



**ГУАП**

Государственный университет  
аэрокосмического приборостроения

[www.guap.ru](http://www.guap.ru)

---

# Информатика

**Рождественская Ксения  
Николаевна**

Кафедра 14

[ksu.khramenkova@gmail.com](mailto:ksu.khramenkova@gmail.com)



# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

$$x = (x_1, \dots, x_n) \quad x_i \in \{0, 1\}$$

Последовательность  $x$  –  $n$ -мерный двоичный вектор с элементами  $x_i$

~~$W(x)$  – вес вектора  $x$  – число его ненулевых элементов~~

Задач

а  
Найти вес двоичного  
вектора

$$x = (x_1, \dots, x_n)$$

# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

- **1 способ** – последовательное рассмотрение элементов вектора и сравнение их с нулем. Аналогично записи:

$$W(x) = \sum_{i=1}^n x_i$$

Это решение **методом перебора**.

Имея конечное число элементов, рассматриваем их один за другим, при этом выполняя сравнение с нулем.

# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

### Определения

- Обозначение  $O(\dots)$  является оценкой сложности алгоритма

$$f(n) = 2n + 1 \longrightarrow O(n)$$

$$f(n) = 3n^3 + 5n^2 + 2 \longrightarrow O(n^3)$$

---

$$f(n) = O(g(n)) \quad \begin{array}{l} \text{если} \\ \text{и} \end{array} \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = C > 0$$

$n$  – размерность

$g(n)$  – некоторая известная функция (линейная, степенная, логарифмическая и пр.)

$f(n)$  – сложность алгоритма (может рассматриваться число некоторых элементарных действий или объем данных)

# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

### Определения

- Анализ сложности с точки зрения  $O(\dots)$  позволяет лишь оценить скорость роста функции  $f(n)$ , т.е. нельзя определить точное значение числа шагов или ячеек памяти
- Такой анализ используется для сравнения двух алгоритмов, при оценке их реализуемости.

---

Возвращаемся к  
задаче

Найти вес двоичного  
вектора

$$x = (x_1, \dots, x_n)$$

# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

- **1 способ – метод перебора.** Требуется выполнить  $(n-1)$  сложений, размер требуемой памяти не изменяется с изменением  $n$ .

Т.е. сложность данного алгоритма  $O(n)$  по времени и  $O(1)$  по памяти

### Переформулируем постановку задачи

Существует ли более эффективный способ вычисления веса двоичного вектора, при котором сложность по времени будет меньше, чем  $O(n)$



# Оценка эффективности алгоритмов по памяти и времени

## памяти и времени

### Вычисление веса двоичного вектора

- **2 способ** – предвычисление веса для всех возможных наборов двоичных векторов длины  $n$ .

Адрес	$x=(x_1x_2x_3)$	$w(x)$
0	(000)	0
1	(001)	1
2	(010)	1
3	(011)	2
4	(100)	1
5	(101)	2
6	(110)	2
7	(111)	3

# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

- **2 способ** – предвычисление веса для всех возможных наборов двоичных векторов длины  $n$ .
- В общем случае объем такой таблицы составит  $2^n$  ячеек памяти, для вычисления вектора нужна одна операция – обращение к таблице.



- Снизили временную сложность до  $O(1)$
- Увеличили сложность по памяти до  $O(2^n)$

т.е. неравноценно **меняем время на память**

# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

- 3 способ
- Операция  $(x-1) \wedge x$ , где  $x$  – десятичное представление двоичного набора  $(x_1, \dots, x_n)$   
приводит к тому, что обнуляется самая правая единица, т.е. кол-во единиц становится на 1 меньше

```

10010100
-      1
-----
10010011
^
10010100
-----
10010000
  
```

Тогда решить задачу можно за  $W(x)$  шагов, каждый раз сравнивая результат вычисления с 0

Каждый шаг это 4 операции: вычитание, побитовое умножение, увеличение счетчика, сравнение с нулем

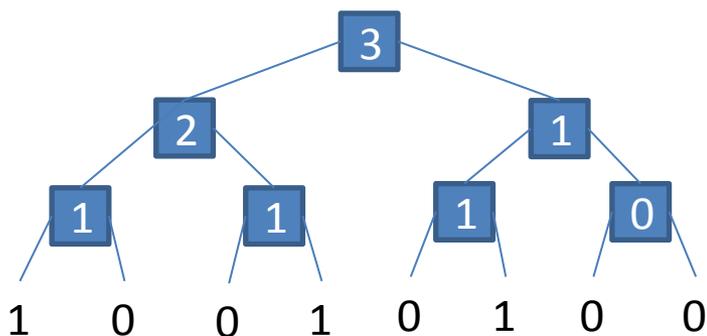
Сложность такого алгоритма  $O(W(x))$  по времени и  $O(1)$  по памяти

# Оценка эффективности алгоритмов по памяти и времени

## памяти и времени

### Вычисление веса двоичного вектора

- **4 способ**
- переборный по времени алгоритм (1 способ) требовал  $(n-1)$  сложение



**Листья** – элементы вектора

**Корень** – сумма всех элементов

Появляется идея уменьшения числа сложений



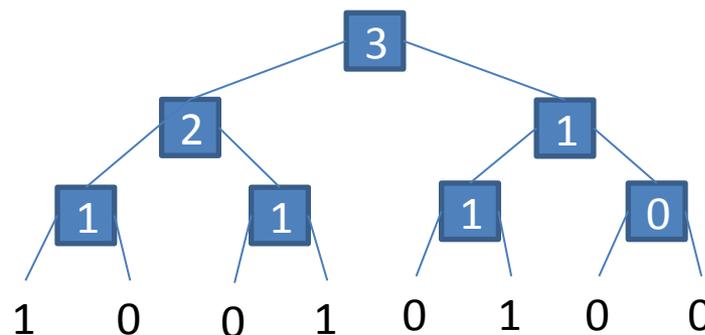
Уменьшение сложности вычислений

# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

- Пусть  $x = 10010100$ 
  - Складываем элементы на соседних позициях

Выделим из  $x$  позиции, имеющие **четные** и **нечетные** номера:  
 умножим побитово вектор  $x$  на двоичную маску



$$m_1^{нч} = 01|01|01|01$$



$$x_1^{нч} = 00|01|01|00$$

$$m_1^ч = 10|10|10|10$$



$$x_1^ч = 01|00|00|00$$

и сдвинем результат на 1 бит вправо

# Оценка эффективности алгоритмов по памяти и времени

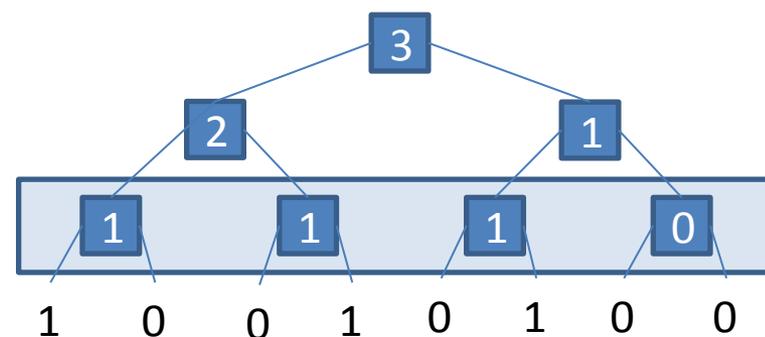
## Вычисление веса двоичного вектора

### Вычисление веса двоичного вектора

- Пусть  $x = 10010100$ 
  - Сумма десятичных представлений  $x^{HC}_1$  и  $x^C_1$  даст вектор  $x_1 = 01|01|01|00$

В десятичном представлении это последовательность

$(1|1|1|0)_{10}$



За одно десятичное сложение  $n$ -битных чисел выполнили  $n/2$  сложений однобитных чисел

# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

- Пусть  $x = 10010100$ 
  - Аналогично за одно сложение можно вычислить результат второго уровня

$$m_2^{HЧ} = 0011|0011$$



$$x_2^{HЧ} = 0001|0000$$

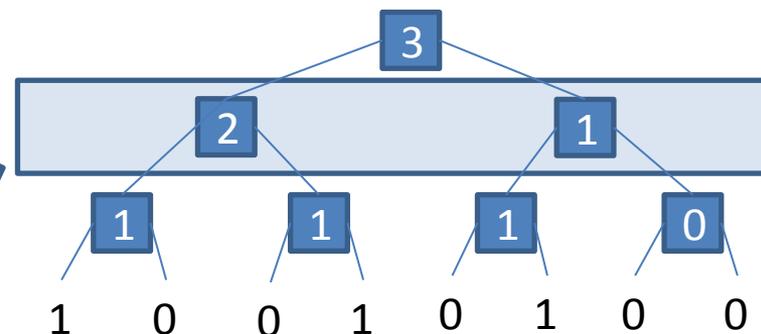
$$m_2^Ч = 1100|1100$$



$$x_2^Ч = 0001|0001$$

и сдвинем результат на 2 бита вправо

$$x_2 = 0010|0001 = (2|1)_{10}$$



# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

- Пусть  $x = 10010100$ 
  - Последний шаг

$$m_3^{HЧ} = 00001111$$



$$x_3^{HЧ} = 00000001$$

$$m_3^У = 11110000$$



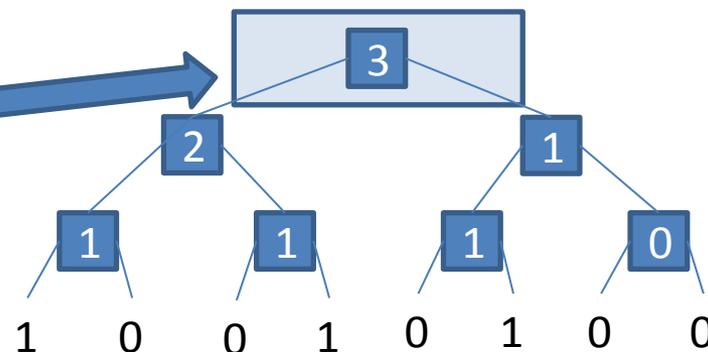
$$x_3^У = 0000010$$

и сдвинем результат на 4 бита вправо

$$x_3 = 00000011 = 3_{10}$$

**Ответ:  $W(x) =$**

**3**





# Оценка эффективности алгоритмов по памяти и времени

## Вычисление веса двоичного вектора

- Один из часто используемых методов построения алгоритмов – **декомпозиция**
- На примере 4го способа: структура дерева; если разобьем вектор на любое число частей, то вес вектора будет равен сумме весов этих частей.
  - **Уровни дерева** – это разбиения числа вектора  $x$  на 2, 4, 8 и т.д. частей. Такое разбиение можно проводить до тех пор, пока веса полученных подвекторов не смогут быть вычислены с помощью простой процедуры, т.е. ищем подзадачу такой размерности, которая имеет простое решение