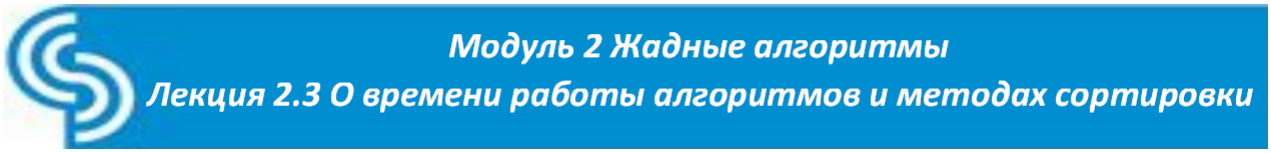

Эта лекция посвящена двум вспомогательным вопросам. Первый из них — оценка времени работы программ. Мы выясним, как определить, при каких ограничениях на исходные данные алгоритм будет работать за приемлемое время. На олимпиаде оценка времени работы позволит понять, какой из нескольких алгоритмов будет эффективнее, и поможет не тратить время на реализацию заведомо неэффективных решений.

Для оценки времени работы программы обычно оценивают количество выполняемых ею элементарных операций в зависимости от размера исходных данных, например, числа n. Чтобы не считать отдельно каждую операцию сложения, присваивания, сравнения и т.д., часто используют O-символику. Например, пишут, что программа выполняет $O(n^2)$ действий. Это значит, что количество выполняемых ею действий не превосходит Cn^2, где C — некоторая константа.

Решение задачи о расписании на рис. 1 выполняет O(nm) действий, где n — количество заказов, а m — максимальный дедлайн. Действительно, внешний цикл выполняет n шагов. На каждом его шаге выполняется проход по вектору used в цикле while. Его число итераций не больше, чем наибольший дедлайн, то есть m. Все остальные операции — элементарные, поэтому получаем, что общее время работы программы не превосходит константы, умноженной на n и на m. В ограничениях задачи $n, m \leqslant 5000$. Их произведение $nm \leqslant 25 \cdot 10^6$. Такое количество операций в условиях контеста (так часто называют олимпиады по программированию, от англ. contest) выполнится меньше, чем за секунду.

```
long long sum = 0;
for (int i = 0; i < n; i++)
{
    int k = d[i];
    while (k >= 1 && used[k])
        k--;
    if (k == 0)
        continue;
    used[k] = true;
    sum += c[i];
}
```

Рис. 1. Решение задачи о расписании

Отметим, что приведённая реализация нашего алгоритма — не самая эффективная. Его можно ускорить, если использовать специальные структуры данных, которые позволяют быстро искать в множестве максимальный элемент не больше заданного числа. Но этот вопрос уже выходит за рамки нашего

курса.

Перейдём к другому важному вопросу — к сортировке. В обеих рассмотренных задачах — задаче о размене и задаче о расписании — нужно сначала отсортировать исходные данные, упорядочить их по убыванию. Выделим основные аспекты, которые нужно знать про сортировку начинающему спортивному программисту.

В первую очередь, нужно знать какой-нибудь простой алгоритм сортировки за $O(n^2)$. Например, сортировку выбором (см. рис. 2).

```
for (int i = 0; i < n; i++)
{
    int mn = i;
    for (int j = i + 1; j < n; j++)
        if (a[j] < a[mn])
            mn = j;
    swap(a[i], a[mn]);
}
```

Рис. 2. Сортировка выбором за $O(n^2)$

Мы сначала находим в векторе a минимальный элемент и меняем его местами с первым элементом при помощи функции swap. Затем среди оставшихся элементов вектора снова находим минимальный элемент и меняем его местами со вторым, и т.д. На i-м шаге мы ставим на своё место i-й элемент. В итоге массив будет отсортирован по возрастанию. Внешний цикл выполняет n итераций, внутренний — не более n, поэтому суммарно будет не более Cn^2 действий.

Существуют более быстрые алгоритмы сортировки, которые работают за $O(n \log n)$. Обозначение $\log n$ используется для *логарифма*. Обычно при анализе алгоритмов возникает двоичный логарифм n, который обозначается как $\log_2 n$. Это такая степень k, в которую нужно возвести число 2, чтобы получить n:

$$\log_2 n = k \quad \Leftrightarrow \quad 2^k = n.$$

Например, $\log_2 8 = 3$, потому что $8 = 2^3$. Логарифм растёт достаточно медленно с ростом n: $\log_2 10^6 \approx 20$.

$O(n \log n)$ больше, чем $O(n)$, но значительно меньше, чем $O(n^2)$. В условиях контеста программы с такой вычислительной сложностью обычно работают для n примерно до 10^6. Это ещё зависит от константы, которую скрывает O-большое. Для $n = 10^6$ значение $n \log_2 n \approx 2 \cdot 10^7$.

Широко известный алгоритм сортировки, который работает за $O(n \log n)$, называется QuickSort, или быстрая сортировка. Он реализован в стандартных библиотеках большинства языков программирования. Например, в библиотеке стандартных шаблонов STL есть функция sort. Она содержится в заголовочном файле algorithm и ей передаются указатели на начало и конец диапазо-

на, который нужно сортировать. Для векторов есть специальные указатели, называемые итераторами. Чтобы сортировать все элементы вектора a по возрастанию, можно вызвать sort(a.begin(), a.end()). Здесь используются итератор a.begin(), указывающий на начало вектора, и a.end() — итератор, указывающий на конец вектора.

Если нужно сортировать более сложные объекты, можно создать структуру и определить для неё оператор $<$. На рис. 3 приведён код, который сортирует заказы с двумя параметрами — дедлайном и стоимостью — по убыванию стоимости.

```
struct Order
{
    int d, c;
};

inline bool operator<(const Order& o1, const Order& o2)
{
    return (o1.c > o2.c);
}
vector<Order> order;

sort(order.begin(), order.end());
```

Рис. 3. Сортировка заказов

Подведём итоги. Мы обсудили оценку времени работы программ с использованием O-большого. Также мы разобрали простейший алгоритм сортировки за $O(n^2)$ и использование функции sort языка C++ для быстрой сортировки. Этих знаний должно быть достаточно, чтобы реализовать решение задачи о расписании.