Algorithms and Data Structures

Master's ITMO, 2024

Theoretical tasks

Plan:

Section 1. Асимптотические оценки алгоритмов.

```
# функция min_search для поиска минимума в массиве arr
def min_search(arr: int[]):
    ans: int = MAX_INT
    for i: int = 0...len(arr)-1:
        if arr[i] < ans:
            ans = arr[i]
    return ans
```

Listing 1: поиск минимума в массиве

Конспект

▷ что такое алгоритм?

- ⋆ это конечная последовательность действий для абстрактного вычислителя
- ⋆ алгоритм получает на вход данные `input` и после завершения работы выдает данные `output`

▷ что такое RAM-модель?

- ⋆ грубо говоря, это очень упрощенная модель компьютера
- ⋆ есть память с произвольным доступом — RAM, Random Access Memory
- ⋆ RAM представляет собой последовательный набор ячеек, где у каждой ячейки есть свой адрес
- ⋆ за одно действие, можем обратится к одной ячейке памяти по ее адресу, для выполнения операции чтения/записи
- ⋆ за одно действие, можем выполнить любую арифметическую/логическую операцию с двумя ячейками

▷ чем отличается реальный компьютер?

- ⋆ про язык ассемблера можно почитать [здесь]
- ⋆ арифметические инструкции (например `ADD`) работают примерно за один такт процессора, в то время как инструкции работы с оперативной памятью (например `MOV`) работают за несколько сотен тактов
- ⋆ для оптимизации процессор *кеширует* данные: при обращении в оперативную память к ячейке с адресом i, процессор заодно выгружает из оперативной памяти несколько соседних с i-й ячеек — это называется *кеш-линией* — и запоминает их в *кеш-память*, которая находится прямо в процессоре и очень быстрая; теперь, если мы захотим обратится в оперативную память к ячейке с адресом $i + 1$, процессор не пойдет в оперативную память, а возьмет данные из *кеш*-а

▷ как оценить производительность алгоритма?

- ⋆ засекать время плохо — у разных компьютеров разная производительность
- ⋆ введем функцию $T(n)$ — количество абстрактных действий, которые делает наш алгоритм в RAM-модели, если на вход алгоритму подаются входные данные `input` и размер этих входных данных равен n

⋆ определим понятия *сложность алгоритма* и *время работы алгоритма* как функцию $T(n)$

⋆ на лекции мы посчитали, что время работы алгоритма [1] поиска минимума в массиве примерно равно $T(n) = 8 \cdot n + 3$ абстрактных операций

Def. Так как нас интересует только *асимптотика* функции $T(n)$ (порядок роста функции $T(n)$ относительно размера n входных данных), мы сказали, что нам интересен только наибольший член этой функции, а так же неинтересны константы. Чтобы формально и коротко это записать, мы ввели следующие обозначения:

⋆ $T(n) = \mathcal{O}(f(n))$

⋆ $T(n) = \Omega(g(n))$

⋆ $T(n) = \Theta(h(n))$

В примерах выше, функции $f(n)$, $g(n)$ — это соответственно *оценка сверху* и *оценка снизу* для нашей функции $T(n)$, в то время как $h(n)$ — это *точная оценка*.

Def. Напомним используемые далее определения:

⋆ $f(n) = \mathcal{O}(g(n)) \;\equiv\; \exists n_0, c > 0 : \forall n \geqslant n_0 : f(n) \leqslant c \cdot g(n)$

⋆ $f(n) = \Omega(g(n)) \;\equiv\; \exists n_0, c > 0 : \forall n \geqslant n_0 : f(n) \geqslant c \cdot g(n)$

⋆ $f(n) = \Theta(g(n)) \;\equiv\; \exists n_0, c_1, c_2 > 0 : \forall n \geqslant n_0 :$
$$c_1 \cdot g(n) \leqslant f(n) \leqslant c_2 \cdot g(n)$$

Практика

☐ Для каждой из рассматриваемых далее функций $f(n)$ найдите наиболее компактно записываемую $g(n)$, такую что $f(n) = \Theta(g(n))$.

1. $f(n) = 7n^2 - 7(n-3)^2$
2. $f(n) = 5n + 2\sqrt[3]{n}$
3. $f(n) = 10(n+1)^2 + 3(n-2)$
4. $f(n) = \log(\sqrt{n}) + \sqrt{\log n}$
5. $f(n) = n \cdot 3^{n+1} + n^{10}$
6. $f(n) = \frac{10n^2+2}{7n-1}$
7. $f(n) = \log(2n \log n)$

☐ Докажите следующие соотношения по определению (подберите константы c и n_0).

8. $\log n = \Omega(20)$
9. $2^n = \mathcal{O}(3^n)$
10. $n(n-8) = \Omega\big(n^2\big)$
11. $3n + 2\sqrt{n} = \mathcal{O}(n \log n)$
12. $n! = \Omega(5^n)$

☐ **Операции с нотацией $\mathcal{O}$-большое**. Докажите, что:

13. если $f(n) = \mathcal{O}(h(n))$ и $g(n) = \mathcal{O}(h(n))$, то верно что $f(n) + g(n) = \mathcal{O}(h(n))$
14. $f(n) + g(n) = \mathcal{O}(\max(f(n), g(n)))$

☐ Для следующих пар функций $f(n)$ и $g(n)$ покажите, верно ли, что $f(n) = \mathcal{O}(g(n))$.

15. $f(n) = \log n$
    $g(n) = \sqrt{n}$
16. $f(n) = n!$
    $g(n) = 2^n$
17. $f(n) = n \log n$
    $g(n) = \log(n!)$

☐ Время работы некоторого алгоритма задано следующим рекуррентным соотношением. Найдите Θ-асимптотику времени работы этого алгоритма, *построив дерево рекурсивных вызовов*.

18. $T(n) = 2 \cdot T(n-1) + 1$
19. $T(n) = T(\frac{n}{2}) + n$
20. $T(n) = 3 \cdot T(\frac{n}{2}) + n$
21. $T(n) = T(\frac{n}{3}) + \log n$
22. $T(n) = T(n-3) + n^3$

☐ Время работы некоторого алгоритма задано следующим рекуррентным соотношением. Построив *дерево рекурсивных вызовов*, докажите, что:

23. если $T(n) = 2T\left(\frac{n}{2}\right) + n$, то $T(n) = \mathcal{O}(n \log n)$
24. если $T(n) = 2T\left(\frac{n}{2}\right) + 1$, то $T(n) = \mathcal{O}(n)$

☐ Гармонический ряд:

25. Докажите, что $\sum\limits_{t=1}^{n} \frac{1}{t} = \Omega(\log n)$.

☐ Для каждой из приведенных ниже программ найдите и аргументируйте точную $\mathcal{O}$-асимптотику времени ее работы.

26.
```
for i = 0..n:
    for j = 0..i:
        for k = 0..j:
            print(i, j, k)
```

27.
```
def f(n):
    if n < 2:
        return 1
    return f(n - 1) + 5 * f(n - 1)
```

28.
```
def f(n):
    if n < 2:
        return 1
    return 3 * f(n / 2)
```

29.
```
for i = 0..n:
    j = i
    while j > 0:
        j = j / 2
```

Section 2. Элементарные структуры данных. Амортизационный анализ.

Конспект-1

▷ какие элементарные структуры данных бывают?

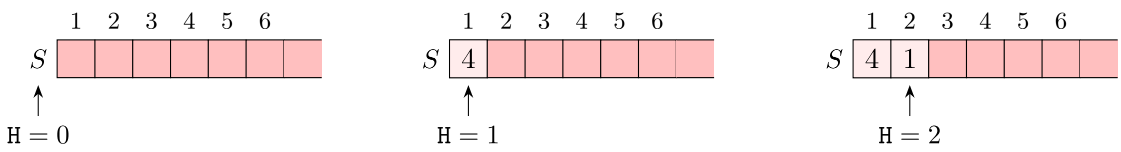
  ⋆ массив, связный список, стек, очередь

▷ массив (фиксированного размера n):

  ⋆ последовательно храним набор элементов в памяти
  ⋆ можем обратиться к i-му элементу за $\mathcal{O}(1)$
  ⋆ можем добавить/удалить элемент на позиции сразу после i-го элемента за $\mathcal{O}(n)$ (придется сдвинуть оставшиеся элементы массива вправо)

▷ стек (на массиве):

  ⋆ девиз: последним вошел — первым вышел (`LIFO`, last-in first-out)
  ⋆ храним указатель `H` (`HEAD`) на верхний элемент стека

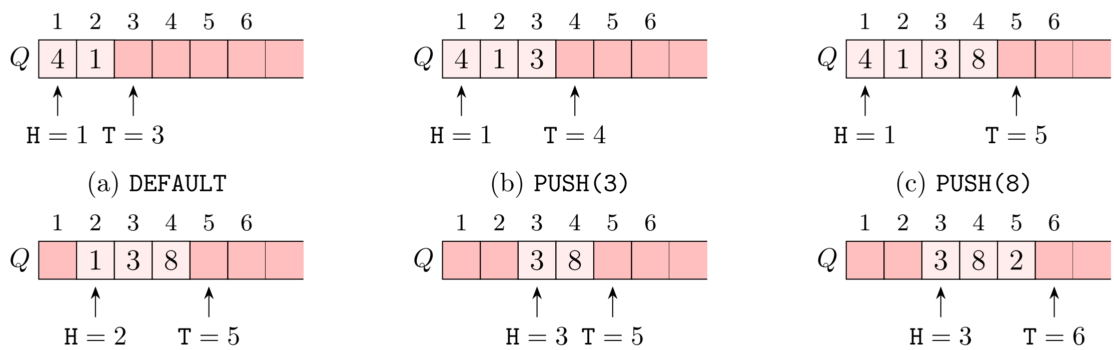


(a) `DEFAULT` (b) `PUSH(4)` (c) `PUSH(1)`
(d) `PUSH(3)` (e) `POP()` (f) `PUSH(8)`

Рис. 1: Операции со стеком

▷ очередь (на массиве):

  ⋆ девиз: первым вошел — первым вышел (`FIFO`, first-in first-out)
  ⋆ храним указатели `H` и `T` (`HEAD` и `TAIL`) на начало и конец очереди соответственно



(a) `DEFAULT` (b) `PUSH(3)` (c) `PUSH(8)`
(d) `POP()` (e) `POP()` (f) `PUSH(2)`

Рис. 2: Операции с очередью

Практика-1

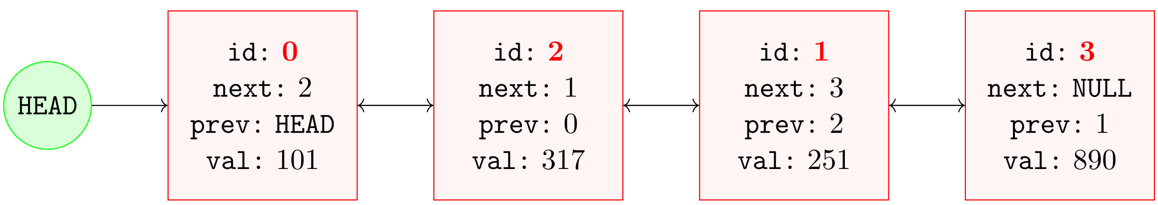
30. Придумайте модификацию стека, позволяющую за $\mathcal{O}(1)$ времени отвечать на запрос «вернуть сумму всех элементов в стеке».
31. Придумайте модификацию стека, позволяющую за $\mathcal{O}(1)$ времени отвечать на запрос «вернуть минимум среди всех элементов в стеке».
32. Придумайте модификацию очереди, позволяющую за $\mathcal{O}(1)$ времени отвечать на запрос «вернуть сумму всех элементов в очереди».
33. Придумайте модификацию очереди, позволяющую за $\mathcal{O}(1)$ времени отвечать на запрос «вернуть минимум среди всех элементов в очереди».
34. Придумайте модификацию очереди (при помощи нескольких стеков), которая поддерживает добавление и удаление элементов из обоих концов, то есть поддерживает следующие операции: `push_back()`, `push_front()`, `pop_back()`, `pop_front()`.

> **Def.** Скобочная последовательность называется *правильной*, если она может быть получена из некоторого арифметического выражения удалением всех не-скобочных символов.
>
> Например: `()[()]`, `[[[]{}]]` и `(()[]{})` — правильные скобочные последовательности, в то время как: `([)]`, `([{}]` и `()]` — неправильные.

35. Дана скобочная последовательность длины n из одного типа скобок: `()`. Определите, является ли она правильной. Время $\mathcal{O}(n)$.
36. Дана скобочная последовательность длины n из трех типов скобок: `()` `[]` и `{}`. Определите, является ли она правильной. Время $\mathcal{O}(n)$.
37. Дано арифметическое выражение в [*постфиксной записи*] длины n. Найдите результат вычисления этого выражения за $\mathcal{O}(n)$.
38. Дано арифметическое выражение в инфиксной записи со скобками (привычное нам арифметическое выражение). Найдите результат вычисления этого выражения за $\mathcal{O}(n)$.

Конспект-2

- ▷ связный список:
  - ⋆ объекты разбросаны по разным местам в памяти (не последовательно)
  - ⋆ для каждого объекта храним:
    - → `id` — уникальный адрес объекта
    - → `next` — адрес следующего за ним объекта
    - → `prev` — адрес предыдущего перед ним объекта
    - → `val` — значение
  - ⋆ храним ссылку `HEAD` на фиктивную голову связного списка
  - ⋆ обращение i-му элементу за $\mathcal{O}(n)$
  - ⋆ добавление/удаление элемента на i-й позиции за $\mathcal{O}(1)$ (при условии что нам дали ссылку на i-й элемент)



Практика-2

39. Как развернуть *односвязный список*, в котором у каждого объекта есть только ссылка `next` и нет ссылки `prev`, за время $\mathcal{O}(n)$ с $\mathcal{O}(1)$ дополнительной памяти?

40. Дан набор из n элементов, в каждом есть ссылка на какой-то другой. Проверьте, правда ли эти элементы образуют один большой кольцевой список (менять ссылки нельзя). Время $\mathcal{O}(n)$, память $\mathcal{O}(1)$.

41. Дан набор из n элементов, в каждом есть ссылка на какой-то другой. Гарантируется, что среди них есть ровно один цикл. Найдите его длину. Время $\mathcal{O}(n)$, память $\mathcal{O}(1)$.

42. Дан набор из n элементов, в каждом есть ссылка на какой-то другой. Проверьте, образуют ли элементы один большой линейный список (начальный элемент неизвестен, менять ссылки нельзя). Время $\mathcal{O}(n)$, память $\mathcal{O}(1)$. Подсказка: вспомните `xor`.

43. Дан набор из n элементов, в каждом есть ссылка на следующий и предыдущий. Проверьте, правда ли эти элементы образуют несколько кольцевых списков (менять ссылки нельзя). Время $\mathcal{O}(n)$, память $\mathcal{O}(1)$.

44. Дан набор из n элементов, в каждом есть ссылка на следующий и предыдущий. Проверьте, правда ли эти элементы образуют несколько линейных списков и, если да, объедините в один большой (в любом порядке). Время $\mathcal{O}(n)$, память $O(1)$.

45. Слить два отсортированных односвязных списка в один за время $\mathcal{O}(n)$ с $\mathcal{O}(1)$ дополнительной памяти.

46. Отсортировать связный список за время $\mathcal{O}(n \log n)$ с $\mathcal{O}(1)$ дополнительной памяти.

47. **Битовый счетчик**. Рассмотрим битовый счетчик, хранящий число в двоичной системе счисления в виде массива бит (младший бит имеет индекс 0, старший $k-1$). Для счетчика реализована операция `increment`, увеличивающая его значение на 1.

    (a) Оцените худшее и лучшее время работы одной операции `increment`. Оцените среднее время работы операции, по-честному посчитав суммарное время работы.

    (b) Придумайте функцию потенциала Φ, позволяющую получить совпадающее с предыдущим пунктом амортизированное время работы `increment`.

    (c) Сохранится ли оценка на среднее время работы одной операции, если добавить операцию `decrement`, уменьшающую значение счетчика?

Section 3. Сортировки и два указателя.

```
def insert_sort(arr: int[]):
    for i: int = 0...len(arr)-1:
        j: int = i
        while j > 0 && a[j - 1] > a[j]:
            swap(a[j], a[j - 1])
            j--
```

Listing 2: Сортировка вставками

Практика

48. В отсортированный массив длины n в произвольное место вставили случайное число. Придумайте алгоритм, который за $\mathcal{O}(n)$ сортирует полученный массив.

49. Дано k отсортированных массивов суммарной длины n. Опишите функцию `merge`, которая сливает эти массивы в один отсортированный массив за время $\mathcal{O}(nk)$.

> **Def.** Сортировка называется *стабильной*, если она не меняет относительный порядок сортируемых элементов, имеющих одинаковые значения.

50. Для каждой из следующих сортировок покажите, является ли она стабильной. Если это зависит от реализации, покажите как именно реализовать стабильный вариант выбранной сортировки.
    (a) сортировка выбором
    (b) сортировка вставками
    (c) сортировка слиянием

> **Def.** *Перестановкой* размера n называется массив размера n, в котором каждое целое число от 1 до n встречается ровно по одному разу.

51. Постройте для произвольного n перестановку, на которой сортировка вставками делает наибольшее число операций `swap(a[i], a[j])` (обмен элементов массива местами). Сколько в точности обменов совершается? Единственна ли такая перестановка?

52. Даны два отсортированных по неубыванию массива a и b. Определите, есть ли в них одинаковые числа. Время $\mathcal{O}(n)$.

53. Даны два отсортированных по неубыванию массива a и b. Найдите такие i и j, что разница $|a_i - b_j|$ минимальна. Время $\mathcal{O}(n)$.

54. Даны два отсортированных по неубыванию массива a и b и число S. Найдите такие i и j, что сумма $a_i + b_j = S$. Время $\mathcal{O}(n)$.

55. Даны два отсортированных по неубыванию массива a и b. Найдите число пар (i, j), таких, что $a_i = b_j$. Время $\mathcal{O}(n)$.

56. Даны два отсортированных по неубыванию массива a и b. Найдите число пар (i, j), таких, что $a_i > b_j$. Время $\mathcal{O}(n)$.

57. Дан массив чисел длины n. Найдите в нем отрезок с максимальной суммой за $\mathcal{O}(n)$ времени.

58. Дан массив из целых чисел. Для каждого элемента найдите ближайший элемент слева, меньший его. Время $\mathcal{O}(n)$.

Def. Дан массив a. Пара (i, j), такая, что $i < j$ и $a_i > a_j$ называется *инверсией*.

59. Пусть в массиве длины n ровно k инверсий. Докажите, что сортировка вставками работает за $\mathcal{O}(n + k)$.

60. Дан массив. Найдите число инверсий в нем при помощи сортировки слиянием за $\mathcal{O}(n \log n)$.

Section 4. Быстрая сортировка.

```python
def quick_sort(arr: int[]):
    if len(arr) <= 1:
        return arr
    q = arr[random.randint(0, len(arr) - 1)]
    left = [x for x in arr if x < q]
    right = [x for x in arr if x >= q]
    return quick_sort(left) + quick_sort(right)
```

Listing 3: Быстрая сортировка с лекции

```python
def quick_sort(arr: int[], left: int, right: int):
    if left >= right:
        return
    q = arr[random.randint(left, right)]
    i = left
    j = right
    while i <= j:
        while arr[i] < q: i += 1
        while q < arr[j]: j -= 1
        if i <= j:
            swap(arr[i], arr[j])
            i += 1
            j -= 1
    quick_sort(arr, left, j)
    quick_sort(arr, i, right)
```

Listing 4: Быстрая сортировка in-place

Конспект

> **Def.** Пусть X — некоторая дискретная *случайная величина*, которая принимает одно из множества значений $\{x_1, x_2, x_3, \ldots\}$.

> **Def.** Вероятность, с которой случайная величина принимает значение x_i обозначим за p_i. Так как случайная величина гарантированно примет одно из своих значений, то:
>
> $$\sum_i p_i = 1$$

> **Def.** *Математическим ожиданием* $\mathbb{E}[X]$ случайной величины X называется взвешенное среднее значение:
>
> $$\mathbb{E}[X] = \sum_i p_i \cdot x_i$$

Def. Так как время работы рандомизированного алгоритма — случайная величина, определим его математическое ожидание:

$$T^*(n) = \mathbb{E}[T(n)]$$

На лекции мы ограничили математическое ожидание времени работы алгоритма быстрой сортировки:

$$T^*(n) \leqslant \frac{1}{3} \cdot \left(T^*\left(\frac{n}{3}\right) + T^*\left(\frac{2n}{3}\right) + n\right) + \frac{2}{3} \cdot \left(T^*(0) + T^*(n) + n\right)$$

полагая $T^*(0) = 0$, упростили:

$$T^*(n) \leqslant T^*\left(\frac{n}{3}\right) + T^*\left(\frac{2n}{3}\right) + 3n \tag{1}$$

Практика

☐ Рассмотрим листинги [3] и [4], на которых представлены две реализации быстрой сортировки:

61. Приведите пример массива на котором реализация быстрой сортировки с лекции (с разделением на две кучки: $< q$, $\geqslant q$) может работать бесконечно долго?
62. Корректна ли in-place реализация быстрой сортировки? Может ли она работать бесконечно долго? Сколько дополнительной памяти для нее требуется?
63. Каким образом происходит разделение массива на части in-place? Может ли опорный элемент q переместиться на другую позицию в процессе работы?
64. При помощи *метода математической индукции* докажите, что из уравнения [1] следует $T^*(n) = \mathcal{O}(n \log n)$.

☐ Приведите пример массива длины n из различных элементов, на котором быстрая сортировка отработает за $\Omega(n^2)$, если в качестве опорного элемента выбирается:

65. Самый левый элемент отрезка: `q = arr[left]`
66. Самый правый элемент отрезка: `q = arr[right]`
67. Центральный элемент отрезка: `q = arr[(left + right) / 2]`
68. Элемент по случайному индексу, однако последовательность случайных чисел известна заранее

☐ Как с помощью генератора случайных чисел получить случайную перестановку размера n? Каждая перестановка должна получаться с вероятностью $\frac{1}{n!}$.

69. Время $\mathcal{O}(n \log n)$.
70. Время $\mathcal{O}(n)$.

☐ Разные задачи на сортировки:

71. Есть массив. Нужно найти наименьшие k элементов массива (сортировать их между собой необязательно). За какое минимальное время это можно сделать? Подсказка: попробуйте запускать только одну ветку рекурсии `quick_sort`.
72. В отсортированном массиве размера n изменили k элементов (неизвестно, каких именно). Отсортируйте полученный массив за $\mathcal{O}(n + k \log k)$.
73. Дан массив из n чисел. Необходимо для каждого элемента найти число элементов, которые меньше его. Время работы $\mathcal{O}(n \log n)$.

74. Покажите, что любую сортировку можно сделать стабильной, не поменяв асимптотику работы и потратив $\mathcal{O}(n)$ дополнительной памяти.
75. У вас есть n болтов и n подходящих к ним гаек, все болты (и, соответственно, все гайки) разного диаметра. Посмотрев на два болта (или две гайки), сложно понять, какой больше, а какой меньше, поэтому единственная операция, которая у вас есть — взяв какой-то болт и какую-то гайку, сравнить их диаметры (если не пролезает — значит болт больше, если болтается — значит меньше). Найдите для каждого болта подходящую гайку за $\mathcal{O}(n \log n)$.

Section 5. Двоичный поиск.

```
# инварианты: arr[left] < x0 and arr[right] >= x0

def check(arr: int[], x0: int, mid: int):
    return arr[mid] >= x0

def binary_search(arr: int[], x0: int):
    left: int = -1
    right: int = arr.length
    while right - left > 1:
        mid: int = (left + right) / 2
        if check(arr, x0, mid):
            right = mid
        else:
            left = mid
    return right
```

Listing 5: Двоичный поиск

Практика

76. Дан отсортированный массив. Отвечать на запросы: сколько элементов равны x?

77. Дан массив положительных чисел. После предподсчета за $\mathcal{O}(n)$, отвечать за $\mathcal{O}(\log n)$ на запросы: какое максимальное число элементов из начала массива можно взять, чтобы их сумма была не больше x?

Def. *Циклическим сдвигом* массива `arr` *вправо* на величину d называется массив, получаемый из исходного переносом последних d элементов в начало:

$$\mathtt{arr}^{\to d} = [\mathtt{arr}_{n-d}, \mathtt{arr}_{n-d+1}, \ldots, \mathtt{arr}_{n-1}, \mathtt{arr}_0, \mathtt{arr}_1, \ldots, \mathtt{arr}_{n-d-1}]$$

78. Задан массив, полученный циклическим сдвигом из отсортированного по возрастанию. Все элементы массива различны. Требуется за $\mathcal{O}(\log n)$ найти в нем заданный элемент.

79. Пусть в предыдущей задаче убрали условие, что все элементы массива различны. Можно ли в таком массиве найти заданный элемент за $\mathcal{O}(\log n)$?

80. Задан массив, полученный приписыванием одного отсортированного по возрастанию массива в конец другому отсортированному по возрастанию. Все элементы массива различны. Требуется за $\mathcal{O}(\log n)$ найти в нем заданный элемент.

81. Задан массив, полученный приписыванием отсортированного по убыванию массива в конец отсортированному по возрастанию. Все элементы массива различны. Требуется за $\mathcal{O}(\log n)$ найти в нем заданный элемент.

82. Задан массив, полученный приписыванием отсортированного по убыванию массива в конец отсортированному по возрастанию и затем циклическим сдвигом получившегося массива. Все элементы массива различны. Требуется за $\mathcal{O}(\log n)$ найти в нем заданный элемент.

83. Есть n кучек предметов, в i-й кучке a_i предметов. Все предметы пронумерованы сквозной нумерацией, так, что в кучке с меньшим номером номера предметов меньше. За $\mathcal{O}(\log n)$ отвечать на запросы: в какой кучке находится предмет номер x?

84. В выборах участвуют n кандидатов. По последним опросам, за кандидата i готовы проголосовать a_i избирателей. Вы хотите, чтобы ваш кандидат победил (набрал больше голосов, чем любой другой кандидат). За s рублей можно изменить мнение одного избирателя. Сколько надо потратить денег на такую избирательную компанию?

85. Есть отсортированный массив из n чисел. Нужно выбрать из них k так, чтобы минимальная разница между выбранными числами была как можно больше.

86. Дан массив чисел длины n. Известно, что каждый элемент массива по абсолютному значению не превосходит C. Нужно выбрать k пар из элементов массива (каждый элемент может быть только в одной паре), так, чтобы максимальная разность чисел в паре была как можно меньше. Время $\mathcal{O}(n \log n + n \log C)$.

87. На прямой стоят n котят. Котенок номер i стоит в точке x_i и может двигаться со скоростью не больше v_i. За какое время все котята могут собраться в одной точке?

88. Домик черепашки расположен в точке 0 числовой прямой. Однажды она узнала, что скоро вырастут n одуванчиков, одуванчик i вырастет в точке x_i во время t_i, причем все x_i и t_i положительные, а так же $x_{i+1} > x_i$ и $t_{i+1} > t_i$. Черепашка выходит из дома в момент 0, она движется со скоростью не больше v. Чтобы съесть одуванчик, ей нужно остановиться около него на время d. За какое время она сможет съесть все одуванчики и вернуться домой?

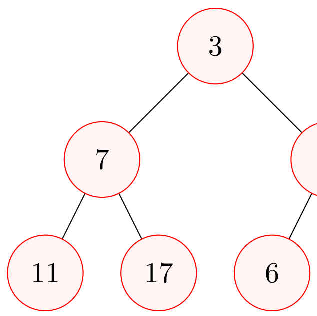
89. Даны n кенгуру с сумками. У каждого кенгуру есть размер (целое число). Кенгуру может поместиться в сумке другого кенгуру тогда и только тогда, когда размер кенгуру-носителя как минимум в два раза больше размера кенгуру-пассажира. Каждый кенгуру может нести не более одного кенгуру, а кенгуру-пассажир не может носить никаких кенгуру. Кенгуру-пассажира не видно, когда он в сумке кенгуру-носителя. Пожалуйста, разработайте такой план рассадки кенгуру, чтобы было видно как можно меньше кенгуру.

Section 6. Двоичная куча.

Конспект

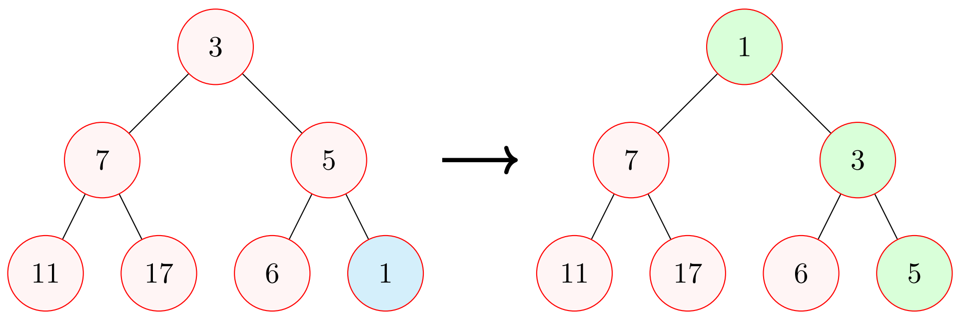
▷ Свойства двоичной кучи:

- ⋆ Подвешенное двоичное дерево
- ⋆ На i-м слое 2^i вершин (кроме последнего слоя)
- ⋆ Последний слой заполнен слева-направо «без пробелов»
- ⋆ Верно одно из двух:
  - → на всех ребрах написано отношение '$\leqslant$' (*куча по минимуму*)
  - → на всех ребрах написано отношение '$\geqslant$' (*куча по максимуму*)

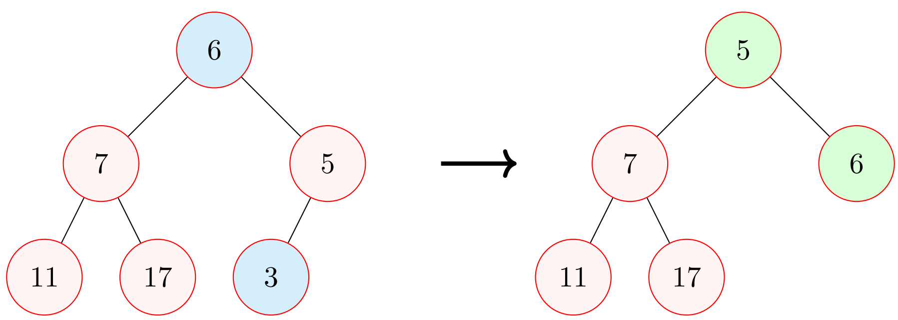


▷ Операции с двоичной кучей (*по минимуму*):

- ⋆ `siftUp(`x`)` — просеять элемент x вверх по дереву
- ⋆ `siftDown(`x`)` — просеять элемент x вниз по дереву
- ⋆ `add(`x`)` — добавить элемент x в кучу (добавляем x последней вершиной на последнем слое, а затем вызываем `siftUp(`x`)`)



- ⋆ `extractMin()` — извлечь текущий минимальный элемент из кучи (свапаем корень с последней вершиной на последнем слое, удаляем последнюю вершину на последнем слое, а затем вызываем `siftDown(root)`)



Практика

90. Пусть в куче лежат числа от 1 до 1023, по одному разу каждое. Какое минимальное число может лежать в куче на самом нижнем слое?

91. Пусть в двоичной куче лежит n элементов. Сколько листьев у соответствующего дерева?

92. Пусть дерево кучи организовано таким образом, что у каждой вершины не два ребенка, а три. Какие номера будут у детей вершины i в этом случае?

93. Пусть дерево кучи организовано таким образом, что у каждой вершины d детей (при этом d – не константа, а параметр). За какое время работают основные операции над кучей в этом случае? Приведите оценку зависимости от n и d.

94. Как добавить в кучу операцию изменения ключа элемента?

95. Как из двух куч сделать структуру данных, которая одновременно может искать и удалять как максимум, так и минимум?

96. На базе двух куч постройте структуру данных, которая может находить и удалять медиану ($\frac{n}{2}$-й элемент в отсортированном порядке).

97. Есть k отсортированных массивов, содержащих в сумме n элементов. Слейте их в один отсортированный массив за время $\mathcal{O}(n \log k)$.

98. Пусть даны n элементов. Как построить на них кучу за $\mathcal{O}(n)$?

99. Даны две кучи с размерами n и m. Придумайте алгоритм, позволяющий слить данные кучи в одну кучу размера $n + m$ за $\mathcal{O}(n + m)$.

100. Докажите, что нельзя сделать операцию `extractMin()` быстрее, чем за $\Theta(\log n)$. Подсказка: на лекции мы разбирали что нельзя построить сортировку на сравнениях быстрее чем за $n \log n$.