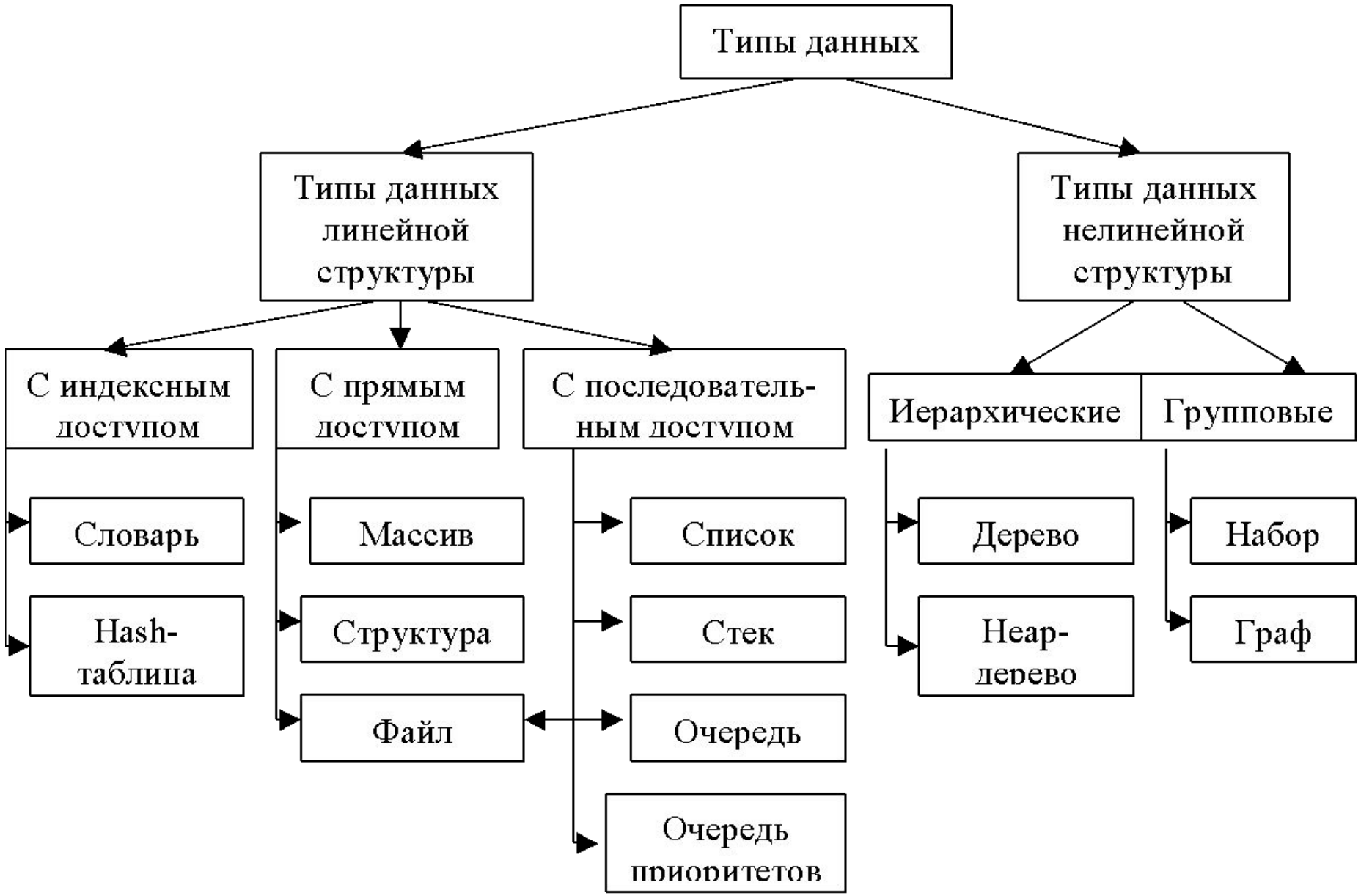
Структуры и алгоритмы обработки данных

- Данные;
- Структура данных;
  - Данные статической структуры;
  - Данные динамической структуры;
- Типы данных линейной структуры;
- Типы данных нелинейной структуры.


Типы данных
Типы данных линейной структуры
Типы данных нелинейной структуры
С индексным доступом
С прямым доступом
С последователь-ным доступом
Иерархические
Групповые
Словарь
Hash-таблица
Массив
Структура
Файл
Список
Стек
Очередь
Очередь приоритетов
Дерево
Heap-дерево
Набор
Граф

Анализ алгоритмов

Факторы, влияющие на время выполнения программы:

- Ввод исходной информации;
- Качество скомпилированного кода;
- Машинные инструкции, используемые для выполнения программы;
- Временная сложность алгоритма.

$T(n)$ $O(n^2)$

c, n_0 $\forall n >= n_0, T(n) <= cn^2$

Пример 1:

$T(n)=(n+1)^2$.

$T(0)=1$, $T(1)=4$.

Положим $n_0=1$, $c=4$.

$T(n)$ имеет порядок $O(n^2)$,

$n\geq 1$, $(n+1)^2\leq 4n^2$.

Пример 2:

$T(n)=3n^3+2n^2$.

$T(0)=0$, $T(1)=5$.

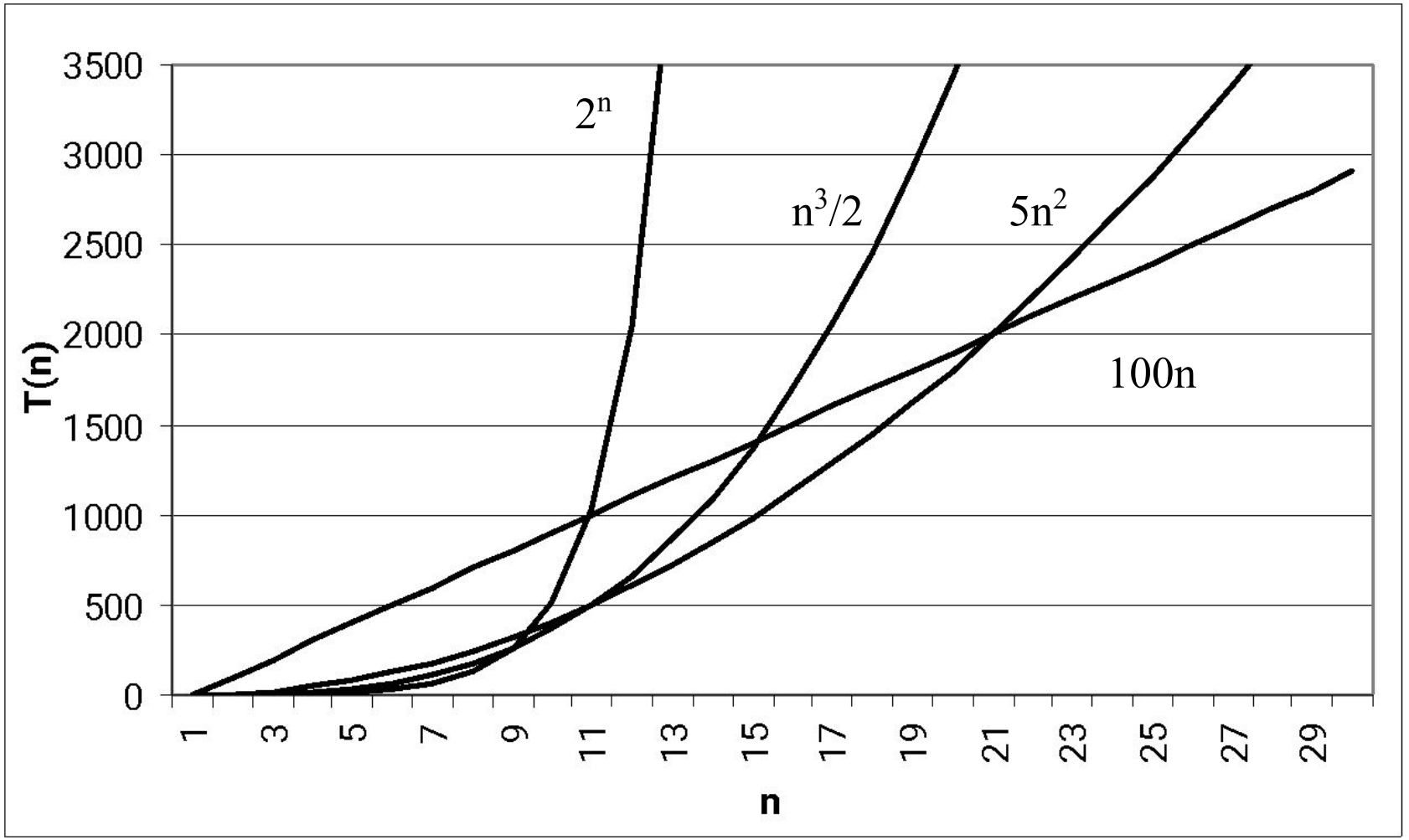
Положим $n_0=0$, $c=5$.

$T(n)$ имеет порядок $O(n^3)$,

$n\geq 0$, $3n^3+2n^2\leq 5n^3$.

Сложность	Описание	Форма повторения
O(1)	Алгоритм константной .сложности	Присваивание. Простое выражение.
O(N)	.Линейный	Нахождение максимального элемента
$O(N^2)$	Квадратический	for (i=0; i<N; i++) for (j=0; j<N; j++) …
$O(N^3)$	Кубический	
$O(N\log_2 N)$ $O(\log_2 N)$	Логарифмический	Бинарный поиск.
$O(a^N)$	NP - сложные .(неполиномиальные) Частный случай - экспоненциальная .сложность $O(2^N)$	Используются при неоднократном поиске дерева решений.

Функции времени выполнения для программ с различной временной сложностью

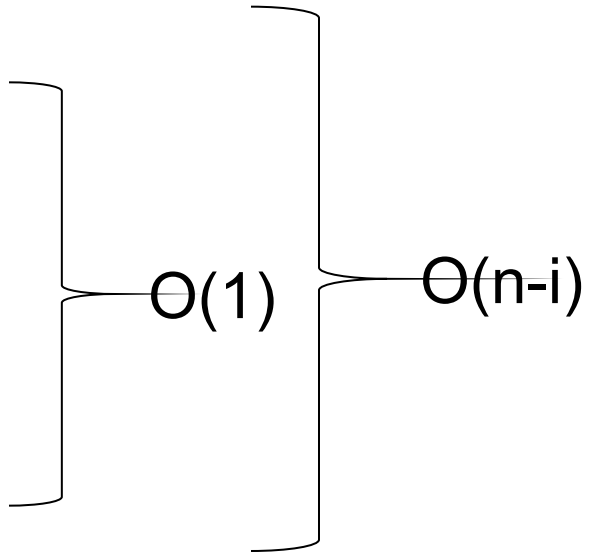


N	$\log_2 N$	$N\log_2 N$	N^2	N^3	2^N
2	1	2	4	8	4
8	3	24	64	512	256
16	4	64	256	4096	65536

Правила определения порядка сложности:

1) $O(k*f)=O(f)$;
2) Правило произведений: $O(fg)=O(f)O(g)$;
3) Правило сумм: $O(n^5+n^3+n)=O(n^5)$

```
(1) for (int i=0; i<n-1; i++)
(2) for (int j=n-1; j>i; j--)
(3)  if (a[j-1]>a[j]) {
(4)       temp=a[j-1];
(5)       a[j-1]=a[j];
(6)       a[j]=temp;
          }
```



$$\sum_{i=1}^{n-1}(n-i) = n(n-1)/2 = n^2/2 - n/2$$

$O(n^2)$

Анализ рекурсивных программ

```
    int fact(int n)
(1) { if (n<=1) return 1;
(2)       else return n*fact(n-1);
    }
```

T(n)

(1) - O(1)

(2) - O(1) + T(n-1)

$$T(n) = \begin{cases} c + T(n-1), \text{если } n > 1, \\ d, \text{если } n \le 1. \end{cases}$$

n>2, T(n) = 2c + T(n-2),

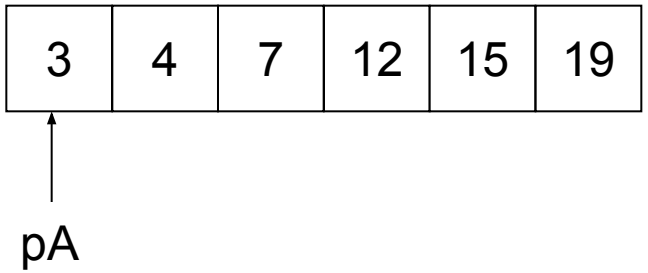
n>3, T(n) = 3c + T(n-3),

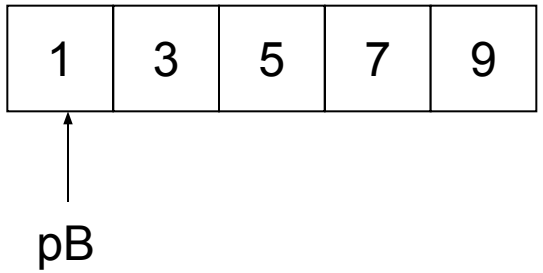
n>i, T(n) = ic + T(n-i),

i=n-1, T(n)=c(n-1)+T(1) = c(n-1) + d

O(n)

Слияние сортированных последовательностей





Последовательный поиск

```
int search(int a[ ],int n,int key)
{
for (int i=0;i<n;i++)
 if (a[i]==key) return i;
return -1;
}
```

Бинарный поиск

A(n), low=0, high=n-1.

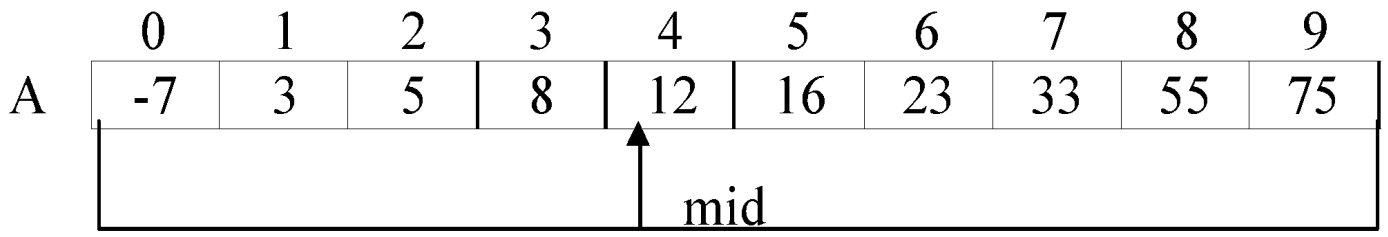
Алгоритм бинарного поиска:

- mid=(low+high)/2.
- A[mid]==key
- A[mid]<key
- A[mid]>key

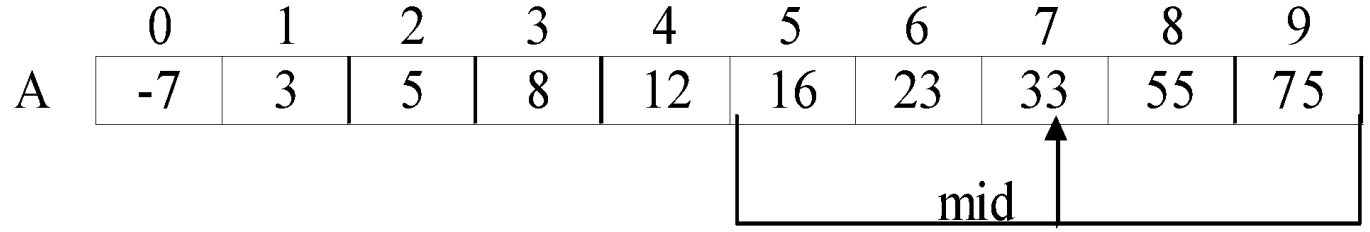
Пример:

A[10], key=16.

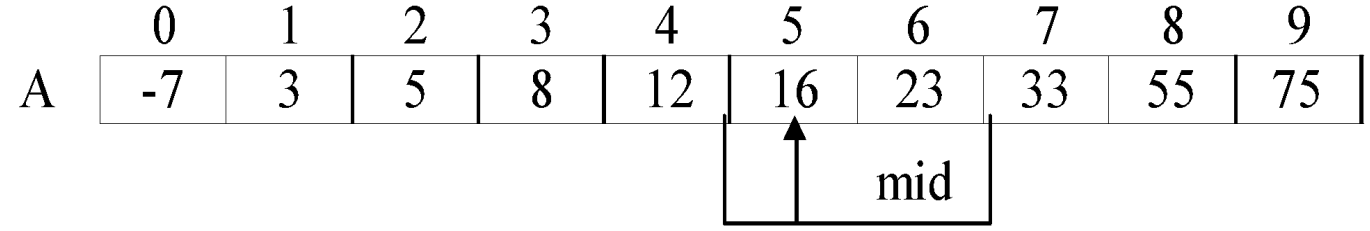
	0	1	2	3	4	5	6	7	8	9
A	-7	3	5	8	12	16	23	33	55	75



low = 0
high = 9
mid = 4
16>A[mid]



low = 5
high = 9
mid = 7
16<A[mid]



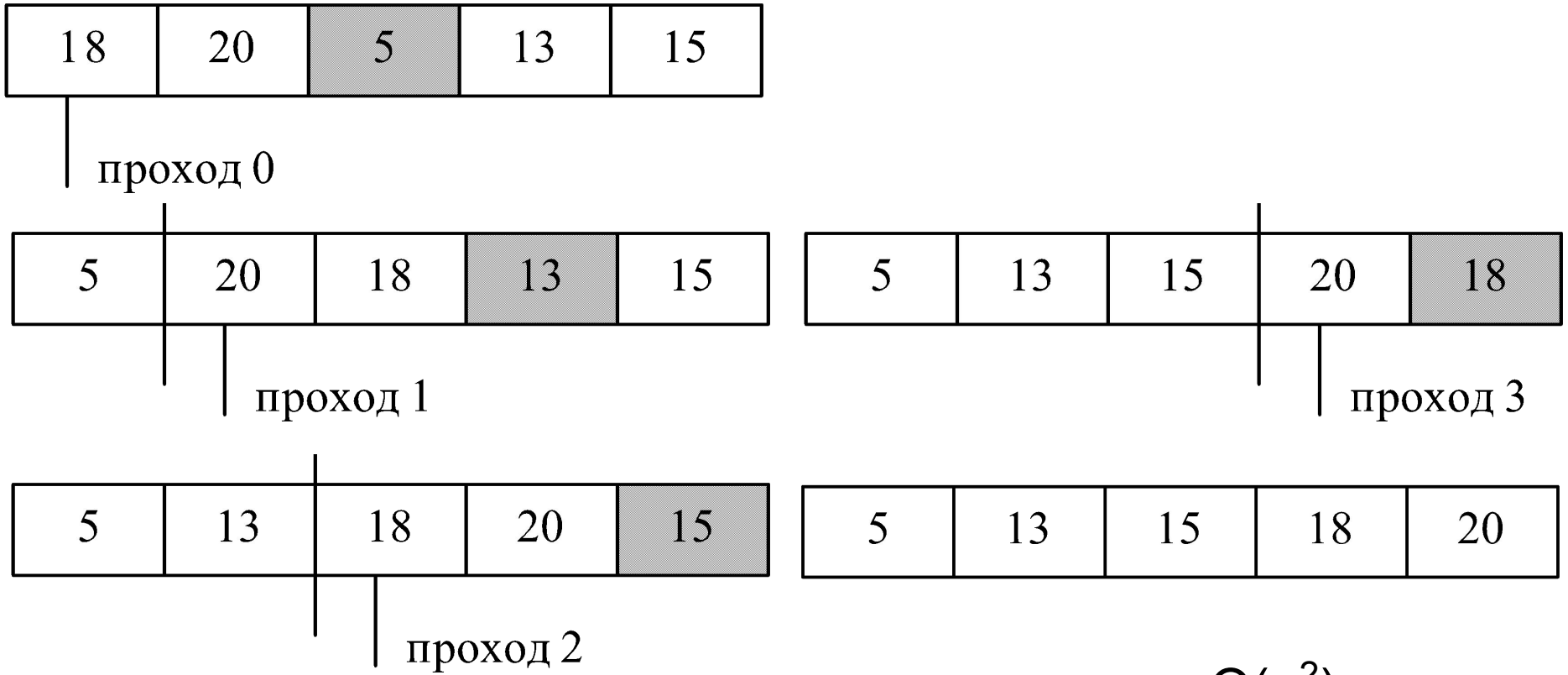
low = 5
high = 6
mid = 5
16=A[mid]

Алгоритмы сортировки массивов

Сортировка посредством выбора

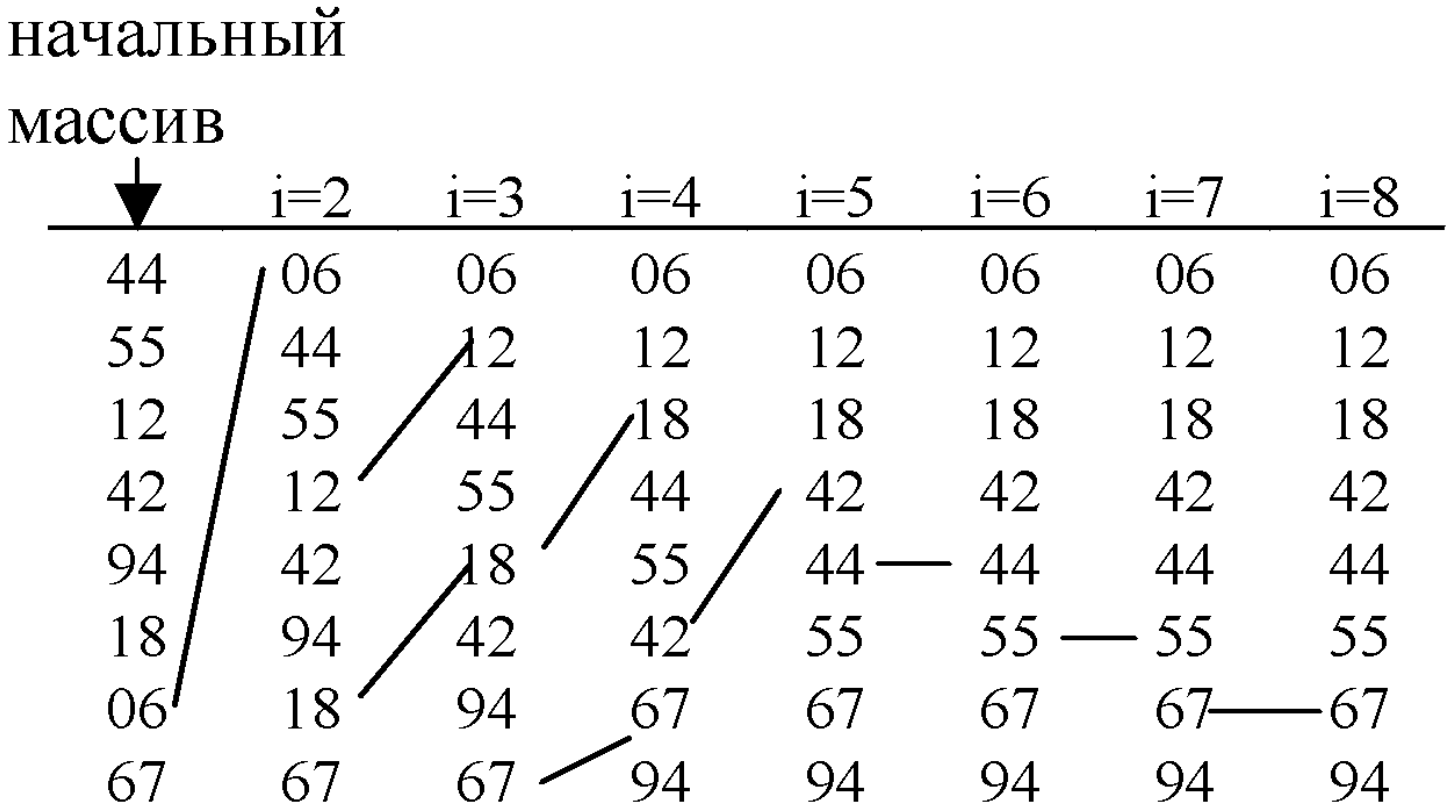
Пример.

Массив содержит целые числа 18, 20, 5, 13, 15.

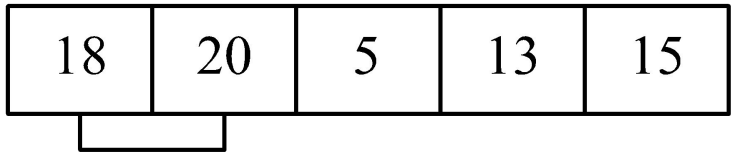


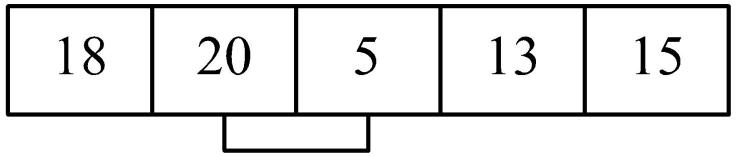
$O(n^2)$

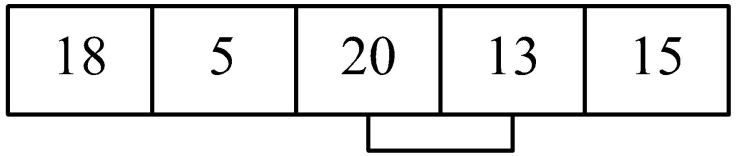
Сортировка обменом (пузырек)

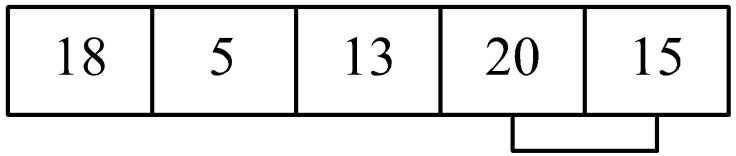


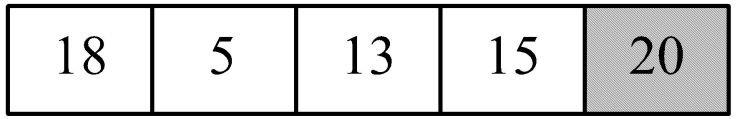
Проход 0





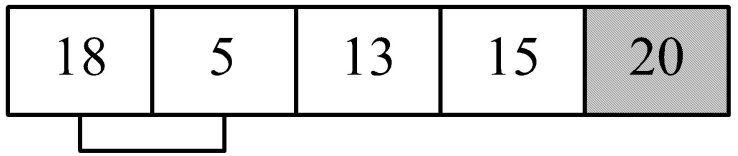


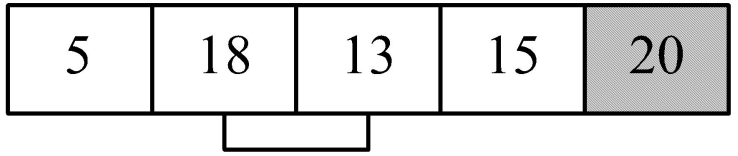


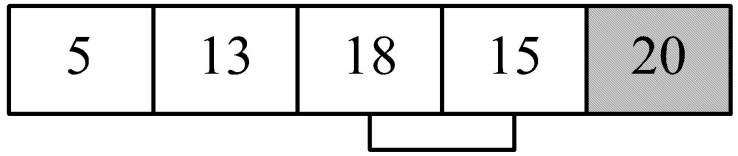


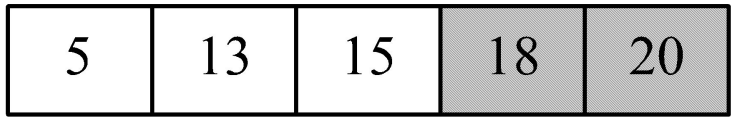
ilast=3

Проход 1



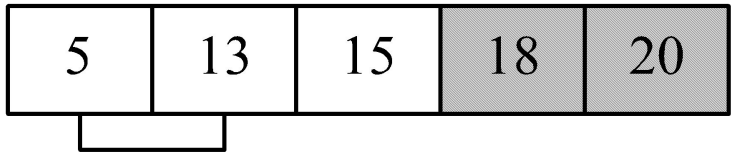


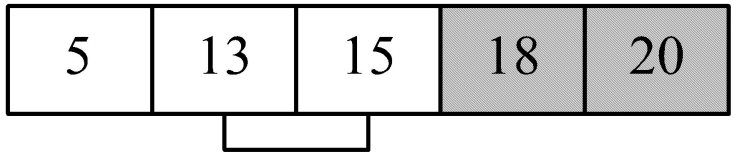




ilast=2

Проход 2

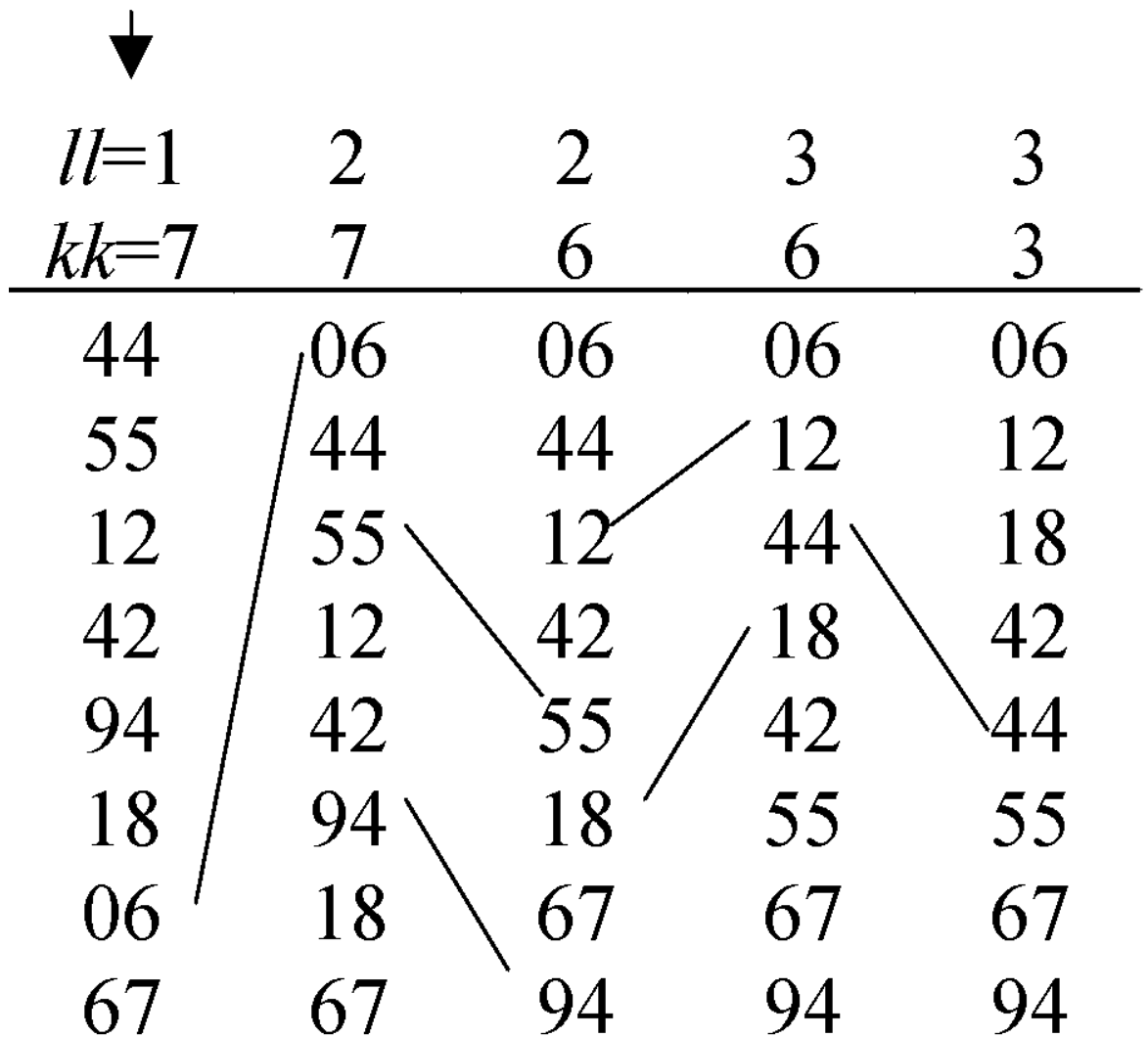




ilast=0

$O(n^2)$

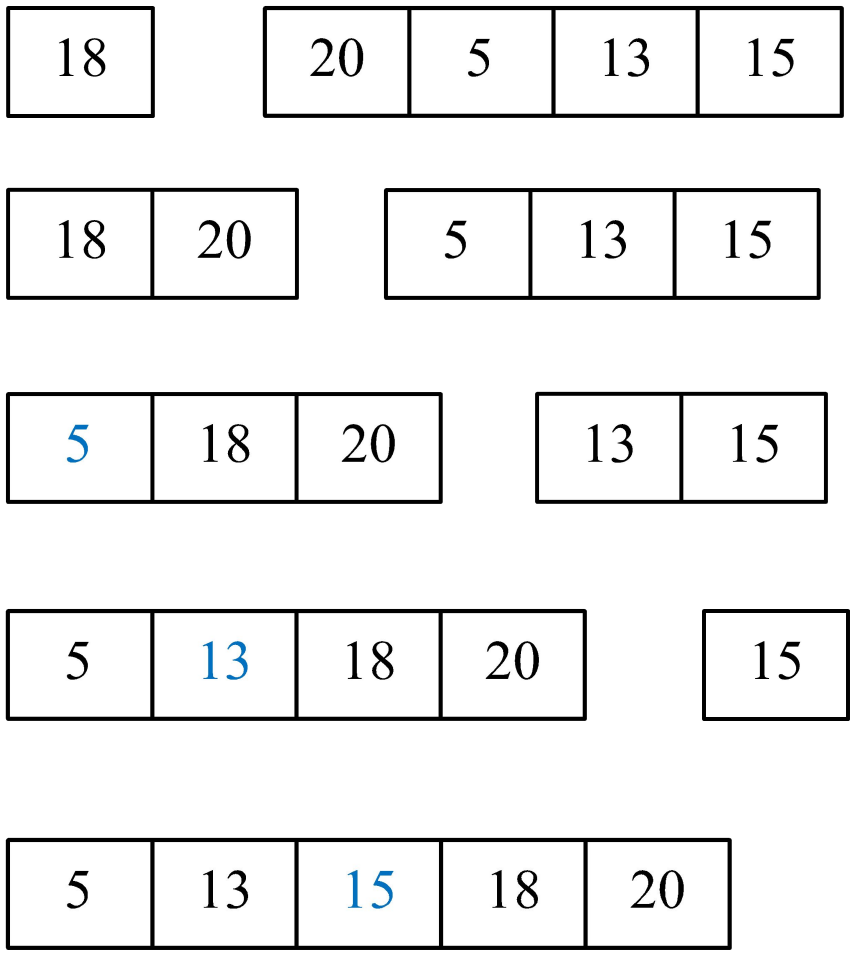
Шейкер-сортировка



$O(n^2)$

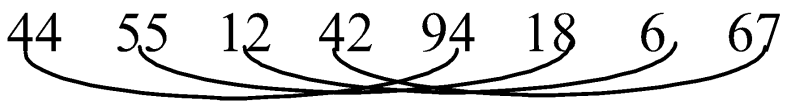
Сортировка вставками

A = 18, 20, 5, 13, 15

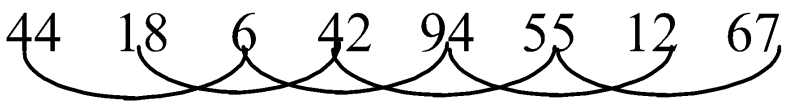


$O(n^2)$

Сортировка Шелла



4 - сортировка



2 - сортировка

1 - сортировка

6 12 18 42 44 55 67 94

h(M)

$M=[\log_2 n]-1$

$h[k]=3h[k+1]+1$

$h[M-1]=1$

Сортировка с разделением (быстрая)

$O(n \log_2 n)$

Сравнение алгоритмов сортировки

1. Сортировка Шелла
2. Сортировка простыми вставками
3. Сортировка бинарными вставками
4. Сортировка простым выбором
5. Шейкер-сортировка
6. Сортировка пузырьком