Лекция №12
по дисциплине
«ТЕОРИЯ АЛГОРИТМОВ»

СЛОЖНОСТЬ АЛГОРИТМОВ

Преподаватель:
Золотоверх Д.О.

АЛГОРИТМЫ ОТЛИЧАЮТСЯ

Некоторые алгоритмы можно выполнить за секунды.

Некоторые алгоритмы не выполнимы за все время жизни вселенной.

Один алгоритм при выполнении задачи может использовать всю доступную память, когда другой – при увеличении количества входных данных использует ее постоянное количество.

ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ

Сложность алгоритма зависит от задачи (от ее размера и природы).

Измеряется в количестве работы выполненной алгоритмом:

- количество циклов работы процессора;
- количество времени;
- количество памяти.

При расчете сложности необходимо учитывать размер задачи.

ПРИМЕР С ПОИСКОМ ЭЛЕМЕНТА В МАССИВЕ

Представим у нас есть отсортированный массив целых чисел;
Необходимо найти порядковый элемент определенного числа;

Для этого можно использовать две программы.

- Первая пересмотрит каждый элемент;
- Вторая использует трюк «разделяй и властвуй».

ПРОСМОТР КАЖДОГО ЭЛЕМЕНТА

Нахождение элемента производится путем перебора всех элементов массива.

Для оценки сложности учитывается худший сценарий.

Поэтому сложность алгоритма прямо пропорциональна сложности задачи (длина массива).

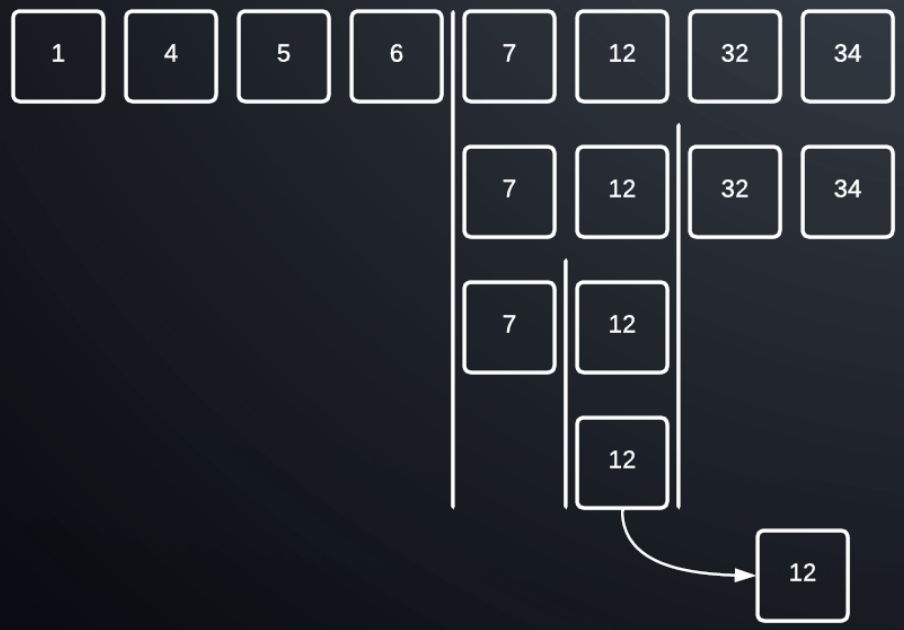
```
int findValue(
    int array[], int len, int number
) {
    int counter = 0;
    int index = -1;

    for (int i = 0; i < len; i++) {
        counter++;

        if (array[i] == number) {
            index = i;
            break;
        }
    }
    printf("It took %d operations\n", counter);
    return index;
}
```

МЕТОД РАЗДЕЛЯЙ И ВЛАСТВУЙ

Поиск элемента производится делением массива на две части, пока не находится искомое число.




```
int findValueBinary(
    int array[], int smallest, int highest, int number
) {
    int counter = 0;
    int index = -1;
    while (smallest <= highest) {
        counter++;
        int middle = smallest + (highest - smallest) / 2;

        if (array[middle] == number)
            index = middle;

        if (array[middle] < number)
            smallest = middle + 1;

        else
            highest = middle - 1;
    }
    printf("It took %d operations\n", counter);
    return index;
}
```

МЕТОД РАЗДЕЛЯЙ И ВЛАСТВУЙ

Особенностью такого алгоритма является то, что скорость роста его сложности как будто замедляется.

Например в худшем случае для нахождения элемента в массиве длинной в 8 эл. нам понадобится 9 операций, для 16 – 12.

Более того, если массив будет размером в 1000000 теоретически нам необходимо всего лишь в районе 60.

Также данный алгоритм можно выполнить рекурсивно.

```
int findValueBinaryR(
    int array[], int smallest,
    int highest, int number)
{
    if (highest >= smallest) {
        int middle = smallest + (highest - smallest) / 2;

        if (array[middle] == number)
            return middle;

        if (array[middle] > number)
            return binarySearch(
                 array, smallest,
                 middle - 1, number);

        return binarySearch(
            array, middle + 1,
            highest, number
        );
    }
}
```

Способ оценки сложности алгоритма

Измерять время выполнения не всегда целесообразно (разные процессоры, реализация разными ЯП и т.д.).

Рост, необходимой для выполнения, памяти зачастую не является проблемой.

Поэтому зачастую измерение сложности сводится к подсчету выполняемых операций в самом коде программы.

Способ оценки сложности алгоритма

Нам необходимо найти количество операций O;

Количество операций, как было описано раньше, зависит от сложности (размера) задачи *n*;

Иногда в коде происходят действия, количество которых не зависит от размера выполняемой задачи – константные значения;

Как говорилось ранее, подсчет совершается для худшего исхода (перебор всех значений, последнее место в массиве).

ОЦЕНКА СЛОЖНОСТИ ПОИСКА ПЕРЕБОРОМ

При запуске одна операция на присвоение значения индекса (счетчик нам не важен);

Цикл может быть вызван n-ое количество раз;

Внутри цикла одна операция – сравнение, и лишь в одном случае – одна операция.

Также в конце – одна операция возврата.

Таким образом имеем:

$O = 1 + n * 1 + 1 + 1$

$O = n + 4$

ОЦЕНКА СЛОЖНОСТИ ДВОИЧНОГО ПОИСКА

Функция точно так же начинается с присвоения значения.

Цикл while не так легко рассчитать, но в данном случае с учётом проверки и постоянным делением на два – максимально количество попыток равно: логарифм с основанием 2.

Каждый цикл происходит 3 операции.

Также в конце происходит возврат значения – одна операция.

$O = 1 + \log_2(n * 3) + 1$

$O = \log_2(3n) + 2$

АСИМПТОТИКА

Полный расчет сложности алгоритма может быть очень трудоемким.

Поэтому для его расчета применяется так называемое асимптотическое равенство.

Простыми словами – убираются малозначимые значения.

Когда задача становится очень большой – их значения незначимы.

Например:

$O = n + 4$ стремится к $O = n$

$O = \log_2(3n) + 2$ стремится к $\log_2(n)$

ТИПЫ АЛГОРИТМОВ

Существуют следующие типы сложности:

- O(1) – кол-во операций не растет с задачей;
- O(log n) – рост кол-ва замедляется с ростом задачи
- O(n) – рост кол-ва пропорционален росту задачи
- $O(n^2)$, $O(2^n)$, O(n!) – рост кол-ва ускоряется с простом задачи

График роста сложности алгоритмов от роста задачи

ПРИМЕРЫ АЛГОРИТМОВ

Сложность	Название алгоритма	Пример задачи
O(1)	Постоянный	Адресация, работа с хеш-таблицами, Работа с очередями
O(log n)	Логарифмический	Бинарный поиск (отсортированный список) Алгоритмы типа разделяй и властвуй
O(n)	Линейный	Перебор массива, Адресация связанного списка, Сравнение строк,
O(n log n)	Линейно-арифмический	Сортировки типа Merge Sort, Heap Sort, Quick Sort
$O(n^2)$	Квадратичный	Работа с двумерным массивом, Сортировки типа Bubble Sort, Insertion Sort, Selection Sort
$O(n^3)$	Кубический	Решение уравнений с 3 переменными
$O(k^n)$	Экспоненциальный	Нахождение всех подмножеств
O(n!)	Факториальный	Найти все перестановки заданного набора, Задача коммивояжера

СПАСИБО ЗА ВНИМАНИЕ!