**Алгоритмы: вводный курс.**

Простые числа – это числа, у которых только два делителя – это единица и само число. Пример: 2,3,5,7,11,132,17,19 и т.д. Как я понимаю – все простые числа нечетные (кроме числа 2).

**Логарифм** – показатель степени, в которую нужно возвести число, называемое основанием, чтобы получить заданное число.

3х = 8

Такое уравнение сложно решить, т.к. х находится в пределах от 1 до 2, но ответ простой:

х = log38

Т.е. х это число, в которое нужно возвести 3, чтобы получить 8.

Получаем такой вывод: х - это **логарифм** восьми по основанию три.

В книге у нас будет такой синтаксис:

lg n - описывается как логарифм числа n по основанию 2, т.е. log2n

Рассмотрим занимательный пример:

n = 2х - log2 n - lg n

210 = 1024, а lg 1024 = 10

220 = 1048576, а lg 1048576 = 20

т.е. мы видим это любопытное соотношение величины числа и логарифма, логарифм семизначного числа равен всего лишь 20, вот такие вот интересные свойства логарифмов.

Задача: отсортировать 10 млн. чисел (107).

ПК А - мощный, выполняет 1010 команд/с.

ПК В – слабый, выполняет 107 команд/с.

Алгоритм ПК А вычисляет n чисел с помощью 2\*n2 команд.

Алгоритм ПК В вычисляет n чисел с помощью 50\*n\*lg n команд.

Посчитаем:

А = = 20 000 с.

В = = 1163 с.

Вот мы видим как, благодаря лучшему алгоритму, более слабый ПК опередил более быстрый.

**Глава 2.**

Опишем некоторые термины этой книги:

**Процедуры** – функции, методы в ЯП.

**Вызов** процедуры – это запуск процедуры на выполнение.

**Параметры** - входные данные процедур.

**Возврат** значения процедурой - это если процедура генерирует выходные данные, которые передаются вызывающему коду.

**Массив** данных – группирует данные в единую сущность.

**Индекс записи**  - позволяет получить **элемент** массива с указанным индексом.

У нас новая задача: нам необходимо найти на полке книгу Шолохова.

Решение с Алгоритом-1:

Принимаем полку книг за массив A[i] с n элементами. Книг может быть даже несколько (нам ведь может нравиться несколько его произведений).

Т.е. нам нужно получить значение х = A[i] или значение индекса i .

Считать будем слева направо, по порядку – такой метод называется **линейным поиском.**

Используем **переменную** answer, с начальным значением NOT-FOUND, затем будем перебирать массив от A[1] до A[n] и если A[i]=x (книга Шолохова найдена), будем **присваивать** answer = i .

На выходе вернуть значение answer.

Т.о. в результате мы либо получим последнее значение индекса i где нашеля х, либо получим значение NOT-FOUND если книг Шолохова на полке нет.

Такой перебор данных, по диапазону с выполнением неоднократных действий наз-ся – **циклом,** а каждое однократное выполнение действия – **итерацией** цикла.

Здесь, в нашем цикле, i - это **переменная цикла**. Наша проверка на равенство называется – **телом** цикла, а постоянное увеличение переменной i называется – **инкрементом.**

Т.о. озвучим, что - цикл вида «для i = 1 до n » (читается как - i от 1 до n) выполняет n итераций и n+1 проверку на превышение значения предела (ведь полка у нас заканчивается на n-й книге).

Кстати, мы можем изменить наш алгоритм-1, чтобы он возвращал значение i сразу же как только A[i] = x и не искал дальше (что гораздо логичнее), мы назовем его – Алгоритм-2.

Сейчас наш алгоритм-2 выполняет две проверки за каждый проход, первая – это проверка i <= n (чтобы не выйти за пределы полки) и вторая – это проверка на наличие самой книги.

Можно сделать так, чтобы за каждый шаг выполнялась только одна проверка на наличие книги.

Для этого мы должны сделать первую дополнительную проверку - мы можем сымитировать наличие книги Шолохова в самом конце полки, а затем когда дойдем до него, то нам останется только проверить – настоящая ли это книга или имитация и книг Шолохова на полке нет.

Вторая дополнительная проверка – это проверить не находится ли книга Шолохова действительно на последнем месте.

Новый ход алгоритма – Алгоритм-3:

- Сохраняем содержимое A[n] в переменную last (настоящее значение последней книги на полке), затем помещаем х в A[n], т.е. делаем x = A[n] (имитируем наличие книги Шолохова на последнем месте полки);

- Теперь, выполняем условие пока A[i] != x выполняем увеличение i на 1 (цикл с проверкой значения х);

- Как только условие из второго пункта становится ложным (т.е. A[i]=x) проводим третий ход - восстанавливаем A[n] из переменной last (см. первый шаг алгоритма);

- Если i < n (т.е. книга нашлась до конца полки) или A[n] = x (вот она действительная проверка значения, ведь мы вернули первоначальное значение A[n], т.е. настоящее значение книги в позиции A[n]), то возвращаем значение i .

- В противном случае вернуть NOT\_FOUND.

В любом случае этот алгоритм быстрее предыдущего, у нас здесь за одну итерацию цикла, в п.2 алгоритма, происходит одна проверка, а в конце всего две дополнительных.

**Описание времени работы алгоритма.**

Вернемся к алгоритму-1 и вычислим формулу времени его выполнения.