Оценка сложности алгоритмов

---

1. Основные понятия

- **Временная сложность** — время выполнения алгоритма в зависимости от размера входных данных (n).
- **Пространственная сложность** — объем памяти, используемый алгоритмом.
- **Асимптотическая нотация** — математическое описание скорости роста сложности.

---

2. Асимптотические обозначения

1. **O (O-большое)**: Верхняя граница (худший случай).
   Пример: Алгоритм с временной сложностью `O(n²)` выполняется не медленнее, чем `c * n²`.
2. **Ω (Омега)**: Нижняя граница (лучший случай).
   Пример: Алгоритм с `Ω(n)` выполняется не быстрее, чем `c * n`.
3. **Θ (Тета)**: Точная оценка (совпадают O и Ω).
   Пример: Алгоритм с `θ(n log n)` имеет сложность, ограниченную сверху и снизу `c * n log n`.

---

3. Формулы и методы анализа

- **Подсчет операций**:
  Например, цикл `for` выполняется `n` раз → `O(n)`.
- **Рекуррентные соотношения**:
  Для рекурсивных алгоритмов (например, `T(n) = 2T(n/2) + O(n)` → `θ(n log n)`).
- **Мастер-теорема**:
  Решение уравнений вида `T(n) = aT(n/b) + f(n)`.
  Пример: Быстрая сортировка → `T(n) = 2T(n/2) + O(n)` ⇒ `O(n log n)`.

---

4. Примеры временной сложности

Алгоритм	Лучший случай	Средний случай	Худший случай	Пространственная сложность
Линейный поиск	O(1)	O(n)	O(n)	O(1)
Бинарный поиск	O(1)	O(log n)	O(log n)	O(1)
Сортировка пузырьком	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Быстрая сортировка	O(n log n)	O(n log n)	$O(n^2)$	O(log n)
Сортировка слиянием	O(n log n)	O(n log n)	O(n log n)	O(n)

---

5. Линейная сложность (O(n))

Линейный поиск в массиве:

```c
#include <stdbool.h>

bool linear_search(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {  // O(n)
        if (arr[i] == target) {
            return true;
        }
    }
    return false;
}
```

---

6. Квадратичная сложность ($O(n^2)$)

Сортировка пузырьком:

```c
void bubble_sort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {          // O(n)
        for (int j = 0; j < size - i - 1; j++) {  // O(n) → Итого O(n²)
            if (arr[j] > arr[j + 1]) {
                // Обмен элементов
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
```

```
            }
        }
    }
}
```

---

7. Логарифмическая сложность (O(log n))

Бинарный поиск в отсортированном массиве:

```c
int binary_search(int arr[], int size, int target) {
    int low = 0;
    int high = size - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;  // Предотвращает переполнение
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;  // Элемент не найден
}
```

---

8. Пространственная сложность (O(n))

Рекурсивное вычисление факториала:

```c
int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);  // O(n) (стек вызовов)
}
```

---

9. Дополнительные примеры

O(1) — Константное время

Доступ к элементу массива:

```c
int get_element(int arr[], int index) {
    return arr[index];  // O(1)
}
```

O(n log n) — Сортировка слиянием (Merge Sort)

```c
#include <stdlib.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    free(L);
    free(R);
}

void merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        merge_sort(arr, left, mid);      // O(log n)
        merge_sort(arr, mid + 1, right); // O(log n)
        merge(arr, left, mid, right);    // O(n) → Итого O(n log n)
    }
}
```

Вывод

- Временная сложность в **C** определяется так же, как и в других языках.
- **Циклы** (`for` , `while`) и **рекурсия** влияют на сложность.
- **Динамическая память** (например, в Merge Sort) может увеличить пространственную сложность.

Анализ алгоритмов из информатики

Тема 1 Линейные вычислительные процессы.

Программа:

```
#include <iostream>
#include <stdio.h>
#include <math.h>

int main()
{
    int a = 2;
    int b = 3;
    double n1 = sin(a + b);
    double n2 = b - sin(a + b);
    double c = (a + abs(n2)) / (3 - abs(b + pow(n2, 3) / (1 + n1)));
    printf("Result:\n %lf", c);
    return 0;
}
```

Сложность: O(1)

Тема 2 ДЦВП с управлением по аргументу .

Программа:

```
#include <stdio.h>

int main()
{
    int r, n;
    printf("Insert a number:\n");
    scanf_s("%d", &n);
    r = 1;
    for (int i = 1; i <= n; i++)
    {
        r = r * i;
    }
    printf("Result: \t %d", r);
}
```

Сложность: O(n)

Тема 3 ДЦВП с управлением по аргументу. Численное интегрирование.

Программа:

```c
#include <stdio.h>
#include <math.h>

int main()
{
    double a, b, n, r, h;
    printf("Enter the lower integration limit (a):\n");
    scanf_s("%lf", &a);
    printf("Enter the upper integration limit (b):\n");
    scanf_s("%lf", &b);
    printf("Enter the number of splits (n):\n");
    scanf_s("%lf", &n);
    h = (b - a) / n;
    r = 0;
    for (double i = a; i <= b - h; i = i + h)
    {
        double f = cos(0.8 * pow(i, 2) + 1) / 1.4 + sin(0.3 * i + 0.5 * i);
        r = r + f;
    }
    r = r * h;
    printf("%lf", r);
    return 0;
}
```

Сложность: O(n)

Тема 4 Процедуры и функции.

Программа:

```c
#include <stdio.h>
#include <math.h>

double f(double x) {
    return cos(0.8 * x*x + 1) / 1.4 + sin(0.3 * x + 0.5 * x);
}

double simpson(double a, double b, double n) {
    double h, s1, s2;
    s1 = 0;
```

```
    s2 = 0;
    h = (b - a) / n;

    for (double i = a + h; i <= b - h; i = i + 2*h) {
        s1 += f(i);
    }

    for (double i = a + 2*h; i < b - h; i = i + 2*h) {
        s2 += f(i);
    }

    return (h / 3) * (f(a) + f(b) + s1*4 + s2*2);
}

int main() {
    double a, b, n;
    printf("Enter the lower integration limit (a):\n");
    scanf_s("%lf", &a);
    printf("Enter the upper integration limit (b):\n");
    scanf_s("%lf", &b);
    printf("Enter the number of splits (n):\n");
    scanf_s("%lf", &n);
    printf("%lf", simpson(a, b, n));
    return 0;
}
```

Сложность: O(n)

Тема 5 ДЦВП с управлением по индексу. Одномерные массивы.

Программа:

```
#include <stdio.h>
#include <math.h>

int main() {
    double w, Xc, Xl, Z, phi;
    double f[] = {49.8, 49.9, 50, 50.1, 50.2};
    double R = 50;
    double C = pow(10, -6);
    double L = pow(10, -2);
    double llength = sizeof(f) / sizeof(double);
    double pi = 3.1415;

    for (int i = 0; i < llength; i++) {
        w = 2 * pi * f[i];
        Xc = 1 / (w * C);
```

```
            Xl = w * L;
            Z = (Xc * sqrt(Xl * Xl + R * R)) / sqrt(R * R + pow(Xl - Xc, 2));
            phi = atan((Xl - Xc) / R) * (180 / pi);
            printf("Freq: %lf", f[i]);
            printf("\tPhase angle: %lf", phi);
            printf("\tImpedance: %lf", Z);
            printf("\n");
        }
        return 0;
    }
```

Сложность: O(n)

Тема 6 ИЦВП по функции .

Программа:

```
#include <iostream>

int main() {
    int x, s;
    do {
        printf("Enter a three digit number:\n");
        scanf_s("%d", &x);
        s = x % 10 + x / 100 + (x % 100) / 10;
        printf("%d", s);
        printf("\n");
    } while (s > 10);
    return 0;
}
```

Сложность: O(1)

Тема 7 ИЦВП с управлением по аргументу и функции.

Программа:

```
#include <iostream>
#include <math.h>

int main() {
    double Uo; // U output
    double R = 2;
    double C = 0.01;
    double accuracy = 0.001;
    double Uin = 50; // U input
    double t = 0.01;

```

```
    do {
        Uo = Uin * (1 - expf(-t / (R * C)));
        printf("t: %lf", t);
        printf("\tUo: %lf", Uo);
        printf("\n");
        t += 0.01;
    } while (fabs(Uo - Uin) > accuracy);

    return 0;
}
```

Сложность: O(1)

Тема 8 ИЦВП с управлением по аргументу и функции. Вычисление элементарных функций.

Программа:

```
#include <stdio.h>
#include <math.h>

int main(void) {
    double M;
    double U = 1;
    double S = 1;
    double x = 0.5;
    double k = 1;

    do {
        M = x / k;
        U = M * U;
        S += U;
        k += 1;  // Исправлено: k должно увеличиваться на 1, а не на S
    } while (U > 0.0001);

    printf("U: %lf\n", U);
    printf("k: %lf\n", k);
    printf("S: %lf", S);

    return 0;
}
```

Сложность: O(1)

Тема 9 ИЦВП с управлением по индексу и функции.

Программа:

```c
#include <stdio.h>
#include <math.h>

int main(void) {
    int arr[] = {1,2,3,4,5,6,7,8,9,9,8,7,6,5,4,3,2,1};
    int num, sum;
    sum = 0;
    int len = sizeof(arr)/sizeof(int);

    printf("Enter number: ");
    scanf("%d", &num);
    printf("Indexes: ");

    for(int i = 0; i < len; i++) {
        if (arr[i] > num) {
            sum += arr[i];
            printf(" %d", i);
        }
    }

    printf("\nSum: %d", sum);
    return 0;
}
```

Сложность: O(n)

Тема 10 Разветвляющиеся вычислительные процессы.

Программа:

```c
#include <math.h>
#include <stdio.h>

int main(void){

double lambd = 0.1;
double pi = 3.1415926536;
double D = 30 * pi / 180;
double phi = 45 * pi / 180;
double sinA = cos(phi) * sin(lambd) / sin(D);
double cosA = (sin(phi)-sin(phi)*cos(D))/cos(phi)*sin(D);
double A = asin(cos(phi)*(sin(lambd)/sin(D)));

if (sinA > 0 && cosA > 0){
    A = fabs(A);
}
if (sinA > 0 && cosA < 0){
    A = pi - fabs(A);
}
```

```
if (sinA < 0 && cosA < 0){
    A = pi + fabs(A);
}
if (sinA < 0 && cosA > 0){
    A = 2*pi - fabs(A);
}
A = A * 180 / pi;
printf("%lf",A);
}
```

Сложность: O(1)

Тема 11 Итерационные циклические вычислительные процессы с управлением по индексу и функции.

Программа:

```
#include <stdio.h>
#include <math.h>

int main(void){

int arr[25] = {42, -7, 15, 98, -23, 0, 16, 72, -45, 8, 34, 67, -12, 9, 3, -50, 21, 5, 78, -99, 13, 27, -36, 4, 81};
int temp;

for (int i = 0; i <= 25 - 2; i++)
{
    for ( int j = 0; j <= 25 - i - 2; j++)
    {
        if (arr[j] < arr[j+1]){
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}

for (int i = 0; i <= 25 - 1; i++)
{
    printf("%d ", arr[i]);
}
}
```

Сложность: O(n^2)

Тема 12 Комбинированные вычислительные процессы.

Программа:

```c
#include <stdio.h>
#include <math.h>
#include <Windows.h>


double sigma(double x)
{
    double s = 0;
    double ten_fact = 3628800;

    for (double z = 0; z <= 10; z+=2)
    {
        s += ((pow(z,x) + x)/ten_fact);
    }

    return s;
}

int main(void)
{
    SetConsoleOutputCP(CP_UTF8);

    double arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int D1 = 1;
    int D2 = 5;
    int D3 = 9;
    double y;


    for (int i = 0; i <= 7; i++)
    {
        if(arr[i] >= D1 && arr[i] < D2)
        {
            y = sqrt(15+arr[i]);
            printf("%.2f\n",y);
        }
        else if(arr[i] > D2 && arr[i] <= D3)
        {
            y = sigma(arr[i]);
            printf("%.2f\n",y);
        }
        else printf("Элемент не принадлжит ни к одному из интервалов\n");
    }
}
```

Сложность: O(n)

Тема 13 Многоступенчатые вычислительные процессы.

Программа:

```c
#include <stdio.h>
#include <math.h>

#define x_start 0      // Начальное значение x
#define x_end 10       // Конечное значение x
#define step 1         // Шаг
#define m_count 5         // Количество значений m

// Функция для расчёта y
double calc_y(double m, double x) {
    return 2 * m * sqrt(pow(1 + pow(m, 2) - pow(x, 2), 2) + 4 * pow(x, 2));
}

int main() {
    double m_values[m_count] = {1, 2, 3, 4, 5}; // Массив значений m
    int x_steps = ((x_end - x_start) / step) + 1; // Количество шагов по x

    // Заголовок таблицы
    printf("x\t");
    for (int i = 0; i < m_count; i++) {
        printf("y(m=%.1f)\t", m_values[i]);
    }
    printf("\n");

    // Расчёт и вывод значений
    for (int step_index = 0; step_index < x_steps; step_index++) {
        double x = x_start + step_index * step;
        printf("%.2f\t", x);

        for (int i = 0; i <= m_count-1; i++) {
            double y = calc_y(m_values[i], x);
            printf("%.6f\t", y);
        }

        printf("\n");
    }
}

```

Сложность: O(n^2)