



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО

ЦЕНТР КОМПЕТЕНЦИЙ ONEAPI В ННГУ

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО
ЦЕНТР КОМПЕТЕНЦИЙ oneAPI в ННГУ
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ

***ВВЕДЕНИЕ В АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ
И ОПТИМИЗАЦИЮ ПРОГРАММ***

**Введение в вычислительную
сложность алгоритмов**

Содержание

- ❑ Введение
- ❑ Показатели эффективности алгоритмов
- ❑ Сравнение алгоритмов по скорости
- ❑ Анализ вычислительной сложности алгоритмов
- ❑ Использование оценок сложности в практике
- ❑ Литература

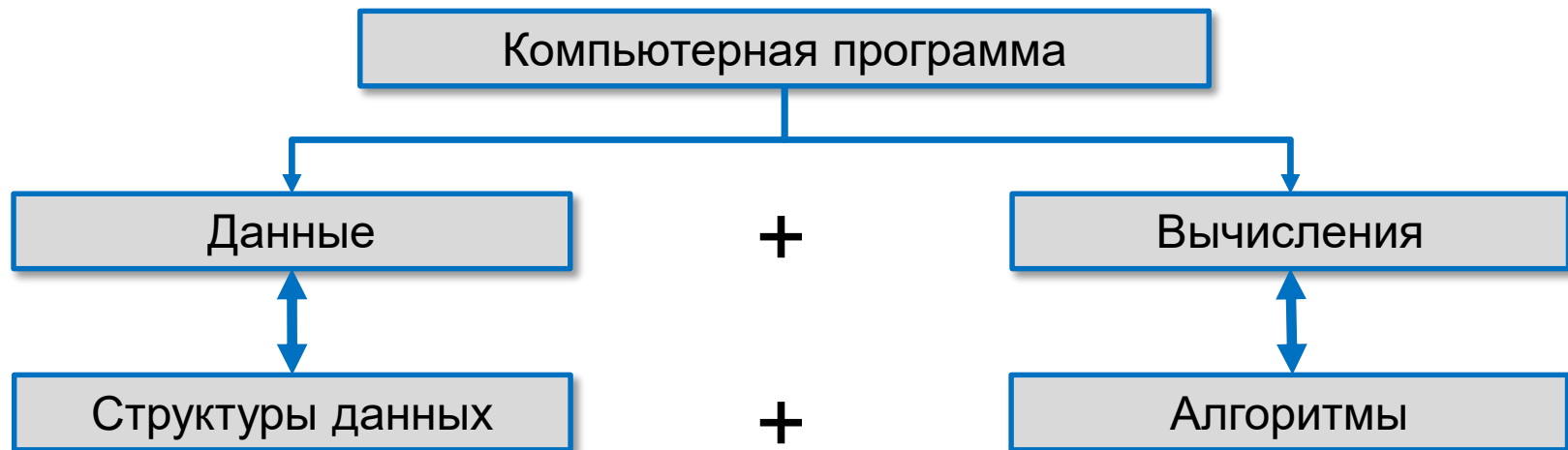
Цель лекции

- Цель лекции – *неформальное** введение в вычислительную сложность алгоритмов
 - Как сравнивать алгоритмы?
 - На что обращать внимание?
 - Какие типичные ошибки допускают разработчики и как их избегать?

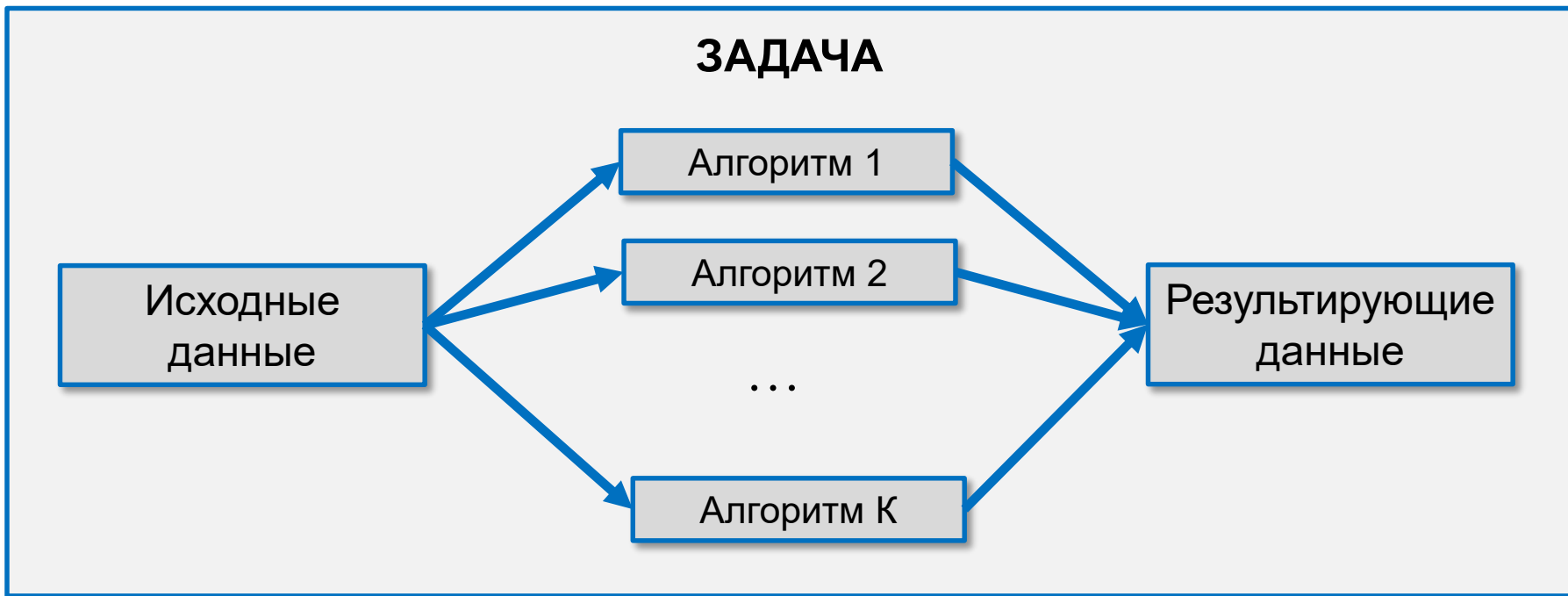
** Желающие улучшить свой кругозор могут ознакомиться с литературой по Теории сложности*
В данной лекции мы фокусируемся только на наиболее интуитивных понятиях и методах, которые, однако, во многих случаях позволят принять верное решение

ВВЕДЕНИЕ

Введение...



Введение...

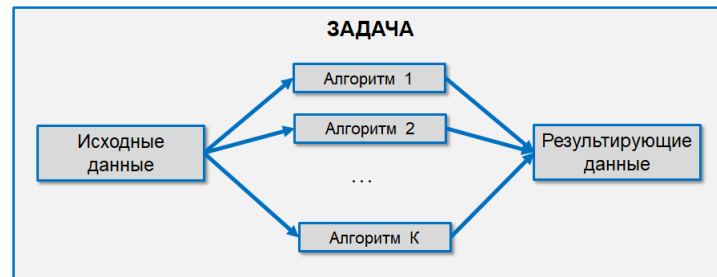


Какой алгоритм выбрать?

Введение...

- Почти всегда для решения конкретной задачи существует несколько алгоритмов

- **Вопрос.** Какой алгоритм выбрать?



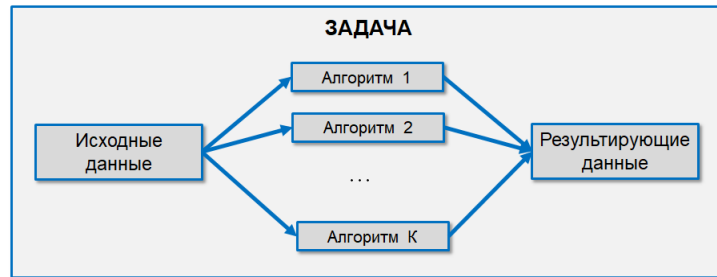
Введение...

- Почти всегда для решения конкретной задачи существует несколько алгоритмов

□ **Вопрос.** Какой алгоритм выбрать?

□ **Ответ.** Конечно же, хороший 😊!

□ **Вопрос.** Как выбрать хороший алгоритм?



Введение

- ❑ Почти всегда для решения конкретной задачи существует несколько алгоритмов

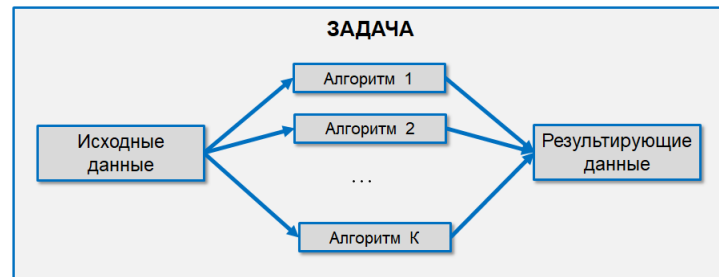
❑ **Вопрос.** Какой алгоритм выбрать?

❑ **Ответ.** Конечно же, хороший 😊!

❑ **Вопрос.** Как выбрать хороший алгоритм?

❑ **Ответ.** По эффективности

❑ **Вопрос.** Каковы критерии эффективности?



ПОКАЗАТЕЛИ ЭФФЕКТИВНОСТИ АЛГОРИТМОВ

Показатели эффективности алгоритмов...

- ❑ **Вопрос.** Как сравнивать алгоритмы? По каким критериям?
- ❑ Обычно рассматривают следующие критерии:
 - **Скорость** (временная сложность)
 - **Память** (объемная сложность)
 - **Сложность разработки** (время на разработку, понятность/читаемость кода)
- ❑ Последний критерий стоит особняком. Вопрос квалификации специалистов
- ❑ Далее будем говорить о первых двух критериях
- ❑ (!) Критерии **Скорость** и **Память** часто противоречивы

Показатели эффективности алгоритмов...

Скорость vs Память...

□ **Вопрос.** Какой критерий важнее – скорость или память? Или оба?

Показатели эффективности алгоритмов

Скорость vs Память

- ❑ **Вопрос.** Какой критерий важнее – скорость или память? Или оба?
- ❑ **Ответ.** Обычно выбирается один главный критерий, а остальные выступают в виде ограничений
- ❑ Наиболее часто главным критерием является **Скорость**
- ❑ Другими словами – **вычислительная эффективность алгоритма**

СРАВНЕНИЕ АЛГОРИТМОВ ПО СКОРОСТИ

Сравнение алгоритмов по скорости...

□ **Вопрос.** Как сравнивать алгоритмы?

Сравнение алгоритмов по скорости...

- ❑ **Вопрос.** Как сравнивать алгоритмы?
- ❑ **Популярный ответ.** запрограммируем и посмотрим
- ❑ (!) Этот ответ ошибочен

Сравнение алгоритмов по скорости...

□ Причины ошибочности «экспериментального» подхода:

- Трудозатраты на программирование и
- Наведенные эффекты – влияние качества реализации
- Погрешность измерения времени – трудности постановки «чистого» эксперимента даже на одной вычислительной системе
- Сложность выбора репрезентативных наборов тестовых входных данных – возможная неадекватность выводов

□ Как результат, такой подход применяется:

- для прототипирования и оценки практической пригодности алгоритма
- в тех случаях, когда не удастся построить аналитические оценки, либо в результате оценивания выясняется, что алгоритмы относятся к одному классу и дальнейший анализ требует вычислительных экспериментов

Сравнение алгоритмов по скорости...

- ❑ Качество алгоритмов оценивают методами специального раздела математики – **Теория сложности**
- ❑ Как правило, сложность алгоритма зависит от объема входных данных (N)
- ❑ Эффективность алгоритмов обычно рассматривают при больших N
- ❑ **Вопрос.** Почему?

Сравнение алгоритмов по скорости...

- ❑ Качество алгоритмов оценивают методами специального раздела математики – **Теория сложности**
- ❑ Как правило, сложность алгоритма зависит от объема входных данных (N)
- ❑ Эффективность алгоритмов обычно рассматривают при больших N
- ❑ **Вопрос.** Почему?
- ❑ **Ответ.** Как правило, при малых N «время» работы алгоритма также мало
- ❑ **Оценка сложности выполняется по порядку величины**

Сравнение алгоритмов по скорости...

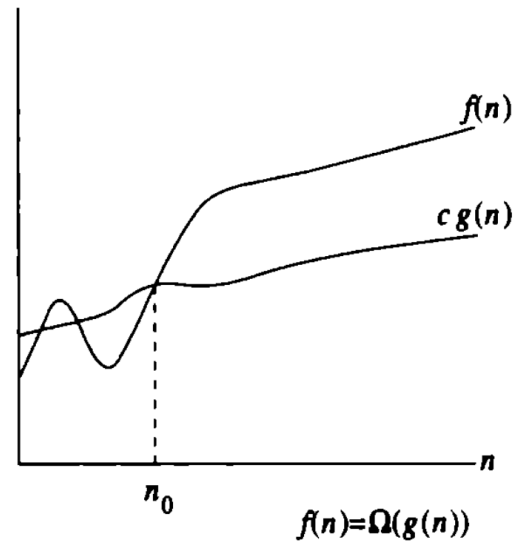
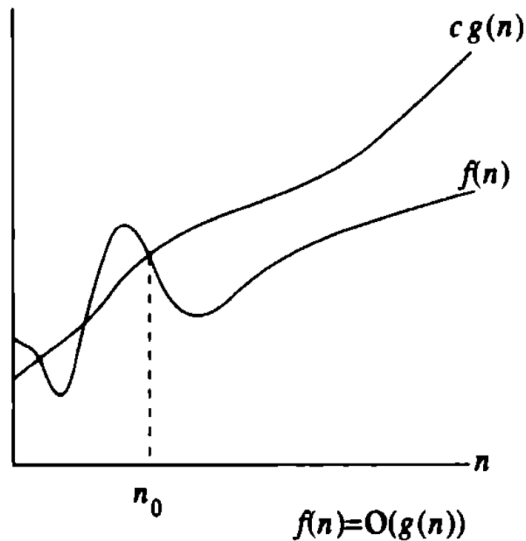
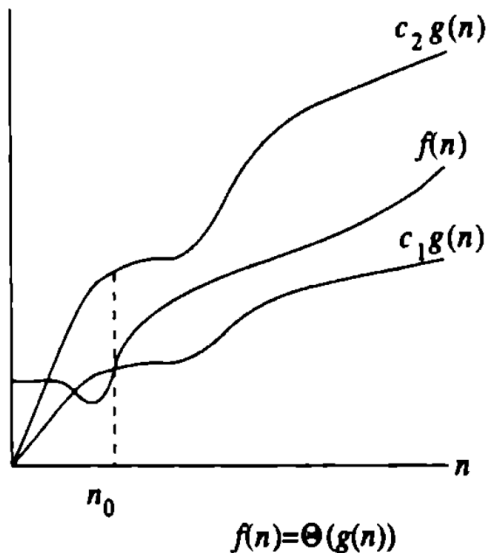
- ❑ Качество алгоритмов оценивают методами специального раздела математики – **Теория сложности**
- ❑ Как правило, сложность алгоритма зависит от объема входных данных (N)
- ❑ Эффективность алгоритмов обычно рассматривают при больших N
- ❑ **Вопрос.** А бывает иначе?

Сравнение алгоритмов по скорости

- ❑ Качество алгоритмов оценивают методами специального раздела математики – **Теория сложности**
- ❑ Как правило, сложность алгоритма зависит от объема входных данных (N)
- ❑ Эффективность алгоритмов обычно рассматривают при больших N
- ❑ **Вопрос.** А бывает иначе?
- ❑ **Ответ.** Да
- ❑ **Вопрос.** Можете привести примеры?

АНАЛИЗ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ АЛГОРИТМОВ

Асимптотическая сложность...



* Рисунок приведен по *Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К.* Алгоритмы: построение и анализ. — 3-е изд. — М.: Вильямс, 2013

Асимптотическая сложность...

O -обозначение

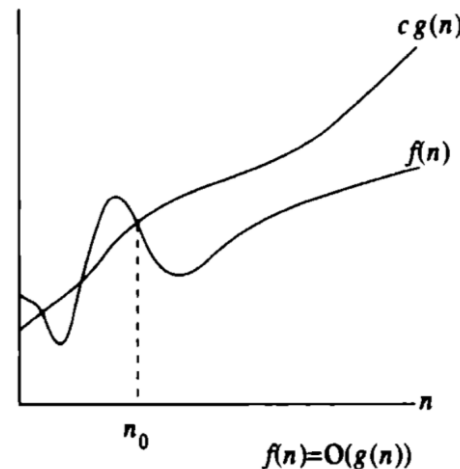
□ Пусть функция $f(n)$ характеризует сложность алгоритма (число операций)

□ O -обозначение

$$f(n) \in O(g(n)) = \left\{ \begin{array}{l} \exists c > 0, n_0 > 0: \\ 0 \leq f(n) \leq c g(n), \forall n \geq n_0 \end{array} \right\}$$

дает **верхнюю асимптотическую** оценку $g(n)$ функции $f(n)$ с точностью до постоянного множителя

□ Т.е. функция $f(n)$ **ограничена сверху** функцией $g(n)$ (асимптотически)



Асимптотическая сложность...

Ω -обозначение

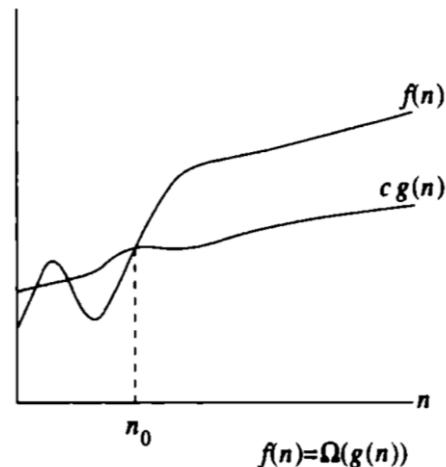
□ Пусть функция $f(n)$ характеризует сложность алгоритма (число операций)

□ Ω -обозначение

$$f(n) \in \Omega(g(n)) = \left\{ \begin{array}{l} \exists c > 0, n_0 > 0: \\ 0 \leq c g(n) \leq f(n), \forall n \geq n_0 \end{array} \right\}$$

дает **нижнюю асимптотическую** оценку $g(n)$ функции $f(n)$ с точностью до постоянного множителя

□ Т.е. функция $f(n)$ **ограничена снизу** функцией $g(n)$ (асимптотически)



Асимптотическая сложность...

Θ -обозначение

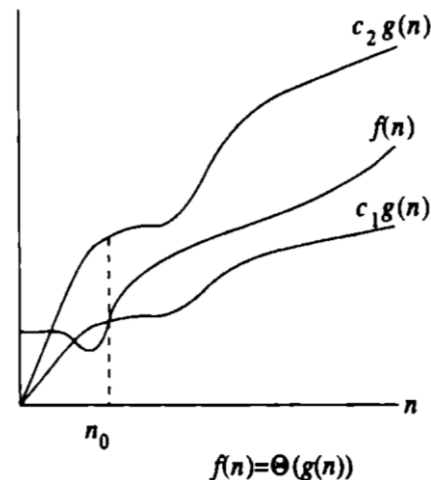
□ Пусть функция $f(n)$ характеризует сложность алгоритма (число операций)

□ Θ -обозначение

$$f(n) \in \Theta(g(n)) = \left\{ \begin{array}{l} \exists c_1 > 0, c_2 > 0, n_0 > 0: \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \end{array} \right\}$$

дает **верхнюю и нижнюю асимптотическую** оценку $g(n)$ функции $f(n)$ с точностью до постоянного множителя

□ Т.е. функция $f(n)$ **ограничена сверху и снизу** функцией $g(n)$ (асимптотически)



Асимптотическая сложность

- ❑ Кроме рассмотренных вариантов, конечно, можно строить и точные оценки числа операций в алгоритме
- ❑ Но!
 - Что брать за «базовую операцию»?
 - Как быть с разной «стоимостью» операций в алгоритме?
 - Сколько усилий нужно для получения точной оценки и окупятся ли они?
- ❑ На практике обычно используют верхнюю оценку сложности алгоритма
- ❑ Пишут
 - Алгоритм бинарного поиска ключа в массиве имеет сложность $O(\log n)$
 - Алгоритм сортировки массива вставками имеет сложность $O(n^2)$
 - Алгоритм сортировки массива слиянием имеет сложность $O(n \log n)$
- ❑ Рассмотрим примеры

Анализ сложности...

Циклы...

```
for (i = 0; i < N; i++)  
    A[i] = A[i] * 2;
```

□ Оценим вычислительную сложность

- Объем входных данных: N
- Операции: умножение, присваивание, сравнение, инкремент
- Число операций: N для каждого вида
- Сложность: $O(N)$

Анализ сложности...

Циклы

```
for (i = 0; i < N; i++)  
    for (j = 0; j < N; j++)  
        A[i][j] = A[i][j] * 2;
```

□ Оценим вычислительную сложность

- Объем входных данных: N^2
- Операции: умножение, присваивание, сравнение (по i и по j), инкремент (i и j)
- Число операций:
 - N операций по i
 - для каждой из них N операций по j
 - Т.е. N^2 умножений и присваиваний
- Сложность: $O(N^2)$

Анализ сложности...

Вызовы функций...

```
void First()
{
    for (i = 0; i < N; i++)
        A[i] = A[i] * 2;
}
void Second()
{
    for (i = 0; i < N; i++)
        B[i] = B[i] + 3;
}
void main()
{
    First();
    Second();
}
```

❑ **Вопрос.** Чему равна сложность?

Анализ сложности...

Вызовы функций...

```
void First()
{
    for (i = 0; i < N; i++)
        A[i] = A[i] * 2;
}
void Second()
{
    for (i = 0; i < N; i++)
        B[i] = B[i] + 3;
}
void main()
{
    First();
    Second();
}
```

❑ **Вопрос.** Чему равна сложность?

❑ **Ответ.** $O(N)$

Анализ сложности...

Вызовы функций...

```
void First()
{
    for (i = 0; i < N; i++)
        A[i] = A[i] * 2;
}
void Second()
{
    for (i = 0; i < N; i++)
        First();
}
void main()
{
    Second();
}
```

❑ **Вопрос.** Чему равна сложность?

Анализ сложности...

Вызовы функций

```
void First()
{
    for (i = 0; i < N; i++)
        A[i] = A[i] * 2;
}
void Second()
{
    for (i = 0; i < N; i++)
        First();
}
void main()
{
    Second();
}
```

❑ **Вопрос.** Чему равна сложность?

❑ **Ответ.** $O(N^2)$

Анализ сложности...

Рекурсия...

```
void Example(unsigned int i)
{
    if (i <= 0)
        return;
    Example(i - 1);
}
void main()
{
    Example(N);
}
```

❑ **Вопрос.** Чему равна сложность?

Анализ сложности...

Рекурсия...

```
void Example(unsigned int i)
{
    if (i <= 0)
        return;
    Example(i - 1);
}
void main()
{
    Example(N);
}
```

❑ **Вопрос.** Чему равна сложность?

❑ **Ответ.** $O(N)$

Анализ сложности...

Рекурсия...

```
void Example(unsigned int i)
{
    if (i <= 0)
        return;
    Example(i - 1);
    Example(i - 1);
}
void main()
{
    Example(N);
}
```

❑ **Вопрос.** Чему равна сложность?

Анализ сложности...

Рекурсия

```
void Example(unsigned int i)
{
    if (i <= 0)
        return;
    Example(i - 1);
    Example(i - 1);
}

void main()
{
    Example(N);
}
```

❑ **Вопрос.** Чему равна сложность?

Пусть $T(N)$ – время работы алгоритма

Очевидно, $T(N) = 1 + T(N-1) + T(N-1) = 1 + 2T(N-1), \dots T(0) = 1$

❑ **Ответ.** $O(2^N)$

Анализ сложности

Поиск подстрок в строке

```
for (size_t i = 0; i <= s.size() - ss.size(); i++) {  
    isFound = true;  
    for (size_t j = 0; j < ss.size(); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        cout << "substring is found, pos = " << i << endl;  
}
```

□ **Вопрос.** Чему равна сложность?

Анализ сложности

Поиск подстрок в строке

```
for (size_t i = 0; i <= s.size() - ss.size(); i++) {  
    isFound = true;  
    for (size_t j = 0; j < ss.size(); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        cout << "substring is found, pos = " << i << endl;  
}
```

❑ **Вопрос.** Чему равна сложность?

Пусть $N = s.size$, $M = ss.size$

Тогда в цикле по i число итераций = $N - M + 1$, в цикле по j – M в худшем случае

❑ **Ответ.** $O((N - M + 1)M) = O(N^2)$

Нетрудно показать, что максимум в $(N - M + 1)M$ достигается при $M = N/2$

Оценки сложности...

□ **Вопрос.** От чего зависит время работы алгоритма, кроме объема данных?

Оценки сложности...

❑ **Вопрос.** От чего зависит время работы алгоритма, кроме объема данных?

❑ **Ответ.** От **самих** данных

❑ В этом случае рассматривают следующие виды оценок:

- оценка в лучшем случае
- оценка в среднем (*)
- оценка в худшем случае (*)

❑ (*) **обычно представляют интерес**

Оценки сложности...

```
int Search(int key)
{
    int Index = -1;
    for (int i = 0; i < N; i++)
        if (A[i] == key)
        {
            Index = i;
            break;
        }
}
```

□ **Вопрос.** Чему равна сложность в лучшем, худшем и среднем случаях?

Оценки сложности

```
int Search(int key)
{
    int Index = -1;
    for (i = 0; i < N; i++)
        if (A[i] == key)
        {
            Index = i;
            break;
        }
}
```

❑ **Вопрос.** Чему равна сложность в лучшем, худшем и среднем?

❑ **Ответ.**

- в лучшем случае $O(1)$ – сразу нашли
- в худшем случае $O(N)$ – не нашли
- в среднем $O(N/2) = O(N)$

Классы сложности

ПОЛИНОМИАЛЬНАЯ СЛОЖНОСТЬ

$O(1)$	константная сложность
$O(\log(n))$	логарифмическая сложность
$O(n)$	линейная сложность
$O(n \log(n))$	$n \cdot \log n$ сложность
$O(n^2)$	квадратичная сложность
$O(n^c), c > 1$	полиномиальная сложность

ЭКСПОНЕНЦИАЛЬНАЯ СЛОЖНОСТЬ

$O(c^n), c > 1$
 $O(n!)$

ИСПОЛЬЗОВАНИЕ ОЦЕНОК СЛОЖНОСТИ В ПРАКТИКЕ

Использование оценок сложности в практике (1)

- Рассмотрим такую реализацию поиска подстрок в строке

```
char *s;  
char *ss;  
bool isFound;  
for (size_t i = 0; i <= strlen(s) - strlen(ss); i++) {  
    isFound = true;  
    for (size_t j = 0; j < strlen(ss); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```

- Мы оценили сложность алгоритма как $O(N^2)$

- **Вопрос.** Как проверить, такова ли сложность для программной реализации?

Использование оценок сложности в практике (1)

□ Итак, оценка сложности алгоритма $f(N) = O(N^2)$

□ **Вопрос.** Как проверить, такова ли сложность программной реализации?

```
for (size_t i = 0; i <= strlen(s) - strlen(ss); i++) {  
    isFound = true;  
    for (size_t j = 0; j < strlen(ss); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```

□ «Наивный» алгоритм проверки

- Добавляем замеры времени
- Меняем размер данных (N) в выбранных пределах с выбранным шагом
- Замеряем времена (T) работы
- Строим график зависимости T(N)
- Сравниваем его с оценкой сложности

□ **Вопрос.** Есть ли проблемы у такого подхода?

Использование оценок сложности в практике (1)

- ❑ **Вопрос.** Есть ли проблемы у такого подхода?
- ❑ Реализовав наивный алгоритм для указанного кода, получим график*
- ❑ **Вопрос.** Можно ли по графику сказать, что характер зависимости $T(N)$ имеет вид $O(N^2)$?

```
for (size_t i = 0; i <= strlen(s) - strlen(ss); i++) {  
    isFound = true;  
    for (size_t j = 0; j < strlen(ss); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```



*по оси ординат значения N в тысячах

Использование оценок сложности в практике (1)

- ❑ **Вопрос.** Есть ли проблемы у такого подхода?
- ❑ Реализовав наивный алгоритм для указанного кода, получим график*
- ❑ **Вопрос.** Можно ли по графику сказать, что характер зависимости $T(N)$ имеет вид $O(N^2)$?
- ❑ **Ответ.** Нельзя
- ❑ **Вопрос.** Что делать?

```
for (size_t i = 0; i <= strlen(s) - strlen(ss); i++) {  
    isFound = true;  
    for (size_t j = 0; j < strlen(ss); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```



*по оси ординат значения N в тысячах

Использование оценок сложности в практике (1)

❑ **Вопрос.** Можно ли по графику сказать, что характер зависимости $T(N)$ имеет вид $O(N^2)$?

❑ **Ответ.** Нельзя

❑ **Вопрос.** Что делать?

❑ Модифицируем **алгоритм проверки**

- Добавляем замеры времени
- Меняем размер данных (N) в выбранных пределах с выбранным шагом
- Замеряем времена (T) работы
- Строим график зависимости $T(N) / f(N)$
- Сравниваем его с **константой** (важно: график может выйти на константу не сразу)

```
for (size_t i = 0; i <= strlen(s) - strlen(ss); i++) {  
    isFound = true;  
    for (size_t j = 0; j < strlen(ss); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```

Использование оценок сложности в практике (1)

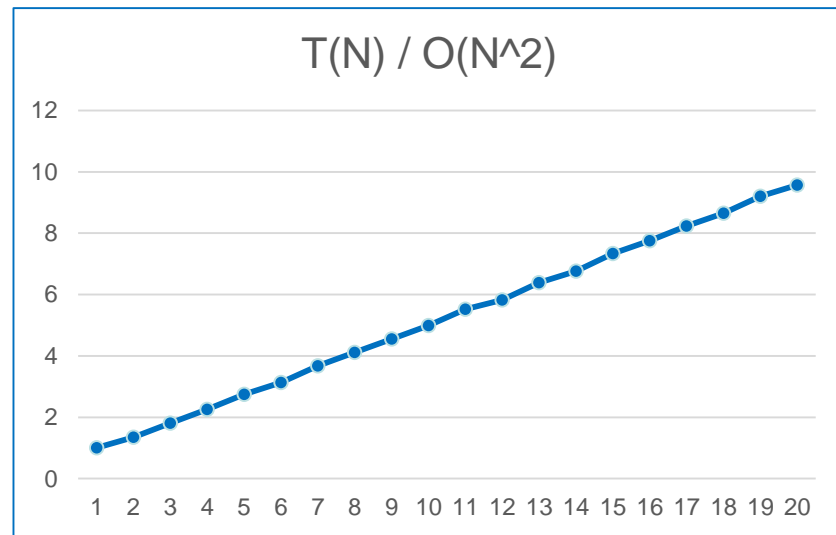
- ❑ Реализовав алгоритм проверки для указанного кода, получим график*
- ❑ Теперь очевидно, что реализация имеет отличную от $O(N^2)$ оценку сложности

```
for (size_t i = 0; i <= strlen(s) - strlen(ss); i++) {  
    isFound = true;  
    for (size_t j = 0; j < strlen(ss); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```

- ❑ **Задание.** Выполните оценку сложности представленного кода и проверьте ее

- ❑ Исправим код

*по оси ординат значения N в тысячах



Использование оценок сложности в практике (1)

□ Исправим код

Было

```
char *s;  
char *ss;  
bool isFound;  
for (size_t i = 0; i <= strlen(s) - strlen(ss); i++) {  
    isFound = true;  
    for (size_t j = 0; j < strlen(ss); j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```

Стало

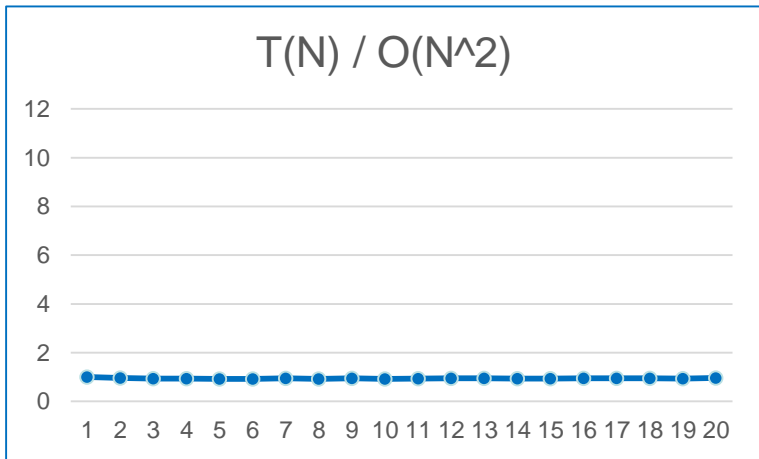
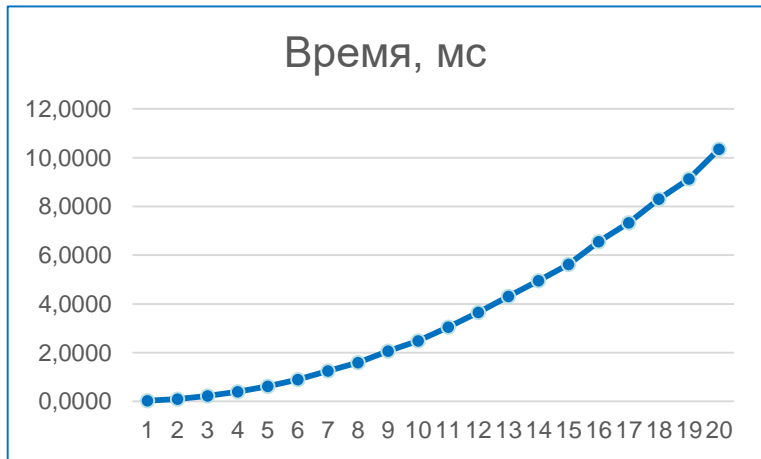
```
char *s;  
char *ss;  
bool isFound;  
size_t sLen = strlen(s);  
size_t ssLen = strlen(ss);  
for (size_t i = 0; i <= sLen - ssLen; i++) {  
    isFound = true;  
    for (size_t j = 0; j < ssLen; j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```

□ И снова применим алгоритм проверки

Использование оценок сложности в практике (1)

- ❑ Реализовав алгоритм проверки для нового кода, получим графики*
- ❑ Теперь очевидно, что реализация имеет оценку сложности $O(N^2)$

```
for (size_t i = 0; i <= sLen - ssLen; i++) {  
    isFound = true;  
    for (size_t j = 0; j < ssLen; j++)  
        if (s[i + j] != ss[j]) {  
            isFound = false;  
            break;  
        }  
    if (isFound)  
        count++;  
}
```



*по оси ординат значения N в тысячах

Использование оценок сложности в практике (2)

- ❑ Предположим, мы рассматриваем 2 алгоритма и хотим выбрать наиболее подходящий
- ❑ Для простоты предположим, что время работы не зависит от входных данных
- ❑ **Пример:** линейный поиск в массиве из N элементов. Ищем любое вхождение. Для чистоты эксперимента строим данные так, чтобы ключа там точно не было.
- ❑ Все знают, как устроен линейный поиск.
Вопрос: сколько в нем операций сравнения в рассматриваемом случае?

Использование оценок сложности в практике (2)

- ❑ Предположим, мы рассматриваем 2 алгоритма и хотим выбрать наиболее подходящий
- ❑ Для простоты предположим, что время работы не зависит от входных данных
- ❑ **Пример:** линейный поиск в массиве из N элементов. Ищем любое вхождение. Для чистоты эксперимента строим данные так, чтобы ключа там точно не было.
- ❑ Все знают, как устроен линейный поиск.
Вопрос: сколько в нем операций сравнения в рассматриваемом случае?
Ответ: $2N$ (цикл это тоже сравнение)
Вопрос №2: как уменьшить число сравнений?

Использование оценок сложности в практике (2)

- ❑ Все знают, как устроен линейный поиск.

Вопрос: сколько в нем операций сравнения в рассматриваемом случае?

Ответ: $2N$ (цикл это тоже сравнение)

Вопрос №2: как уменьшить число сравнений?

- ❑ **Постановка барьера** (см. Д. Кнут. Искусство программирования):

- Проверить последний элемент и сохранить его
- Записать в последний элемент ключ
- Сделать поиск, пока не найдем ключ
- Восстановить последний элемент
- Проверить, как мы вышли из цикла (чему равно i)

- ❑ **Число проверок уменьшается почти вдвое!**

Использование оценок сложности в практике (2)

- ❑ **Постановка барьера** (см. Д. Кнут. Искусство программирования):
- ❑ **Число проверок уменьшается почти вдвое!**
- ❑ **Вопрос:** а что происходит с временем работы? Оно уменьшается вдвое или нет?
- ❑ **Вопрос:** почему так?

Использование оценок сложности в практике (2)

- ❑ **Важный вывод:** сравнение констант у алгоритмов одного класса «на бумаге» чревато ошибками (неверными прогнозами), т.к. архитектуры ВС и компиляторы устроены достаточно сложно.
- ❑ В таких случаях зачастую приходится прибегать к экспериментам.

Литература

1. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ. – 3-е изд. – М.: Вильямс, 2013.
2. Ахо А.В., Хопкрофт Д., Ульман Д.Д. Структуры данных и алгоритмы. – М.: Вильямс, 2001. – 384 с.
3. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/Поиск. – К.: ДиаСофт, 2001. – 688 с.
4. Левитин А.В. Алгоритмы: введение в разработку и анализ. – М.: Вильямс, 2006. – 576 с.

Авторский коллектив

- ❑ Мееров Иосиф Борисович, к.т.н., доцент, зам. зав. каф. МОСТ
- ❑ Сысоев Александр Владимирович, к.т.н., доцент каф. МОСТ
- ❑ Линев Алексей Владимирович, зав. лаб. интернета вещей, каф. ПРИН
- ❑ Волокитин Валентин Дмитриевич, программист лаборатории СТиВВ, каф. МОСТ
- ❑ Козинов Евгений Александрович, к.т.н., преподаватель каф. МОСТ
- ❑ Панова Елена Анатольевна, инженер лаборатории СТиВВ, каф. МОСТ

Контакты

Нижегородский государственный университет

<http://www.unn.ru>

Центр компетенций oneAPI в ННГУ

<http://hpc-education.unn.ru/ru/центр-компетенций-oneapi-в-ннгу>

Институт информационных технологий, математики и механики

<http://www.itmm.unn.ru>