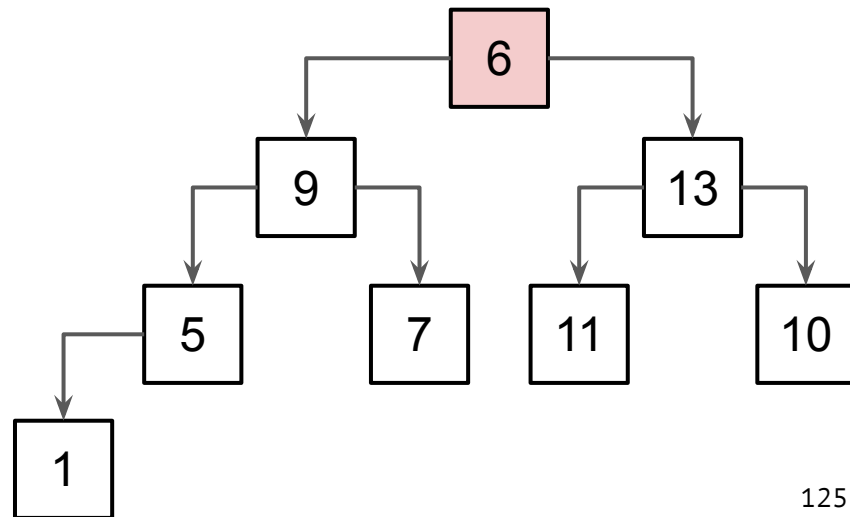
$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$



Пусть теперь наш
корень **сломали**.

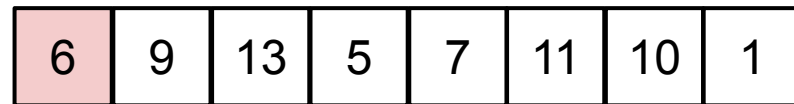
Как починить?



Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

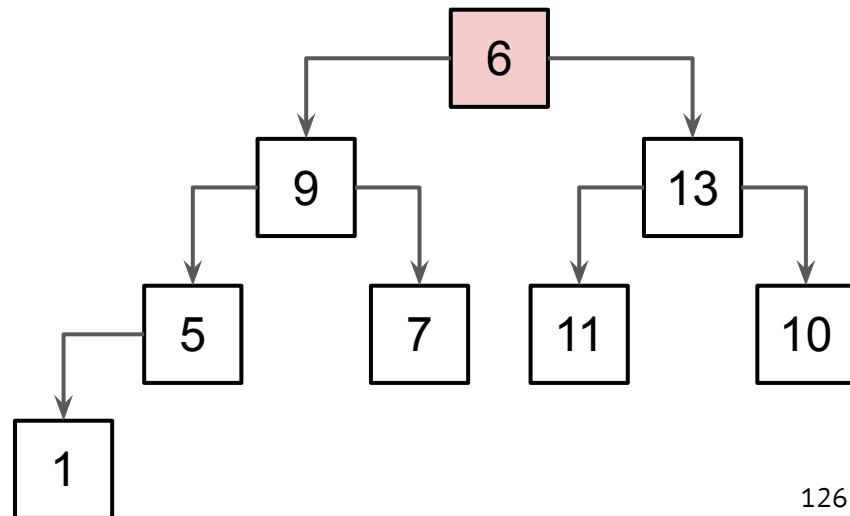
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$



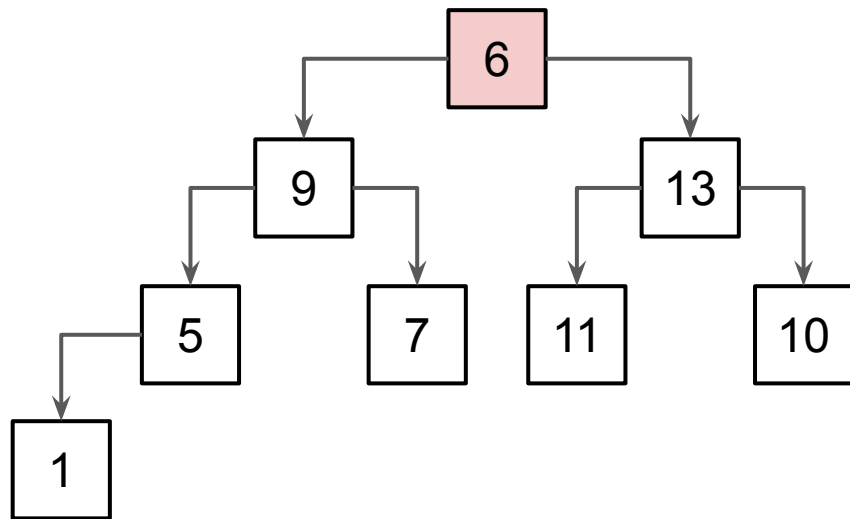
Пусть теперь наш
корень **сломали**.

Как починить?

Операция **просеивания** вниз



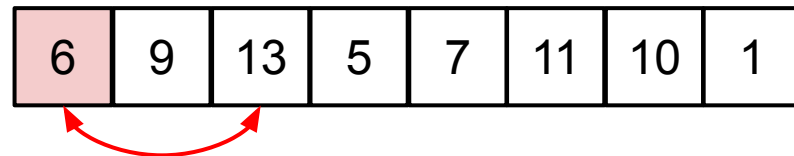
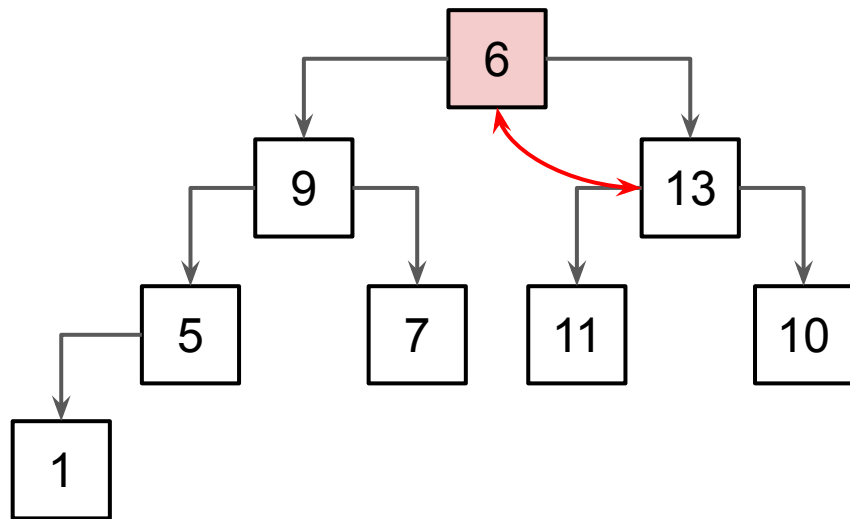
Структура данных: невозрастающая пирамида



6	9	13	5	7	11	10	1
---	---	----	---	---	----	----	---

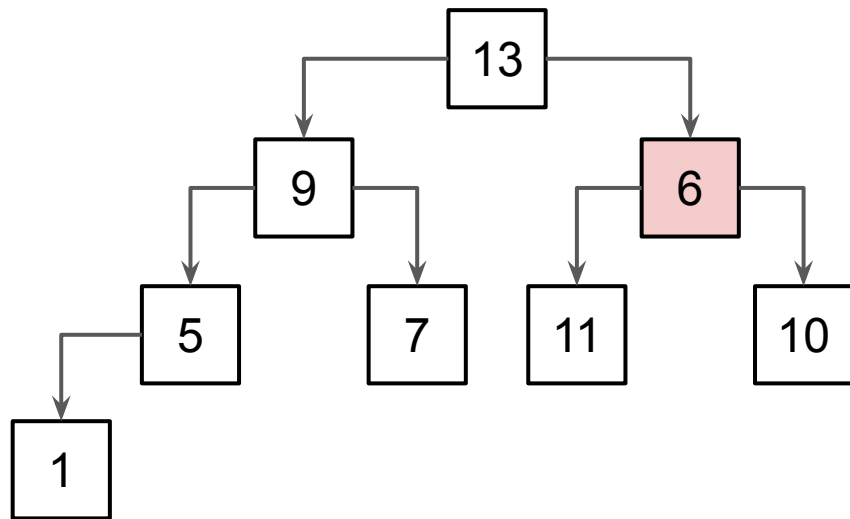
Поменяем местами с
максимальным из детей

Структура данных: невозрастающая пирамида



Поменяем местами с
максимальным из детей

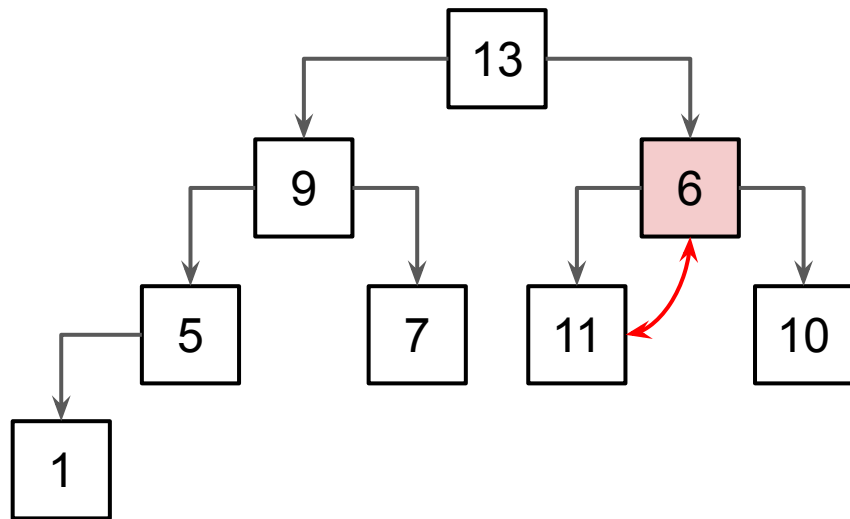
Структура данных: невозрастающая пирамида



13	9	6	5	7	11	10	1
----	---	---	---	---	----	----	---

Поменяем местами с
максимальным из детей

Структура данных: невозрастающая пирамида

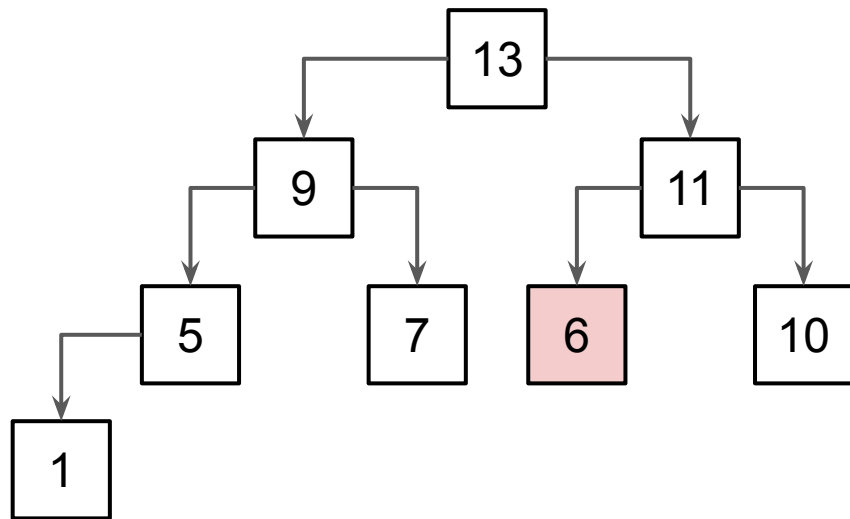


13	9	6	5	7	11	10	1
----	---	---	---	---	----	----	---

Поменяем местами с
максимальным из детей

Теперь чиним маленькую
пирамиду

Структура данных: невозрастающая пирамида

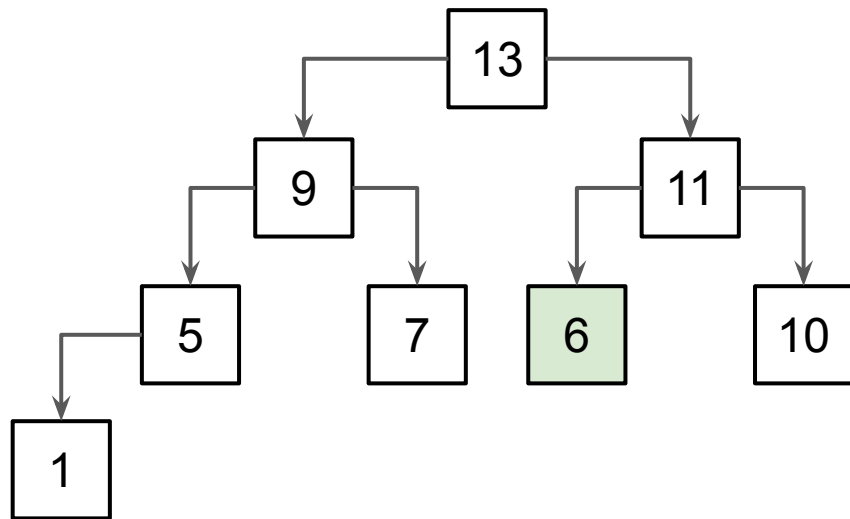


13	9	11	5	7	6	10	1
----	---	----	---	---	---	----	---

Поменяем местами с
максимальным из детей

Теперь чиним маленькую
пирамиду

Структура данных: невозрастающая пирамида



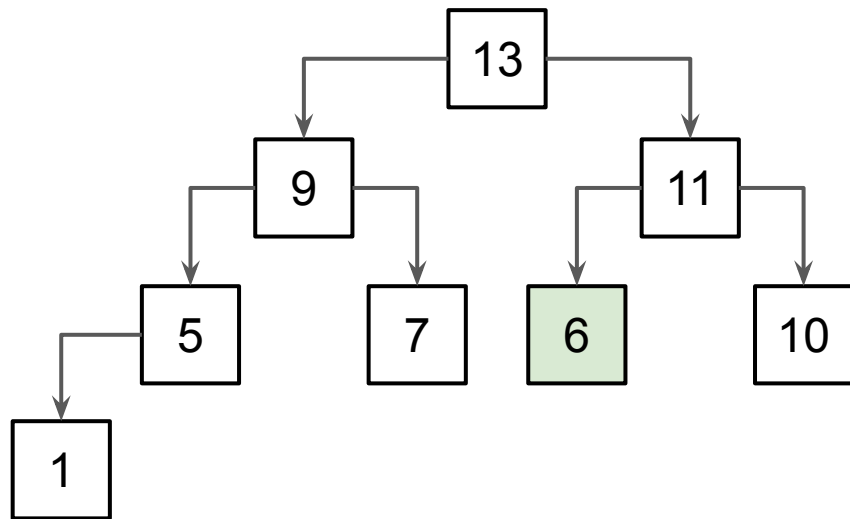
13	9	11	5	7	6	10	1
----	---	----	---	---	---	----	---

Поменяем местами с
максимальным из детей

Теперь чиним маленькую
пирамиду

Готово! Восстановили свойство
пирамиды

Структура данных: невозрастающая пирамида



13	9	11	5	7	6	10	1
----	---	----	---	---	---	----	---

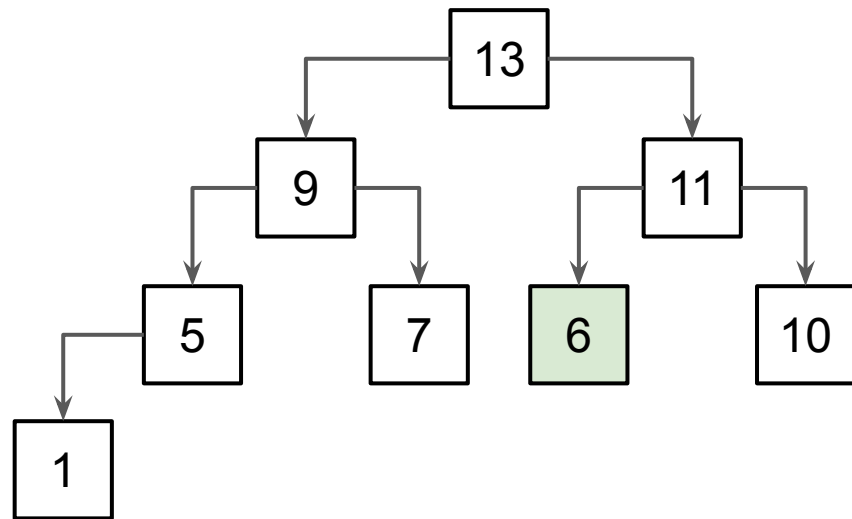
Поменяем местами с
максимальным из детей

Теперь чиним маленькую
пирамиду

Готово! Восстановили свойство
пирамиды

Сложность операции?

Структура данных: невозрастающая пирамида



13	9	11	5	7	6	10	1
----	---	----	---	---	---	----	---

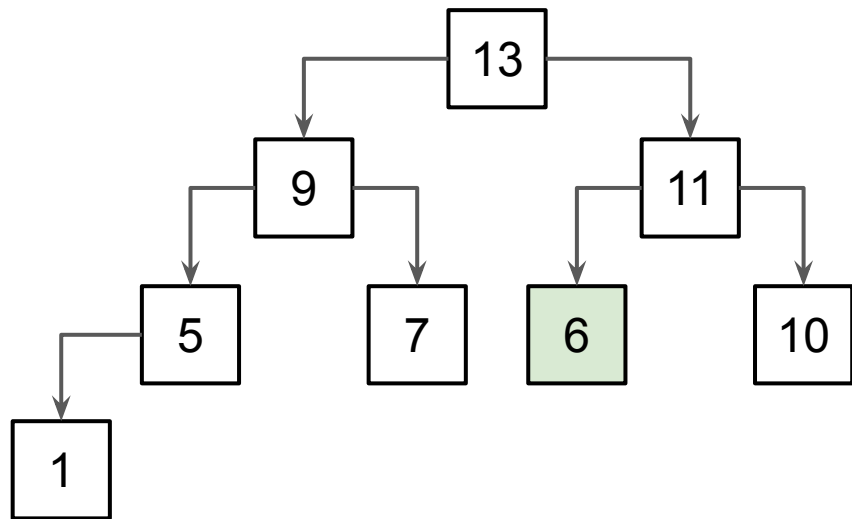
Поменяем местами с
максимальным из детей

Теперь чиним маленькую
пирамиду

Готово! Восстановили свойство
пирамиды

Сложность операции? $O(\log_2 N)$

Структура данных: невозрастающая пирамида



13	9	11	5	7	6	10	1
----	---	----	---	---	---	----	---

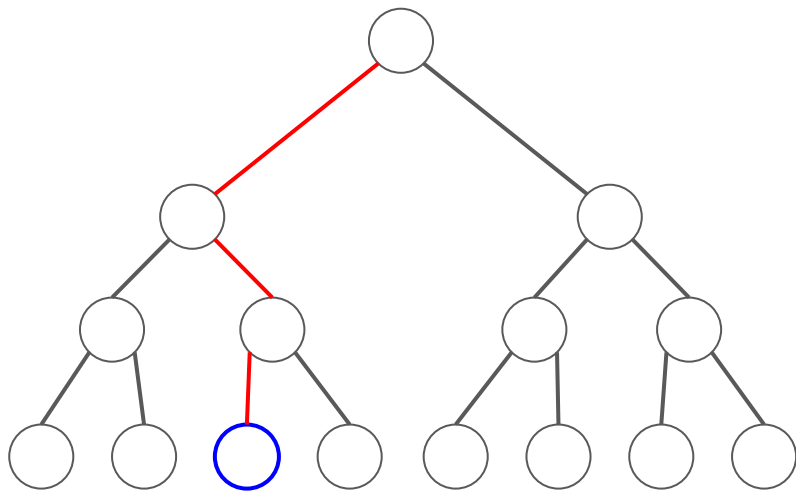
Поменяем местами с
максимальным из детей

Теперь чиним маленькую
пирамиду

Готово! Восстановили свойство
пирамиды

Сложность операции? $O(\log_2 N)$
Кстати, а почему?

Дерево решений



Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

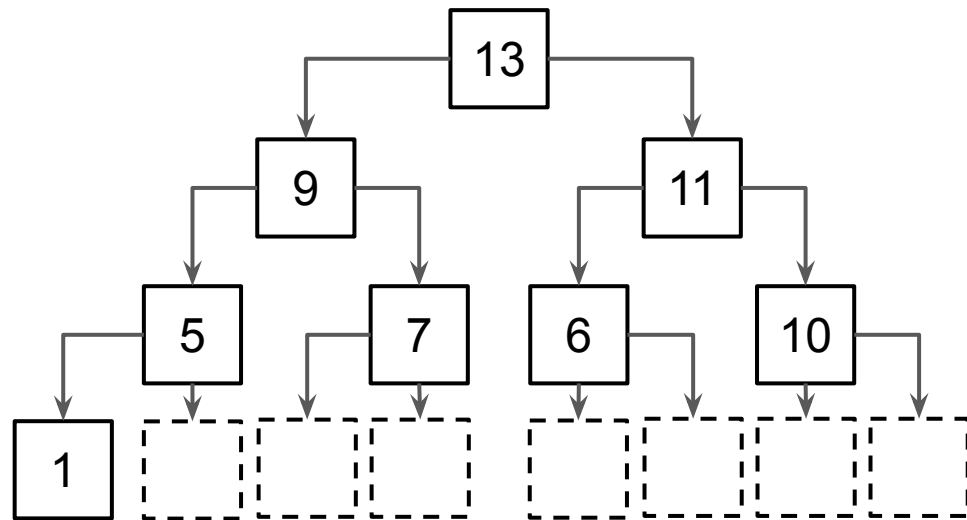
Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

А сколько у бинарного дерева высоты k может быть листьев?

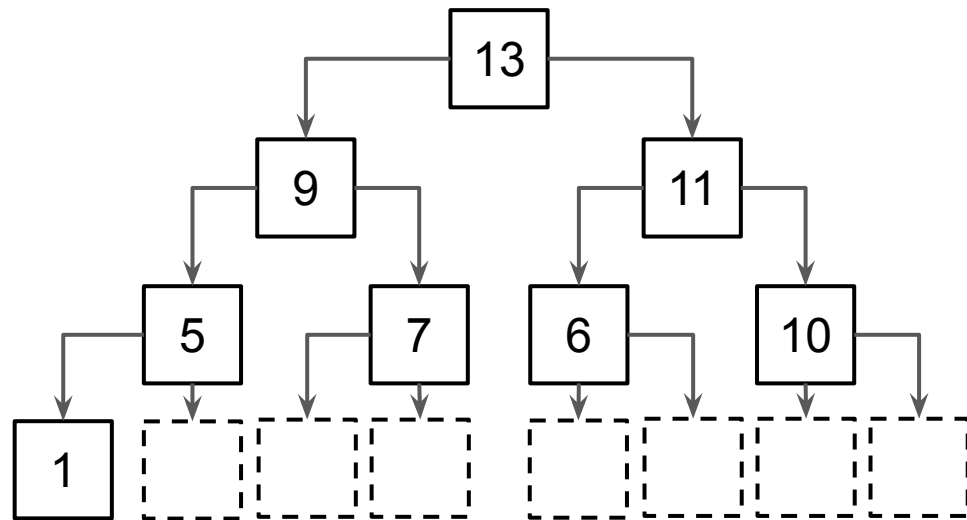
$$\#(\text{листьев}) \leq 2^k$$

Структура данных: невозрастающая пирамида



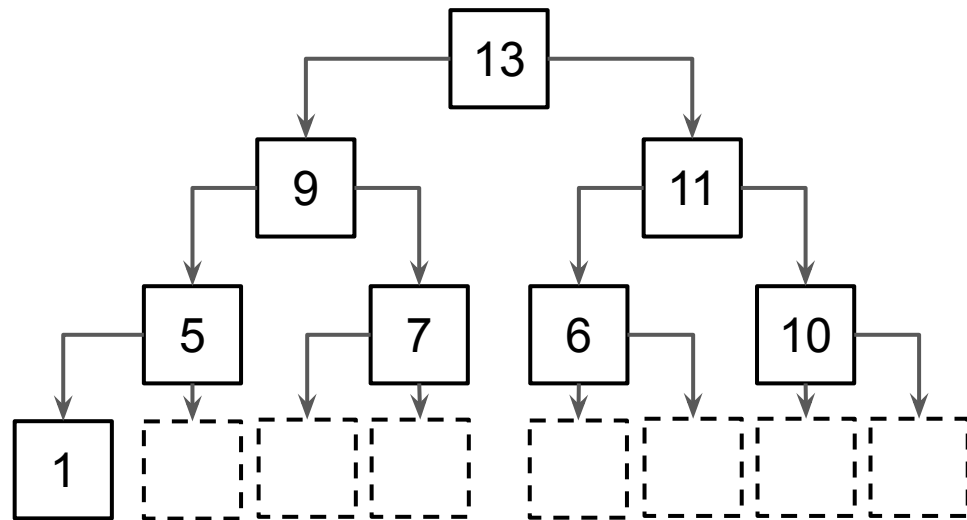
Худший случай - когда куча является полным бинарным деревом

Структура данных: невозрастающая пирамида



Худший случай - когда куча является полным бинарным деревом (и просеивать нужно до конца)

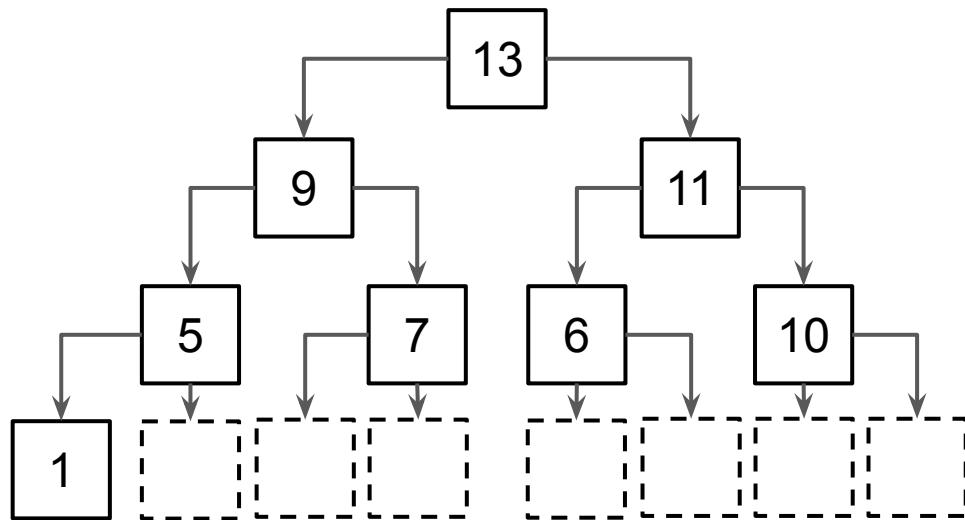
Структура данных: невозрастающая пирамида



Худший случай - когда куча является полным бинарным деревом (и просеивать нужно до конца)

Сколько тогда будет смены элементов?

Структура данных: невозрастающая пирамида

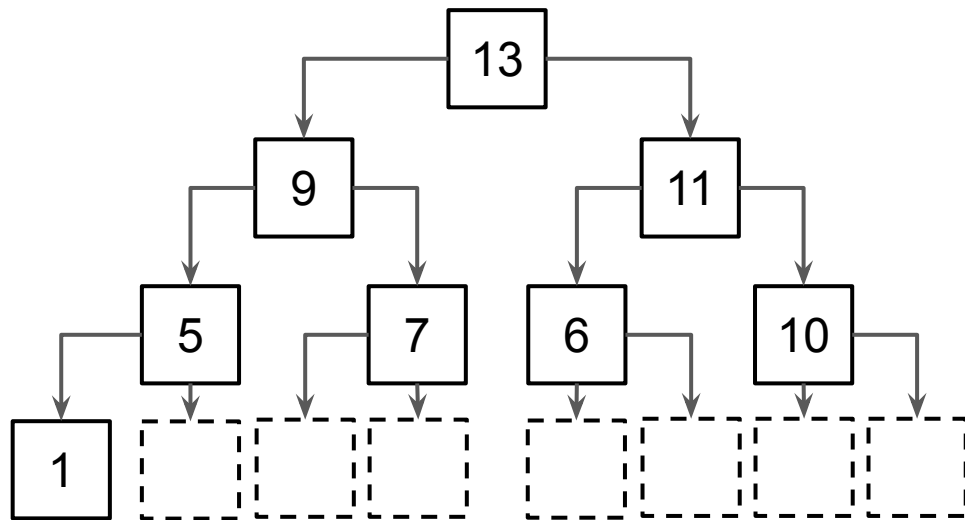


Худший случай - когда куча является полным бинарным деревом (и просеивать нужно до конца)

Сколько тогда будет смены элементов?

Высота дерева!

Структура данных: невозрастающая пирамида



Худший случай - когда куча является полным бинарным деревом (и просеивать нужно до конца)

Сколько тогда будет смены элементов?

Высота дерева!

Как связаны высота полного бинарного дерева и количество элементов в нем?

Структура данных: невозрастающая пирамида

Утверждение: пусть h - высота полного бинарного дерева, а N - количество элементов в нем.

Тогда: $2^h = \frac{N+1}{2}$

Структура данных: невозрастающая пирамида

Утверждение: пусть h - высота полного бинарного дерева, а N - количество элементов в нем.

Тогда: $2^h = \frac{N+1}{2}$

Доказательство:

Для дерева высоты 0 (из одного элемента верно): $2^0 = \frac{1+1}{2} = 1$

Структура данных: невозрастающая пирамида

Утверждение: пусть h - высота полного бинарного дерева, а N - количество элементов в нем.

Тогда: $2^h = \frac{N+1}{2}$

Доказательство:

Для дерева высоты 0 (из одного элемента верно): $2^0 = \frac{1+1}{2} = 1$

Пусть верно для дерева высоты k : $2^k = \frac{n_k+1}{2}$

Структура данных: невозрастающая пирамида

Утверждение: пусть h - высота полного бинарного дерева, а N - количество элементов в нем.

Тогда: $2^h = \frac{N+1}{2}$

Доказательство:

Для дерева высоты 0 (из одного элемента верно): $2^0 = \frac{1+1}{2} = 1$

Пусть верно для дерева высоты k : $2^k = \frac{n_k+1}{2}$

Заметим: $n_{k+1} = n_k + 2^{k+1}$

Структура данных: невозрастающая пирамида

Утверждение: пусть h - высота полного бинарного дерева, а N - количество элементов в нем.

$$\text{Тогда: } 2^h = \frac{N+1}{2}$$

Доказательство:

Для дерева высоты 0 (из одного элемента верно): $2^0 = \frac{1+1}{2} = 1$

Пусть верно для дерева высоты k : $2^k = \frac{n_k+1}{2}$

Заметим: $n_{k+1} = n_k + 2^{k+1}$

$$\text{Тогда: } \frac{n_{k+1}+1}{2} = \frac{n_k+2^{k+1}+1}{2}$$

Структура данных: невозрастающая пирамида

Утверждение: пусть h - высота полного бинарного дерева, а N - количество элементов в нем.

$$\text{Тогда: } 2^h = \frac{N+1}{2}$$

Доказательство:

Для дерева высоты 0 (из одного элемента верно): $2^0 = \frac{1+1}{2} = 1$

Пусть верно для дерева высоты k : $2^k = \frac{n_k+1}{2}$

Заметим: $n_{k+1} = n_k + 2^{k+1}$

$$\text{Тогда: } \frac{n_{k+1}+1}{2} = \frac{n_k+2^{k+1}+1}{2} = \frac{n_k+1}{2} + 2^k$$

Структура данных: невозрастающая пирамида

Утверждение: пусть h - высота полного бинарного дерева, а N - количество элементов в нем.

Тогда: $2^h = \frac{N+1}{2}$

Доказательство:

Для дерева высоты 0 (из одного элемента верно): $2^0 = \frac{1+1}{2} = 1$

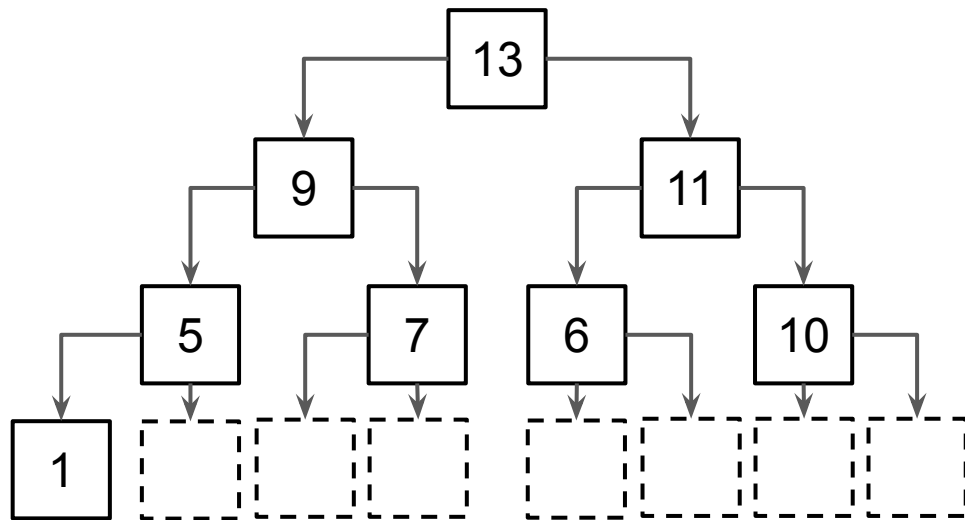
Пусть верно для дерева высоты k : $2^k = \frac{n_k+1}{2}$

Заметим: $n_{k+1} = n_k + 2^{k+1}$

Тогда: $\frac{n_{k+1}+1}{2} = \frac{n_k+2^{k+1}+1}{2} = \frac{n_k+1}{2} + 2^k = 2^k + 2^k = 2^{k+1}$



Структура данных: невозрастающая пирамида



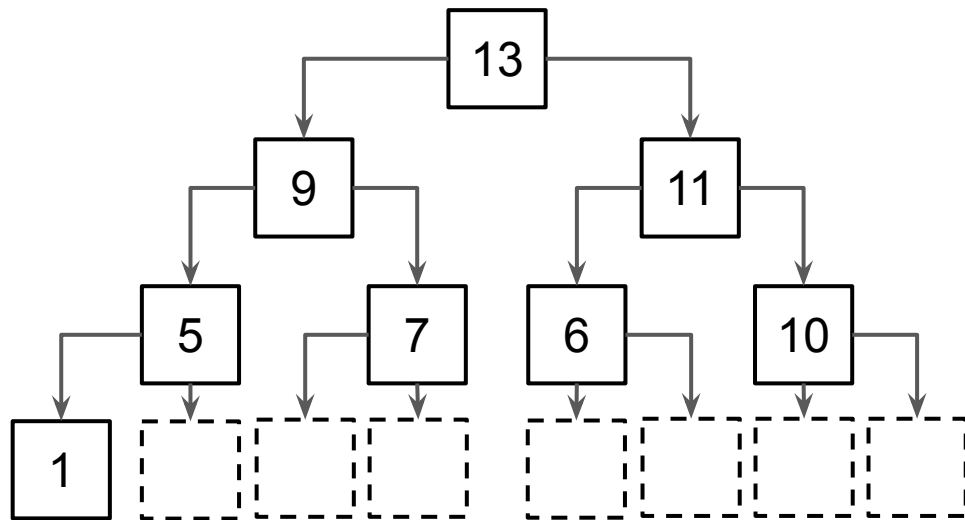
Худший случай - когда куча является полным бинарным деревом (и просеивать нужно до конца)

Сколько тогда будет смены элементов?

Высота дерева!

Как связаны высота полного бинарного дерева и количество элементов в нем?

Структура данных: невозрастающая пирамида



Худший случай - когда куча является полным бинарным деревом (и просеивать нужно до конца)

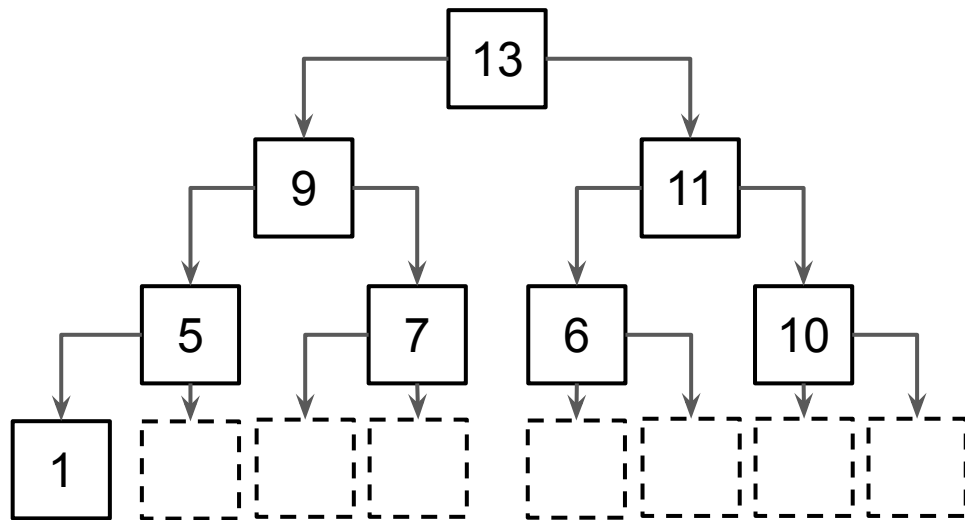
Сколько тогда будет смены элементов?

Высота дерева!

Как связаны высота полного бинарного дерева и количество элементов в нем?

$$2^h = \frac{N+1}{2}$$

Структура данных: невозрастающая пирамида



Худший случай - когда куча является полным бинарным деревом (и просеивать нужно до конца)

Сколько тогда будет смены элементов?

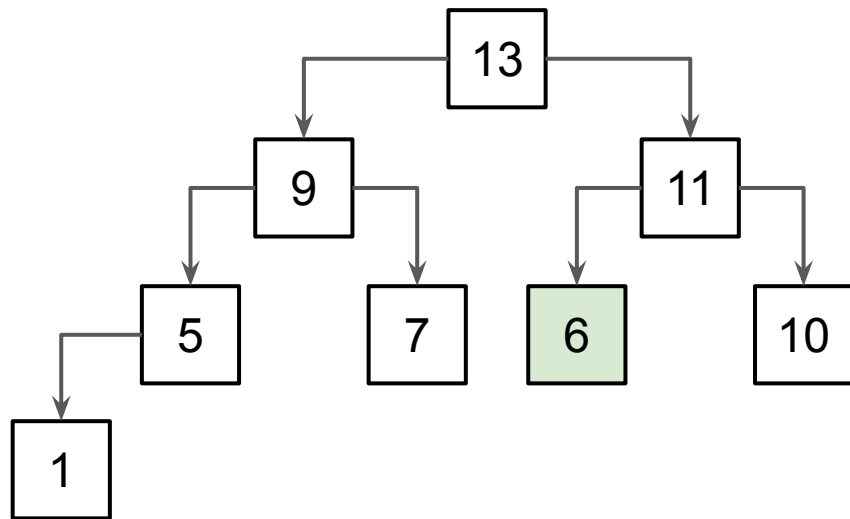
Высота дерева!

Как связаны высота полного бинарного дерева и количество элементов в нем?

$$2^h = \frac{N+1}{2}$$

Тогда имеем $\log_2\left(\frac{N+1}{2}\right) = O(\log N)$

Структура данных: невозрастающая пирамида



13	9	11	5	7	6	10	1
----	---	----	---	---	---	----	---

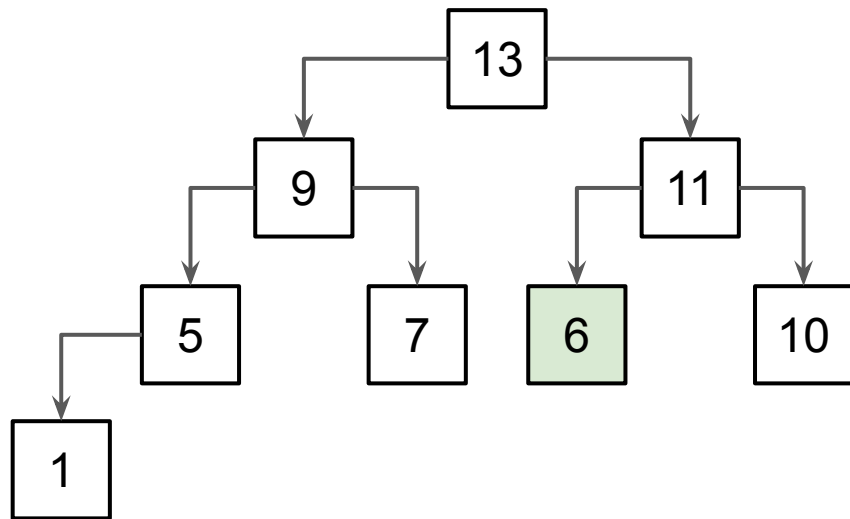
Поменяем местами с
максимальным из детей

Теперь чиним маленькую
пирамиду

Готово! Восстановили свойство
пирамиды

Сложность операции? $O(\log_2 N)$
~~Кстати, а почему?~~

Структура данных: невозрастающая пирамида



13	9	11	5	7	6	10	1
----	---	----	---	---	---	----	---

Поменяем местами с
максимальным из детей

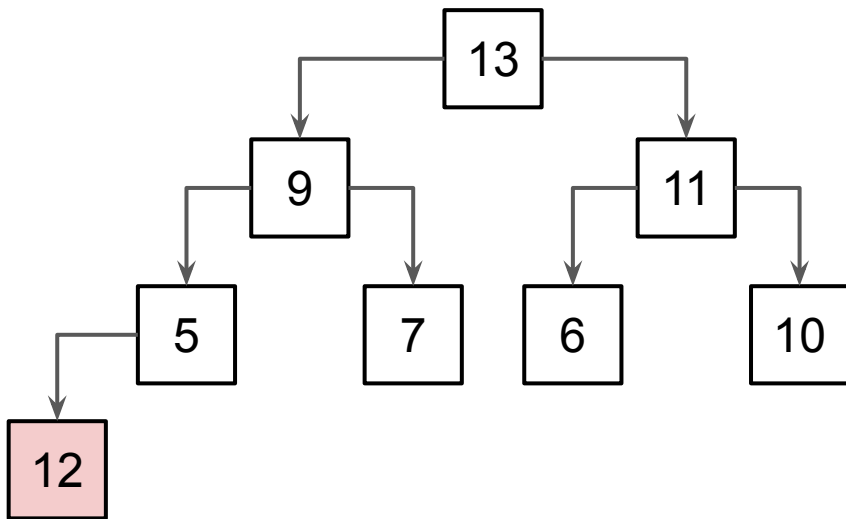
Теперь чиним маленькую
пирамиду

Готово! Восстановили свойство
пирамиды

Сложность операции? $O(\log_2 N)$

Альтернативно можно было через [Master Method](#)

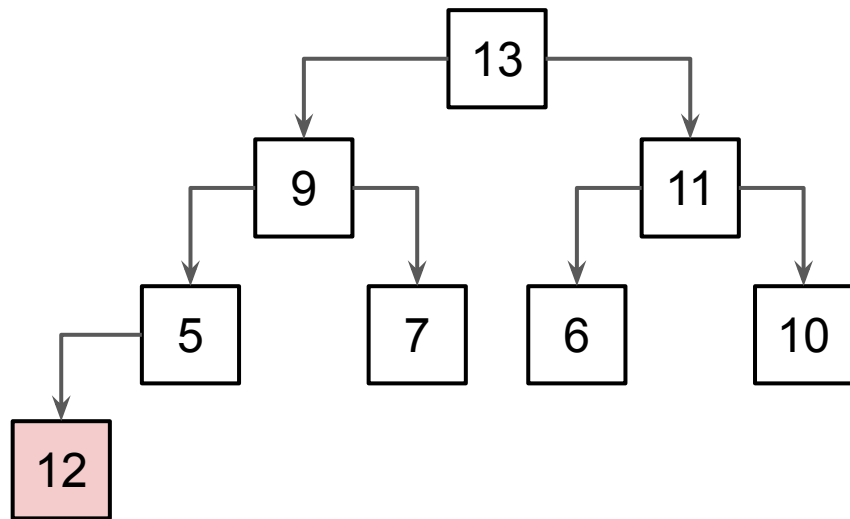
Структура данных: невозрастающая пирамида



13	9	11	5	7	6	10	12
----	---	----	---	---	---	----	----

Теперь пусть пирамиду сломали
снизу. Как починить?

Структура данных: невозрастающая пирамида

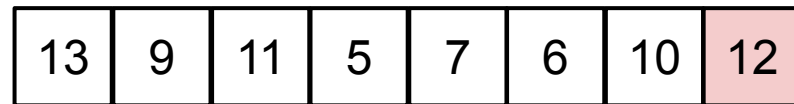
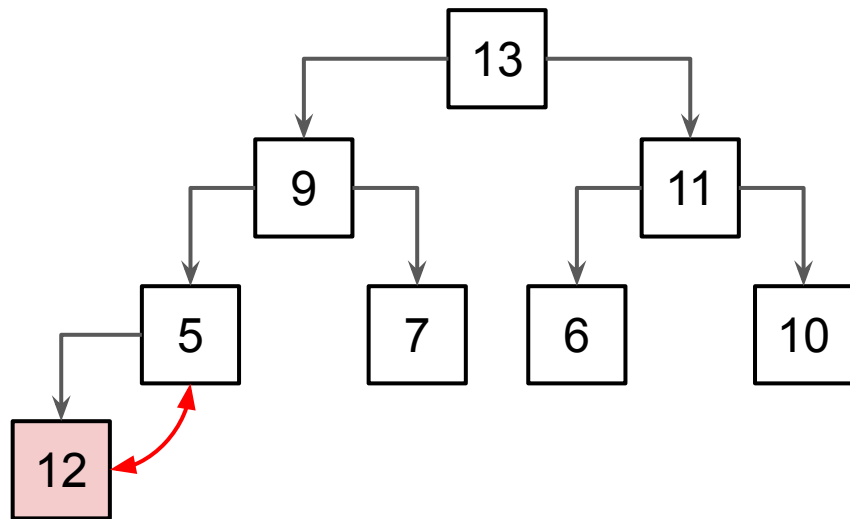


13	9	11	5	7	6	10	12
----	---	----	---	---	---	----	----

Теперь пусть пирамиду сломали снизу. Как починить?

Смотрим на предка, если порядок нарушен - меняем.

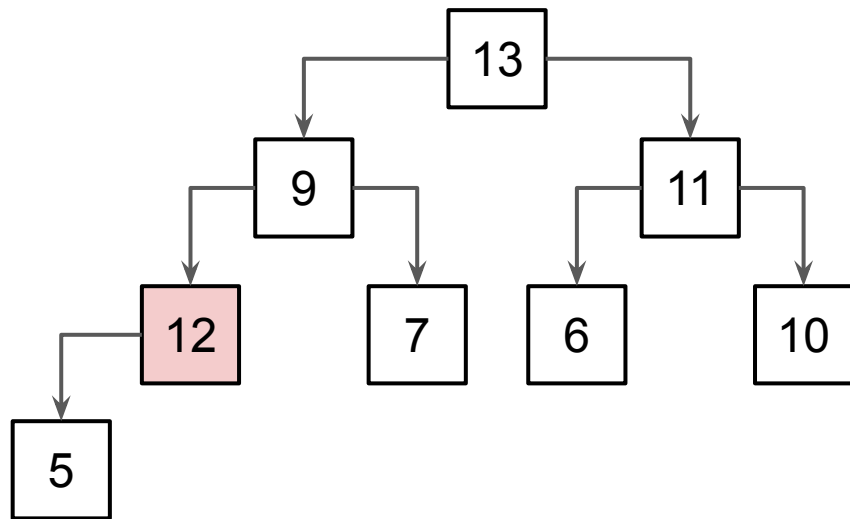
Структура данных: невозрастающая пирамида



Теперь пусть пирамиду сломали снизу. Как починить?

Смотрим на предка, если порядок нарушен - меняем.

Структура данных: невозрастающая пирамида

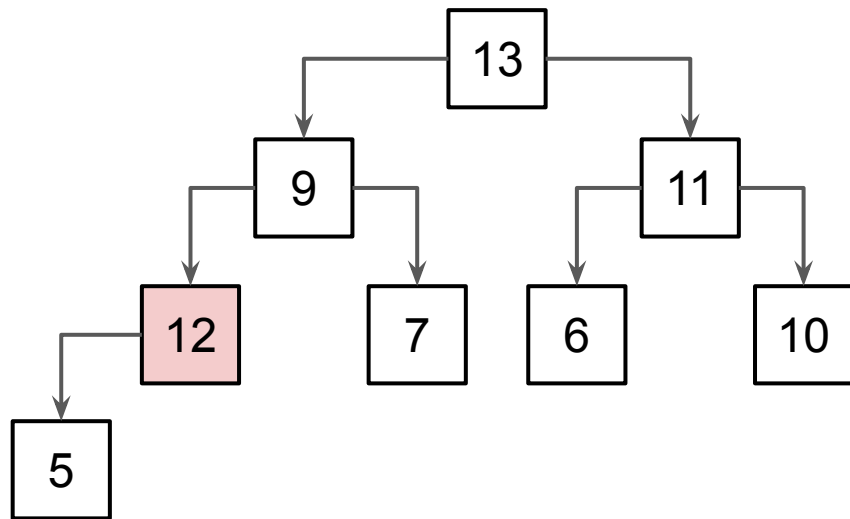


13	9	11	12	7	6	10	5
----	---	----	----	---	---	----	---

Теперь пусть пирамиду сломали снизу. Как починить?

Смотрим на предка, если порядок нарушен - меняем.

Структура данных: невозрастающая пирамида



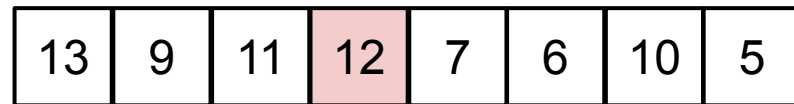
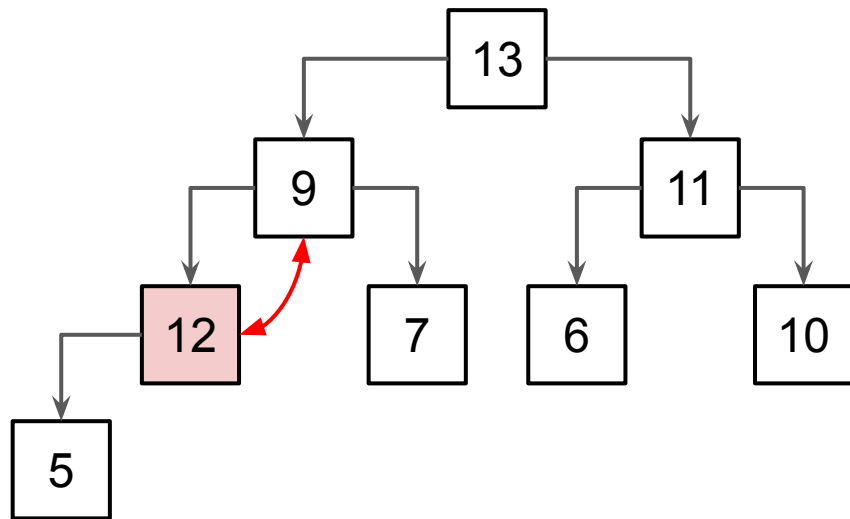
13	9	11	12	7	6	10	5
----	---	----	----	---	---	----	---

Теперь пусть пирамиду сломали снизу. Как починить?

Смотрим на предка, если порядок нарушен - меняем.

Продолжаем, пока не починим.

Структура данных: невозрастающая пирамида

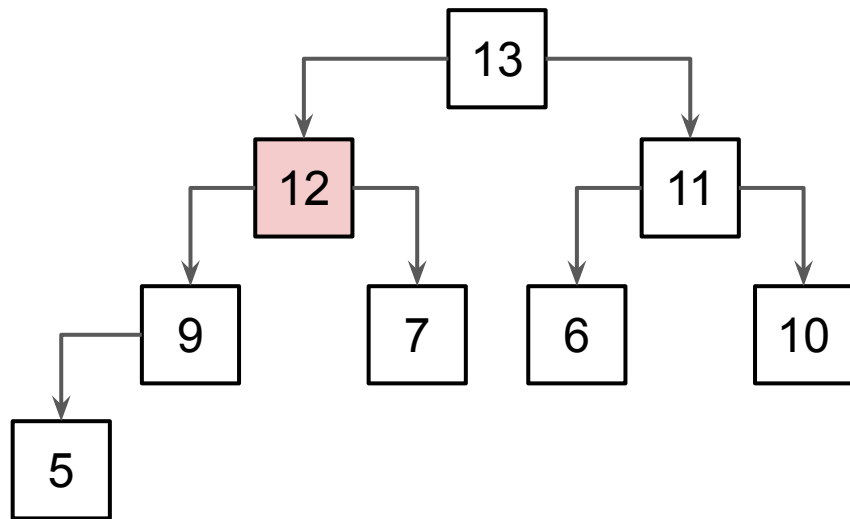


Теперь пусть пирамиду сломали снизу. Как починить?

Смотрим на предка, если порядок нарушен - меняем.

Продолжаем, пока не починим.

Структура данных: невозрастающая пирамида



13	12	11	9	7	6	10	5
----	----	----	---	---	---	----	---

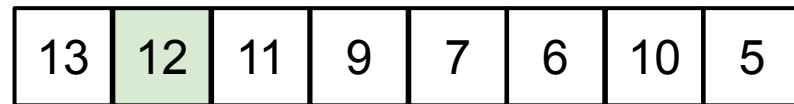
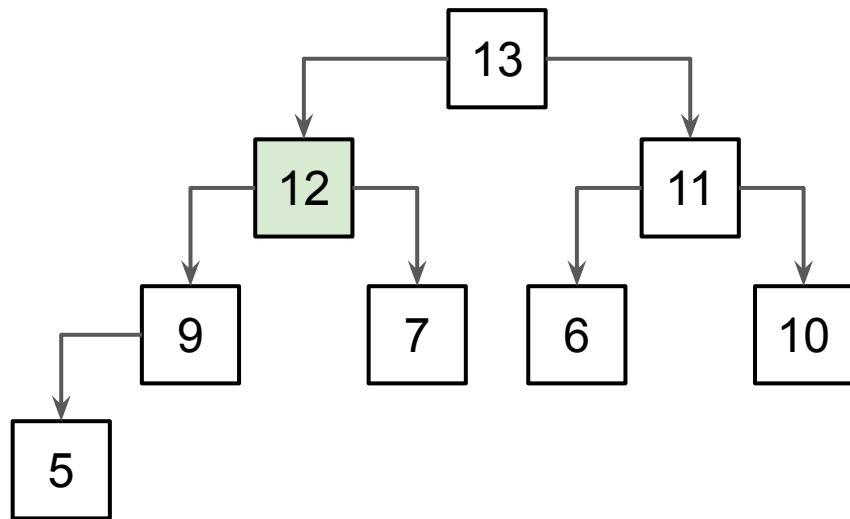
Теперь пусть пирамиду сломали снизу. Как починить?

Смотрим на предка, если порядок нарушен - меняем.

Продолжаем, пока не починим.

Операция просеивания вверх.

Структура данных: невозрастающая пирамида



Теперь пусть пирамиду сломали снизу. Как починить?

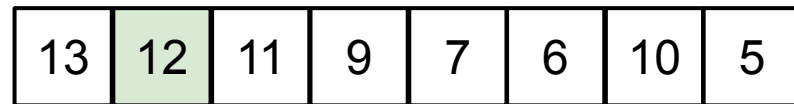
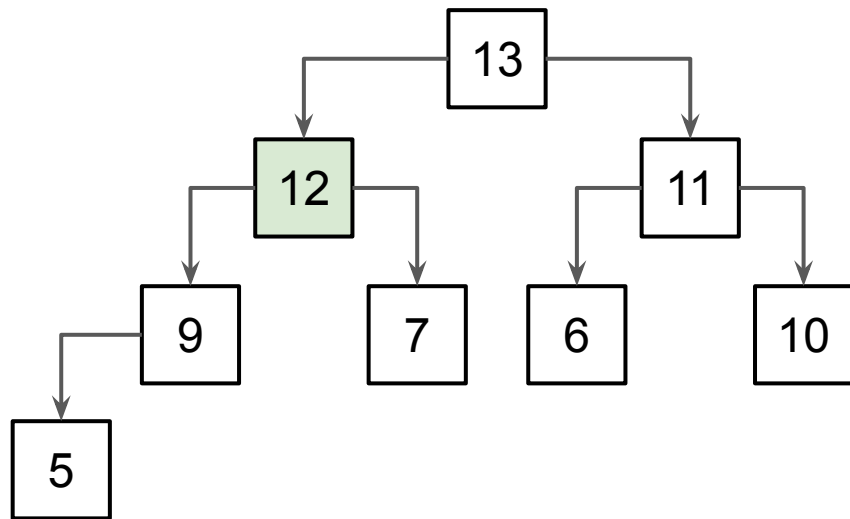
Смотрим на предка, если порядок нарушен - меняем.

Продолжаем, пока не починим.

Сложность операции?

Операция просеивания вверх.

Структура данных: невозрастающая пирамида



Теперь пусть пирамиду сломали снизу. Как починить?

Смотрим на предка, если порядок нарушен - меняем.

Продолжаем, пока не починим.

Сложность операции? $O(\log_2 N)$

Операция просеивания вверх.

Структура данных: невозрастающая пирамида

Массив A размерности N называется невозрастающей пирамидой*, если:

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Другие названия: бинарная куча, `binary heap`

Операции:

1. Узнать максимум - $O(1)$

Структура данных: невозрастающая пирамида

Массив A размерности N называется невозрастающей пирамидой*, если:

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Другие названия: бинарная куча, **binary heap**

Операции:

1. Узнать максимум - $O(1)$
2. Починить, если сломали корень - $O(\log N)$
3. Починить, если сломали лист - $O(\log N)$

Абстрактный тип данных: очередь с приоритетами

Множество значений: пары `<priority: int, value: T>`

Операции:

1. `insert(priority, value)` - добавить в очередь задание с указанным приоритетом
2. `peek_max()` $\rightarrow T$ - взять элемент с максимальным* приоритетом (без изменения очереди)
3. `extract_max()` $\rightarrow T$ - взять элемент с максимальным* приоритетом (удалив его из очереди)

Как реализовывать будем?

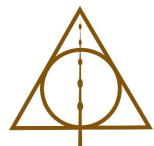
Абстрактный тип данных: очередь с приоритетами

Множество значений: пары `<priority: int, value: T>`

Операции:

1. `insert(priority, value)` - добавить в очередь задание с указанным приоритетом

2. `peek_max()` $\rightarrow T$ - взять элемент с максимальным* приоритетом (без изменения очереди)



3. `extract_max()` $\rightarrow T$ - взять элемент с максимальным* приоритетом (удалив его из очереди)

Как реализовывать будем? Пирамида!

Очередь с приоритетами через пирамиду

Множество значений: пары `<priority: int, value: T>`



Будем хранить массив из пар `<priority, value>`
(но относимся к этому массиву, как к пирамиде из ключей)

Очередь с приоритетами через пирамиду

Множество значений: пары `<priority: int, value: T>`



Будем хранить массив из пар `<priority, value>`
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. `insert(priority, value)`
2. `peek_max() -> T`
3. `extract_max() -> T`

Очередь с приоритетами через пирамиду

Множество значений: пары `<priority: int, value: T>`



Будем хранить массив из пар `<priority, value>`
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. `insert(priority, value)`
2. `peek_max()` -> T
3. `extract_max()` -> T

`peek_max` - это просто взятие максимального элемента из пирамиды за $O(1)$

Очередь с приоритетами через пирамиду

Множество значений: пары `<priority: int, value: T>`



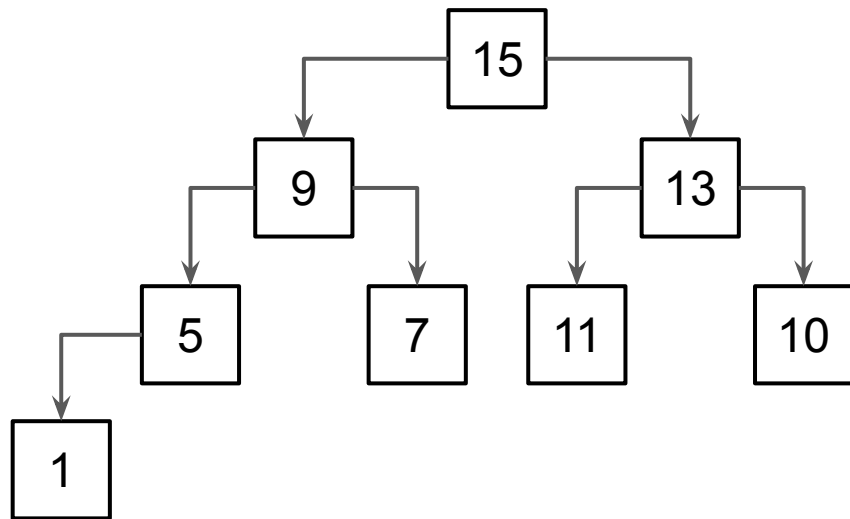
Будем хранить массив из пар `<priority, value>`
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. `insert(priority, value)`
2. `peek_max()` -> T
3. `extract_max()` -> T

`extract_max?`

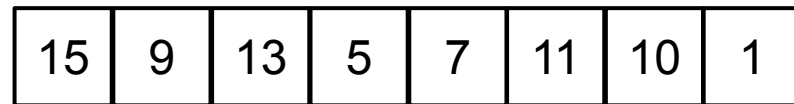
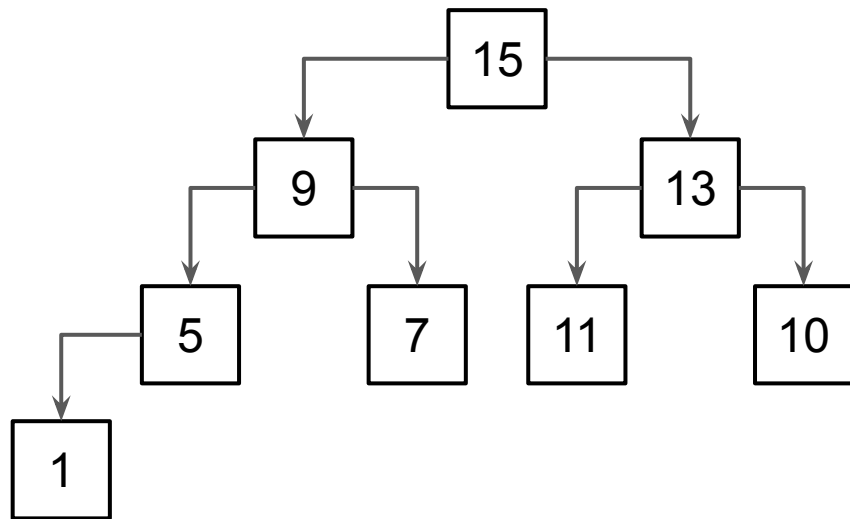
Структура данных: невозрастающая пирамида



15	9	13	5	7	11	10	1
----	---	----	---	---	----	----	---

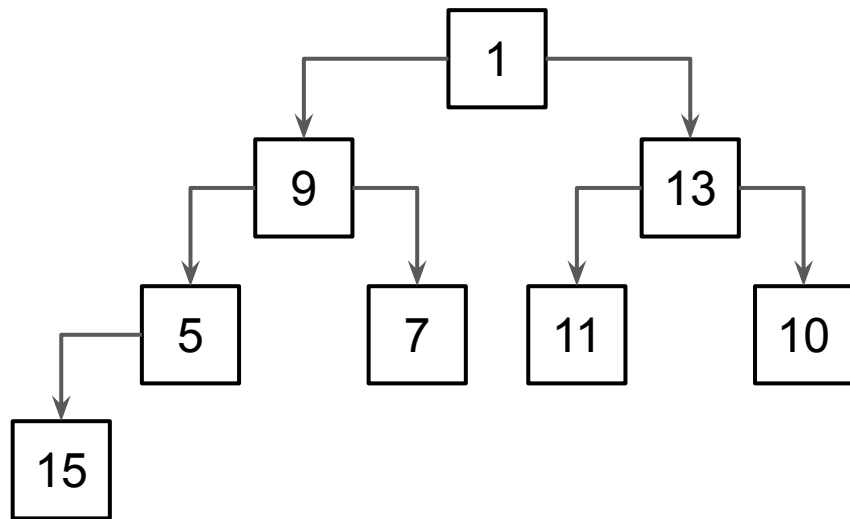
1. `res = A[0]`

Структура данных: невозрастающая пирамида



1. `res = A[0]`
2. меняем первый и последний элементы

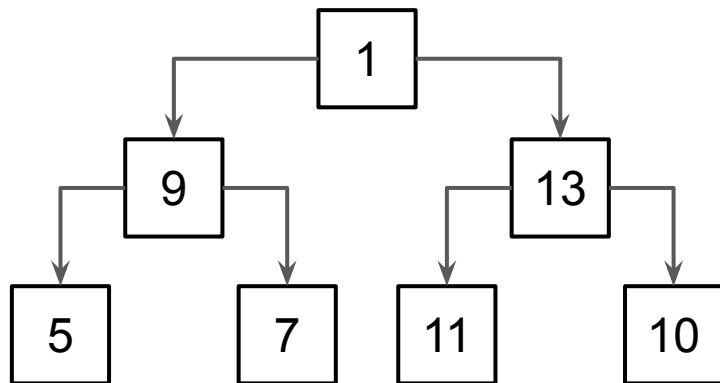
Структура данных: невозрастающая пирамида



1	9	13	5	7	11	10	15
---	---	----	---	---	----	----	----

1. `res = A[0]`
2. меняем первый и последний элементы

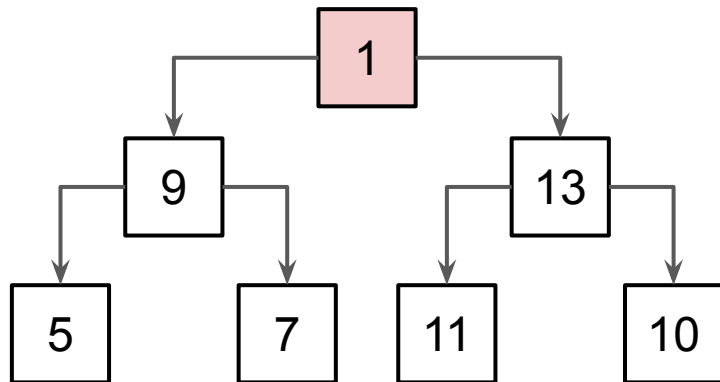
Структура данных: невозрастающая пирамида



1	9	13	5	7	11	10
---	---	----	---	---	----	----

1. `res = A[0]`
2. меняем первый и последний элементы
3. уменьшаем размер массива на 1

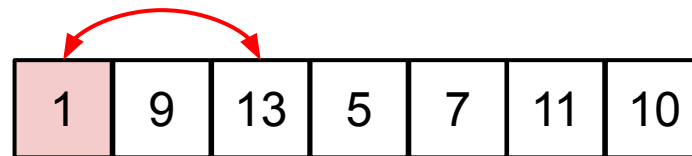
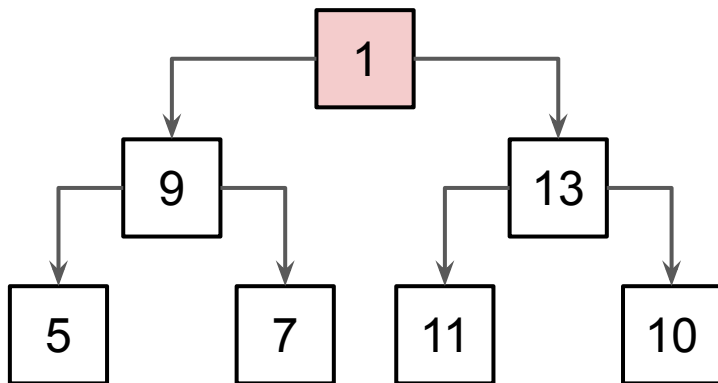
Структура данных: невозрастающая пирамида



1	9	13	5	7	11	10
---	---	----	---	---	----	----

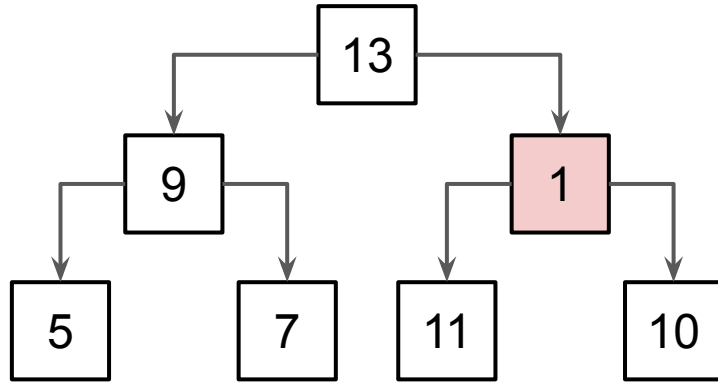
1. `res = A[0]`
2. меняем первый и последний элементы
3. уменьшаем размер массива на 1
4. получаем сломанную пирамиду

Структура данных: невозрастающая пирамида



1. `res = A[0]`
2. меняем первый и последний элементы
3. уменьшаем размер массива на 1
4. получаем сломанную пирамиду
5. чиним просеиванием

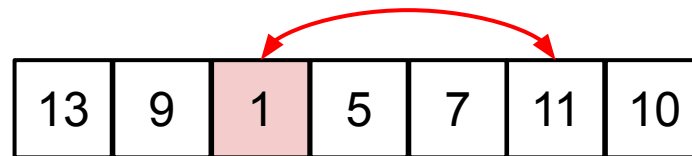
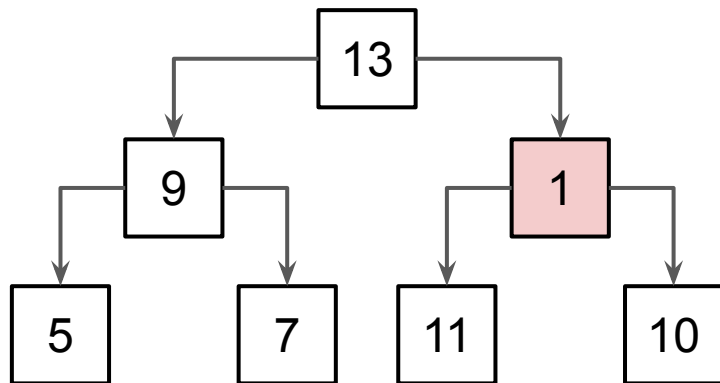
Структура данных: невозрастающая пирамида



13	9	1	5	7	11	10
----	---	---	---	---	----	----

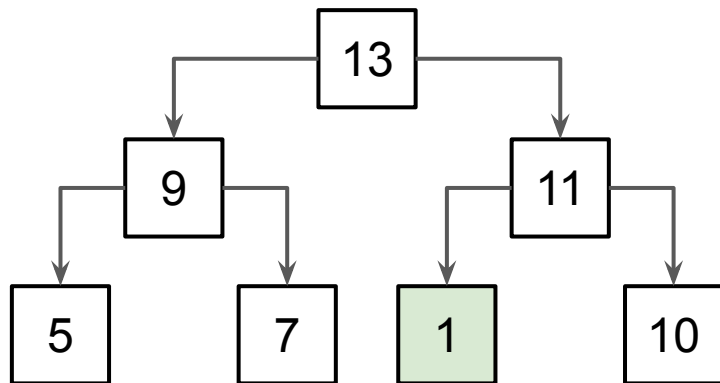
1. `res = A[0]`
2. меняем первый и последний элементы
3. уменьшаем размер массива на 1
4. получаем сломанную пирамиду
5. чиним просеиванием

Структура данных: невозрастающая пирамида



1. `res = A[0]`
2. меняем первый и последний элементы
3. уменьшаем размер массива на 1
4. получаем сломанную пирамиду
5. чиним просеиванием

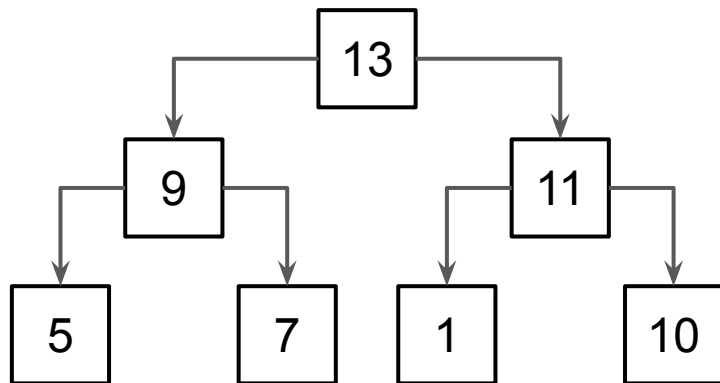
Структура данных: невозрастающая пирамида



13	9	11	5	7	1	10
----	---	----	---	---	---	----

1. `res = A[0]`
2. меняем первый и последний элементы
3. уменьшаем размер массива на 1
4. получаем сломанную пирамиду
5. чиним просеиванием

Структура данных: невозрастающая пирамида



1	9	13	5	7	11	10
---	---	----	---	---	----	----

1. `res = A[0]`
2. меняем первый и последний элементы
3. уменьшаем размер массива на 1
4. получаем сломанную пирамиду
5. чиним просеиванием
6. возвращаем `res`

Очередь с приоритетами через пирамиду

Множество значений: пары `<priority: int, value: T>`



Будем хранить массив из пар `<priority, value>`
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. `insert(priority, value)`
2. `peek_max()` $\rightarrow T$
3. `extract_max()` $\rightarrow T$

`extract_max`? Процедура, работающая за $O(\log N)$

Очередь с приоритетами через пирамиду

Множество значений: пары `<priority: int, value: T>`



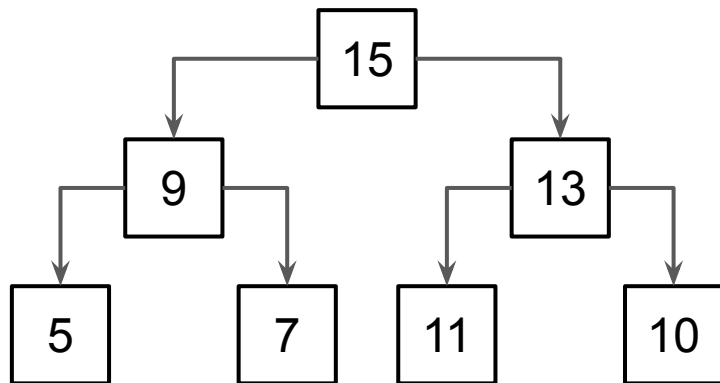
Будем хранить массив из пар `<priority, value>`
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. `insert(priority, value)`
2. `peek_max()` $\rightarrow T$
3. `extract_max()` $\rightarrow T$

`insert?`

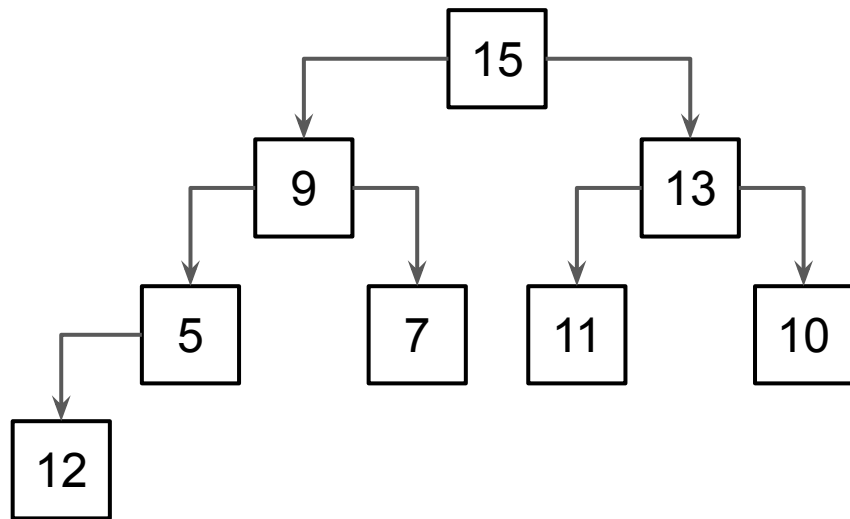
Структура данных: невозрастающая пирамида



15	9	13	5	7	11	10
----	---	----	---	---	----	----

`insert(12, val):`

Структура данных: невозрастающая пирамида

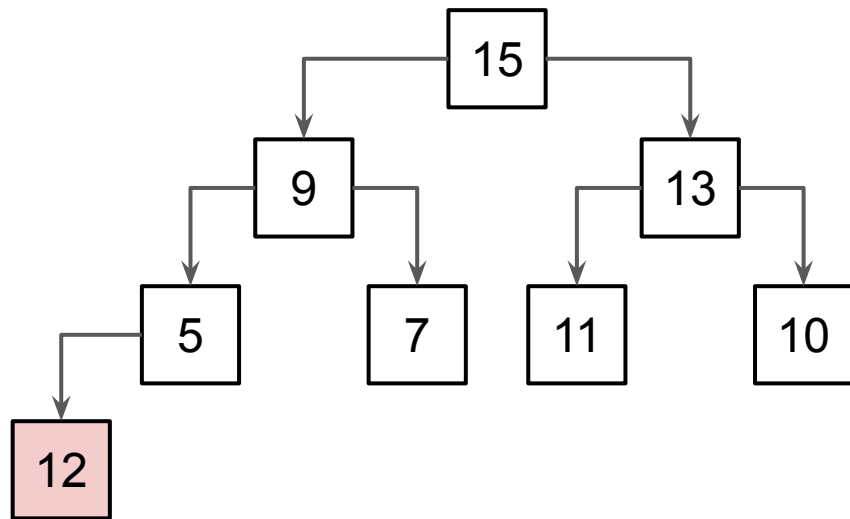


15	9	13	5	7	11	10	12
----	---	----	---	---	----	----	----

`insert(12, val):`

- 1) добавляем новый элемент в конец массива

Структура данных: невозрастающая пирамида

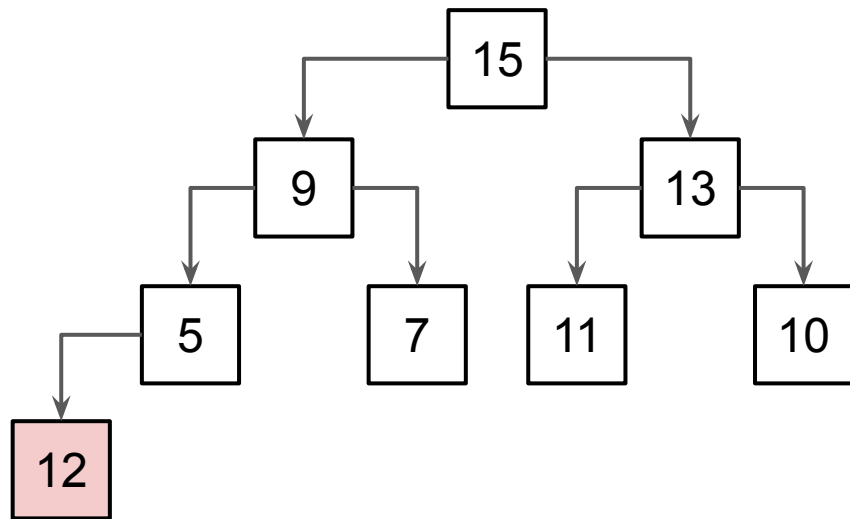


15	9	13	5	7	11	10	12
----	---	----	---	---	----	----	----

`insert(12, val):`

- 1) добавляем новый элемент в конец массива
- 2) получаем сломанную пирамиду!

Структура данных: невозрастающая пирамида

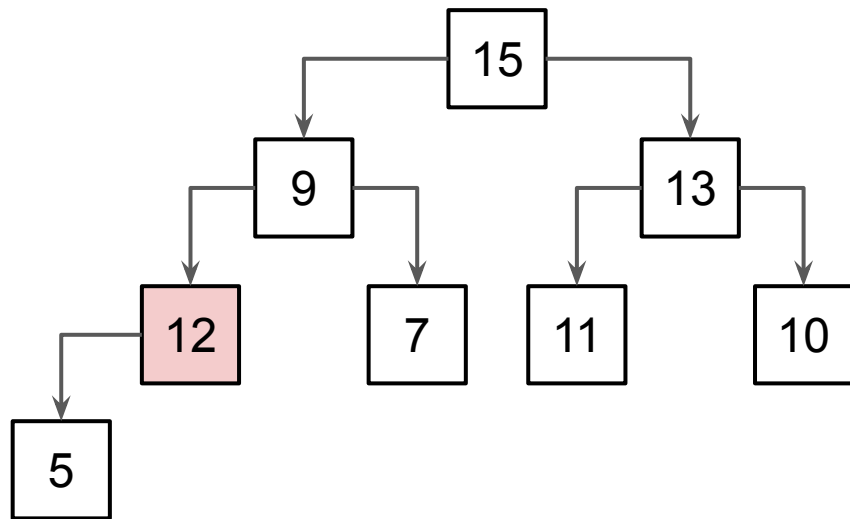


15	9	13	5	7	11	10	12
----	---	----	---	---	----	----	----

`insert(12, val):`

- 1) добавляем новый элемент в конец массива
- 2) получаем сломанную пирамиду!
- 3) просеиваем вверх

Структура данных: невозрастающая пирамида

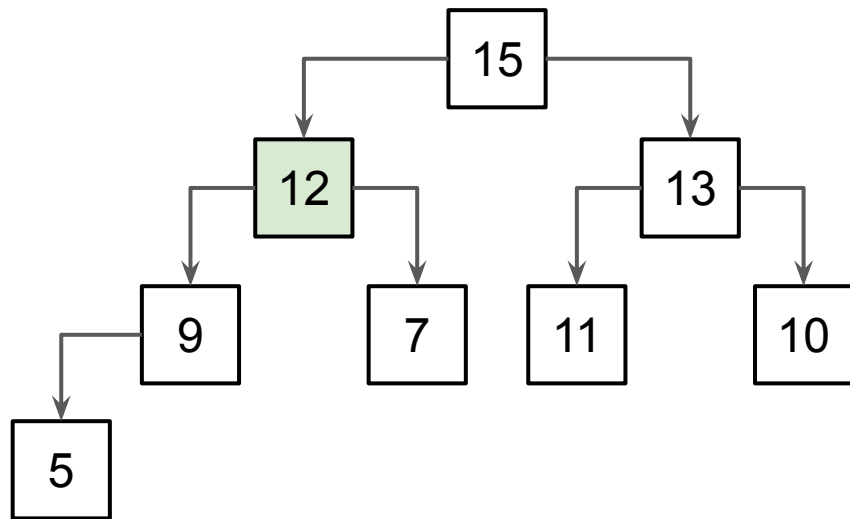


15	9	13	12	7	11	10	5
----	---	----	----	---	----	----	---

`insert(12, val):`

- 1) добавляем новый элемент в конец массива
- 2) получаем сломанную пирамиду!
- 3) просеиваем вверх

Структура данных: невозрастающая пирамида



15	12	13	9	7	11	10	5
----	----	----	---	---	----	----	---

`insert(12, val):`

- 1) добавляем новый элемент в конец массива
- 2) получаем сломанную пирамиду!
- 3) просеиваем вверх

Очередь с приоритетами через пирамиду

Множество значений: пары `<priority: int, value: T>`



Будем хранить массив из пар `<priority, value>`
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. `insert(priority, value)`
2. `peek_max()` $\rightarrow T$
3. `extract_max()` $\rightarrow T$

`insert`? Процедура, работающая за $O(\log N)$

Очередь с приоритетами через пирамиду

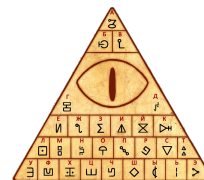
Множество значений: пары `<priority: int, value: T>`



Будем хранить массив из пар `<priority, value>`
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. `insert(priority, value)` $O(\log N)$
2. `peek_max()` $\rightarrow T$ $O(1)$
3. `extract_max()` $\rightarrow T$ $O(\log N)$



Мини-задача #19 (2 балла)

Пусть есть k отсортированных связанных списков.

Написать программу, строящую один отсортированный связный список из всех элементов.

<https://leetcode.com/problems/merge-k-sorted-lists/>

Для решения используйте очередь с приоритетами.

Альтернативные решения приветствуются, но не отменяют необходимость продемонстрировать решение через priority queue.

Мини-задача #20 (1 балл)

Реализовать стек, который бы поддерживал одну дополнительную операцию:

`get_min()` -> T: возвращение минимального элемента в стеке
(без изменения стека)

Новая операция (как и все старые) должна работать за $O(1)$

<https://leetcode.com/problems/min-stack/>

Пирамидальная сортировка

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

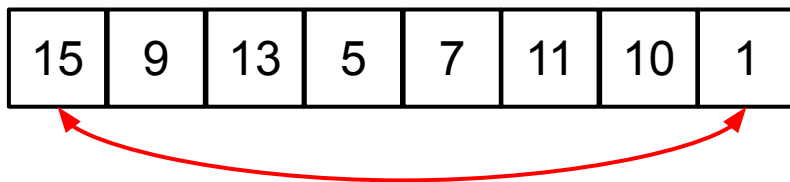
Допустим, у нас есть пирамида. Как ее отсортировать?

15	9	13	5	7	11	10	1
----	---	----	---	---	----	----	---

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Допустим, у нас есть пирамида. Как ее отсортировать?



1. Знаем, где должен стоять первый элемент!

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Допустим, у нас есть пирамида. Как ее отсортировать?

1	9	13	5	7	11	10	15
---	---	----	---	---	----	----	----

1. Знаем, где должен стоять первый элемент!
2. Меняем

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Допустим, у нас есть пирамида. Как ее отсортировать?

1	9	13	5	7	11	10	15
---	---	----	---	---	----	----	----

1. Знаем, где должен стоять первый элемент!
2. Меняем, получаем сломанную пирамиду (размера $N-1$).

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Допустим, у нас есть пирамида. Как ее отсортировать?

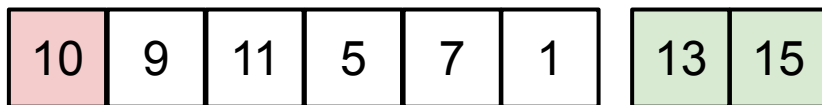
13	9	11	5	7	1	10	15
----	---	----	---	---	---	----	----

1. Знаем, где должен стоять первый элемент!
2. Меняем, получаем сломанную пирамиду (размера $N-1$).
3. Чиним за $O(\log N)$

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Допустим, у нас есть пирамида. Как ее отсортировать?



1. Знаем, где должен стоять первый элемент!
2. Меняем, получаем сломанную пирамиду (размера $N-1$).
3. Чиним за $O(\log N)$
4. Повторяем, пока пирамида не кончится

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Допустим, у нас есть пирамида. Как ее отсортировать?

11	9	10	5	7	1	13	15
----	---	----	---	---	---	----	----

1. Знаем, где должен стоять первый элемент!
2. Меняем, получаем сломанную пирамиду (размера $N-1$).
3. Чиним за $O(\log N)$
4. Повторяем, пока пирамида не кончится

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Допустим, у нас есть пирамида. Как ее отсортировать?

1	5	7	9	10	11	13	15
---	---	---	---	----	----	----	----

1. Знаем, где должен стоять первый элемент!
2. Меняем, получаем сломанную пирамиду (размера $N-1$).
3. Чиним за $O(\log N)$
4. Повторяем, пока пирамида не кончится

Пирамидальная сортировка

Допустим, у нас есть пирамида. Как ее отсортировать?

1	5	7	9	10	11	13	15
---	---	---	---	----	----	----	----

1. Знаем, где должен стоять первый элемент!
2. Меняем, получаем сломанную пирамиду (размера $N-1$).
3. Чиним за $O(\log N)$
4. Повторяем, пока пирамида не кончится

Пирамидальная сортировка

Допустим, у нас есть пирамида. Как ее отсортировать?

1	5	7	9	10	11	13	15
---	---	---	---	----	----	----	----

1. Знаем, где должен стоять первый элемент!
2. Меняем, получаем сломанную пирамиду (размера $N-1$).
3. Чиним за $O(\log N)$
4. Повторяем, пока пирамида не кончится

Сложность?

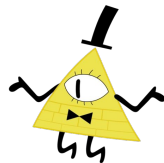
Пирамидальная сортировка

Допустим, у нас есть пирамида. Как ее отсортировать?

1	5	7	9	10	11	13	15
---	---	---	---	----	----	----	----

1. Знаем, где должен стоять первый элемент!
2. Меняем, получаем сломанную пирамиду (размера $N-1$).
3. Чиним за $O(\log N)$
4. Повторяем, пока пирамида не кончится

Сложность? $O(N * \log N)$



Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Проблема: не любой массив является пирамидой

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Проблема: не любой массив является пирамидой

Решение: нужно преобразовать произвольный массив к виду пирамиды

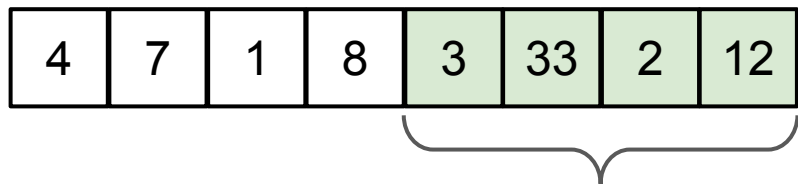
Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды

4	7	1	8	3	33	2	12
---	---	---	---	---	----	---	----

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



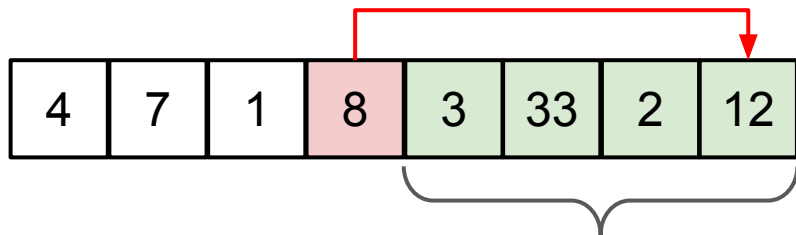
не нарушают условие
пирамиды

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



не нарушают условие пирамиды

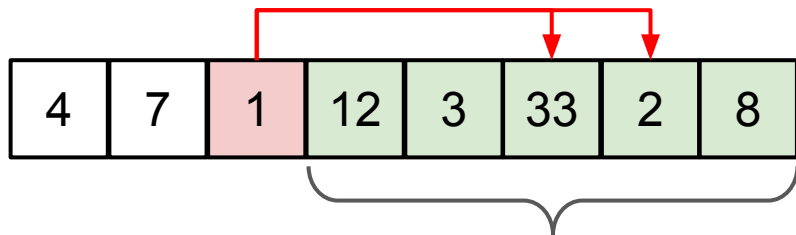
8 нарушает => просеиваем

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



не нарушают условие пирамиды

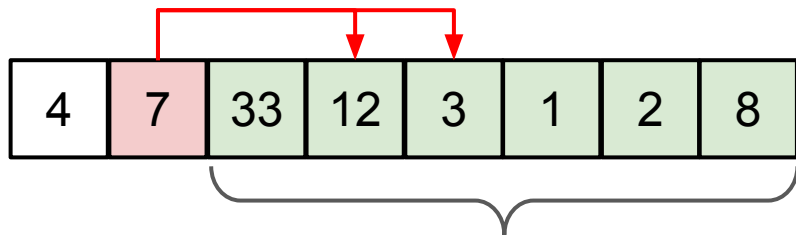
1 нарушает => просеиваем

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



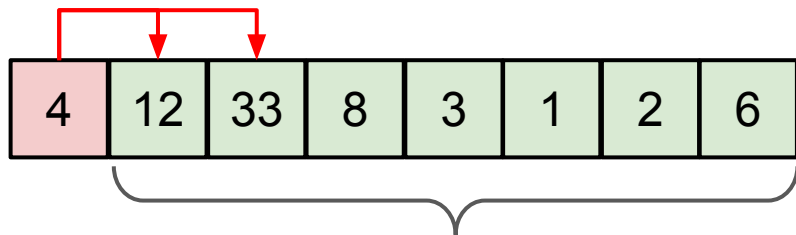
7 нарушает => просеиваем

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



не нарушают условие
пирамиды

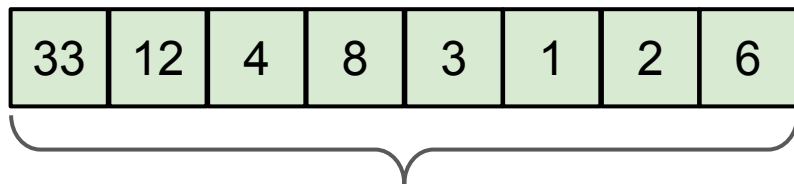
4 нарушает => просеиваем

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



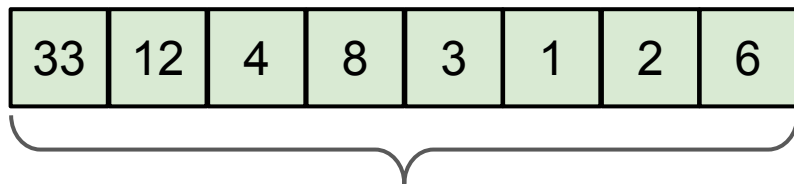
не нарушают условие
пирамиды

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



не нарушают условие
пирамиды

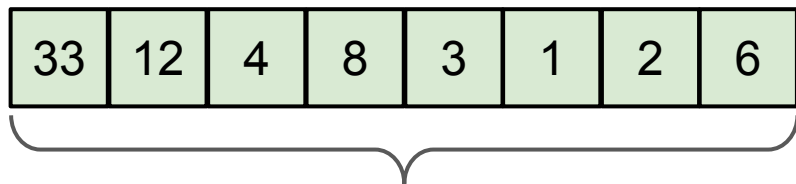
Сложность?

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



не нарушают условие
пирамиды

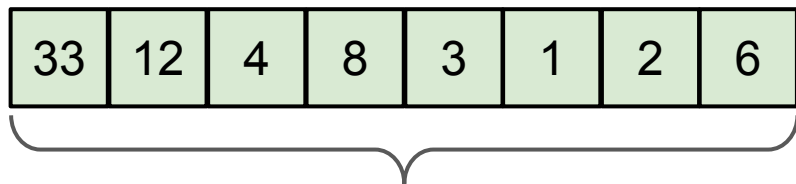
$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Сложность? $O(N * \log N)$

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



не нарушают условие
пирамиды

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

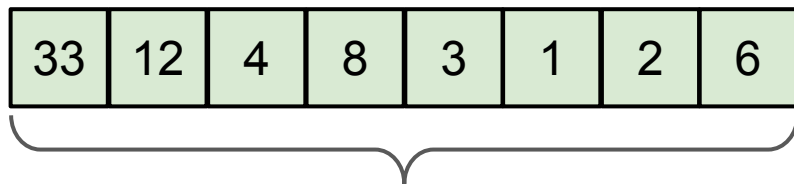
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Сложность? $O(N * \log N)$

На самом деле даже лучше!

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



не нарушают условие
пирамиды

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

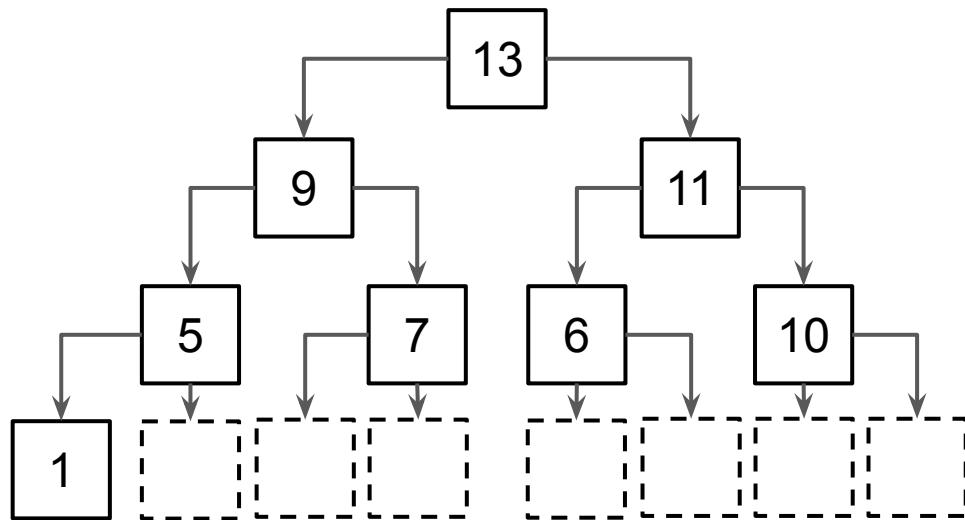
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Сложность? $O(N * \log N)$

На самом деле даже лучше! **Интуиция**: просеивание работает быстрее для **НИЗКИХ** узлов, а таких узлов большинство.

Факты о пирамидах

Структура данных: невозрастающая пирамида



Худший случай - когда куча является полным бинарным деревом (и просеивать нужно до конца)

Сколько тогда будет смены элементов?

Высота дерева!

Как связаны высота полного бинарного дерева и количество элементов в нем?

$$2^h = \frac{N+1}{2}$$

Факты о пирамидах (упражнения)

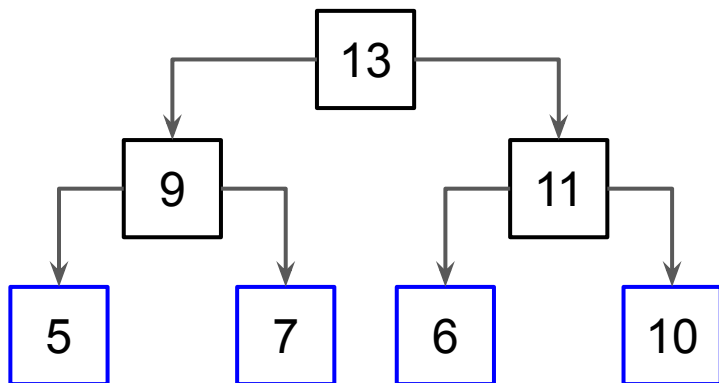
1. h - высота, N - количество вершин $\Rightarrow h = \lfloor \log N \rfloor$

Факты о пирамидах (упражнения)

1. h - высота, N - количество вершин $\Rightarrow h = \lfloor \log N \rfloor$
2. пусть l - количество вершин на высоте h

Факты о пирамидах (упражнения)

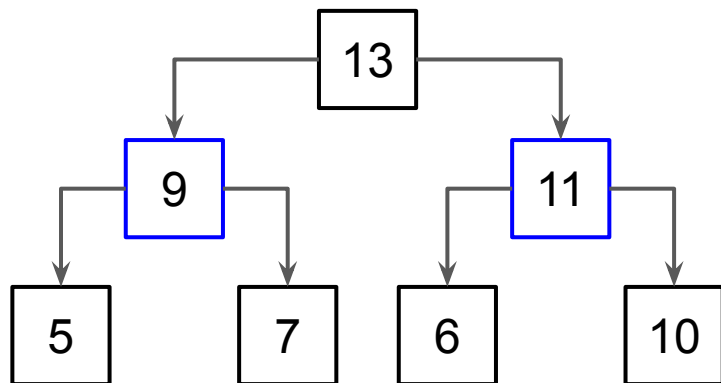
1. h - высота, N - количество вершин $\Rightarrow h = \lfloor \log N \rfloor$
2. пусть l - количество вершин на высоте h



на высоте 0

Факты о пирамидах (упражнения)

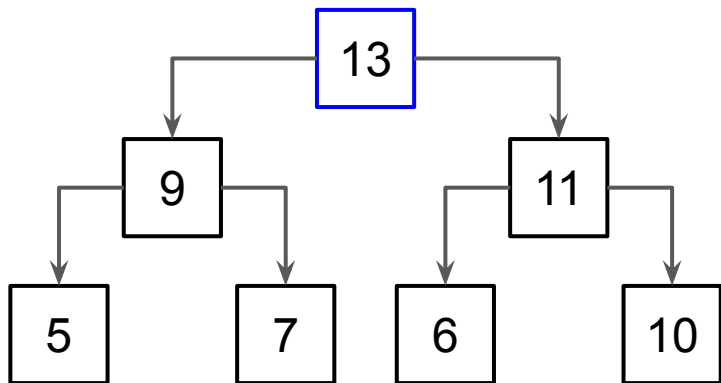
1. h - высота, N - количество вершин $\Rightarrow h = \lfloor \log N \rfloor$
2. пусть l - количество вершин на высоте h



на высоте 1

Факты о пирамидах (упражнения)

1. h - высота, N - количество вершин $\Rightarrow h = \lfloor \log N \rfloor$
2. пусть l - количество вершин на высоте h



на высоте 2

Факты о пирамидах (упражнения)

1. h - высота, N - количество вершин $\Rightarrow h = \lfloor \log N \rfloor$
2. пусть l - количество вершин на высоте h , тогда

$$l \leq \frac{2^{\lfloor \log N \rfloor}}{2^h} \leq \frac{N}{2^h}$$

Факты о пирамидах (упражнения)

1. h - высота, N - количество вершин $\Rightarrow h = \lfloor \log N \rfloor$

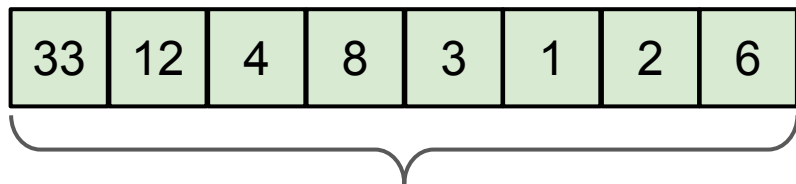
2. пусть l - количество вершин на высоте h , тогда

$$l \leq \frac{2^{\lfloor \log N \rfloor}}{2^h} \leq \frac{N}{2^h}$$

Используя эти факты, оценим количество операций при построении кучи.

Пирамидальная сортировка

Нужно преобразовать произвольный массив к виду пирамиды



не нарушают условие
пирамиды

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

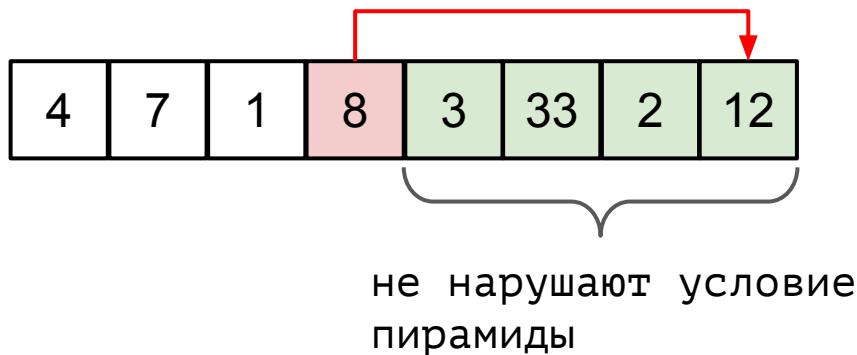
$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Сложность? $O(N * \log N)$

На самом деле даже лучше! **Интуиция**: просеивание работает быстрее для **НИЗКИХ** узлов, а таких узлов большинство.

Построение пирамиды

Рассмотрим построение так: идем по уровням от **кроны**, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня.



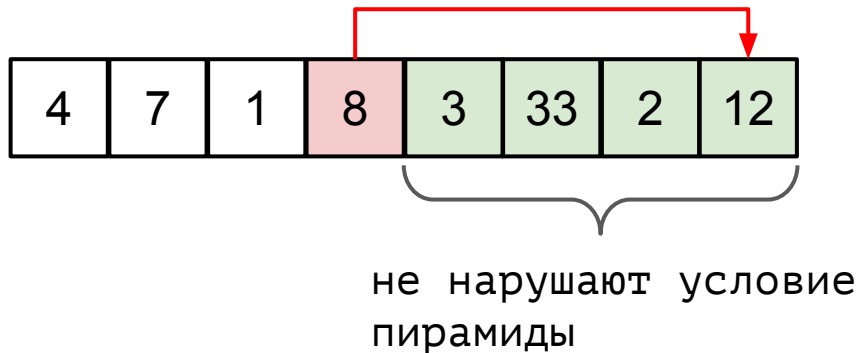
$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Построение пирамиды

Рассмотрим построение так: идем по уровням от **кроны**, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня. Тогда работу можно оценить:

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{N}{2^h} * O(h)$$



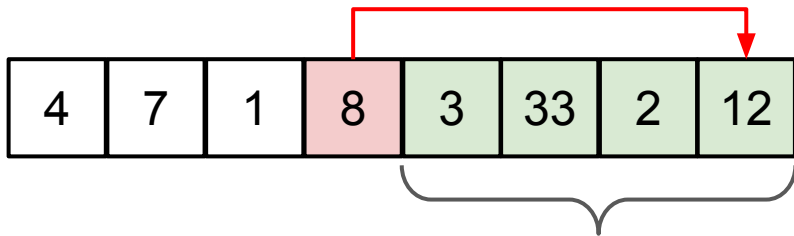
$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Построение пирамиды

Рассмотрим построение так: идем по уровням от **кроны**, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня. Тогда работу можно оценить:

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{N}{2^h} * O(h) = O(N * \sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h})$$



не нарушают условие
пирамиды

$$A[i] \geq A[2i + 1] \quad \forall i : 2i + 1 < N,$$

$$A[i] \geq A[2i + 2] \quad \forall i : 2i + 2 < N.$$

Построение пирамиды

Рассмотрим построение так: идем по уровням от **кроны**, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня. Тогда работу можно оценить:

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{N}{2^h} * O(h) = O(N * \sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h})$$

Построение пирамиды

Рассмотрим построение так: идем по уровням от **кроны**, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня. Тогда работу можно оценить:

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{N}{2^h} * O(h) = O(N * \sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h})$$

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{\infty} \frac{h}{2^h}$$

Построение пирамиды

Рассмотрим построение так: идем по уровням от **кроны**, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня. Тогда работу можно оценить:

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{N}{2^h} * O(h) = O(N * \sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h})$$

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{\infty} \frac{h}{2^h}$$

$$|x| < 1 \Rightarrow \sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

Построение пирамиды

Рассмотрим построение так: идем по уровням от **кроны**, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня. Тогда работу можно оценить:

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{N}{2^h} * O(h) = O(N * \sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h})$$

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{\infty} \frac{h}{2^h}$$

$$|x| < 1 \Rightarrow \sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \Rightarrow \begin{array}{l} \text{дифференцируем и} \\ \text{умножаем на } x \end{array} \Rightarrow \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Построение пирамиды

Рассмотрим построение так: идем по уровням от кроны, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня. Тогда работу можно оценить:

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{N}{2^h} * O(h) = O(N * \sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h})$$

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{\infty} \frac{h}{2^h} \Rightarrow \text{подставляем в формулу ниже } 1/2 \Rightarrow \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

$$|x| < 1 \Rightarrow \sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \Rightarrow \text{дифференцируем и умножаем на } x \Rightarrow \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Построение пирамиды

Рассмотрим построение так: идем по уровням от **кроны**, просеиваем каждый элемент вниз, тратим на это $O(h)$ - где h - высота этого уровня. Тогда работу можно оценить:

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{N}{2^h} * O(h) = O(N * \sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h}) = O(N) \quad \square$$

$$\sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{\infty} \frac{h}{2^h} \Rightarrow \text{подставляем в формулу ниже } 1/2 \Rightarrow \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

$$|x| < 1 \Rightarrow \sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \Rightarrow \text{дифференцируем и умножаем на } x \Rightarrow \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за $O(N * \log N)$

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$
2. Сортируем пирамиду за $O(N * \log N)$

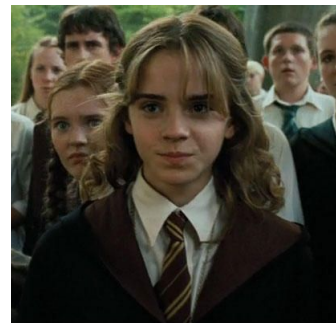
Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$
2. Сортируем пирамиду за $O(N * \log N)$

Общая сложность: $O(N * \log N)$



Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$
2. Сортируем пирамиду за $O(N * \log N)$

Общая сложность: $O(N * \log N)$

In-place?



Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$
2. Сортируем пирамиду за $O(N * \log N)$

Общая сложность: $O(N * \log N)$

In-place: **да**



Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

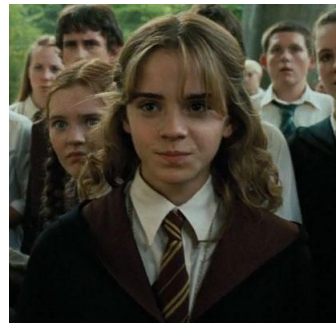
Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$
2. Сортируем пирамиду за $O(N * \log N)$

Общая сложность: $O(N * \log N)$

In-place: **да**

Стабильность?



Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$
2. Сортируем пирамиду за $O(N * \log N)$

Общая сложность: $O(N * \log N)$

In-place: **да**

Стабильность: **нет**

Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$
2. Сортируем пирамиду за $O(N * \log N)$

Общая сложность: $O(N * \log N)$

In-place: **да**

Стабильность: **нет**



Пирамидальная сортировка

Идея: нельзя ли приспособить пирамиды для сортировки?

Алгоритм пирамидальной сортировки:

1. Приводим массив к виду пирамиды за ~~$O(N * \log N)$~~ $O(N)$
2. Сортируем пирамиду за $O(N * \log N)$

Общая сложность: $O(N * \log N)$, но константы так себе

In-place: **да**

(merge-sort с заготовленным буфером работает лучше)

Стабильность: **нет**

Takeaways

- Разница между абстрактным типом данных и структурой данных

Takeaways

- Разница между абстрактным типом данных и структурой данных
- Стеки и очереди, как рабочие лошадки для решения задач и реализации других алгоритмов

Takeaways

- Разница между абстрактным типом данных и структурой данных
- Стеки и очереди, как рабочие лошадки для решения задач и реализации других алгоритмов
- Пирамиды для очередей с приоритетами и пирамидальной сортировки

Мини-задача #18 (2 балла)

Реализовать алгоритм сортировочной станции для преобразования выражения в инфиксной нотации к обратной польской записи.

Приоритеты и ассоциативность операторов взять отсюда:

https://en.cppreference.com/w/c/language/operator_precedence

В алгоритме поддержать:

1. Базовые арифметические, битовые и логические операторы
2. Подвыражения в скобках

Мини-задача #19 (2 балла)

Пусть есть k отсортированных связных списков.

Написать программу, строящую один отсортированный связный список из всех элементов.

<https://leetcode.com/problems/merge-k-sorted-lists/>

Для решения используйте очередь с приоритетами.

Альтернативные решения приветствуются, но не отменяют необходимость продемонстрировать решение через priority queue.

Мини-задача #20 (1 балл)

Реализовать стек, который бы поддерживал одну дополнительную операцию:

`get_min()` -> T: возвращение минимального элемента в стеке
(без изменения стека)

Новая операция (как и все старые) должна работать за $O(1)$

<https://leetcode.com/problems/min-stack/>