Во всех домашних заданиях можно основываться на тех фактах, которые мы доказывали с вами на лекциях, все остальные факты доказывать. Исключения составляют достаточно известные и в то же время сложнодоказуемые математические факты, которые иногда могут приниматься на веру. Если вы не знаете, нужно ли какое-то конкретное утверждение (не имеющее отношение к нашей теме) доказывать, то задумайтесь о решении без этого факта. В большинстве заданий это можно сделать.

1 Неделя 1

1.1 Письменная часть

1. Докажите по определению: $\frac{n^3}{6} - 7n^2 = \Omega(n^3)$
2. Упорядочите функции так, чтобы, если $f(n)$ стоит раньше $g(n)$, то $f(n) = \mathcal{O}(g(n))$: $(\sqrt{2})^{\log n}$, n^2, $n!$, $(\log n)!$, $(\frac{3}{2})^2$, n^3, $\log^2 n$, $\log n!$, $\log\log n$, 2^{2^n}, $n^{\frac{1}{\log n}}$, 1, n, $(n+1)!$, $2^{2^{n+1}}$, e^n, $n\log n$, $(\log n)^{\log n}$, $2^{\log n}$, $\sqrt{\log n}$, $n^{\log\log n}$, $n \cdot 2^n$, $4^{\log n}$.
   P.S. Старайтесь, чтобы ваши формулы выглядели как можно более похоже на данные.
3. Докажите или опровергните, что $\log(n!) = \Theta(n \log n)$.
4. Решите рекурренту: найдите верхнюю и нижнюю границы ($\mathcal{O}$ и Ω), докажите по индукции. $T(n) = 2T(\frac{n}{2}) + \frac{n}{\log n}$.
5. Пусть у нас есть два целых числа, записанных в десятичной системе счисления: x и y. Пусть $x = a \cdot 10^k + b$ и $y = c \cdot 10^k + d$. Рассмотрим рекурсивный метод умножения этих чисел:

```
fun multiply(x, y):
  k = max(len(x), len(y)) / 2
  a = shiftRight(x, k)
  b = getLowest(x, k)
  c = shiftRight(y, k)
  d = getLowest(y, k)
  ac = multiply(a, c)
  bd = multiply(b, d)
  ad = multiply(a, d)
  cb = multiply(c, b)
  e = add(ad, cb)
  ans = shiftLeft(ac, 2 * k)
  ans = add(ans, bd)
  ans = add(ans, shiftLeft(e, k))
  return ans
```

   Заметим следующий факт:

$$(a \cdot 10^k + b) \cdot (c \cdot 10^k + d) = a \cdot c \cdot 10^{2k} + b \cdot d + (a \cdot d + b \cdot c) \cdot 10^k,$$

а так как $a \cdot d + c \cdot b = (a+b) \cdot (c+d) - a \cdot c - b \cdot d$, то

$$(a \cdot 10^k + b) \cdot (c \cdot 10^k + d) = a \cdot c \cdot 10^{2k} + b \cdot d + ((a+b) \cdot (c+d) - a \cdot c - b \cdot d) \cdot 10^k.$$

Рассмотрим второй метод умножения, основанный на этом факте:

```
fun multiply2(x, y):
  k = max(len(x), len(y)) / 2
  a = shiftRight(x, k)
  b = getLowest(x, k)
  c = shiftRight(y, k)
  d = getLowest(y, k)
  ac = multiply2(a, c)
  bd = multiply2(b, d)
  e = multiply2(add(a, b), add(c, d))
  e = subtract(e, ac)
  e = subtract(e, bd)
  ans = shiftLeft(ac, 2 * k)
  ans = add(ans, bd)
  ans = add(ans, shiftLeft(e, k))
  return ans
```

В нашем коде `ac` $= a \cdot c$, `bd` $= b \cdot d$, `e` $= a \cdot d + c \cdot b = (a+b) \cdot (c+d) - a \cdot c - b \cdot d$. Функции `len`, `shiftRight`, `shiftLeft`, `getLowest`, `add`, `subtract` работают за линейное от длины аргумента и длины результата время. Оцените время работы обоих алгоритмов, если считать, что их вызывают от двух чисел длины n.

2 Неделя 2

2.1 Письменная часть

1. Приращение бинарного счетчика. В качестве счетчика используется массив $a[0..k-1]$. Младший бит находится в элементе $a[0]$, а старший — в $a[k-1]$. В массиве хранится число $x = \sum_{i=0}^{k-1} a[i] \cdot 2^i$. Изначально массив заполнен нулями. Рассмотрим операцию:

```
increment():
  i = 0
  while i < k and a[i] = 1:
    a[i] = 0
    i++
  if i < k:
    a[i] = 1
```

(a) Посчитайте истинное время работы `increment` в худшем случае.

(b) Посчитайте среднее время работы `increment`.

Добавим операцию `decrement`:

```
decrement():
  i = 0
  while i < k and a[i] = 0:
    a[i] = 1
    i++
  if i < k:
    a[i] = 0
```

(c) Докажите, что если выполнять операции `increment` и `decrement`, то n операций в худшем случае выполнятся за $\Theta(nk)$.

(d) Рассмотрим приращение бинарного счетчика с операциями `increment` (из предыдущего задания), `setZero` и `get i`.

```
get(i):
  return a[i]

setZero():
  i = 0
  while i < k:
    a[i] = 0
    i++
```

Предложите, как модифицировать операции `increment`, `setZero` и `get i`, чтобы амортизированная стоимость операций была $\mathcal{O}(1)$, с использованием $\mathcal{O}(1)$ дополнительной памяти.

2. Проанализируйте стек на саморасширяющемся массиве, если при полном заполнении происходит увеличение в 2 раза, а при заполнении менее чем на $\frac{1}{4}$ — сужение в 2 раза, с помощью метода потенциалов. Потенциал должен зависеть только от текущего состояния стека (размера выделенного массива и числа заполненных элементов) и не должен зависеть от истории операций.

3. Использования памяти без инициализации. Задан массив $a[1..n]$. Требуется поддержать две операции: `set(i, x)` и `get(i)`:

- Операция `set` должна присваивать i-му элементу массива значение x
- Операция `get` должна возвращать последнее присвоенное i-му элементу значение, либо 0, если присвоения не было

Вы можете использовать $\mathcal{O}(n)$ памяти и выделять куски памяти, заполненные недетерминированными значениями. Выделение памяти работает за $\mathcal{O}(1)$. Операции `set` и `get` должны работать за $\mathcal{O}(1)$. Препроцессинг должен работать за $\mathcal{O}(1)$.

4. Предложите реализацию менеджера памяти, позволяющего выделять и возвращать блоки одинакового размера (размер больше машинного слова) за $\mathcal{O}(1)$ истинного времени и $\mathcal{O}(1)$ дополнительной памяти. Пояснение: вам изначально выдается большой кусок памяти, которую вам нужно выдавать по запросам аналогичным `malloc` и `free`.

5. Найдите в массиве все элементы, которые встречаются больше $\frac{n}{k}$ раз. Требуется найти их за $\mathcal{O}(nk)$ с $\mathcal{O}(k)$ дополнительной памяти.

6. Дайте амортизированную оценку следующей структуре данных с помощью метода потенциалов. Структура хранит массивы $a_0, a_1, \ldots, a_k$: каждый из массивов a_i либо пуст, либо содержит 2^i элементов. Также поддерживается число n — число добавленных элементов. Каждый из добавленных элементов находится ровно в одном массиве. Каждый из массивов упорядочен по неубыванию.

   - `add x` — добавить элемент x в множество:

```
add(x):
  v = [x]
  i = 0
  while a[i] is not empty:
    v = merge(v, a[i])
    a[i] = []
    i++
  a[i] = v
  n++
```

   - `contains x` — проверить, принадлежит ли x множеству:

```
contains(x):
  i = 0
  while i <= log(n):
    if binarysearch(a[i], x):
      return True
    i++
  return False
```

   Функция `merge(a, b)` сливает два упорядоченных массива a и b за $\mathcal{O}(|a| + |b|)$, а `binarysearch(a, x)` определяет, содержится ли элемент x в упорядоченном массиве a, за $\mathcal{O}(\log |a|)$. $|a|$ — длина массива a.

3 Неделя 3

3.1 Практика

Сортировка выбором:

```
for (i in 0 until n) {
  var min = i
  for (j in i + 1 until n) {
    if (a[j] < a[min]) {
      min = a[j]
    }
  }
  if (min != i) {
    val t = a[min]
    a[min] = a[i]
    a[i] = t
  }
}
```

Сортировка вставками:

```
for (i in 0 until n) {
  var j = i
  while (j > 0 && a[j] < a[j - 1]) {
    val t = a[j]
    a[j] = a[j - 1]
    a[j - 1] = t
    j--
  }
}
```

Сортировка пузырьком:

```
for (i in 0 until n) {
  for (j in 0 until n - i - 1) {
    if (a[j] < a[j + 1]) {
      val t = a[j]
      a[j] = a[j + 1]
      a[j + 1] = t
    }
  }
}
```

1. Посчитайте сколько сравнений делает на заданном массиве a за время $\mathcal{O}(n \log n)$:
   (a) сортировка выбором (все элементы в массиве различны);
   (b) сортировка вставками;
   (c) сортировка пузырьком.
2. Посчитайте сколько обменов делает на заданном массиве a за время $\mathcal{O}(n \log n)$:
   (a) сортировка вставками;
   (b) сортировка выбором;
   (c) сортировка пузырьком.
3. Посчитайте четность перестановки за $\mathcal{O}(n)$.
4. Постройте два массива длинами n и m, на которых операция `merge` выполняет как можно больше сравнений.
5. Постройте массив длины n, на котором сортировка слиянием выполняет как можно больше сравнений.
6. Посчитайте, сколько есть перестановок длины n, на которых сортировка слиянием выполняет как можно больше сравнений.

3.2 Письменная часть

1. Постройте для любого n перестановку, на которой классическая сортировка выбором делает максимальное число обменов. Сколько всего таких перестановок существует?
2. Постройте для любого n перестановку, для которой требуется выполнить все $n-1$ итераций сортировки пузырьком. Сколько всего таких перестановок существует?
3. Постройте для любого n биекцию между перестановками, на которых классическая сортировка выбором делает максимальное число обменов, и перестановками, для которых требуется выполнить все $n-1$ итераций сортировки пузырьком. Биекция и обратная ей функция должна вычисляться за время $\mathcal{O}(n)$.
4. Задан почти отсортированный массив длины n, каждый элемент отстоит от своей правильной позиции не более чем на k. Предложите алгоритм, который сортирует заданный массив за $\mathcal{O}(n \log k)$ (без использования сложных структур данных).
5. Задан массив a длины n и массив p длины m. Найдите p_i-ю порядковую статистику массива a для всех i за время $\mathcal{O}(n \log m + m)$.
6. Предложите алгоритм «разделяй-и-властвуй» с временем работы $\mathcal{O}(n \log n)$ для следующей задачи: задан массив из чисел и число k, определите число отрезков с суммой, не превышающей k.

4 Неделя 4

4.1 Практика

1. Задан массив, полученный циклическим сдвигом из отсортированного, как за $\mathcal{O}(\log n)$ времени найти в нем элемент x? Все элементы в массиве различны.
2. А как решить предыдущую задачу, если в массиве разрешены равные элементы.
3. 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.
4. 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.
5. Как в массиве найти любые два равных элемента, используя только сравнения?
   (a) Алгоритм за $\mathcal{O}(n^2)$ сравнений.
   (b) Алгоритм за $\mathcal{O}(n \log n)$ сравнений.
   (c) Докажите, что любой алгоритм делает $\Omega(n)$ сравнений.
   (d) Докажите, что любой алгоритм делает $\Omega(n \log n)$ сравнений.
6. Пусть выполняется целочисленный двоичный поиск с начальными значениями $L = 0$, $R = 2^k$. Какие пары (L, R) могут возникнуть в процессе двоичного поиска?
   (a) Предложите критерий для определения таких пар.
   (b) Посчитайте точно, сколько таких пар.
   (c) Предложите алгоритм определения за $\mathcal{O}(1)$ по заданным значениям L и R, могут ли они возникнуть в процессе двоичного поиска.
7. Пусть выполняется целочисленный двоичный поиск с начальными значениями $L = 0$, $R = n$. Предложите алгоритм определения за $\mathcal{O}(\log n)$ по заданным значениям L и R, могут ли они возникнуть в процессе двоичного поиска.
8. Задан массив длины n из чисел от 1 до n. Предложите модификацию алгоритма сортировки подсчетом, который сортирует данный массив за $\mathcal{O}(n)$, используя лишь $\mathcal{O}(\sqrt{n})$ дополнительной памяти.

4.2 Письменная часть (до 20 октября)

1. Задан отсортированный массив чисел a длины n и число x. Научитесь искать такую позицию p, что $a[p] = x$, за время $\mathcal{O}(\log p)$.
2. Задан массив длины n, состоящий из чисел от 1 до n, и целое число k. Предложите алгоритм, работающий за $\mathcal{O}(n)$ времени, который находит число пар (L, R), что среди $a_L, a_{L+1}, \ldots, a_R$ есть ровно k различных чисел.

3. Задано два упорядоченных массива размеров n и m, найдите k-ю порядковую статистику объединения двух массивов за $\mathcal{O}(\log \min(n, m))$.

5 Неделя 5

5.1 Практика

1. Постройте сортирующую сеть для 5 проводов с минимальным числом компараторов.
2. Докажите, что сеть компараторов из n элементов сортирует перестановку $(n, n-1, \ldots, 1)$ тогда и только тогда, когда она сортирует следующие $n-1$ двоичных последовательностей: $(1, 0, \ldots, 0), (1, 1, 0, \ldots, 0), \ldots, (1, 1, \ldots, 1, 0)$.
3. Докажите, что любая сортирующая сеть имеет глубину $\Omega(\log n)$.
4. Для сортировки битонических двоичных последовательностей на лекции первым шагом было разделение последовательности на две битонические такие, что все элементы первой половины были не больше любого элемента второй половины. Пусть последовательности не обязательно двоичные, и к ним применили первый шаг, докажите или опровергните каждый из пунктов:

   (a) обе половины после этого стали битоническими последовательностями;

   (b) все элементы первой половины не больше любого элемента второй половины.

5. Докажите или опровергните аналог теоремы о сортирующих сетях. Сеть компараторов сортирует любую битоническую последовательность тогда и только тогда, когда она сортирует любую двоичную битоническую последовательность.
6. Докажите или опровергните, что для любого заданного неотсортированного набора из 0 и 1 существует сеть компараторов, которая сортирует все наборы кроме заданного.
7. Докажите или опровергните, что для любой заданной неотсортированной перестановки чисел от 1 до n существует сеть компараторов, которая сортирует все перестановки, кроме заданной.
8. Докажите или опровергните следующее утверждение: вставка любого компаратора в сеть сортировки оставляет ее сетью сортировки.

5.2 Письменная часть (до 27 октября)

1. Докажите, что в любой сети сортировки, для любых двух соседних входных позиций i и $i+1$ найдется компаратор, который их сравнивает.
2. Докажите, что любая сеть компараторов, которая сливает 1 элемент с $n-1$ другими элементами, имеет глубину хотя бы $\log_2 n$.
3. Пусть у нас есть $2n$ элементов $a_1, a_2, \ldots a_{2n}$, и хотим разделить ее на n минимальных и n максимальных. Докажите, что нам понадобится $\mathcal{O}(1)$ дополнительной глубины после сортировки $a_1, a_2, \ldots, a_n$ и сортировки $a_{n+1}, a_{n+2}, \ldots, a_{2n}$.

4. Допустим, что кроме обычных компараторов существуют антикомпараторы: они сортируют в обратном порядке — минимальный из двух ниток элемент ставят в нижнюю из ниток, а максимальный в верхнюю. Покажите, как из сортирующей сети компараторов с c_1 компараторами и c_2 антикомпараторами построить сортирующую сеть из $c_1 + c_2$ обычных компараторов.

5. Предположим, что в сети компараторов по каждой нитке идет не один элемент, а список из k отсортированных. Компаратор в такой сети принимает два отсортированных списка из k элементов и в верхнюю нитку отправляет k минимальных, а в нижнюю — k максимальных. Докажите, что любая обычная сортирующая сеть на n нитках сортирует в новоописанной сети nk чисел.

6 Неделя 6

6.1 Практика

1. Постройте двоичную кучу за время $\mathcal{O}(n)$.
2. У вас есть k отсортированных массивов суммарной длины n. Как слить эти массивы в отсортированный длины n за время $\mathcal{O}(n \log k)$?
3. Модифицируйте `siftUp` для двоичной кучи, чтобы время работы не изменилось, а число сравнений стало $\mathcal{O}(\log \log n)$.
4. В d-куче выполняется m операций `decreaseKey` и n операций `extractMin`. Какое оптимальное асимптотически d следует выбрать?
5. Для всех $h > 0$ предложите последовательность операций `insert`, чтобы левосторонняя куча стала высоты h и имела $2^{h+1} - 1$ вершин.
6. Для всех $n > 0$ предложите последовательность операций `insert`, чтобы левосторонняя куча стала высоты n и имела n вершин.
7. Пусть подряд выполняется n операций `insert` в пустую биномиальную кучу. Какое среднее время операции?
8. Докажите, что амортизированная стоимость операции `insert` в биномиальную кучу равна $\mathcal{O}(1)$, при амортизированной стоимости остальных операций $\mathcal{O}(\log S)$, где S — размер кучи, на котором выполняется операция.
9. Придумайте, как добавить в двоичную кучу k элементов за $\mathcal{O}(k + \log n \cdot \log \log n)$.

6.2 Письменная часть (до 3 ноября)

1. Предложите алгоритм поиска k-го минимума в двоичной куче размера n за время $\mathcal{O}(k \log k)$, используя дополнительную двоичную кучу и не прибегая к использованию алгоритма поиска k-й порядковой статистики.
2. Модифицируйте левостороннюю кучу, которая хранит вещественные числа, так, чтобы поддержать операцию `changeKeys h x`, которая прибавляет ко всем ключам, хранящимся

в куче h значение $x \in \mathbb{R}$. Время работы операции `changeKeys` — $\mathcal{O}(1)$, время выполнения других операций не должно поменяться, разрешается использовать дополнительно $\mathcal{O}(1)$ памяти в каждой вершине.

3. С помощью кучи сделайте структуру данных, которая поддерживает три операции за $\mathcal{O}(\log n)$:

   (a) `insert x` — добавить элемент x в множество;

   (b) `medianElement` — возвращает медиану, в этой задаче *медиана* — это $\left\lfloor \frac{n}{2} \right\rfloor$-я порядковая статистика множества размера n;

   (c) `deleteMedian` — удаляет медиану.

   Научитесь строить такую структуру данных из n элементов за время $\mathcal{O}(n)$.

4. Будем сортировать элементы матрицы $m \times m$, повторяя следующую процедуру:

   (a) отсортировать каждую нечетную строку в возрастающем порядке;

   (b) отсортировать каждую четную строку в убывающем порядке;

   (c) отсортировать каждый столбец в возрастающем порядке.

   Через сколько итераций матрица перестанет изменяться? Как она будет выглядеть в итоге?