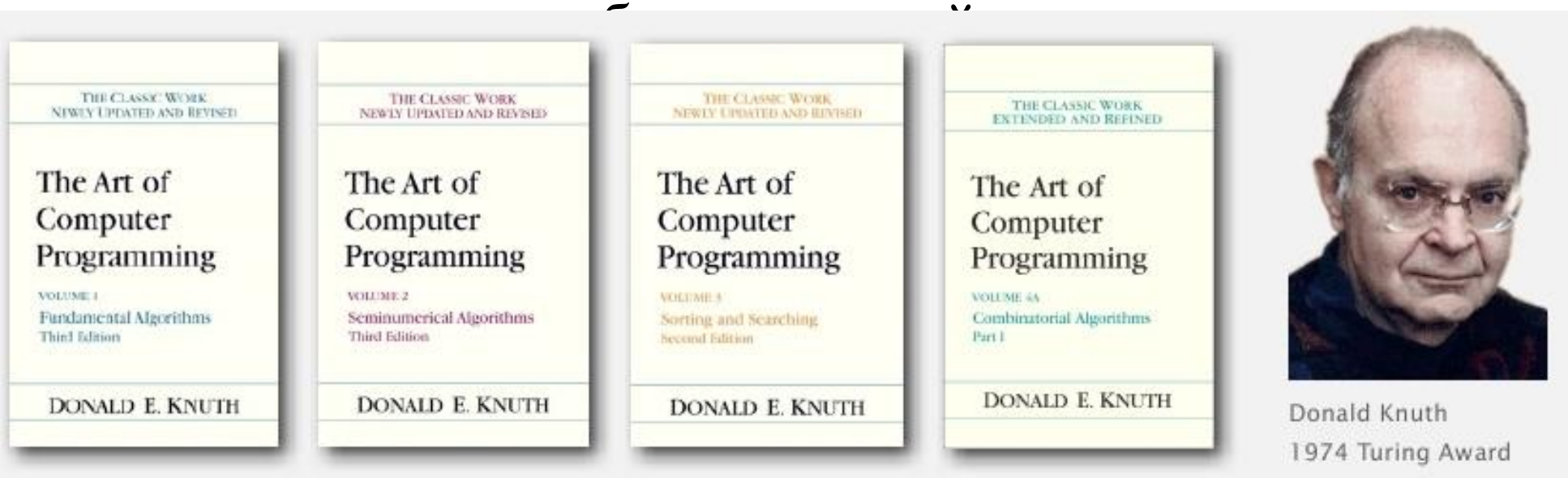
Математические модели

Математические модели для времени выполнения

- Общее время выполнения. Сумма: стоимость каждой операции * частоту, для всех операций
  - Анализ программ нужно производить на



- В принципе, создать точную математическую модель возможно.

Стоимость основных операций

operation	example	nanoseconds †
integer add	a + b	2.1
integer multiply	a * b	2.4
integer divide	a / b	5.4
floating-point add	a + b	4.6
floating-point multiply	a * b	4.2
floating-point divide	a / b	13.5
sine	Math.sin(theta)	91.3
arctangent	Math.atan2(y, x)	129.0
...	...	...

† Running OS X on Macbook Pro 2.2GHz with 2GB RAM

Стоимость основных операций

operation	example	nanoseconds †
variable declaration	`int a`	c_1
assignment statement	`a = b`	c_2
integer compare	`a < b`	c_3
array element access	`a[i]`	c_4
array length	`a.length`	c_5
1D array allocation	`new int[N]`	c_6 N
2D array allocation	`new int[N][N]`	c_7 N^2
string length	`s.length()`	c_8
substring extraction	`s.substring(N/2, N)`	c_9
string concatenation	`s + t`	c_{10} N

- Ошибка новичков: неправильная оценка

Пример: 1-Sum

- Подсчет количества инструкций, как функции от N.

```
int count = 0;
for (int i = 0; i < N; i++)
   if (a[i] == 0)
      count++;
```

operation	frequency
variable declaration	2
assignment statement	2
less than compare	N + 1
equal to compare	N
array access	N
increment	N to 2 N

Пример: 2-Sum

- Подсчет количества инструкций, как функции от]

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2} N (N-1) = \binom{N}{2}$$

operation	frequency
variable declaration	N + 2
assignment statement	N + 2
less than compare	½ (N + 1) (N + 2)
equal to compare	½ N (N – 1)
array access	N (N – 1)
increment	½ N (N – 1) to N (N – 1)

tedious to count exactly

Упрощение вычислений

" It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process and then given them various weights. We might, for instance, count the number of additions, subtractions, multiplications, divisions, recording of numbers, and extractions of figures from tables. In the case of computing with matrices most of the work consists of multiplications and writing down numbers, and we shall therefore only attempt to count the number of multiplications and recordings. " — *Alan Turing*

ROUNDING-OFF ERRORS IN MATRIX PROCESSES

By A. M. TURING

(*National Physical Laboratory, Teddington, Middlesex*)

[Received 4 November 1947]

SUMMARY

A number of methods of solving sets of linear equations and inverting matrices are discussed. The theory of the rounding-off errors involved is investigated for some of the methods. In all cases examined, including the well-known 'Gauss elimination process', it is found that the errors are normally quite moderate: no exponential build-up need occur.

Упрощение 1: модель стоимости

- Модель стоимости. Использовать некоторые основные операции для приближенного расчета в

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2} N (N-1) = \binom{N}{2}$$

operation	frequency
variable declaration	N + 2
assignment statement	N + 2
less than compare	½ (N + 1) (N + 2)
equal to compare	½ N (N − 1)
array access	N (N − 1)
increment	½ N (N − 1) to N (N − 1)

cost model = array accesses

(we assume compiler/JVM do not optimize array accesses away!)

Упрощение 2: тильда-нотация

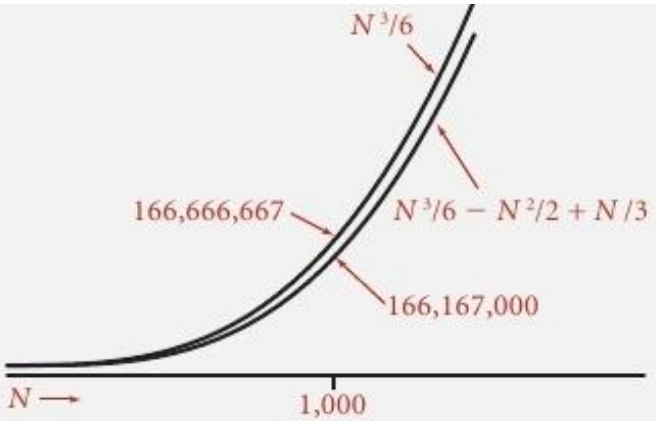
- Оценить время выполнение (или память), как функцию от входных данных N
- Игнорировать младшие члены

Ex 1. $\frac{1}{6} N^3 + 20 N + 16 \sim \frac{1}{6} N^3$

Ex 2. $\frac{1}{6} N^3 + 100 N^{4/3} + 56 \sim \frac{1}{6} N^3$

Ex 3. $\frac{1}{6} N^3 - \underbrace{\frac{1}{2} N^2 + \frac{1}{3} N}_{\text{discard lower-order terms}} \sim \frac{1}{6} N^3$

(e.g., N = 1000: 500 thousand vs. 166 million)



Leading-term approximation

Technical definition. $f(N) \sim g(N)$ means $\lim_{N \to \infty} \frac{f(N)}{g(N)} = 1$

Упрощение 2: тильда-нотация

- Оценить время выполнение (или память), как функцию от входных данных N
- Игнорировать младшие члены
  - Когда N велико, младшие члены незначительны
  -

operation	frequency	tilde notation
variable declaration	N + 2	~ N
assignment statement	N + 2	~ N
less than compare	½ (N + 1) (N + 2)	~ ½ N^2
equal to compare	½ N (N – 1)	~ ½ N^2
array access	N (N – 1)	~ N^2
increment	½ N (N – 1) to N (N – 1)	~ ½ N^2 to ~ N^2

Пример: 2-Sum

Q. Approximately how many array accesses as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

"inner loop"

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2} N (N-1) = \binom{N}{2}$$

A. $\sim N^2$ array accesses.

- Нижняя оценка. Использовать модель стоимости и тильда-нотацию для упрощение вычислений

Пример: 3-Sum

Q. Approximately how many array accesses as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

"inner loop"

$$\binom{N}{3} = \frac{N(N-1)(N-2)}{3!} \sim \frac{1}{6}N^3$$

A. $\sim \frac{1}{2} N^3$ array accesses.

- Нижняя оценка. Использовать модель стоимости и тильда-нотацию для упрощение вычислений

Оценка дискретной суммы

- Как оценить дискретную сумму?

1) Средствами дискретной математики.

2) Заменить сумму на определенный интеграл и

Ex 1. $1 + 2 + \ldots + N.$

$$\sum_{i=1}^{N} i \sim \int_{x=1}^{N} x\,dx \sim \frac{1}{2} N^2$$

Ex 2. $1^k + 2^k + \ldots + N^k.$

$$\sum_{i=1}^{N} i^k \sim \int_{x=1}^{N} x^k dx \sim \frac{1}{k+1} N^{k+1}$$

Ex 3. $1 + 1/2 + 1/3 + \ldots + 1/N.$

$$\sum_{i=1}^{N} \frac{1}{i} \sim \int_{x=1}^{N} \frac{1}{x} dx = \ln N$$

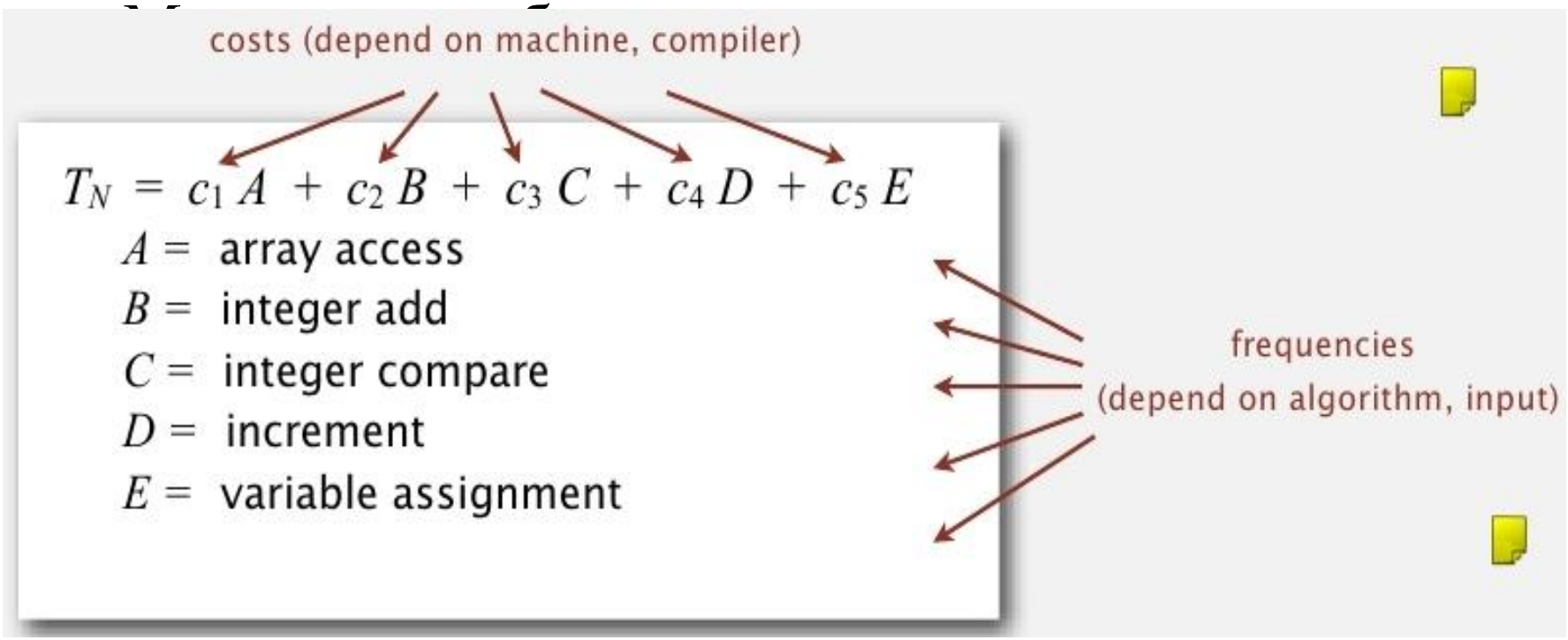
Ex 4. 3-sum triple loop.

$$\sum_{i=1}^{N} \sum_{j=i}^{N} \sum_{k=j}^{N} 1 \sim \int_{x=1}^{N} \int_{y=x}^{N} \int_{z=y}^{N} dz\,dy\,dx \sim \frac{1}{6} N^3$$

Математическая модель для времени выполнения

- В принципе, всегда возможно построить точную математическую модель.

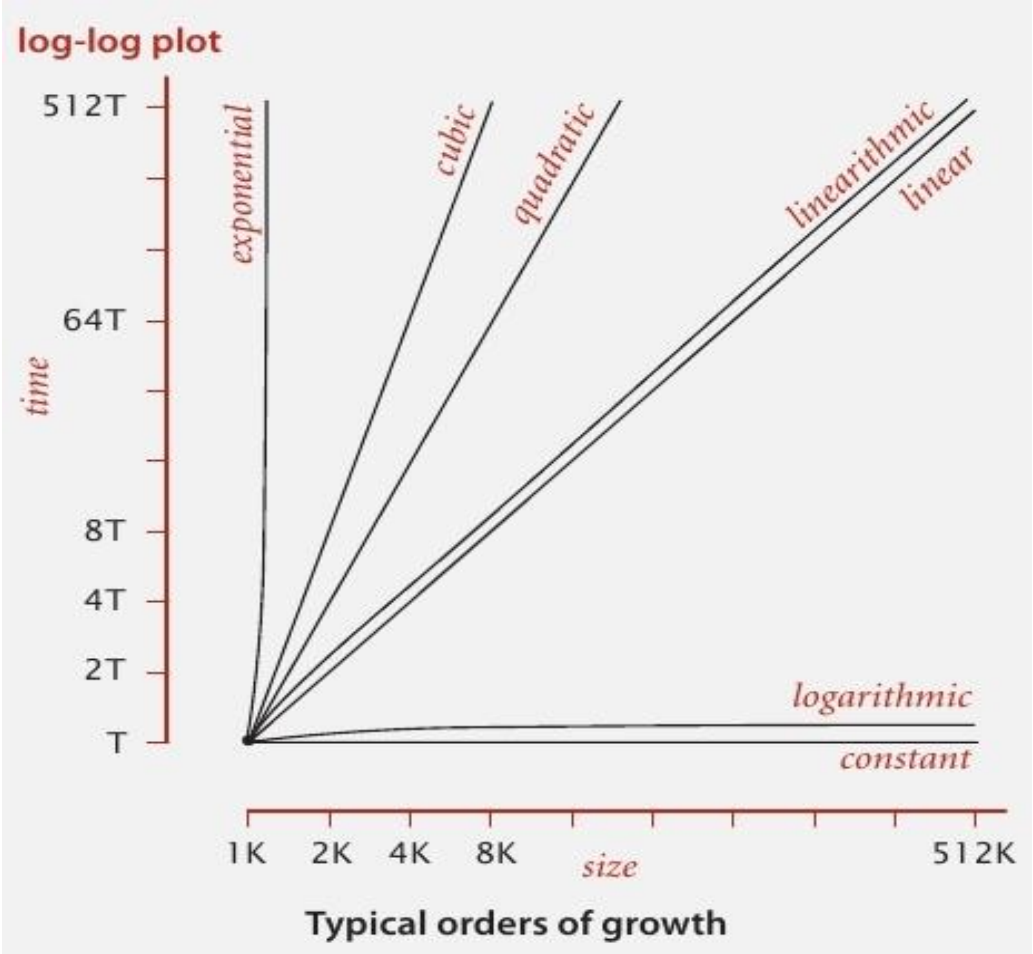
- На практике
  - Формула может быть сложной



Классификация порядков роста

Общая классификация порядков роста

- Малое число функций описывающих порядок роста основных алгоритмов
  - 1, log N, N, Nlog N, N^2, N^3 и 2^N



Общая классификация порядков роста

order of growth	name	typical code framework	description	example	T(2N) / T(N)
1	constant	`a = b + c;`	statement	add two numbers	1
log N	logarithmic	`while (N > 1) { N = N / 2; ... }`	divide in half	binary search	~ 1
N	linear	`for (int i = 0; i < N; i++) { ... }`	loop	find the maximum	2
N log N	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	quadratic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }`	double loop	check all pairs	4
N^3	cubic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }`	triple loop	check all triples	8
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	T(N)

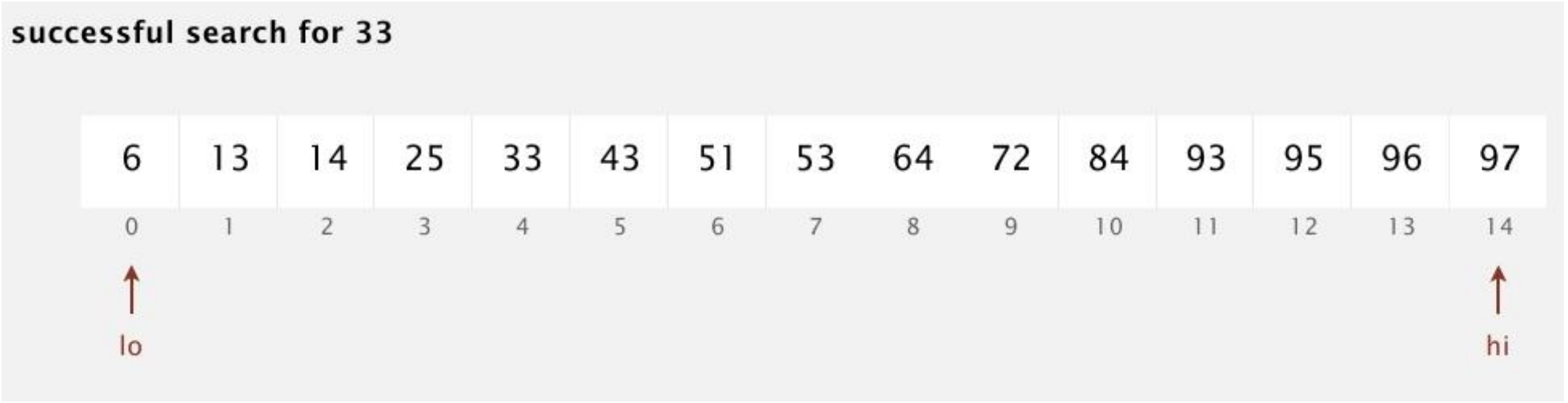
Практическое применение порядков роста

growth rate	problem size solvable in minutes			
	1970s	1980s	1990s	2000s
1	any	any	any	any
log N	any	any	any	any
N	millions	tens of millions	hundreds of millions	billions
N log N	hundreds of thousands	millions	millions	hundreds of millions
N^2	hundreds	thousand	thousands	tens of thousands
N^3	hundred	hundreds	thousand	thousands
2^N	20	20s	20s	30

- Нижняя оценка. Нужны линейные или линейно-логарифмические алгоритмы, чтобы идти в ногу

Бинарный поиск

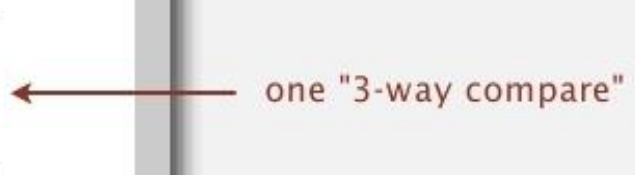
- Цель. Найти индекс ключа в отсортированном массиве
- Бинарный поиск
  - Ключ меньше — идем влево
  - Ключ больше — идем вправо
  - Равен — возвращаем результат



Бинарный поиск: реализация

- Впервые бинарный поиск был опубликован в 1946; первая безошибочная реализация в 1962
- Ошибка в Java.binarySearch() найдена в 2006

```
public static int binarySearch(int[] a, int key)
{
   int lo = 0, hi = a.length-1;
   while (lo <= hi)
   {
       int mid = lo + (hi - lo) / 2;
       if      (key < a[mid]) hi = mid - 1;
       else if (key > a[mid]) lo = mid + 1;
       else return mid;
   }
   return -1;
}
```



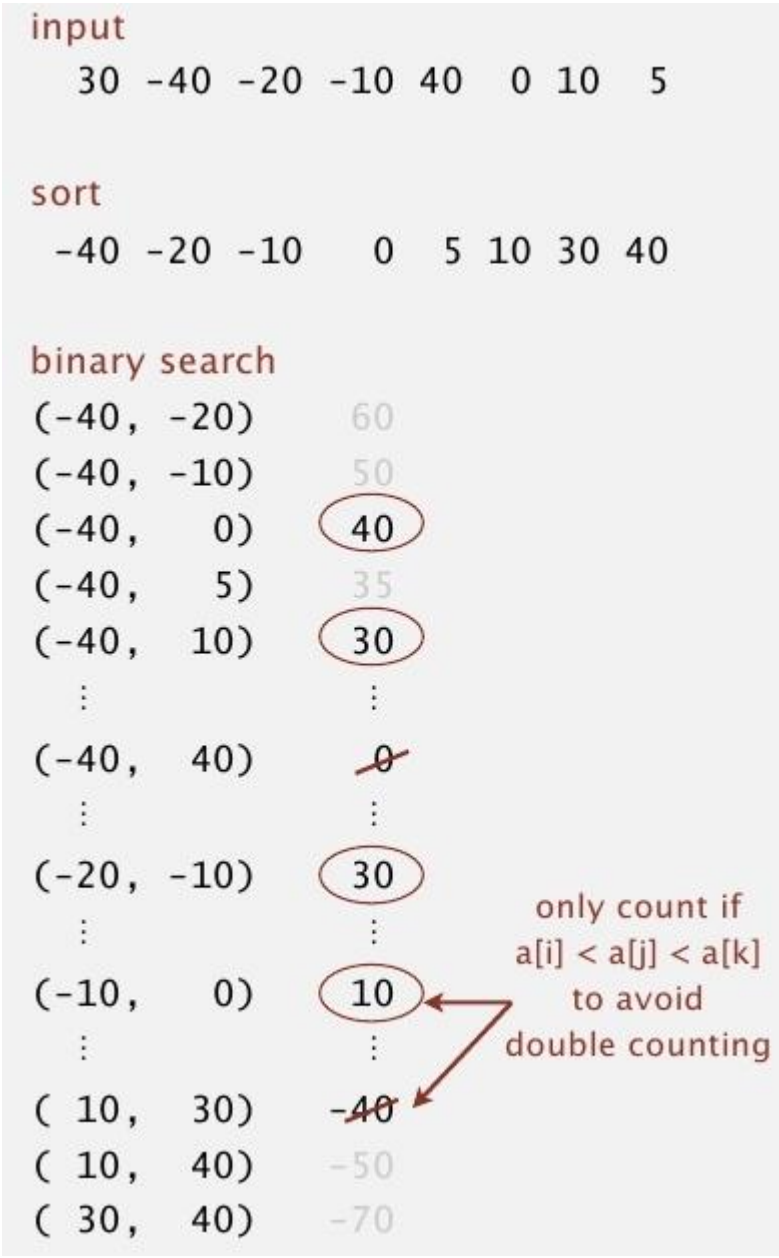
Бинарный поиск: математический анализ

- Предположение. Бинарный поиск использует 1 + lg N сравнений ключа в отсортированном массиве N
- T(N) количество сравнений ключа в отс
- T(

$$
\begin{aligned}
T(N) &\le T(N/2) + 1 && \text{given} \\
&\le T(N/4) + 1 + 1 && \text{apply recurrence to first term} \\
&\le T(N/8) + 1 + 1 + 1 && \text{apply recurrence to first term} \\
&\ldots \\
&\le T(N/N) + 1 + 1 + \ldots + 1 && \text{stop applying, T(1) = 1} \\
&= 1 + \lg N
\end{aligned}
$$

$N^2 \log N$ алгоритм для 3-Sum

- Алгоритм основанный на сортировке
  - Шаг 1: Сортировка N чисел
  - Шаг 2: Для каждой пары чисел a[i] и a[j] сделать бинарный поиск для -(a[i] + a[j])
- Анализ. Порядок роста $N^2 \log N$
  - Шаг 1: N^2 сортировка вставками
  - Шаг 2: $N^2 \log N$ бинарный поиск



Сравнение программ

- Гипотеза. Основанный на сортировке 3-Sum алгоритм $N^2 \log N$ однозначно быстрее метода грубой силы N^3

N	time (seconds)
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1

ThreeSum.java

N	time (seconds)
1,000	0.14
2,000	0.18
4,000	0.34
8,000	0.96
16,000	3.67
32,000	14.88
64,000	59.16

ThreeSumDeluxe.java

- Главный принцип. Лучший порядок роста => быстрота на практике

Теория алгоритмов

Типы анализа

- Лучший случай. Нижняя граница по стоимости
  - Определяется самыми «простыми» входными данными
  - Цель для любых входных данных
- Худший случай. Верхняя граница
  - Определяется «самыми сложными» входными данными
  - Предоставляет гарантии для всех возможных

Ex 1. Array accesses for brute-force 3-Sum.	
Best:	$\sim \frac{1}{2} N^3$
Average:	$\sim \frac{1}{2} N^3$
Worst:	$\sim \frac{1}{2} N^3$

Ex 2. Compares for binary search.	
Best:	~ 1
Average:	$\sim \lg N$
Worst:	$\sim \lg N$

  - Нужна модель для случайных входных данных

Типы анализа

- Лучший случай. Нижняя граница по стоимости
- Худший случай. Верхняя граница
- Средний случай. Ожидаемая стоимость для случайных входных данных

- Реальные входные данные могут не соответствовать модели
  - Нужно понимать, что может быть на входе, чтобы эффективно обрабатывать данные
  - Подход 1: строить модель для худшего случая
  - Подход 2: строить модель для случайных данных, в

notation	provides	example	shorthand for	used to
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$ ⋮	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$ ⋮	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$ ⋮	develop lower bounds

Пример: два алгоритма сортировки

- Быстрая сортировка
  - Количество сравнений в худшем случае: N^2
  - $O(N^2)$
- Сортировка слиянием
  - Количество сравнений в худшем случае: N logN
  - O(N logN)

- Известно, что на практике быстрая сортировка в два раза быстрее и использует в два раза меньше памяти, чем сортировка слиянием.
- Не используйте O для предсказания

notation	provides	example	shorthand for	used to
Tilde	leading term	$\sim 10\ N^2$	$10\ N^2$ $10\ N^2 + 22\ N \log N$ $10\ N^2 + 2\ N + 37$	provide approximate model
Big Theta	asymptotic growth rate	$\Theta(N^2)$	$\frac{1}{2}\ N^2$ $10\ N^2$ $5\ N^2 + 22\ N \log N + 3N$	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10\ N^2$ $100\ N$ $22\ N \log N + 3\ N$	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2}\ N^2$ N^5 $N^3 + 22\ N \log N + 3\ N$	develop lower bounds

Сортировка выбором

Сортировка выбором

- На итерации i найти минимальный оставшийся элемент с индексом min
- Поменять местами a[i] и a[min]

- Видео 1

Сортировка выбором

- Алгоритм. Сканирование идет слева направо
- Элементы слева от стрелки отсортированы и не меняются
- Нет элемента справа от стрелки, который был бы меньше элемента слева от стрелки

Сортировка выбором: внутренний цикл

- Move the pointer to the right.

```
i++;
```

- Identify index of minimum entry on right.

```
int min = i;
for (int j = i+1; j < N; j++)
   if (less(a[j], a[min]))
      min = j;
```

- Exchange into position.

```
exch(a, i, min);
```

Сортировка выбором: реализация на Java

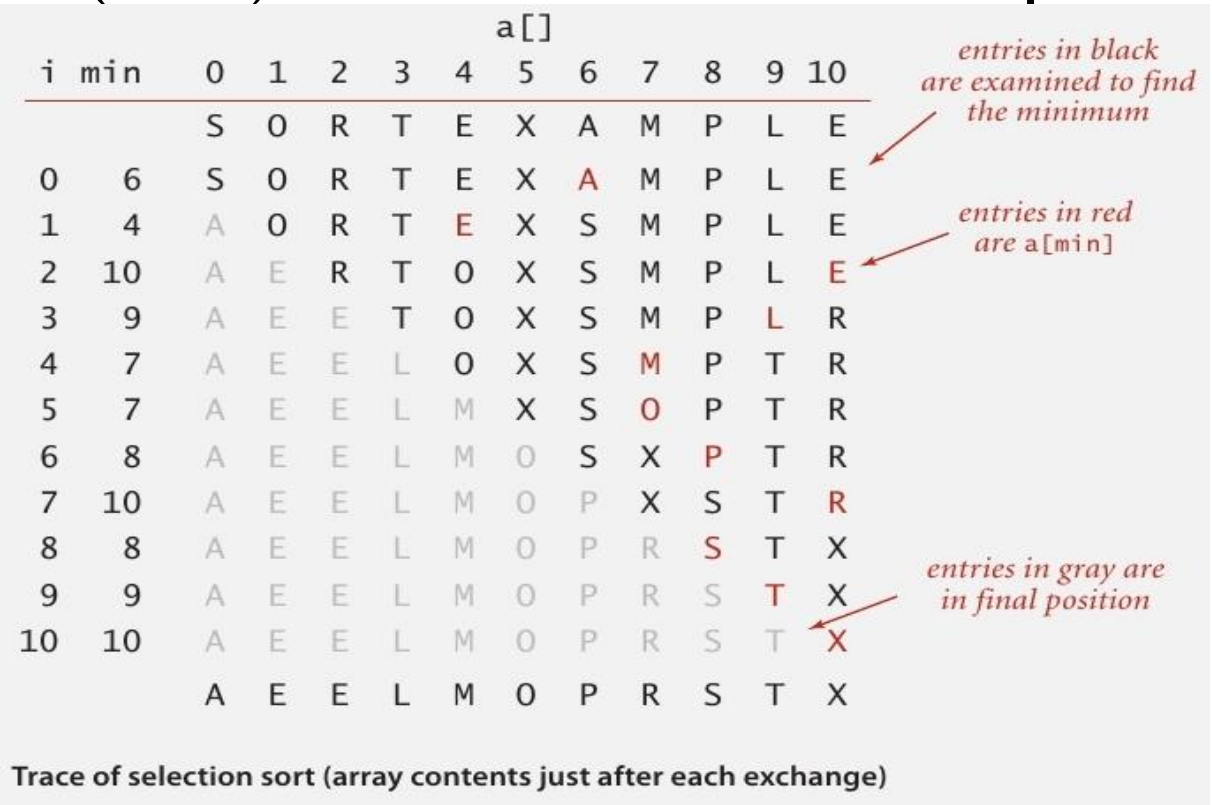
```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int min = i;
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Comparable[] a, int i, int j)
   {  /* as before */  }
}
```

Сортировка выбором: математический анализ

- Утверждение. Сортировка выбором использует (N-1) + (N-2) + … + 1 + 0 ~ $N^2/2$ сравнений и N перес



		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position

Trace of selection sort (array contents just after each exchange)

- Время выполнения не зависит от входных данных. Квадратичное время, даже если массив отсортирован