Оценка сложности алгоритмов

Лекция 1.

Сложность алгоритма: понятие, виды сложности. Классы сложности.

Простые и составные числа

Число n (n>1) называется простым, если имеет только два положительных делителя (1 и n), иначе – составное.

Идея алгоритма: Перебор всех делителей (k) от 2 до n-1 и проверка делимости на них.

При больших составных $n=k_1*k_2$ (k_1 и k_2 больше 1) достаточно среди нечетных чисел проверить делители до

$$k = \sqrt{n}$$

Основные понятия*

- Решение задачи программируют так, чтобы с помощью программы решить любой возможный *экземпляр задачи, который определяется конкретными входными данными, характеризуемыми некоторым числовым параметром (n)*
- *Экземпляры: «Является ли число 997 простым?»*
- *Операции над значениями скалярных типов (присваивание, сравнение, сложение, умножение и др.) называются элементарным действием.*
- *Время работы программы прямо пропорционально числу выполняемых операций, т.е. измеряется количеством действий.*

** Рассматриваются однопоточные алгоритмы*

Алгоритмы

- Вспомним, что такое «алгоритм».
- Под «алгоритмом» обычно понимают четко определенную последовательность действий, приводящую через конечное число шагов к результату — решению задачи, для которой разработан алгоритм.

Алгоритмы

Основные свойства, присущие любому алгоритму:

- массовость — алгоритм предназначен для решения задачи с некоторым множеством допустимых входных данных;
- конечность — алгоритм должен завершаться за конечное число шагов.

Алгоритмы

- Не для любой задачи существует алгоритм решения. Существуют *алгоритмически неразрешимые* задачи.
- Но даже если алгоритм существует, он может оказаться неприменимым на практике из-за высокой *сложности*.

Алгоритмически неразрешимые задачи

- **Проблема 1** : Распределение девяток в записи числа π;
- Определим функцию $f(n) = i$, где n – количество девяток подряд в десятичной записи числа π, а i – номер самой левой девятки из n девяток подряд: π=3,141592… $f(1) = 5$.
- Задача состоит в вычислении функции $f(n)$

Неразрешимость:

- **Проблема 1** : Распределение девяток в записи числа π;
- Определим функцию $f(n) = i$, где n – количество девяток подряд в десятичной записи числа π, а i – номер самой левой девятки из n девяток подряд:
  π=3,141592… $f(1) = 5$.

Проблема 2: Вычисление совершенных чисел

- Совершенные числа – это числа, которые равны сумме своих делителей, например: 28 = 1+2+4+7+14.

- Определим функцию $S(n)$ = n-ое по счёту совершенное число и поставим задачу: вычисления $S(n)$ по произвольно заданному n.

Неразрешимость:

- Нет общего метода вычисления совершенных чисел, мы даже не знаем, множество совершенных чисел конечно или счетно, поэтому наш алгоритм должен перебирать все числа подряд, проверяя их на совершенность. Отсутствие общего метода решения не позволяет ответить на вопрос о останове алгоритма через конечное число шагов. Если мы проверили M чисел при поиске *n*-ого совершенного числа – означает ли это, что его вообще не существует?

Сложность алгоритма

- **Сложность алгоритма** – это количественная характеристика ресурсов, необходимых алгоритму для успешного решения поставленной задачи.
- Основные ресурсы:
  - **время** (*временна́я сложность*) и
  - **объем** памяти (*ёмкостная сложность*).
- Наиболее важной характеристикой является время.

Модель вычислений RAM
Random Access Machine

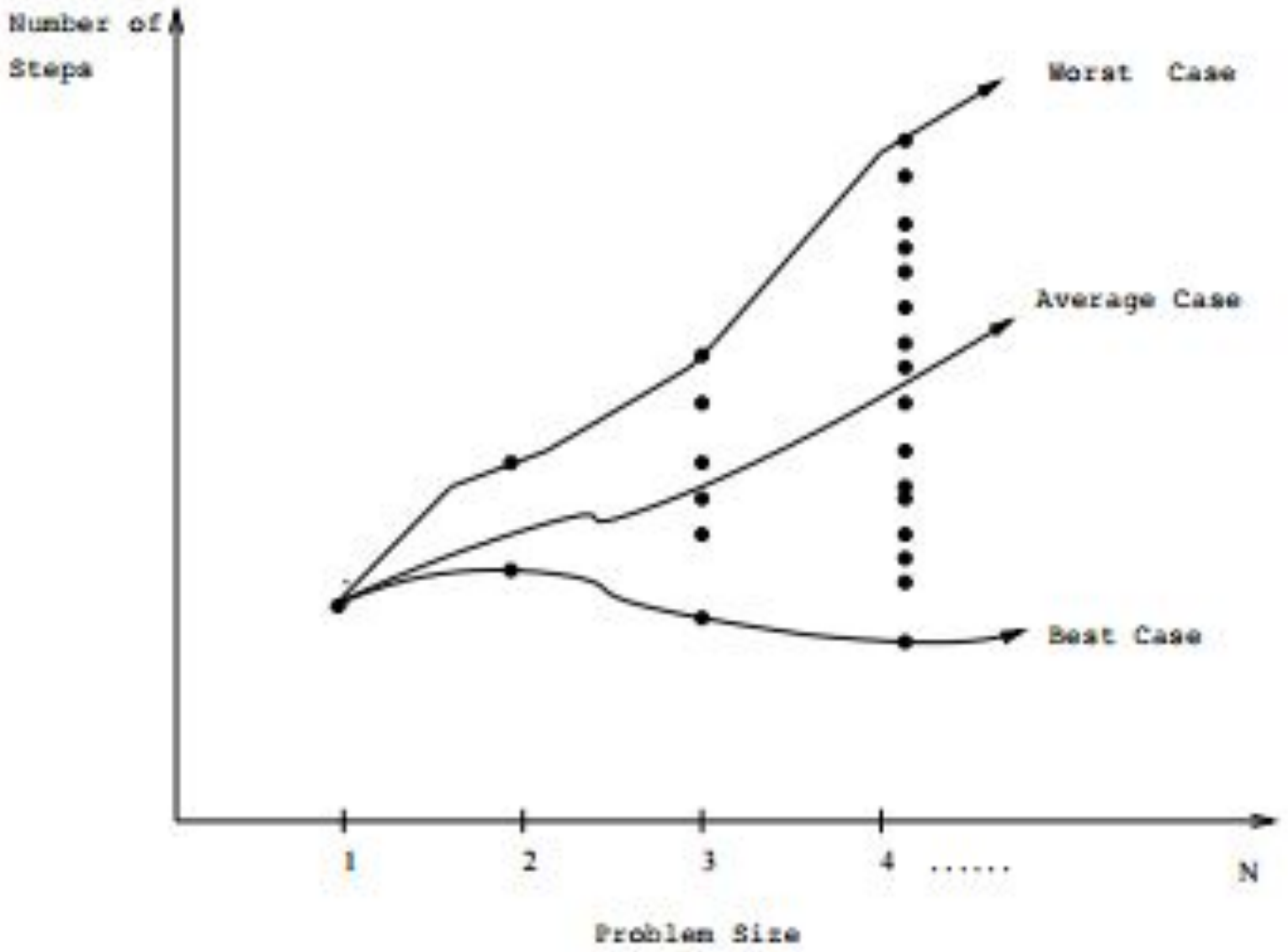
- Исполнение каждой "простой" операции (+, -, =, if, call) занимает один временной шаг;
- Циклы и подпрограммы не считаются простыми операциями, а состоят из нескольких простых операций;
- Каждое обращение к памяти занимает один временной шаг/

- Время исполнения алгоритма в RAM-модели вычисляется по общему количеству шагов, требуемых алгоритму для решения данного экземпляра задачи.

Чтобы получить общее представление о сложности алгоритма, необходимо знать, как он работает со всеми экземплярами задачи

Анализ сложности наилучшего, наихудшего и среднего случаев

- *OX: размер входа задачи (кол-во эл-тов и при сортировке и проч.)*
- *OY: кол-во шагов алгоритма для обработки данного входного экземпляра задачи*

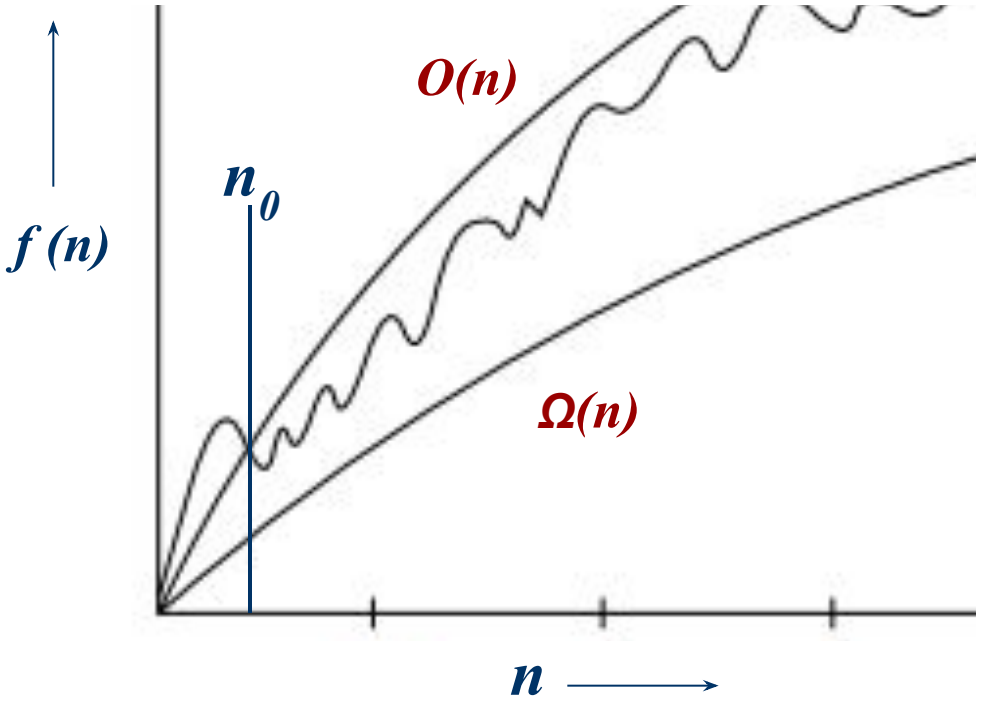

Number of
Steps
Worst Case
Average Case
Best Case
1
2
3
4
.....
N
Problem Size

Сложность алгоритма --

- В наихудшем случае -- функция, определяемая максимальным количеством шагов, требуемых для обработки любого входного экземпляра размером ***n;***
- В наилучшем случае -- функция, определяемая минимальным количеством шагов, требуемых для обработки любого входного экземпляра размером ***n;***
- В среднем случае -- функция, определяемая средним количеством шагов, требуемых для обработки всех экземпляров размером ***n;***

Асимптотические обозначения

- **«Лучший, худший и средний»: затруднено точное определение именно потому, что детали алгоритма являются очень сложными**



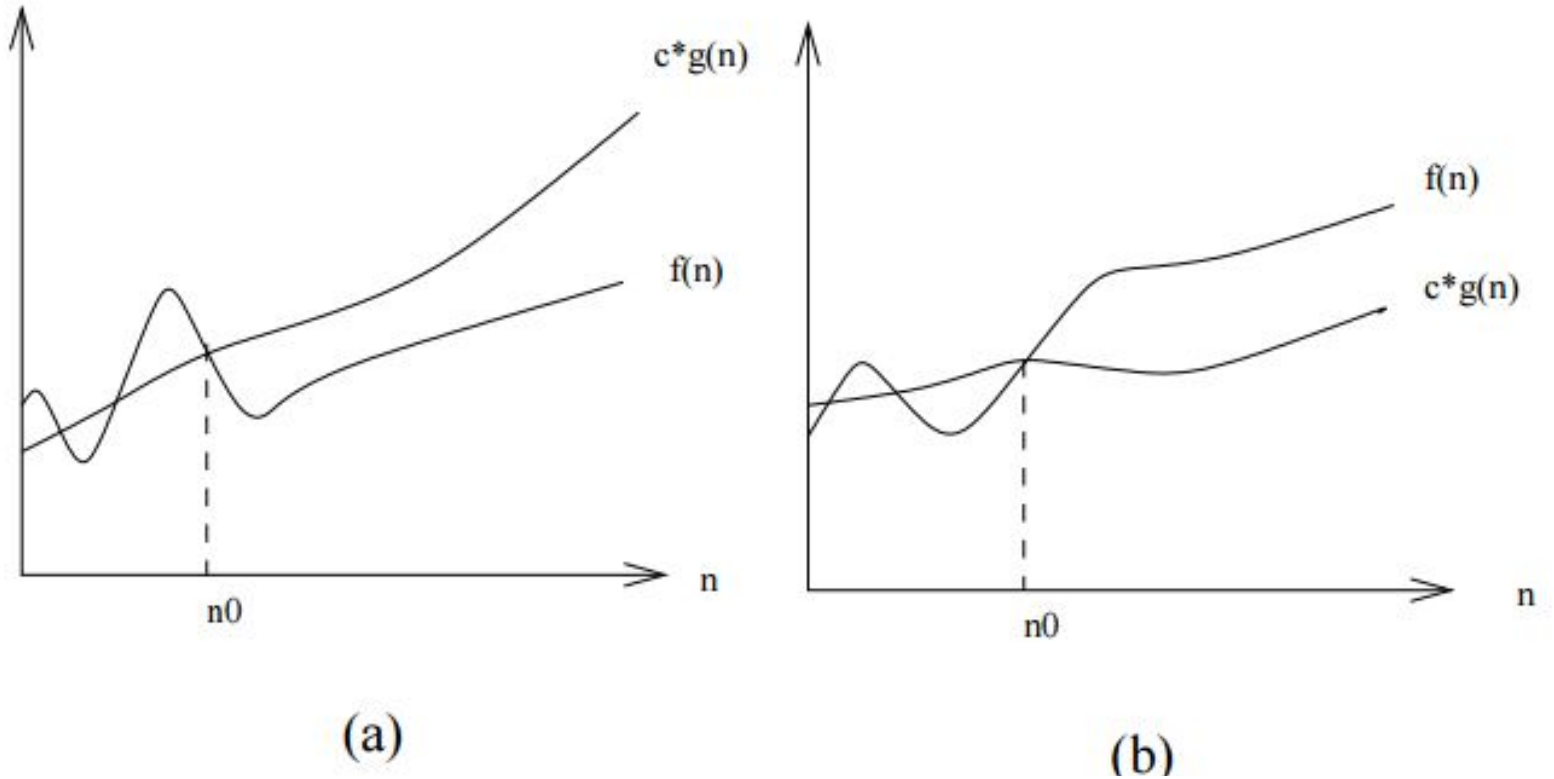
Легче говорить о верхних и нижних пределах функции

Асимптотическая нотация (O, Θ, Ω)

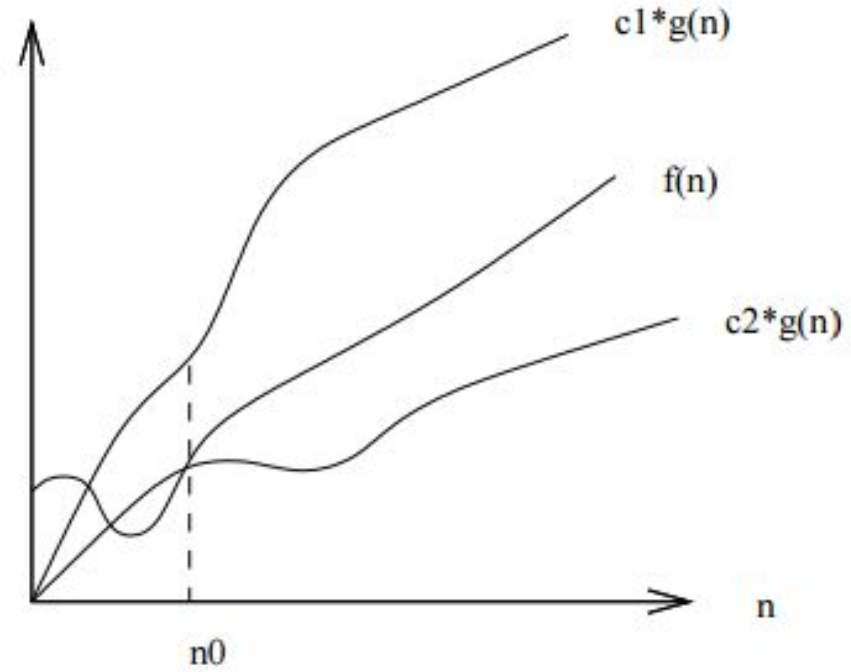
Смысл асимптотических функций:

- $g(n) = O(f(n))$ означает, что $C \times f(n)$ является верхней границей функции $g(n)$
- $g(n) = \Omega(f(n))$ означает, что $C \times f(n)$ является нижней границей функции $g(n)$.
- • $g(n) = \Theta(f(n))$ означает, что $C1 \times f(n)$ выше функции $g(n)$ и $C2 \times f(n)$ ниже функции $g(n)$.

!!! C, C1, и C2 не зависят от n


c*g(n)
f(n)
n
n0
(a)
f(n)
c*g(n)
n
n0
(b)

В каждом из этих определений фигурирует константа n_0, после которой эти определения всегда верны



(c)

Формальные определения:

- $f(n) = O(g(n))$ означает, что функция $f(n)$ ***ограничена сверху*** функцией $c \cdot g(n)$, т. е. существует такая константа c , для которой $f(n) <= c \cdot g(n)$ при достаточно большом n $(n>=n_0)$;
- $f(n) = \Omega(g(n))$ означает, что функция $f(n)$ ***ограничена снизу*** функцией $c \cdot g(n)$, т. е. существует такая константа c , для которой $f(n) >= c \cdot g(n)$ для всех n $(n>=n_0)$;
- $f(n) = \Theta(g(n))$ означает, что функция $f(n)$ ***ограничена сверху*** функцией $c_1 \cdot g(n)$, ***а снизу --*** функцией $c_2 \cdot g(n)$, т. е. существуют такие константы c_1 и c_2, для которых $c_2 \cdot g(n) <= f(n) <= c_1 \cdot g(n)$ для всех n $(n>=n_0)$

Пример. Алгоритм нахождения минимального элемента массива A из n элементов.

```
min=A[0];
for(i=1;i<n;i++)
{
    if(A[i]<min)
        min=A[i];
}
```

Временная сложность алгоритма в наилучшем случае – **n**

Пример входных данных: **(1,4,6,8,2,5)**

Временная сложность алгоритма в худшем случае – f(n)=**(n-1)*2+1=2n-1**

Пример входных данных: **(8,6,5,4,2,1)**

Асимптотическая сложность алгоритма - ***O(n)***

c=2, 2n-1<2n

Примеры

$$3n^2 - 100n + 6 = O(n^2) \quad 3n^2 > 3n^2 - 100n + 6$$
$$3n^2 - 100n + 6 = O(n^3) \quad 1n^3 > 3n^2 - 100n + 6$$
$$3n^2 - 100n + 6 \neq O(n) \quad c \cdot n < 3n^2 \ when \ n > c$$

$$3n^2 - 100n + 6 = \Omega(n^2) \quad 2n^2 < 3n^2 - 100n + 6$$
$$3n^2 - 100n + 6 \neq \Omega(n^3) \quad 3n^2 - 100n + 6 < n^3$$
$$3n^2 - 100n + 6 = \Omega(n) \quad 10^{10^{10}} n < 3n^2 - 100 + 6$$

$$3n^2 - 100n + 6 = \Theta(n^2) \quad O \ \Omega$$
$$3n^2 - 100n + 6 \neq \Theta(n^3) \quad O$$
$$3n^2 - 100n + 6 \neq \Theta(n) \quad \Omega$$

Скорость роста O-функций

n (размер задачи)	O(n)	$O(2^n)$
50	1 сек	1 сек
51	1,02 сек	2 сек
60	1,2 сек	17 мин
70	1,4 сек	12 суток
80	1,6 сек	34 года
90	1,8 сек	~35 тыс.лет

Свойства асимптотических функций

$$O(c \cdot f(n)) \to O(f(n))$$
$$\Omega(c \cdot f(n)) \to \Omega(f(n))$$
$$\Theta(c \cdot f(n)) \to \Theta(f(n))$$

1) Умножение на константу ***c>0*** – не меняет асимптотических функций

2) При возрастании функций сложение и произведение определяются соотношениями:

$$O(f(n)) + O(g(n)) = O(max(f(n),g(n))$$
$$\Omega(f(n)) + \Omega(g(n)) = \Omega(max(f(n),g(n))$$
$$\Theta(f(n)) + \Theta(g(n)) = \Theta(max(f(n),g(n))$$

$$O(f(n)) * O(g(n)) \to O(f(n) * g(n))$$
$$\Omega(f(n)) * \Omega(g(n)) \to \Omega(f(n) * g(n))$$
$$\Theta(f(n)) * \Theta(g(n)) \to \Theta(f(n) * g(n))$$

Класс алгоритмов

№ п.п.(От сложного к простому)	Название сложности	Мат. формула	Примеры алгоритмов
1	Факториальная	N!	Алгоритмы комбинаторики (сочетания, перестановки и т.д.)
2	Экспоненциальная	K^N	Алгоритмы перебора (*brute force*)
3	Полиномиальная	N^K	Алгоритмы простой сортировки (*buble sort*)
4	Линейный логарифм	N * log(N)	Алгоритмы быстрой сортировки (*heap sort*)
5	Линейная	K * N	Перебор элементов массива
6	Логарифмическая	K * log(N)	Бинарный поиск
7	Константная	K	Обращение к элементу массива по индексу

ЗАПОМНИТЬ

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1$$

$$n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\epsilon} \gg n \log n \gg n \gg \sqrt{n} \gg$$
$$\log^2 n \gg \log n \gg \log n / \log \log n \gg \log \log n \gg \alpha(n) \gg 1$$

1) Расположите функции в возрастающем асимптотическом порядке:

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1$$

- **Проблема 1** : Распределение девяток в записи числа π;
- Определим функцию $f(n) = i$, где n – количество девяток подряд в десятичной записи числа π, а i – номер самой левой девятки из n девяток подряд: π=3,141592... $f(1) = 5$.
- Задача состоит в вычислении функции $f(n)$ для произвольно заданного n.

Оценка сложности алгоритмов

1) Какое значение возвращает функция? Ответ должен быть в виде функции числа n. Определите сложность алгоритма в наихудшем случае (O(n)):

```
function mistery(n)
 r:=0
 for i:=1 to n-1 do
    for j:=i+1 to n do
       for k:=1 to j do
            r:=r+1
 return(r)
```

Оценка сложности алгоритмов

```
function mystery(n)
    r:=0
    for i:=1 to n-1 do
        for j:=i+1 to n do
            for k:=1 to j do
                r:=r+1
     return(r)
```

$$mystery(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{j} 1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} j$$

Сумма членов арифметической прогрессии

1-го порядка:

$$\sum_{x=1}^{n} x = \frac{1}{2} n(n+1)$$

2 порядка:

$$\sum_{x=1}^{n} x^2 = \frac{1}{6} n(n+1)(2n+1)$$

3 порядка:

- **Проблема 1** : Распределение девяток в записи числа π;
- Определим функцию $f(n) = i$, где n – количество девяток подряд в десятичной записи числа π, а i – номер самой левой девятки из n девяток подряд: π=3,141592… $f(1) = 5$.
- Задача состоит в вычислении функции $f(n)$ для произвольно заданного n.

Решение задачи (1 вариант)

$$\sum_{j=i+1}^{n} j = \sum_{j=1}^{n} j - \sum_{j=1}^{i} j = \sum_{i=1}^{n-1} \left(\frac{1}{2} n(n+1) - \frac{1}{2} i(i+1) \right)$$

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

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

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

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

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

$$= \frac{1}{2}\, n(n-1)(n+\frac{1}{2}) - \frac{1}{2}\,\frac{1}{6}\, n(n-1)(2n-1) =$$
$$= \frac{1}{2}\, n(n-1)(n+\frac{1}{2} - \frac{1}{6}\,(2n-1)) =$$
$$= \frac{1}{2}\, n(n-1)(n+\frac{1}{2} - \frac{1}{3}\, n + \frac{1}{6}) =$$
$$= \frac{1}{2}\, n(n-1)(\frac{2}{3}\, n + \frac{2}{3}) =$$
$$= \frac{1}{3}\, n(n-1)(n+1)$$

Оценка сложности алгоритмов

Сортировка методом выбора:

$$S(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1} n - i - 1$$

$$S(n) = (n-2) + (n-3) + \ldots + 2 + 1 + 0 =$$

$$= (n-1)(n-2)/2$$

$$\Theta(n^2)$$

```
selection_sort(int s[], int n)
{
    int i,j;          // счетчики
    int min;          //указатель min элемента

    for (i=0; i<n; i++) {
        min=i;
        for (j=i+1; j<n; j++)
            if (s[j] < s[min]) min=j;
        swap(&s[i],&s[min]);
    }
}
```

Домашнее задание:

- **Проблема 1** : Распределение девяток в записи числа π;
- Определим функцию $f(n) = i$, где n – количество девяток подряд в десятичной записи числа π, а i – номер самой левой девятки из n девяток подряд: π=3,141592… $f(1) = 5$.
- Задача состоит в вычислении функции $f(n)$ для произвольно заданного n.

- 2) Какое значение возвращает функция? Ответ должен быть в виде функции числа n. Определите сложность алгоритма в наихудшем случае (O(n)):

```
function pesky(n)
        r:=0
              for i:=1 to n do
                 for j:= 1 to i do
                    for k:=j to j+i do
                     r:=r+1
        return(r)
```