1 Кучи, бинарный поиск и простые структуры данных

1.1 Практика

Двоичная куча:

```
def siftUp(cur): # Просеивание вверх
  while cur != 0 and H[cur] < H[(cur - 1) // 2]:
    swap(H[cur], H[(cur - 1) // 2])
    cur = (cur - 1) // 2

def siftDown(cur): # Просеивание вниз
  while True:
    pmin = cur
    for child in [cur * 2 + 1, cur * 2 + 2]:
      if child < len(H) and H[child] < H[pmin]:
        pmin = child

    if cur == pmin:
      break
    swap(H[cur], H[pmin])
    cur = pmin

def getMin():
  return H[0]

def add(x):
  H.append(x)
  siftUp(len(H) - 1)

def extractMin():
  swap(H[0], H[len(H) - 1])
  result = H.pop()
  siftDown(0)
  return result
```

1. **Повторение мать заикания**
   Добавить к интерфейсу "очередь" функцию `min`, так чтобы она, а также все предыдущие функции работали за $\mathcal{O}(1)$ в среднем.

2. **Долги**
   Дана последовательность $a_1, a_2, \cdots, a_n \in \mathbb{N}$ и $S \in \mathbb{N}$. Найти l, r $(1 \le l \le r \le n)$ такие, что сумма $\sum_{i=l}^{r} a_i = S$. Задачу требуется решить за линейное от n время.

   Можно воспользоваться следующим планом решения:

   - Для каждого i найдите максимальное такое r_i, что $\sum_{j=i}^{r_i} a_j \le S$ за суммарное время $\mathcal{O}(n^2)$.
   - Найдите за $\mathcal{O}(n)$ ответ задачи, если известны $r_1, \ldots, r_n$.
   - Докажите, что $r_i \le r_{i+1}$.
   - Пользуясь предыдущим пунктом найдите все r_i за $\mathcal{O}(n)$.

3. (!) **Рваные кеды**
   Есть n верёвок, которые можно резать, каждая имеет целую длину l_i. Нужно получить k одинаковых кусков максимальной целой положительной длины (можно оставлять неиспользованные обрезки). $\mathcal{O}(n \log l_{\max})$.

4. `Count(l, r, x)`
   Сделайте предподсчет за $\mathcal{O}(n \log n)$, чтобы за $\mathcal{O}(\log n)$ отвечать на запросы вида: "сколько раз число x встречается на отрезке $[l..r]$"?

5. (!) **k-Merge**
   Есть k отсортированных массивов. В сумме массивы содержат n элементов. Слить массивы за $\mathcal{O}(n \log k)$.

6. **Интермеццо**

   (a) Вычислите $\sum_{i=0}^{\infty} \frac{1}{2^i}$.

   (b) Вычислите $\sum_{i=0}^{\infty} \frac{i}{2^i}$.

7. **Апгрейды `Build`, `HeapSort`**
   Пусть в массиве H записаны какие-то объекты (ключи) в произвольном порядке (т. е. свойство кучи может не выполняться).

   (a) Что делает следующий код? Оцените время работы.

```
for i in range(len(H)):
  siftUp(i)
```

   (b) (!) Что делает следующий код? Оцените время работы.

```
for i in range(len(H) - 1, -1, -1):
  siftDown(i)
```

   (c) (!) Опишите как сделать HeapSort используя только $\mathcal{O}(1)$ дополнительной памяти.

8. **Частичная сортировка**
   Дан массив длины n, выдать в порядке возрастания наименьшие k элементов за $\mathcal{O}(n+k\log n)$. (Другими словами, выдать префикс длины k отсортированной версии массива.)

9. Дана обычная бинарная куча (с минимумом в голове), требуется узнать k-й минимум.

   (a) $\mathcal{O}(k\log n)$

   (b) $\mathcal{O}(k^2)$

   (c) $\mathcal{O}(k\log k)$

10. Придумайте структуру данных, которая умеет делать `add(x)`, `getMedian()`, `extractMedian()` — все за $\mathcal{O}(\log n)$.

    Медиана множества — это такой элемент, что половина оставшихся элементов меньше его, а другая половина — больше. Другими словами, это элемент, стоящий посередине в отсортированном порядке (причём такое определение подходит и для мультимножества).

    Если n нечётно, медиана определена однозначно. Если n чётно, то существуют разные подходы (например, полусумма двух «кандидатов»). В этой задаче для определённости мы выбираем меньшего из «кандидатов». Например, медиана $\{1,1,6,3,4,7\}$ — это число 3, а медиана $\{1,6,3,5,4\}$ — это число 4.

11. (*) **Переаллокация в реализации дека на массиве**
    Рассмотрим реализацию дека на массиве. Как её правильно дополнить переаллокацией памяти, так чтобы все операции работали за $\mathcal{O}(1)$ амортизированно?

```
Q = [None for _ in range(n)]
L = 0
R = 0 # храним данные в [L, R).

def push_back(x):
  Q[R] = x
  R = (R + 1) % n

def pop_back():
  R = (R + n - 1) % n
  return Q[R]

def push_front(x):
  L = (L + n - 1) % n
```

```
15    Q[L] = x
16
17  def pop_front():
18    result = Q[L]
19    L = (L + 1) % n
20    return result
```

12. (*) Модифицируйте операцию `siftUp` для бинарной кучи так, чтобы она по-прежнему работала за $\mathcal{O}(\log n)$, но при этом делала лишь $\mathcal{O}(\log \log n)$ сравнений ключей.

13. (*) Модифицируйте операцию `siftDown` для бинарной кучи так, чтобы она по-прежнему работала за $\mathcal{O}(\log n)$, но при этом делала лишь $\log_2 n + \mathcal{O}(\log \log n)$ сравнений ключей.

1.2 Домашнее задание

Запись **(2.5)** означает, что за полное решение этой задачи или пункта даётся 2.5 балла. Запись `(*)` означает, что задача дополнительная: за её решение даются баллы, но она не учитывается в знаменателе оценки.
Не забывайте смотреть на задачи, которые обсуждались в классе! В них много полезных идей.
В некоторых задачах ограничен размер дополнительной памяти (то есть используемой памяти без учёта памяти для входа). В частностни, использовать $\mathcal{O}(1)$ дополнительной памяти неформально означает, что разрешено использовать лишь константное количество новых переменных, а создавать новые массивы (или структуры) неконстантного размера запрещено, также как и делать рекурсивные функции более чем константной глубины.

1. **(1)** Есть m стойл с координатами $x_1, \ldots, x_m$ и n коров. Расставить коров по стойлам (не более одной в стойло) так, чтобы минимальное расстояние между коровами было максимально. $\mathcal{O}(m(\log m + \log x_{\max}))$.

2. **(1.5 + 0.5)** Даны два массива a и b длины n, сгенерировать все попарные суммы $a_i + b_j$ в сортированном порядке. Складывать все суммы в один массив необязательно, можно печатать ответ постепенно по ходу выполнения.

   (a) **(0.5)** За $\mathcal{O}(n^2 \log n)$.

   (b) **(0.5)** За $\mathcal{O}(n^3)$ с использованием $\mathcal{O}(n)$ дополнительной памяти.

   (c) **(0.5)** За $\mathcal{O}(n^2 \log n)$ с использованием $\mathcal{O}(n)$ дополнительной памяти.

   (d) **(+0.5)** `(*)` За $\mathcal{O}(n^3)$ с использованием $\mathcal{O}(1)$ дополнительной памяти.
   Разрешается изменять исходные массивы. (В предыдущих пунктах тоже разрешается, но это ни на что не влияет: там есть хотя бы линейный запас по памяти, поэтому их можно просто скопировать.)

3. **(1)** Даны два отсортированных массива длины n, которые нельзя модифицировать. Найдите k-ю порядковую статистику в объединении массивов (то есть элемент, который окажется на k-й позиции, если массивы слить), используя $\mathcal{O}(1)$ дополнительной памяти.

   (a) **(0.5)** За $\mathcal{O}(\log^2 n)$.

   (b) **(0.5)** За $\mathcal{O}(\log n)$.