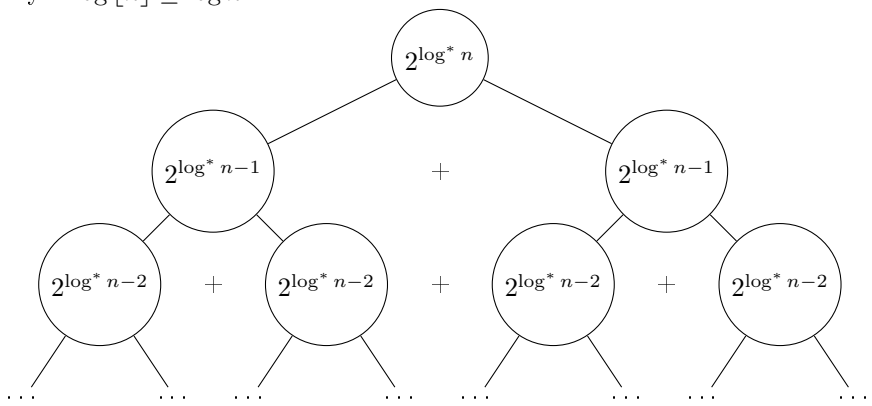
Алгоритмы и структуры данных
Домашняя работа 3

Мирзоев Денис

1 Определить ассимптотику

$T(n) = 2 \cdot T(\lfloor \log n \rfloor) + 2^{\log^* n}$

Рассмотрим дерево вычисления данной функции для n. Округление вниз для логарифма можно отбросить, потому что мы доказываем верхнюю оценку и $\log \lfloor n \rfloor \leq \log n$.



Глубина дерева равна $h = \log^* n$.
Возьмём сумму по слоям:

$$T(n) = O(2^h + 2 \cdot 2^{h-1} + \ldots + 2^{n-1} \cdot 2 + 2^h)$$
$$T(n) = O(h \cdot 2^h) = O(\log^* n \cdot 2^{\log^* n})$$

2 Коровы — в стойла!

Для начала выполним сортировку за $O(m \log m)$ координат стойл и найдём x_{max}. Далее осуществим бинарный поиск по условию: $f(d) =$ "коров можно расставить по стойлам с минимальным расстоянием не меньше, чем d". При $d = 0$ ответ положительный, при $d = x_{max}$ ответ отрицательный. Функция f монотонно убывает. Найдём максимальное возможное d бинарным поиском.

Найти способ расставить коров по стойлам с минимальным рассноянием не больше чем d или определить что такого способа нет можно за $O(m)$. Массив стойл уже упорядочен. Поставим два указателя i и j в начало массива координат($i = 0, j = 1$). От i до $j - 1$ будут находиться стойла запрещённые к использованию. Запрет на использование этих стойл будет гарантировать нам некоторое минимальное расстояние. Будем увеличивать j до тех пор, пока расстояние между крайними запрещёнными стойлами не станет больше d. Если количество оставшихся стойл $i + (m - j)$ больше n, то возвращаем положительный ответ в противном случае увеличим j на единицу, а потом будем увеличивать i пока расстояние между стойлом $i + 1$ и j больше d. Если количество оставшихся стойл $i + (m - j)$ больше n, то возвращаем отрицательный ответ в противном случае продолжим увеличивать i и j, пока не достигнем m. В этом случае возвращаем отрицательный ответ.

Эта же процедура позволяет определить точное максимальное расстояние между коровами не больше d. После завершения бинарного поиска вернём это значение. это будет максимальное возможное расстояние между коровами вообще.

3 Второй максимум в массиве

```
secondMax(array, N)
    a = array[1]
    b = 0
    for i = 1 to N:
        if array[i] > a: # или >=
            b = a
            a = array[i]
    return b
```

4 K-порядковая статистика двух отсортированных массивов

- (a)

  Для каждого элемента(a_i) первого массива можно вычислить его индекс в отсортированном массиве за логарифм. Для этого находим бинарным поиском во втором массиве первый элемент больше a_i назовём его b_j. Тогда его индекс в отсортированном массиве будет равен $i + j$. Найдем бинарным поиском в первом массиве последний элемент с индексом в отсортированном массиве меньше k(a_i и b_j первый больше a_i). Ответом будет $b[j - 1 + (k - i)]$.

- (b)

  a и b - массивы в которых ищем k-порядковую статистику. Пусть $i = \lfloor \frac{n}{2} \rfloor, j = k - 1 - i$. Рассмотрим случай, когда $a[i] < b[j]$, другой вариант рассматривается аналогично. Если $a[i] > b[j - 1]$, то $a[i]$ — k-порядковая статистика. Рассмотрим случай $a[i] < b[j - 1]$. В этом случае на месте $a[i]$ стоит не более чем $(i + (j - 2) + 2) = k - 1$-порядковая статистика, значит слева от него k-порядковой статистики нет. Мы рассматриваем случае, когда $b[j] > a[i]$. Если $b[j] < a[i + 1]$, то $b[j]$ — k-порядковая статистика. В слитом массиве слева от $b[j]$ будет стоять на $i + 1$ элементов больше, значит он будет минимум с индексом $i + j + 1 = k$, но k он быть не может так как этот случай мы уже рассмотрели. Индексы остальных элементов b справа от b_j будут ещё больше. Продолжитм искать k-порядковую статистику таким же способом в a от i до конца и в b от начала до j.

5 Сортировка файла

Будем сортировать файл с использованием двух дополнительных файлов модифицированной сортировкой слиянием. Рекурсивных вызовов не будет, мы будем сортировать как бы по слоям, начиная с самого нижнего. Сначала отсортируем подмассивы из одного элемента, потому из двух, потом из четырёх и так далее. Отсортированные части будем складывать в два дополнительных файла, а из них обратно.

```
# 1     - исходный файл
# 2, 3  - вспомогательные файлы
# read(n)       - прочитать один элемент из файла n
# write(n, x)   - записать один элемент в файл n
# rewind(n)     - вернуть указатель в начало файла n
k = 1
while k < n:
    p = 1
    while p < n:
        for i = p to min(n, p + k):
            write(2, read(1))
        p = min(n, p + k) + 1:
        for i = p to min(n, p + k):
            write(3, read(1))
        p = min(n, p + k) + 1:
    rewind(1)
    rewind(2)
    rewind(3)
    p = 1
    while p < n:
        a = read(2)
        b = read(3)
        for i = p + 2 to min(n, p + 2 * k):
            if a < b:
                write(1, a)
                a = read(2)
            else
                write(1, b)
                b = read(3)
        p = min(n, p + 2 * k) + 1
    k = k * 2
```

6 Чёрные и белые шары

Отсортируем коробки по возрастанию значения $\frac{w_i}{W} + \frac{b_i}{B}$. Выберем вторую половину.