Алгоритмы и структуры данных

Поиск порядковых статистик

Задача

Пусть дан массив размера N из уникальных элементов. Найти i-ую порядковую статистику.

i-ая порядковая статистика - это i-ый элемент в порядке возрастания.

Задача

Пусть дан массив размера N из уникальных элементов. Найти i-ую порядковую статистику.

i-ая порядковая статистика - это i-ый элемент в порядке возрастания.

33	2	3	12	7	4	8
z_7	z_1	z_2	z_6	z_4	z_3	z_5

Задача

Пусть дан массив размера N из уникальных элементов. Найти i-ую порядковую статистику.

Как решать?

33	2	3	12	7	4	8
z_7	z_1	z_2	z_6	z_4	z_3	z_5

Задача

Пусть дан массив размера N из уникальных элементов. Найти i-ую порядковую статистику.

Как решать? Отсортируем и возьмем i-ый.

2	3	4	7	8	12	33
z_1	z_2	z_3	z_4	z_5	z_6	z_7

Задача

Пусть дан массив размера N из уникальных элементов. Найти i-ую порядковую статистику.

Как решать? Отсортируем и возьмем i-ый.

Сложность?

2	3	4	7	8	12	33
z_1	z_2	z_3	z_4	z_5	z_6	z_7

Задача

Пусть дан массив размера N из уникальных элементов. Найти i-ую порядковую статистику.

Как решать? Отсортируем и возьмем i-ый.

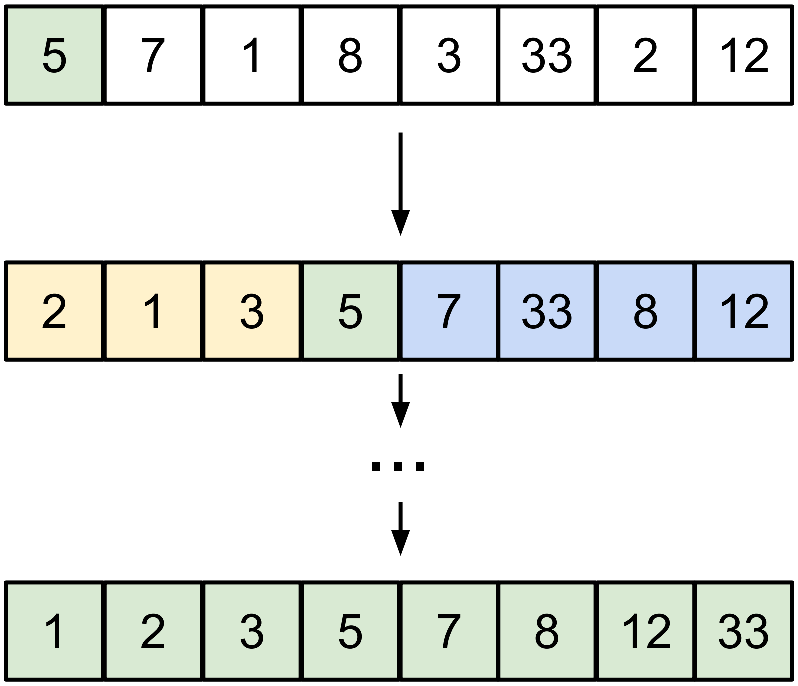
Сложность? Ω(N*logN), если сорт. сравнением

2	3	4	7	8	12	33
z_1	z_2	z_3	z_4	z_5	z_6	z_7

Быстрая сортировка

Алгоритм:

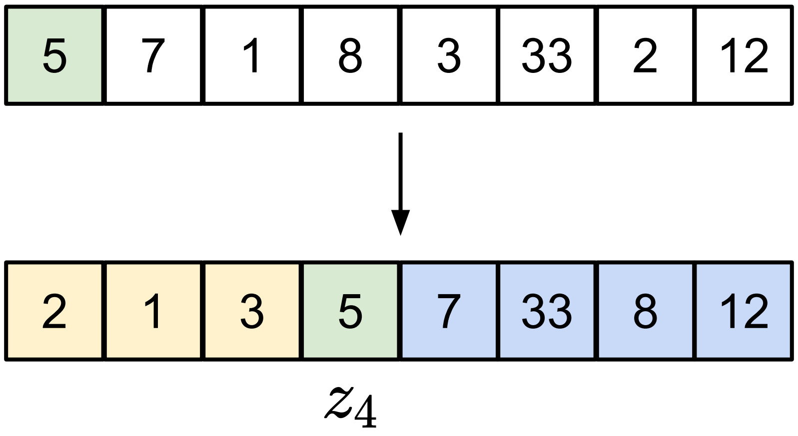
1. Выбрать опорный элемент
2. Выполнить разбиение
3. Рекурсия на левую и правые части разбиения



Быстрая сортировка

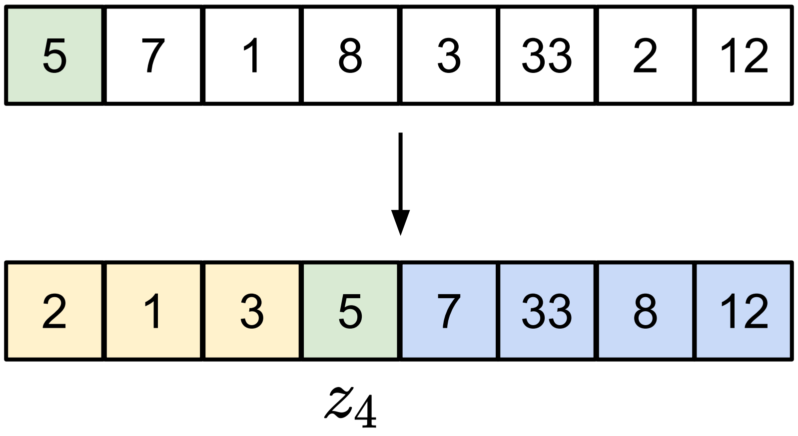
Алгоритм:

1. Выбрать опорный элемент
2. Выполнить разбиение
3. Рекурсия на левую и правые части разбиения



Быстрая сортировка

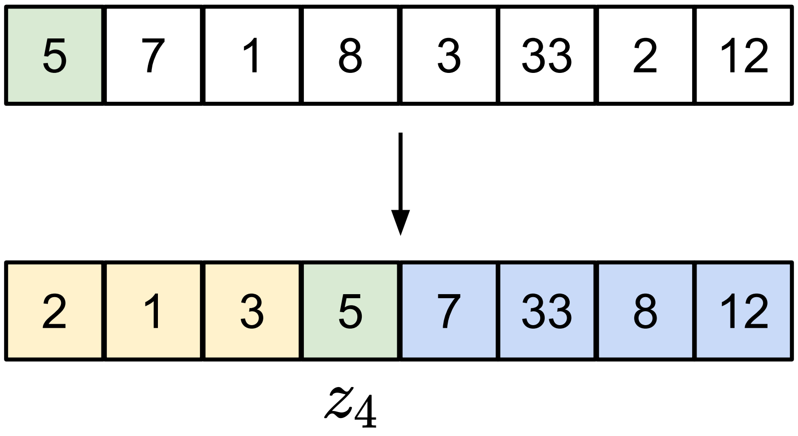
Если вдруг мы искали z_4 и выбрали опорным элементом пятерку, то после разбиения она как раз стоит на 4-ом месте.



Быстрая сортировка

Если вдруг мы искали z_4 и выбрали опорным элементом пятерку, то после разбиения она как раз стоит на 4-ом месте.

А если нет?

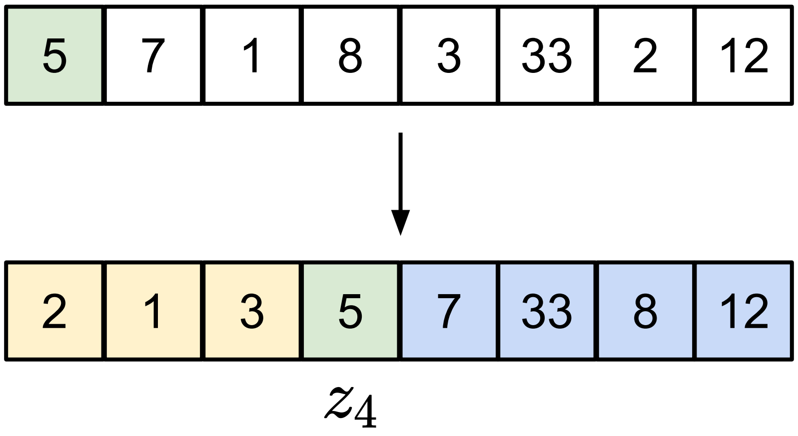


Быстрая сортировка

Если вдруг мы искали z_4 и выбрали опорным элементом пятерку, то после разбиения она как раз стоит на 4-ом месте.

Если искали i-ый меньше четвертого, то он где-то слева.

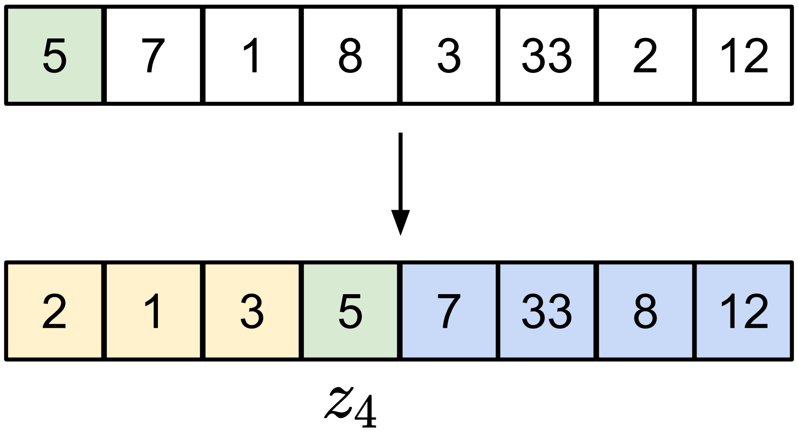
А иначе - где-то справа.



Быстрая сортировка

Допустим ищем z_3.

Какой рекурсивный вызов сделать?

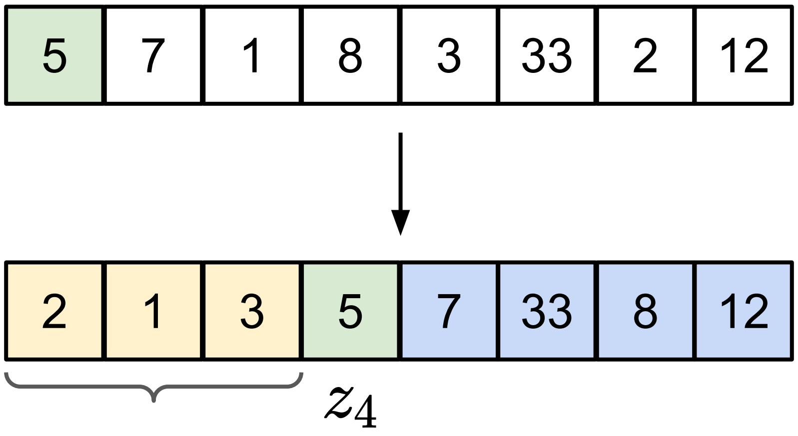


Быстрая сортировка

Допустим ищем z_3.

Какой рекурсивный вызов сделать?

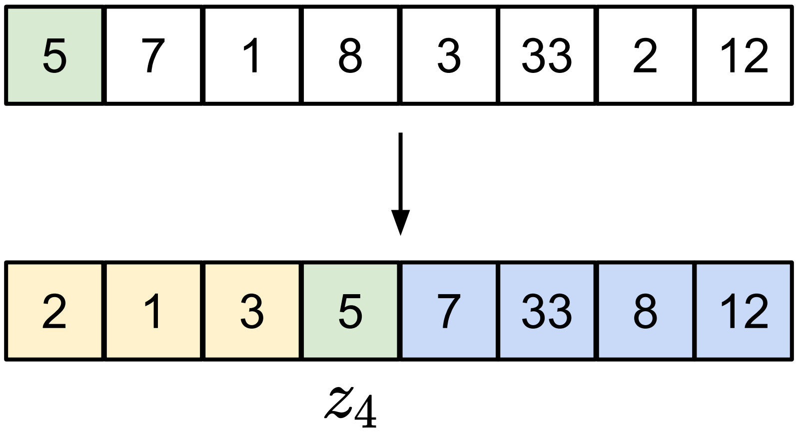
Ищем z_3 в левом подмассиве.



Быстрая сортировка

Допустим ищем z_6.

Какой рекурсивный вызов сделать?

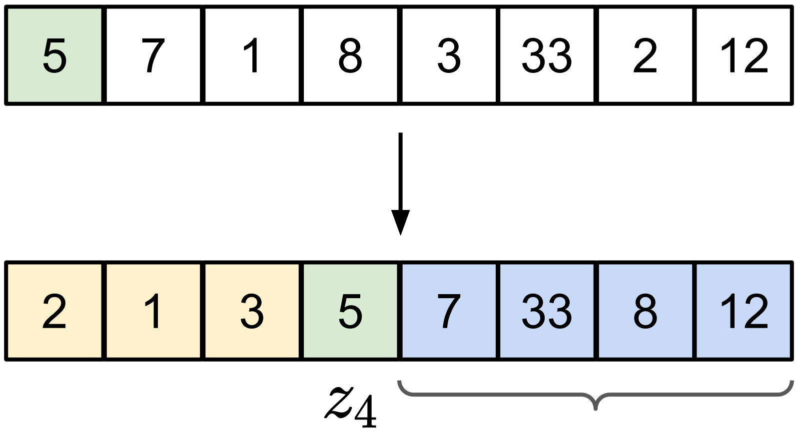


Быстрая сортировка

Допустим ищем z_6.

Какой рекурсивный вызов сделать?

Ищем z_2 в правом подмассиве.



(шестая статистика глобально,
но в правом уже на 4 элемента меньше)

Поиск k-ой статистики

```
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]
```

Поиск k-ой статистики

```
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]

    pivot = random(0, len(array) - 1)
    p = partition(array, pivot)
```

Поиск k-ой статистики

```
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]

    pivot = random(0, len(array) - 1)
    p = partition(array, pivot)

    if p + 1 == k:
        return array[p]
```

Поиск k-ой статистики

```python
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]

    pivot = random(0, len(array) - 1)
    p = partition(array, pivot)

    if p + 1 == k:
        return array[p]
    elif p + 1 > k:
        return kth(array[:p], k)
```

Поиск k-ой статистики

```
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]

    pivot = random(0, len(array) - 1)
    p = partition(array, pivot)

    if p + 1 == k:
        return array[p]
    elif p + 1 > k:
        return kth(array[:p], k)
    else
        return kth(array[p + 1:], k - p - 1)
```

Поиск k-ой статистики

```
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]

    pivot = random(0, len(array) - 1)
    p = partition(array, pivot)

    if p + 1 == k:
        return array[p]
    elif p + 1 > k:
        return kth(array[:p], k)
    else
        return kth(array[p + 1:], k - p - 1)
```

Сложность?

Поиск k-ой статистики

```
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]

    pivot = random(0, len(array) - 1)
    p = partition(array, pivot)

    if p + 1 == k:
        return array[p]
    elif p + 1 > k:
        return kth(array[:p], k)
    else
        return kth(array[p + 1:], k - p - 1)
```

Сложность?

В худшем: $O(n^2)$

Поиск k-ой статистики

```
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]

    pivot = random(0, len(array) - 1)
    p = partition(array, pivot)

    if p + 1 == k:
        return array[p]
    elif p + 1 > k:
        return kth(array[:p], k)
    else
        return kth(array[p + 1:], k - p - 1)
```

Сложность?

В худшем: $O(n^2)$

А среднее (ожидаемое) время работы?

Поиск k-ой статистики

Теорема: для любого входного массива длины N среднее время работы алгоритма поиска k-ой статистики (со случайным выбором опорных элементов) равно O(N)

Поиск k-ой статистики

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Поиск k-ой статистики

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Обозначим: j-ой фазой алгоритма ситуацию, когда размер подмассива l: $(\frac{3}{4})^{j+1} n \leq l \leq (\frac{3}{4})^{j} n$

Поиск k-ой статистики

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Обозначим: j-ой фазой алгоритма ситуацию, когда размер подмассива l: $(\frac{3}{4})^{j+1}n \leq l \leq (\frac{3}{4})^{j}n$

В зависимости от выбора очередного опорного элемента можем либо остаться в той же фазе, либо перейти в более позднюю.

Поиск k-ой статистики

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Обозначим: j-ой фазой алгоритма ситуацию, когда размер подмассива l: $(\frac{3}{4})^{j+1} n \leq l \leq (\frac{3}{4})^{j} n$

В зависимости от выбора очередного опорного элемента можем либо остаться в той же фазе, либо перейти в более позднюю.

В каждой фазе может быть много рекурсивных вызовов!

Поиск k-ой статистики

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Обозначим: j-ой фазой алгоритма ситуацию, когда размер подмассива l: $(\frac{3}{4})^{j+1}n \leq l \leq (\frac{3}{4})^{j}n$

X_j = количество рекурсивных вызовов на j-ой фазе.

Поиск k-ой статистики

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Обозначим: j-ой фазой алгоритма ситуацию, когда размер подмассива l: $(\frac{3}{4})^{j+1} n \leq l \leq (\frac{3}{4})^{j} n$

X_j = количество рекурсивных вызовов на j-ой фазе.

Зависит от выбора опорных элементов!
Так что это случайная величина.

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Обозначим: j-ой фазой алгоритма ситуацию, когда размер подмассива l: $(\frac{3}{4})^{j+1}n \leq l \leq (\frac{3}{4})^{j}n$

X_j = количество рекурсивных вызовов на j-ой фазе.

Пусть теперь R - общее количество операций за все время работы алгоритма.

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Обозначим: j-ой фазой алгоритма ситуацию, когда размер подмассива l: $(\frac{3}{4})^{j+1} n \leq l \leq (\frac{3}{4})^{j} n$

X_j = количество рекурсивных вызовов на j-ой фазе.

Пусть теперь ***R*** - общее количество операций за все время работы алгоритма.

Тогда: $R \leq \sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

Заметим: процедура разбиения работает за линейное время. Тогда зафиксируем константу c, такую что количество операций потраченное на разбиение любого подмассива длины l: $\leq c * l$

Обозначим: j-ой фазой алгоритма
ситуацию, когда размер подмассива l: $(\frac{3}{4})^{j+1}n \leq l \leq (\frac{3}{4})^{j}n$

X_j = количество рекурсивных вызовов на j-ой фазе.

Пусть теперь ***R*** - общее количество операций за все время работы алгоритма.

Тогда: $R \leq \sum_{\text{фаза } j} X_j * c * \boxed{(\frac{3}{4})^j * n}$

Максимальная длина подмассива на фазе j

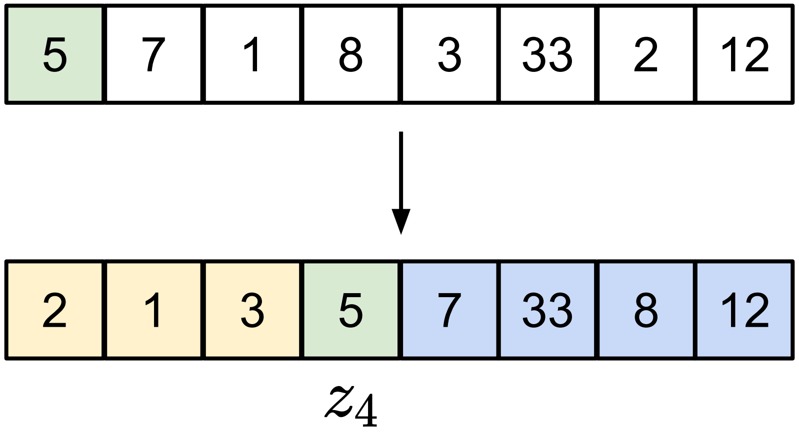
Пусть: мы сейчас находимся на фазе j.

Т.е. размер подмассива сейчас l: $(\frac{3}{4})^{j+1}n \leq l \leq (\frac{3}{4})^{j}n$

Пусть: мы сейчас находимся на фазе j.

Т.е. размер подмассива сейчас l: $(\frac{3}{4})^{j+1}n \leq l \leq (\frac{3}{4})^{j}n$

Что должно произойти, чтобы мы перешли в следующую фразу?

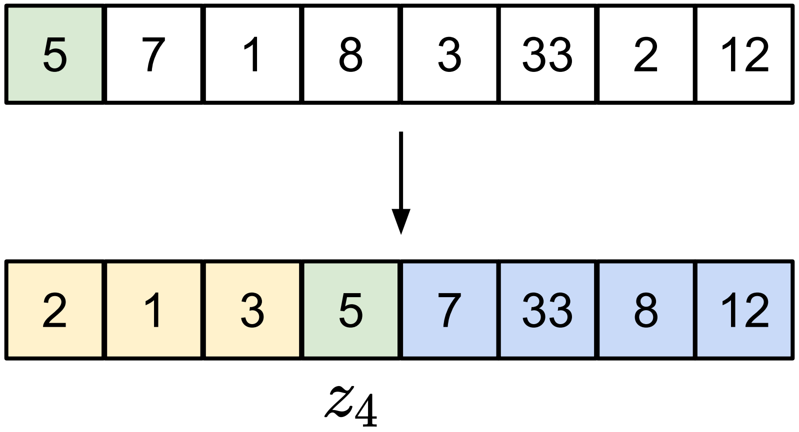


Пусть: мы сейчас находимся на фазе j.

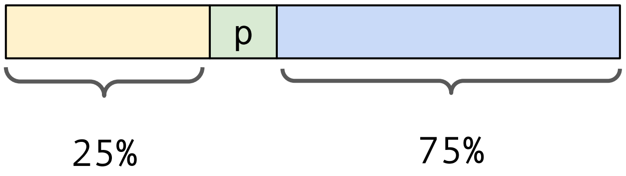
Т.е. размер подмассива сейчас l: $(\frac{3}{4})^{j+1}n \le l \le (\frac{3}{4})^{j}n$

Что должно произойти, чтобы мы перешли в следующую фразу?

Мы должны сделать достаточно хорошее разбиение! Лучше, чем 75 к 25.



Немного теории вероятностей #2



Событие – подмножество $S \subseteq \Omega$

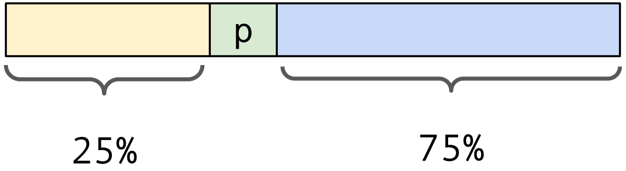
Вероятность события S: $Pr[S] = \sum_{i \in S} p(i)$

Пример #5: Событие – выбор случайного опорного элемента дал разбиение 25% к 75% или лучше (ближе к 50:50)

1	2	3	4	5	6	7	8	9	10	11	12

Попадаем в один из зеленых элементов => разбиение 25:75 или лучше

Немного теории вероятностей #2



Событие – подмножество $S \subseteq \Omega$

Вероятность события S: $Pr[S] = \sum_{i \in S} p(i)$

Пример #5: Событие – выбор случайного опорного элемента дал разбиение 25% к 75% или лучше (ближе к 50:50)

$S = \{$ выбрали (n/4 + 1)-ый элемент,… , выбрали (3*n/4 - 1)-ый элемент $\}$

$$Pr[S] = \sum_{i \in S} p(i) = \frac{n}{2} * \frac{1}{n} = \frac{1}{2}$$

Пусть: мы сейчас находимся на фазе j.

Т.е. размер подмассива сейчас l: $(\frac{3}{4})^{j+1}n \leq l \leq (\frac{3}{4})^{j}n$

Что должно произойти, чтобы мы перешли в следующую фразу?

Мы должны сделать достаточно хорошее разбиение! Лучше, чем 75 к 25.

Т.е. на каждом шаге алгоритма шанс хорошего разбиения и перехода в следующую фазу - 50%!

Пусть: мы сейчас находимся на фазе j.

Т.е. размер подмассива сейчас l: $(\frac{3}{4})^{j+1}n \le l \le (\frac{3}{4})^{j}n$

Что должно произойти, чтобы мы перешли в следующую фразу?

Мы должны сделать достаточно хорошее разбиение! Лучше, чем 75 к 25.

Т.е. на каждом шаге алгоритма шанс хорошего разбиения и перехода в следующую фазу - 50%!

X_j = количество рекурсивных вызовов на j-ой фазе.

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

Пусть $L =$"кол-во бросков монеты, пока не выпал орёл"

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

Пусть $L =$"кол-во бросков монеты, пока не выпал орёл"

Тогда $\mathbb{E}[L] = \frac{1}{2}(1+0) +$

Повезло с первого же броска.

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

Пусть $L =$"кол-во бросков монеты, пока не выпал орёл"

Тогда $\mathbb{E}[L] = \frac{1}{2}(1 + 0) + \frac{1}{2}(1 +$

Повезло с первого же броска.

Не повезло с первого броска.

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

Пусть $L =$"кол-во бросков монеты, пока не выпал орёл"

Тогда $\mathbb{E}[L] = \frac{1}{2}(1+0) + \frac{1}{2}(1+\mathbb{E}[L])$

Повезло с первого же броска.

Не повезло с первого броска.

Начинаем все с самого начала (про прошлый бросок забыли)

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

Пусть $L =$"кол-во бросков монеты, пока не выпал орёл"

Тогда $\mathbb{E}[L] = \frac{1}{2}(1+0) + \frac{1}{2}(1+\mathbb{E}[L]) = 1 + \frac{1}{2}\mathbb{E}[L]$

Повезло с первого же броска.

Не повезло с первого броска.

Начинаем все с самого начала (про прошлый бросок забыли)

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

Пусть $L =$"кол-во бросков монеты, пока не выпал орёл"

Тогда $\mathbb{E}[L] = \frac{1}{2}(1+0) + \frac{1}{2}(1+\mathbb{E}[L]) = 1 + \frac{1}{2}\mathbb{E}[L]$

$$\mathbb{E}[L] = 2$$

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

Пусть $L =$"кол-во бросков монеты, пока не выпал орёл"

Тогда $\mathbb{E}[L] = \frac{1}{2}(1+0) + \frac{1}{2}(1+\mathbb{E}[L]) = 1 + \frac{1}{2}\mathbb{E}[L]$

$$\mathbb{E}[L] = 2$$

У такой случайной величины
геометрическое распределение

X_j = количество рекурсивных вызовов на j-ой фазе.

На каждом рекурсивном вызове мы с вероятностью 50% вылетаем из фазы.

Получается, что эта случайная величина - это фактически "сколько раз нужно бросить монетку, пока не выпадет орёл"

Пусть $L =$"кол-во бросков монеты, пока не выпал орёл"

Тогда $\mathbb{E}[L] = \frac{1}{2}(1+0) + \frac{1}{2}(1+\mathbb{E}[L]) = 1 + \frac{1}{2}\mathbb{E}[L]$

$\mathbb{E}[L] = 2$

У такой случайной величины геометрическое распределение

Не является строгим док-вом, легко доказывается строго по определению мат. ожидания

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \leq \mathbb{E}[L]$

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \le \mathbb{E}[L]$

Вспоминаем, что: $R \le \sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \leq \mathbb{E}[L]$

Вспоминаем, что: $R \leq \sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

Тогда: $\mathbb{E}[R] \leq \mathbb{E}[\sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n]$

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \leq \mathbb{E}[L]$

Вспоминаем, что: $R \leq \sum_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

Тогда: $\mathbb{E}[R] \leq \mathbb{E}[\sum_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n]$

$$= \sum_{\text{фаза } j} \mathbb{E}[X_j] * c * (\frac{3}{4})^j * n$$

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \leq \mathbb{E}[L]$

Вспоминаем, что: $R \leq \sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

Тогда: $\mathbb{E}[R] \leq \mathbb{E}[\sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n]$

$$= \sum_{\text{фаза } j} \mathbb{E}[X_j] * c * (\frac{3}{4})^j * n \leq 2 * c * n * \sum_{\text{фаза } j} (\frac{3}{4})^j$$

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \leq \mathbb{E}[L]$

Вспоминаем, что: $R \leq \sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

Тогда: $\mathbb{E}[R] \leq \mathbb{E}[\sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n]$

$$= \sum_{\text{фаза } j} \mathbb{E}[X_j] * c * (\frac{3}{4})^j * n \leq 2 * c * n * \boxed{\sum_{\text{фаза } j} (\frac{3}{4})^j}$$

Школьные факты про суммы

Пусть $r \geq 0, r \neq 1$

Тогда $1 + r + r^2 + r^3 + \cdots + r^k = \frac{r^{k+1}-1}{r-1}$ $(**)$ сумма геометрической прогрессии

Если $r < 1$, то $(**) \leq \boxed{\frac{1}{1-r}}$

Если $r > 1$, то $(**) \leq r^k \boxed{(1 - \frac{1}{r-1})}$

Это константы не зависящие от k!

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \le \mathbb{E}[L]$

Вспоминаем, что: $R \le \sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

Тогда: $\mathbb{E}[R] \le \mathbb{E}[\sum\limits_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n]$

$$= \sum_{\text{фаза } j} \mathbb{E}[X_j] * c * (\frac{3}{4})^j * n \le 2 * c * n * \boxed{\sum_{\text{фаза } j} (\frac{3}{4})^j}$$

$$\le \frac{1}{1 - \frac{3}{4}} = 4$$

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \leq \mathbb{E}[L]$

Вспоминаем, что: $R \leq \sum_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

Тогда: $\mathbb{E}[R] \leq \mathbb{E}[\sum_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n]$

$$= \sum_{\text{фаза } j} \mathbb{E}[X_j] * c * (\frac{3}{4})^j * n \leq 2 * c * n * \boxed{\sum_{\text{фаза } j} (\frac{3}{4})^j}$$

$$\leq \frac{1}{1-\frac{3}{4}} = 4$$

$$\leq 8 * c * n$$

X_j = количество рекурсивных вызовов на j-ой фазе.

Тогда: $\mathbb{E}[X_j] \leq \mathbb{E}[L]$

Вспоминаем, что: $R \leq \sum_{\text{фаза } j} X_j * c * (\frac{3}{4})^j * n$

Тогда: $\mathbb{E}[R] \leq 8 * c * n$ $\square$

Поиск k-ой статистики

```
def kth(array: int[], k: int) -> int:
    if len(array) == 1: return array[0]

    pivot = random(0, len(array) - 1)
    p = partition(array, pivot)

    if p + 1 == k:
        return array[p]
    elif p + 1 > k:
        return kth(array[:p], k)
    else
        return kth(array[p + 1:], k - p - 1)
```

Сложность?

В худшем: $O(n^2)$

А среднее (ожидаемое) время работы?

$O(n)$

Takeaways

- Поиск — это не сортировка, так что пробиваем O(N*logN) и получаем O(N)

Takeaways

- Поиск — это не сортировка, так что пробиваем O(N*logN) и получаем O(N)
- Есть и детерминированный (без рандома) линейный алгоритм поиска k-ой статистики, но у него значительно хуже константы

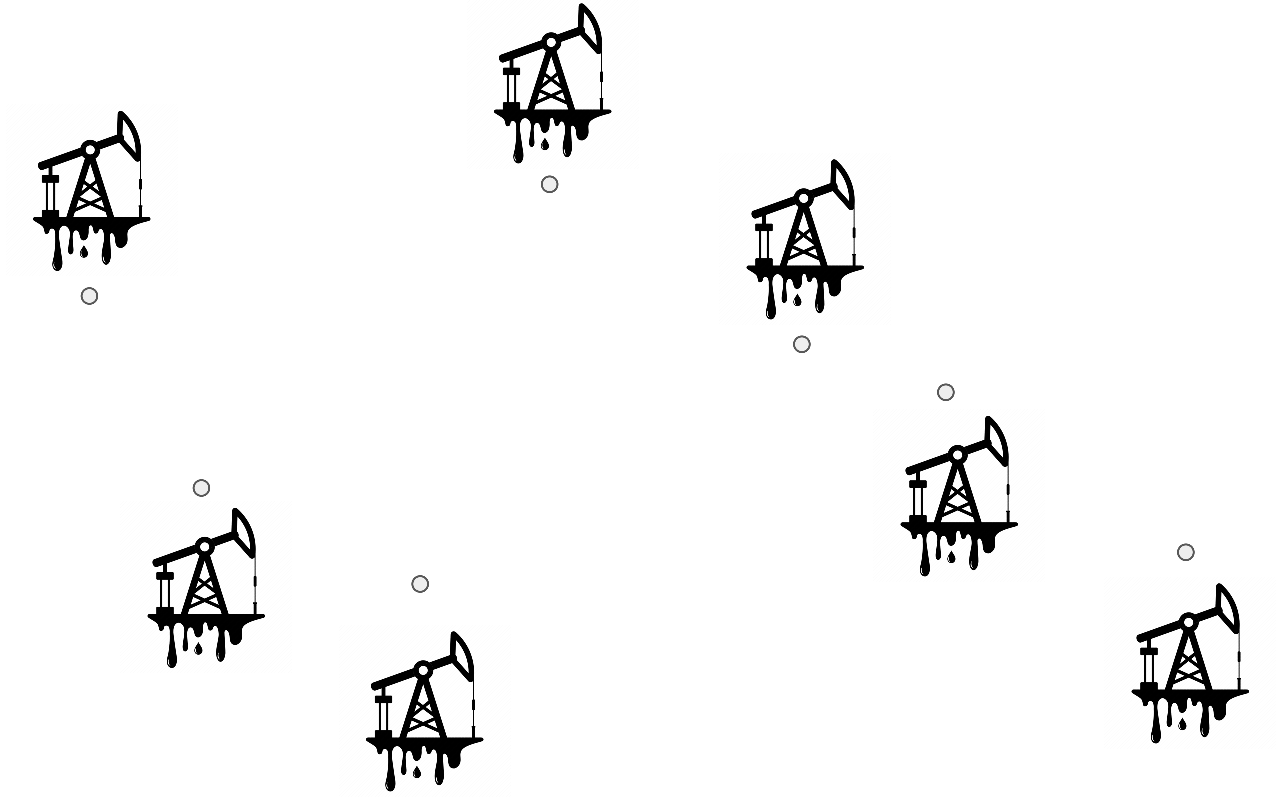
Мини-задача #14 (1 балл)

Необходимо принять решение о месте магистрального нефтепровода с запада на восток через нефтеносное поле. На этом поле расположены N нефтяных скважин (их координаты заданы). От магистрали до скважин отходят перпендикулярные трубопроводы.

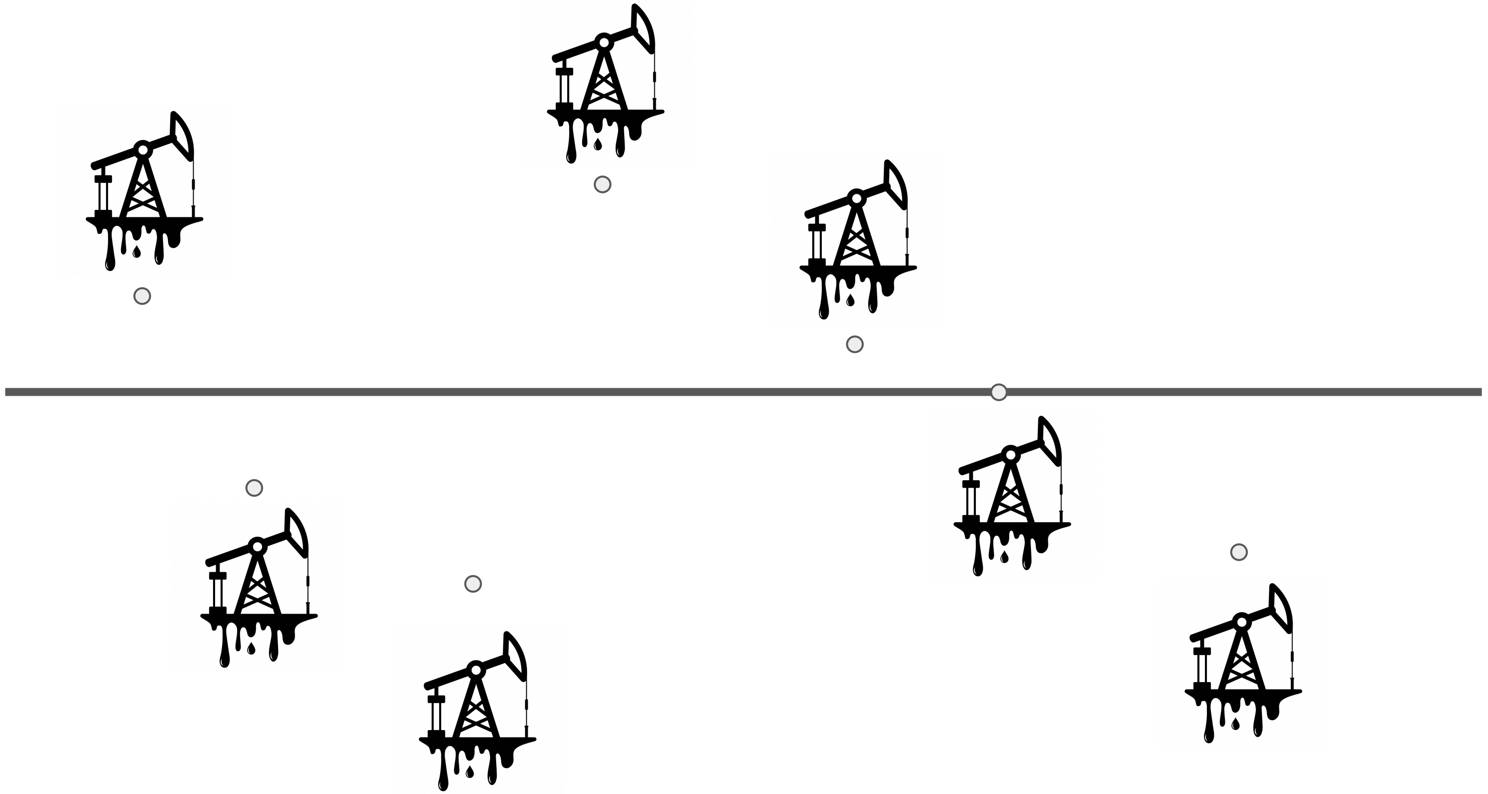
Необходимо выбрать место магистрального нефтепровода так, чтобы суммарная длина трубопроводов была минимальной.

Решить задачу нужно за линейное.

Мини-задача #14 (1 балл)



Мини-задача #14 (1 балл)



Мини-задача #14 (1 балл)