Сортировка

Пусть есть последовательность $a_0, a_1 \dots a_n$ и **функция сравнения**, которая на любых двух элементах последовательности принимает одно из трех значений: меньше, больше или равно.

Задача сортировки

перестановка членов последовательности таким образом, чтобы выполнялось условие: $a_i <= a_{i+1}$, для всех i от 0 до n.

Сложные данные

Данные состоят из нескольких полей:

struct element { field x; field y; }

Пусть значение функции сравнения зависит только от поля x

Поле **x** называют ключом, по которому производится сортировка.

Пример

Фамилия	История	Информатика
Алексеев	5	4
Андреев	4	5
Балашова	3	5
Васильев	5	5
Воронина	3	4
Гольцова	4	3
Градов	4	3
Донцов	3	3
Дорофеева	5	5

Фамилия	История	Информатика
Алексеев	5	4
Васильев	5	5
Дорофеева	5	5
Андреев	4	5
Гольцова	4	3
Градов	4	3
Балашова	3	5
Воронина	3	4

Фамилия	История	Информатика
Васильев	5	5
Дорофеева	5	5
Андреев	4	5
Балашова	3	5
Алексеев	5	4
Воронина	3	4
Гольцова	4	3
Градов	4	3

Характеристики сортировки

Вычислительная сложность алгоритма - функция, определяющая зависимость объёма работы, выполняемой некоторым алгоритмом, от размера входных данных -

f(n)

Объём работы = { Время , Память }

Количество элементарных шагов

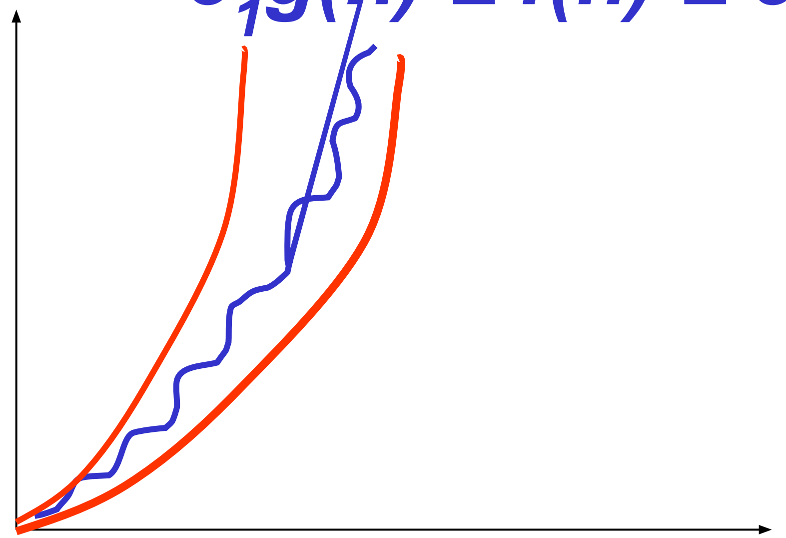
Объем памяти

Поиск точной функции, оценивающей временную сложность – трудоемкая задача

ОЦЕНКИ ФУНКЦИИ СЛОЖНОСТИ АЛГОРИТМА

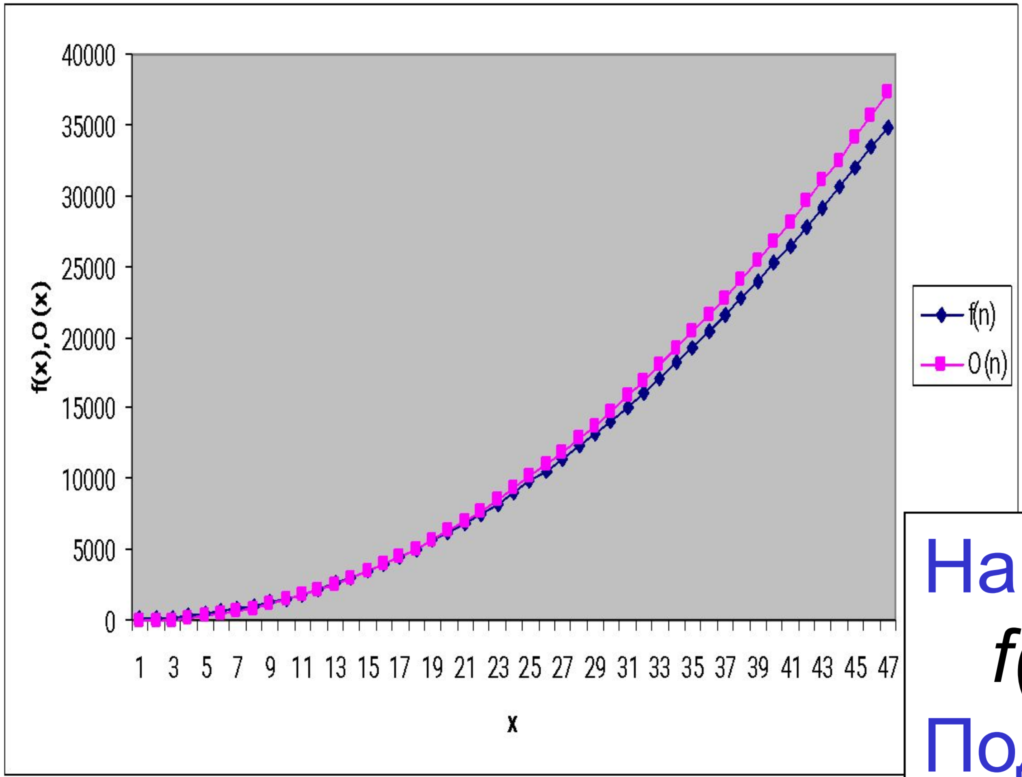
Θ(g(n)) - если $\exists$ $c_1>0$, $c_2>0$, $n_0>0$ **:**

$$c_1 g(n) \le f(n) \le c_2 g(n), \ \forall\ n>n_0$$



$O(g(n))$ - **если** $\exists\ c>0,\ n_0>0:$

$$0 \le f(n) \le cg(n),\ \forall\ n > n_0$$



Например:

$f(n) = n^2 + 5n + 100$

Подберем

$O(n) = 1.1n^2$

O-функция – верхняя асимптотическая оценка трудоемкости алгоритма

Если получена верхняя асимптотическая оценка *f(n),* то говорят, что класс функций *f(n)* растет не быстрее, чем функция *g(n)* с точностью до постоянного множителя

Ω-функция – нижняя асимптотическая оценка трудоемкости алгоритма

Ω(g(n)) - если ∃ $c>0, n_0>0$:

$$0 \le cg(n) \le f(n), \forall\ n > n_0$$

Порядок функции *f(n)* - $O(N^2)$

О – функции выражают относительную скорость алгоритма в зависимости от некоторой переменной (или переменных).

Правила преобразования

1. $O(k*f)=O(f)$
2. $O(f*g)=O(f)*O(g)$ или $O(f/g)=O(f)/O(g)$
3. $O(f+g)$ равна доминанте $O(f)$ и $O(g)$

$$O(1{,}5*N) = O(N)$$

$$O((17*N)*N) = O(17*N)*O(N)$$

$$= O(N)*O(N) = O(N*N) =$$

$$= O(N^2)$$

$$O(N^5+N^2) = O(N^5)$$

Типы сложности алгоритма

- $O(1)$ – константная сложность
- $O(n)$ – линейная сложность
- $O(n^k)$, $k = 2,3,4,\dots$ полиномиальная сложность
- $O(\log N)$ – логарифмическая сложность
- $O(N*\log N)$
- $O(2^n)$ – экспоненциальная сложность

```
int y,n;
float x;
y = scanf(“%d”,&n);
if (n==0)
            { printf(“Введены неверные данные…\n ”);
            printf(“Для продолжения нажмите любую
              клавишу…”);
              getch();
              exit(0);}
else {
x = 25.5*sin(2*n)/n;
printf (“Значение x = %8.2f”, x);}
}
```

O(1)+O(1)   O(1)   O(1)   O **O(1)** ов)   *Доминанта* O(*true*) *и* O(*false*)

Алгоритмы без циклов и рекурсивных вызовов имеют константную сложность.

Оцените сложность алгоритма поиска минимального элемента квадратной матрицы размерности *n*.

```
…
int  n,min = x[0][0];
for(int i=0;i<n;i++)
    for(int j = 0;j<n;j++)
        if (x[i][j] < min) min = x[i][j];
…
```

$O(1)$ +

$O(N)$ *

$O(N)$*

O(выч. условия) +

O(выч. выражения)

$O(N^2)$

ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМА НАХОЖДЕНИЯ СУММЫ n ЭЛЕМЕНТОВ РЯДА

$$\sum_{n=1}^{\infty} \frac{-1^{n} \cdot x^{2n+1}}{n!(2n+1)}$$

I вариант:

```
int n = 10;
float x = 2;
double q,q1;
double S = 0;
for(int i=1;i<=n;i++)
{
q = pow(-1,i);
q1 = pow(x, 2*i+1);
q = q*q1;
S+=q/(fact(i)*(2*i+1));
}
```

O(1) O(1) O(1) O(1) O(1)

O(n)* O(n)

O(i)+O(2i+1)+O(1)+O(i)

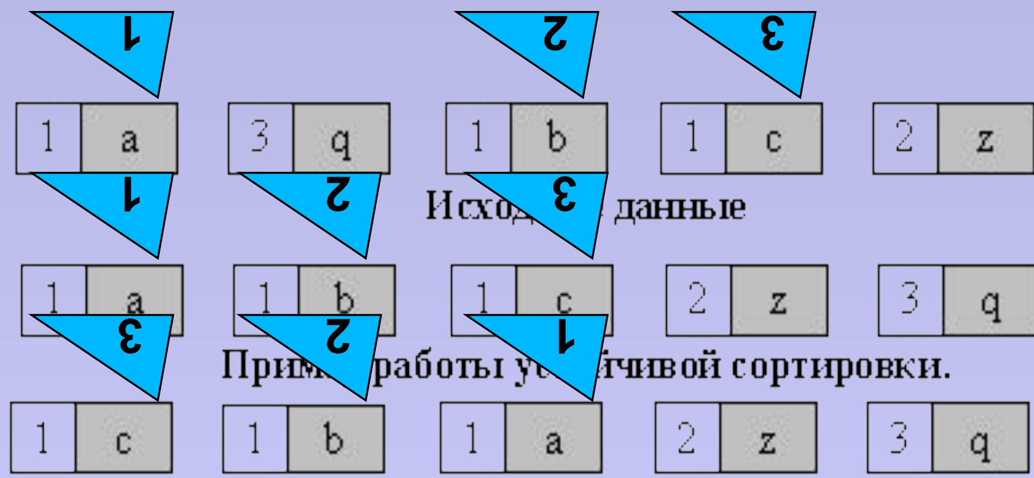
Среднее O(2n+1) ≈ O(n)

II вариант

```
int n = 10;
float x = 2;
double q,q1;
double S = 0;
q = x*x*x/3;
for(int i=1;i<=n;i++)
{
if(i%2) S-=q; else S+=q;
q = (q*x*x*(2*i+1))/((i+1)*(2*i+3));
}
```

O(n)

Устойчивость - устойчивая сортировка не меняет взаимного расположения равных элементов.



Пример работы неустойчивой сортировки.

Естественность поведения - эффективность метода при обработке уже отсортированных, или частично отсортированных данных.

Вставка

?

Выбор

?

Сфера применения

Внутренние

работают с данными в оперативной памяти с произвольным доступом

Внешние

упорядочивают информацию, расположенную на внешних носителях.

Способы сортировки

Квадратичные и субквадратичные алгоритмы:

- Сортировка выбором (*SelectSort*)
- Сортировка вставками *(InsertSort)*
- Сортировка обменом *(BubbleSort)*
- Сортировка Шелла *(ShellSort)*
- Комбинированная сортировка *(CombSort)*

Логарифмические и линейные алгоритмы

- Пирамидальная сортировка (HeapSort)
- Сортировка Хоара (QuickSort)
- Поразрядная сортировка (RadixSort)

Сортировка обменом

Сортировка сравнивает пары рядом стоящих элементов и меняет их местами, если значение первого элемента из пары больше значения второго элемента.

1. 4 7 8 5 2
2. 4 7 5 2 8
3. 4 5 2 7 8
4. 4 2 5 7 8
5. 2 4 5 7 8

- Алгоритм содержит 2 цикла
- На каждом шаге внутреннего цикла самый «большой» элемент занимает «свое» место в массиве
- На каждом шаге внутреннего цикла самый маленький элемент передвигается ровно на одну позицию к «своему» месту

Оптимизация алгоритма

1. Фиксировать уже поставленные на свои места элементы
2. Проверять, был ли выполнен хотя бы один обмен во внутреннем цикле
3. Просматривать массив в двух направлениях

Характеристики алгоритма

Классическая реализация

Лучший	Средний	Худший
$M = 0$	$M = 3(n^2 - n)/4$	$M = 3(n^2 - n)/2$
$C = (n-1)n/2$	$C = (n-1)n/2$	$C = (n-1)n/2$

С проверкой обменов

Лучший	Средний	Худший
$M = 0$	$M = 3(n^2 - n)/4$	$M = 3(n^2 - n)/2$
$C = n-1$	$C = (n-1)n/4$	$C = (n-1)n/2$

Оценка временной сложности алгоритма

от $O(n)$ *до* $O(n^2)$

Естественность

естественная

Устойчивость

устойчива

Сортировка выбором

При первом просмотре элементов массива ищется индекс *минимального* элемента *k*, элемент с индексом *k* меняется местом с первым элементом.

При втором просмотре ищется индекс *минимального* элемента k среди оставшихся, элемент с индексом *k* меняется местом со вторым элементом.

На *i*-том просмотре ищется индекс *минимального* элемента *k* среди элементов с номерами от *i* до *n* и меняются местами элементы с номерами *i и k.*

4 7 8 5 2 $i=0$ $k = 4$ $\rightarrow$ 2 7 8 5 4
2 7 8 5 4 $i=2$ $k = 5$ $\rightarrow$ 2 4 8 5 7
2 4 8 5 7 $i=3$ $k = 4$ $\rightarrow$ 2 4 5 8 7
2 4 5 8 7 $i=4$ $k = 5$ $\rightarrow$ 2 4 5 7 8

Характеристики алгоритма

Классическая реализация

Лучший	Средний	Худший
M = 3(n-1)	M =3(n-1)	M =3(n-1)
C = (n-1)n/2	C = (n-1)n/2	C = (n-1)n/2

С проверкой индексов

Лучший	Средний	Худший
M = 0	M =3n/2	M =3(n-1)
C = (n-1)n/2	C = (n-1)n/2	C = (n-1)n/2

Оценка временной сложности алгоритма

$O(n^2)$

неестественная

неустойчива

Сортировка вставками

Полагается *i = 1*, считается что массив от 0 до *i* -1 *упорядочен*. В этом упорядоченном массиве ищется место для *x[i]*. В дальнейшем, значение *i* увеличивается на единицу и алгоритм повторяется.

4 7 8 5 2	i=1	→	***4 7*** 8 5 2
4 7 8 5 2	i=2	→	***4 7 8*** 5 2
4 7 8 5 2	i=3	→	***4 5 7 8*** 2
4 5 7 8 2	i=4	→	***2 4 5 7 8***

Оптимизация алгоритма

1. Использование сторожевого элемента

    1.1. Ищется минимальный элемент , выставляется в позицию 0.

    1.2. Размер массива увеличивается на 1 , элементы массива располагаются, начиная с 1-го элемента, на каждом шаге сортировки в 0 позицию выставляется вставляемый элемент

2. Поиск места с помощью бинарного деления

Алгоритм сортировки бинарными вставками

```
1. для i от 2 до n
   если a[i-1]>a[i] то
    x:= a[i];
    left:= 1;
    right:= i-1;
    пока left<=right
       sred:= (left+right) / 2;
       если a[sred]<x то left:= sred+1
                         иначе right:= sred-1;
    для j:= i-1 до left шаг -1
      a[j+1]:= a[j];
    a[left]:= x;
```

Характеристики алгоритма

Классическая реализация

Лучший	Средний	Худший
$M = 2(n-1)$	$M = 1/4\ (n^2+9n-10)$	$M = 1/2\ (n^2+3n-4)$
$C = 2(n-1)$	$C = 1/4\ (n^2+n-2)$	$C = 1/2\ (n^2+n-2)$

Со сторожевым элементом (I)

Лучший	Средний	Худший
$M = 2(n-1)$	$M = 1/4\ (n^2+9n-10)$	$M = 1/2\ (n^2+3n-4)$
$C = n-1$	$C = 1/8\ (n^2 + n - 2)$	$C = 1/4\ (n^2 + n -2)$

Со сторожевым элементом (II)

Лучший	Средний	Худший
M = 0	M =(n-1)(n+3)/2	M = (n-1)(n+3)
C = n-1	C = (n+1)n/4	C = (n+1)n/2

Бинарными вставками

Лучший	Средний	Худший
M = 0	M =(n-1)(n+3)/2	M = (n-1)(n+3)
C = n-1	C = $N/2*\log_2 N$	C = $N*\log_2 N$

Естественная

Устойчивая

Сортировка Шелла

Сортирует элементы массива, отстоящие друг от друга на заданный интервал .

После того, как все элементы массива, отстоящие друг от друга на будут отсортированы, интервал изменяется по правилу $H_{i+1}=(H_i - 1)/2$ для массивов, содержащих более 500 элементов и $H_{i+1}=(H_i-1)/3$ для массивов, содержащих менее 500 элементов. За H_0 принимается число элементов массива. Метод заканчивает работу, когда становится меньше 1. Модификация сортировки вставками

Сортировка Шелла

Например, для массива из 7 элементов:

- 23 12 43 54 83 11 2 □ 23 12 43 54 83 11 2

23 12 43 54 83 11 2 □ 23 12 43 54 83 11 2

23 12 43 54 83 11 2 □ 23 12 43 54 83 11 2

23 12 43 **54** 83 **11** 2 □ 23 12 43 11 83 54 2

23 12 43 54 **83** 11 **2** □ 23 12 43 11 2 54 83

Так как обмены при первом проходе были, то шаг остается прежним:

- 23 12 43 11 2 54 83 □ 23 12 43 11 2 54 83

23 **12** 43 **11** 2 54 83 □ 23 11 43 12 2 54 83

23 11 **43** 12 **2** 54 83 □ 23 11 2 12 43 54 83

23 11 2 12 43 54 83 □ 23 11 2 12 43 54 83

23 11 2 12 43 54 83 □ 23 11 2 12 43 54 83

Сортировка комбинированная

Комбинация пузырька и сортировки Шелла. На каждом шаге сравниваются значения отстоящие друг от друга на заданное значение шага $H_{i+1}=8H_i/11$, но такое сравнение происходит всего один раз. Как только значение смещения становится равным 1, выполняется сортировка до конца методом пузырька. За принимается число элементов массива.

Сортировка комбинированная

23 12 43 54 83 11 H_0=7 H_1= (7*8)/11=5

23 12 43 54 83 **11** 2 □ 11 12 43 54 83 23 2

11 **12** 43 54 83 23 **2** □ 11 2 43 54 83 23 12

H_2= (5*8)/11=3

11 2 43 54 83 23 12 □ 11 2 43 54 83 23 12

11 2 43 54 83 23 12 □ 11 2 43 54 83 23 12

11 2 **43** 54 83 **23** 12 □ 11 2 23 54 83 43 12

11 2 23 **54** 83 43 **12** □ 11 2 23 12 83 43 54

H_3= (3*8)/11=2

11 2 23 12 83 43 54 □ 11 2 23 12 83 43 54

11 2 23 12 83 43 54 □ 11 2 23 12 83 43 54

11 2 23 12 83 43 54 □ 11 2 23 12 83 43 54

11 2 23 12 83 43 54 □ 11 2 23 12 83 43 54

11 2 23 12 **83** 43 **54** □ 11 2 23 12 54 43 83

Пирамидальная сортировка

Пирамида – это частично упорядоченное двоичное дерево, элементы которого расположены в узлах дерева по следующему правилу – каждый элемент родительского узла обязательно больше элементов, расположенных в дочерних узлах.