Эффективные сортировки, бинарный поиск

1 A. Сортировка слиянием

Разбор

В этой задаче требовалось реализовать алгоритм сортировки слиянием. Используя возможности языка Python, можно было написать меньше кода при помощи "срезов". Однако, срезы требуют дополнительной памяти и времени на копирование подмассива, поэтому наиболее эффективным решением является разделение не всего массива пополам, а только индексов, которые передаются ниже в рекурсию.

Асимптотика решения: $O(N \times \log N)$

Решение

```
def merge_sort(a, l, r):
    if r - l < 2:
        return 0
    m = (r + l) // 2
    merge_sort(a, l, m)
    merge_sort(a, m, r)
    merge(a, l, m, r)


def merge(a, l, m, r):
    l1, r1 = l, m
    l2, r2 = m, r
    result = []
    while l1 < r1 or l2 < r2:
        if l1 < r1 and l2 < r2:
            if a[l1] < a[l2]:
                result.append(a[l1])
                l1 += 1
            else:
                result.append(a[l2])
                l2 += 1
        elif l1 >= r1 and l2 < r2:
            result.append(a[l2])
            l2 += 1
        elif l2 >= r2 and l1 < r1:
            result.append(a[l1])
            l1 += 1
    for i in range(l, r):
        a[i] = result[i - l]


n = int(input())
x = list(map(int, input().split()))
merge_sort(x, 0, n)
print(' '.join(map(str, x)))
```

2 B. Сортировка подсчётом

Разбор

В этой задаче требовалось реализовать алгоритм сортировки подсчётом. При решении задачи следовало обратить внимание на минимальное и максимальное значения элементов массива, чтобы сократить объем выделяемой памяти для подсчёта.

Асимптотика решения: $O(N + M)$

Решение

```
n = int(input())
x = list(map(int, input().split()))
min_e, max_e = min(x), max(x)
cnt = [0] * (max_e - min_e + 1)
for v in x:
    cnt[v - min_e] += 1
k = 0
for i in range(len(cnt)):
    for c in range(cnt[i]):
        x[k] = i + min_e
        k += 1
print(' '.join(map(str, x)))
```

3 C. Ноутбуки

Разбор

Основная идея решения этой задачи заключалась в том, что с точки зрения Димы, если ноутбуки будут отсортированы по возрастанию цены, то они также будут отсортированы по качеству.

Нам нужно было проверить, что хотя бы для одной пары отсортированных по возрастанию цены ноутбуков **не выполнялось** условие строгого возрастания качества. Более формально, нам нужно было найти такое значение $1 < i <= N$, при котором было бы верно, что:

$$cost_{i-1} < cost_i, quality_{i-1} > quality_i$$

Асимптотика решения: $O(N \times \log N)$

Решение

```
n = int(input())
p, q = [0] * n, [0] * n
for i in range(n):
    p[i], q[i] = map(int, input().split())
z = list(zip(p, q))
z.sort(key=lambda x: x[0])
ans = 'Poor Alex'
for i in range(1, n):
    if z[i][1] < z[i - 1][1]:
        ans = 'Happy Alex'
        break
print(ans)
```

4 D. Двоичный поиск

Разбор

В этой задаче требовалось реализовать алгоритм двоичного поиска. Следовало обратить внимание на начальные значения левой и правой границ поиска (они должны выходить за пределы допустимых значений индексации массива), а также на проверки правильности ответа в конце поиска.

Асимптотика решения: $O(K \times \log N)$

Решение

```
def solve(a, n, x):
    l, r = -1, n
    while r - l > 1:
        m = (r + l) // 2
        if a[m] <= x:
            l = m
        else:
            r = m
    if l < 0 and a[r] == x:
        return "YES"
    if a[l] == x:
        return "YES"
    return "NO"


n, k = map(int, input().split())
a = list(map(int, input().split()))
b = list(map(int, input().split()))
for i in range(k):
    print(solve(a, n, b[i]))
```

5 E. Приближенный двоичный поиск

Разбор

В этой задаче требовалось реализовать алгоритм двоичного поиска. Следовало обратить внимание на начальные значения левой и правой границ поиска (они должны выходить за пределы допустимых значений индексации массива), а также на проверки правильности ответа в конце поиска.

Асимптотика решения: $O(K \times \log N)$

Решение

```
def solve(a, n, x):
    l, r = -1, n
    while r - l > 1:
        m = (r + l) // 2
        if a[m] <= x:
            l = m
        else:
            r = m
    if l < 0:
        return a[r]
    if r == n or x - a[l] <= a[r] - x:
        return a[l]
    return a[r]


n, k = map(int, input().split())
a = list(map(int, input().split()))
```

```
b = list(map(int, input().split()))
for i in range(k):
    print(solve(a, n, b[i]))
```

6 F. Отгадай число

Разбор

В этой задаче требовалось реализовать алгоритм двоичного поиска. Начальные границы поиска в этой задаче должны были выходить за пределы возможного ответа. Исходя из ограничения сверху на максимальное загаданное число, можно было понять, что нам потребуется не более $6 \times \log_2 10 \approx 20$ операций.

Сложность решения: $\log 10^6$

6.1 Решение

```
l = 0
r = 10 ** 6 + 1
while r - l > 1:
    m = (r + l) // 2
    print(m)
    s = input()
    if s == '<':
        r = m
    else:
        l = m
print('! ' + str(l))
```

7 G. Квадратный корень и квадратный квадрат

Разбор

В этой задаче нам было необходимо найти такое значение x, что равенство $x^2 + \sqrt{x} = C$ при заданном значении C было бы верным. Иначе говоря, от нас требовалось найти корень функции $f(x) = x^2 + \sqrt{x} - C$. Нетрудно показать, что у этой функции существует единственный корень, если $x > 0$: для этого найдём её производную и убедимся, что она не равна нулю при любом положительном x.

Тогда условием сдвига границ будет являться знак функции в точке $m = \frac{l+r}{2}$, а для достижения желаемой точности нам достаточно 36 итераций.

Как посчитать достаточное число итераций? Поймём для начала, что достижение точности до 6 знака эквивалентно тому, что бинпоиск в целых числах сойдётся для границ 0 и 10^6. Однако в целых числах мы также имеем границы 0 и 10^5. Тогда сложим показатели степеней и посчитаем $11 \times log_2 10 \approx 36$.

Сложность решения: $log 10^{11} \approx 36$

Решение

```
def f(x, c):
    return x ** 2 + x ** (0.5) - c
```

```
def solve(c):
    l, r = 0, 1E5
    for _ in range(35):
        m = (r + l) / 2
        if f(m, c) < 0:
            l = m
        else:
            r = m
    return (r + l) / 2


c = float(input())
print(f'{solve(c):.10f}')
```

8 H. Коровы — в стойла

Разбор

Для решения этой задачи мы можем просто перебрать все возможные значения расстояния d между коровами и для каждого из них проверить, является ли оно правильным. Асимптотика такого решения будет $O(N \times M)$, где M - максимально возможное расстояние между коровами.

Так как естественно, что расстояние - величина монотонно возрастающая, давайте искать подходящее расстояние двоичным поиском.

Как нам проверить, что коровы поместятся в стойла при выбранном расстоянии d? Поместим первую корову в первое стойло, запомним её позицию p и будем перебирать все следующие стойла до тех пор, пока $p + d > a_i$. Как только мы нашли подходящее стойло a_k, запомним новую позицию текущей коровы $p := a_k$ и будем перебирать стойла дальше.

Если в какой-то момент все коровы размещены, то вернём 1, иначе 0. Если нам удалось разместить всех коров, то сдвинем левую границу вправо (попробуем большую дистанцию), иначе сдвинем правую границу влево (попробуем меньшую дистанцию).

Асимптотика решения: $O(N \times \log M)$, где M - максимально возможное расстояние между коровами.

Решение

```
def ok(a, k, d):
    pos, cnt = a[0], 1
    for i in range(1, len(a)):
        if pos + d <= a[i]:
            pos = a[i]
            cnt += 1
        if cnt == k:
            return True
    return False


def solve(a, n, k):
    l, r = 0, max(a)
    while r - l > 1:
        m = (r + l) // 2
        if ok(a, k, m):
            l = m
        else:
            r = m
    if ok(a, k, r):
        return r
    return l
```

```
n, k = map(int, input().split())
a = list(map(int, input().split()))
print(solve(a, n, k))
```

9 I. Письма

Разбор

В этой задаче нам нужно научиться для заданного x быстро находить такой индекс i, что

$$\sum_{k=1}^{i} a_k \leq x < \sum_{k=1}^{i+1} a_k$$

Давайте предпосчитаем префиксные суммы номеров общежитий: $p_{k+1} = p_k + a_k$. Так как префиксные суммы упорядочены по возрастанию, то мы можем двоичным поиском найти необходимый индекс $p_i \leq x < p_{i+1}$, где i - это номер общежития, а затем посчитать номер комнаты в этом общежитии как $x - p_i$

Асимптотика решения: $O(M \times \log N)$

Решение

```
def solve(pref, d, n):
    l, r = -1, n
    while r - l > 1:
        m = (r + l) // 2
        if pref[m] < d:
            l = m
        else:
            r = m
    if l < 0:
        return 1, d - pref[r]
    return l + 1, d - pref[l]


n, m = map(int, input().split())
a = list(map(int, input().split()))
b = list(map(int, input().split()))

pref = [0] * (n + 1)
for i in range(n):
    pref[i + 1] = pref[i] + a[i]
for d in b:
    p, q = solve(pref, d, n)
    print(f'{p} {q}')
```