Понятие алгоритма. Краткое введение в теорию вычислимости и теорию формальных грамматик.

Хайрулин Сергей Сергеевич
s.khayrulin@gmail.com

Overview

- Понятие алгоритм, свойства
- Примеры
- Формализованное понятие алгоритма.
- Машина тьюринга.
- Тезис Черча
- Классы задач P и NP
- Примеры NP полных проблем
- O - нотация
- Асимптотическая сложность алгоритмов

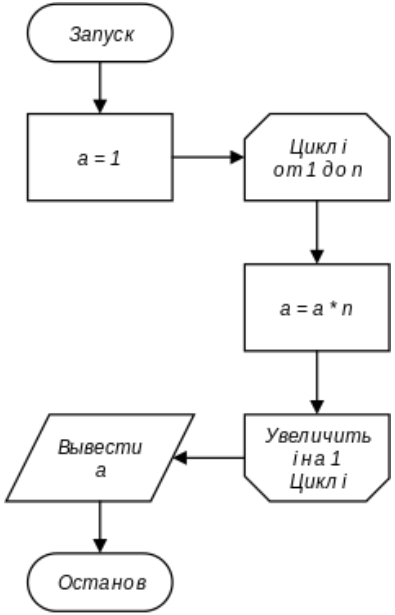
Литература и др. источники

- Дональд Эрвин Кнут. Искусство программирования (Том 1, 2, 3) // Вильямс 2015.
- Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. Структуры данных и алгоритмы // Вильямс 2000.
- Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И. Лекции по теории графов // М.: Наука, 1990.
- Харари Ф. Теория графов // М.: Мир, 1973.
- Косточка А. В. Дискретная математика. Часть 2 //Новосибирск: НГУ, 2001.
- Котов В. Е., Сабельфельд В. К. Теория схем программ // Наука 1991.
- http://algolist.manual.ru
- ….

Алгоритм

Алгори́тм — набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата. Независимые инструкции могут выполняться в произвольном порядке, параллельно, если это позволяют используемые исполнители.

(Википедия)



Алгоритм

```
algorithm ford-fulkerson is
    input: Graph G with flow capacity c, source node s, sink node t
    output: Flow f such that f is maximal from s to t
    (Note that f(u,v) is the flow from node u to node v, and c(u,v) is the
     flow capacity from node u to node v)
    for each edge (u, v) in GE do
        f(u, v) ← 0
        f(v, u) ← 0
    while there exists a path p from s to t in the residual network Gf do
        let cf be the flow capacity of the residual network Gf
        cf(p) ← min{cf(u, v) | (u, v) in p}
        for each edge (u, v) in p do
            f(u, v) ←  f(u, v) + cf(p)
            f(v, u) ← −f(u, v)
    return f
```

Свойства

- **Конечность описания** — любой алгоритм задается как набор инструкций конечных размеров, т. е. программа имеет конечную длину.
- **Дискретность** — алгоритм выполняется по шагам, происходящим в дискретном
- времени. Шаги четко отделены друг от друга. В алгоритмах нельзя использовать аналоговые устройства и непрерывные методы.
- **Направленность** — у алгоритма есть входные и выходные данные. В алгоритме четко указывается, когда он останавливается, и что выдается на выходе после остановки.
- **Массовость** — алгоритм применим к некоторому достаточно большому классу
- однотипных задач, т. е. входные данные выбираются из некоторого, как правило, бесконечного множества.
- **Детерминированность** (или конечная недетерминированность) — вычисления продвигаются вперед детерминировано, т. е. вычислитель однозначно представляет, какие инструкции необходимо выполнить в текущий момент. Нельзя использовать случайные числа или методы. Конечная недетерминированность означает, что иногда в процессе работы алгоритма возникает несколько вариантов для дальнейшего хода вычислений, но таких вариантов лишь конечное.

Алгоритм вычисления чисел Фибоначчи

$$F_1 = 1, F_2 = 1, \ldots, F_n = F_{n-1} + F_{n-2}$$

```
function Fibo(n)
  if n = 1 or n = 2
    return 1
  end if
  return Fibo(n - 1) + Fibo(n - 2)
end function
```

Вычисление корней квадратного уравнения

```
function sqrt_roots(a, b, c):
    d = b * b - 4 * a * c
    if d < 0:
        print(“There are no roots for equation”)
    else:
        if d == 0:
            x_1 = b/(2*a)
            print(“Equation has only one root x_1 = ” + str(x_1))
        else:
            x_1 =  (b + sqrt(d))/(2*a)
            x_2 =  (b - sqrt(d))/(2*a)
            print(“Equation has two roots x_1 =  ” + str(x_1) + “ x_2 = ” + str(x_2))
```

Формализация понятия алгоритма

- Машина Поста
- Нормальный алгоритм Маркова
- Рекурсивные функции
- Машина Тьюринга
- ...

Машина Тьюринга

Определение. *Машина Тьюринга* — это пятерка $T = \langle A, Q, P, q_1, q_0 \rangle$, где:

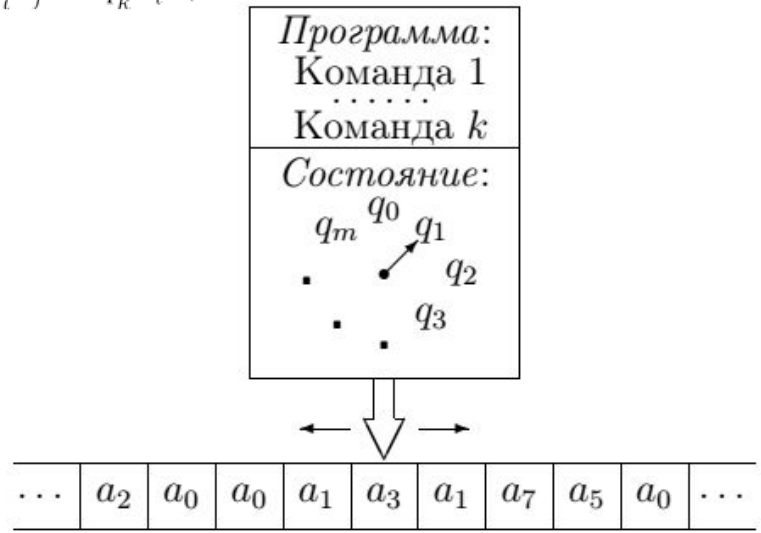
a) $A = \{a_1, \dots, a_n\}$ – конечный внешний алфавит (мы будем всегда предплогать, что $n \geq 1$ и $a_0 = 0, a_1 = 1$);

b) $Q = \{q_0, \dots, q_m\}$ – конечный алфавит внутренних состояний;

c) $P = \{T(i,j) | 1 \leq i \leq m, 1 \leq j \leq n\}$- программа, состоящая из команд $T(i,j)$, каждая из которых есть слово вида: $q_i a_j \to q_k a_l$, или $q_i a_j \to q_k a_l R$, или $q_i a_j \to q_k a_l L$, где $1 \leq k \leq m$, $1 \leq l \leq n$;

d) q_1 - начальное состояние

e) q_0 - конечное состояние



Пример

$$T = (\{(,), 0, 1, *\}, \{q_0, q_1, q_2, q_3, q_4\}, P)$$

1. $q_0(\rightarrow q_0(R$
2. $q_0) \rightarrow q_1 * L$
3. $q_0* \rightarrow q_0 * R$
4. $q_0\# \rightarrow q_2\#L$
5. $q_1(\rightarrow q_0 * R$
6. $q_1* \rightarrow q_1 * L$
7. $q_1\# \rightarrow q_4 0R$
8. $q_2(\rightarrow q_3\#L$
9. $q_2* \rightarrow q_2\#L$
10. $q_2\# \rightarrow q_1 1$
11. $q_3(\rightarrow q_3\#L$
12. $q_3* \rightarrow q_3\#L$
13. $q_3\# \rightarrow q_1 0$
14. $q_4(\rightarrow q_4\#R$
15. $q_4) \rightarrow q_4\#R$
16. $q_4* \rightarrow q_4\#R$

Тезис Черча

Тезис Чёрча. Класс интуитивно вычислимых функций совпадает с классом всех функций вычислимых по Тьюрингу.

Как посчитать эффективность алгоритма???

В чем измерять эффективность алгоритма описанного с помощью какого-либо Тьюринг-полного языка программирования? От чего вообще может зависеть этот показатель?

Как посчитать эффективность алгоритма???

В чем измерять эффективность алгоритма описанного с помощью какого-либо Тьюринг-полного языка программирования? **От чего вообще может зависеть этот показатель?**

- Данные

Как посчитать эффективность алгоритма???

От чего вообще может зависеть этот показатель?

- Данные
- Память и какие-либо другие ресурсы

Как посчитать эффективность алгоритма???

В чем измерять эффективность алгоритма описанного с помощью какого-либо Тьюринг-полного языка программирования (в зависимости от входных данных)?

- Количество действий сделанных алгоритмом во время работы от начала и до конца.

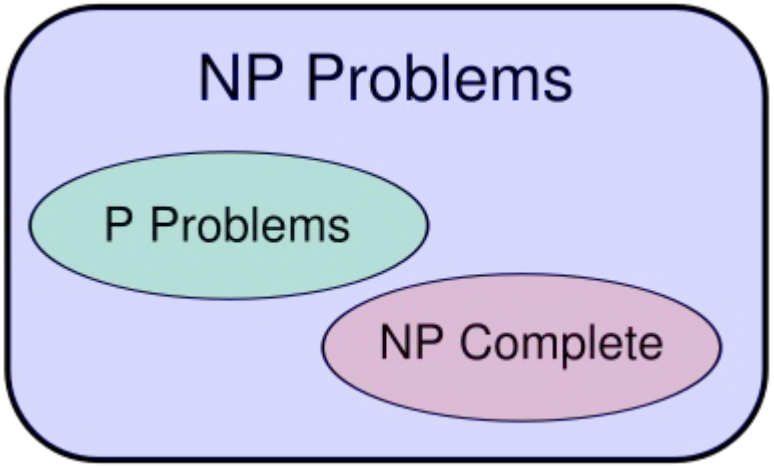
Как посчитать эффективность алгоритма???

В чем измерять эффективность алгоритма описанного с помощью какого-либо Тьюринг-полного языка программирования (в зависимости от входных данных)?

- Количество действий сделанных алгоритмом во время работы от начала и до конца.
- Количество памяти необходимой алгоритму для работы.

P=NP?

Имеет место включение **P ⊆ NP**



Примеры NP полных проблем.

- Задача о клике: по данному графу узнать, есть ли в нём клики (полные подграфы) заданного размера.
- Оптимизационный вариант задачи о коммивояжере (существует ли маршрут не длиннее, чем заданное значение k) — расширенный и более приближенный к реальности вариант предыдущей задачи.
- Сапер
- Быки и коровы
- Полный список в Интернете

O - нотация

- Пусть $f(x)$ и $g(x)$ — две функции, определенные в некоторой проколотой окрестности точки x_0, причем в этой окрестности g не обращается в ноль. Говорят, что:
  - f является «O» большим от g при $x \to x_0$, если существует такая константа $C > 0$, что для всех x из некоторой окрестности точки x_0 имеет место неравенство
    $$|f(x)| \leq C|g(x)|;$$
  - f является «o» малым от g при $x \to x_0$, если для любого $C > 0$ найдется такая проколотая окрестность U'_{x_0} точки x_0, что для всех $x \in U'_{x_0}$ имеет место неравенство
    $$|f(x)| < C|g(x)|.$$

Иначе говоря, в первом случае отношение $|f|/|g|$ в окрестности точки x_0 ограничено сверху, а во втором оно стремится к нулю при $x \to x_0$.

O - нотация

Пусть f(x) и g(x) - две функции определенные в некоторой проколотой окрестности точки x_0, причем в этой окрестности g(x) не обращается в 0. Говорят что:

- f(x) является “O” от g(x) при $x \to x_0$, если существует такая константа $C > 0$, что для любой точки x из некоторой окрестности x_0, верно неравенство:
  $|f(x)| \leq C|g(x)|$
- f(x) является “o” от g(x) при $x \to x_0$, если для любой константы $C > 0$, существует проколотая окрестность x_0(), что для всех точек x из окрестности U_{x0} верно неравенство:
  $|f(x)| < C|g(x)|$

Асимптотическая сложность алгоритмов

Обозначение	**Граница**	**Рост**
(Тета) Θ	Нижняя и верхняя границы, точная оценка	Равно
(О - большое) O	Верхняя граница, точная оценка неизвестна	Меньше или равно
(о - малое) o	Верхняя граница, не точная оценка	Меньше
(Омега - большое) Ω	Нижняя граница, точная оценка неизвестна	Больше или равно
(Омега - малое) ω	Нижняя граница, не точная оценка	Больше

Асимптотическая сложность алгоритмов

Алгоритм	Эффективность
$o(n)$	$< n$
$O(n)$	$\leq n$
$\Theta(n)$	$= n$
$\Omega(n)$	$\geq n$
$\omega(n)$	$> n$

Свойства

Транзитивность

$$
\begin{aligned}
f(n) = \Theta(g(n)) \land g(n) = \Theta(h(n)) &\Rightarrow f(n) = \Theta(h(n)) \\
f(n) = O(g(n)) \land g(n) = O(h(n)) &\Rightarrow f(n) = O(h(n)) \\
f(n) = \Omega(g(n)) \land g(n) = \Omega(h(n)) &\Rightarrow f(n) = \Omega(h(n)) \\
f(n) = o(g(n)) \land g(n) = o(h(n)) &\Rightarrow f(n) = o(h(n)) \\
f(n) = \omega(g(n)) \land g(n) = \omega(h(n)) &\Rightarrow f(n) = \omega(h(n))
\end{aligned}
$$

Свойства

Рефлективность

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

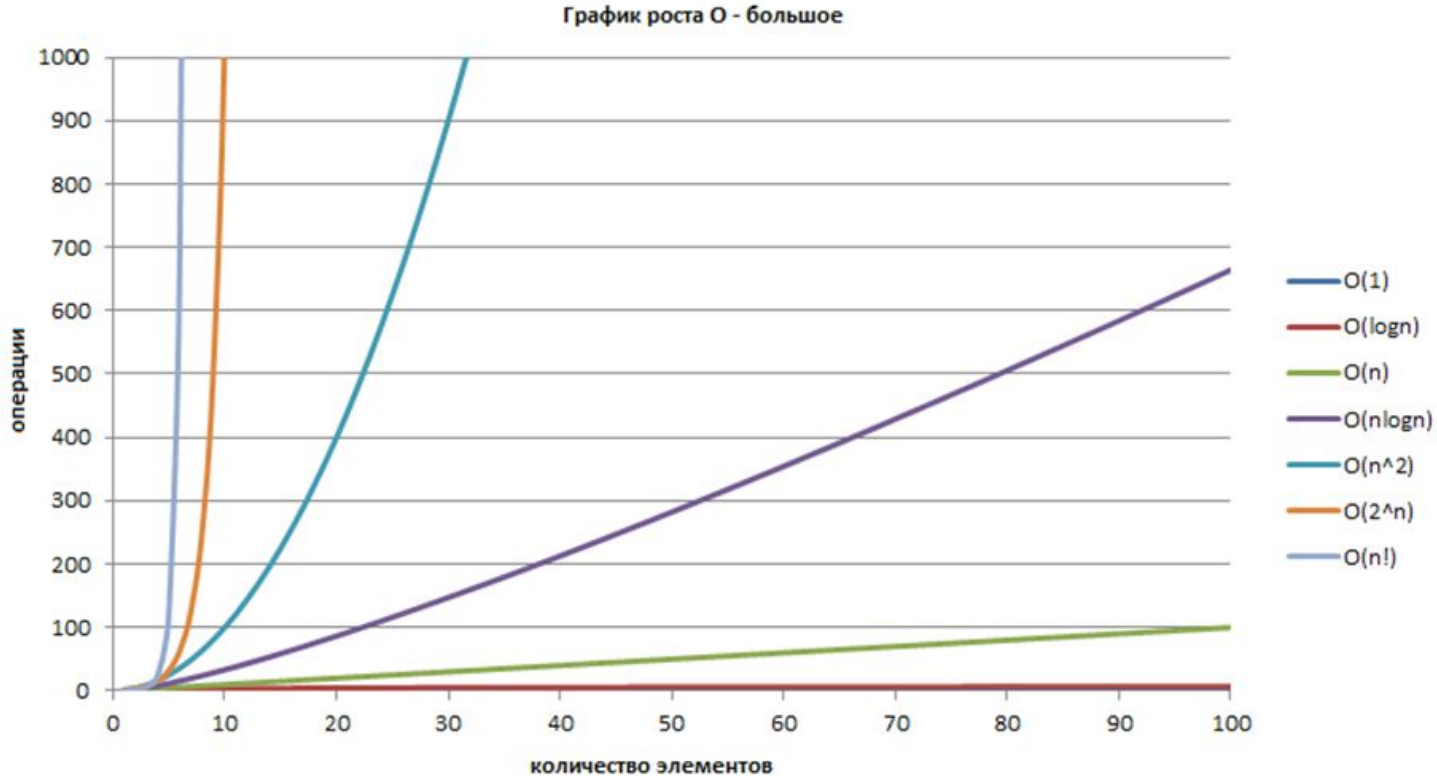
Симметричность

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

Свойства

$$C \cdot o(f(n)) = o(f(n))$$
$$C \cdot O(f(n)) = O(f(n))$$
$$o(C \cdot f(n)) = o(f(n))$$
$$O(C \cdot f(n)) = O(f(n))$$
$$o(-f(n)) = o(f(n))$$
$$O(-f(n)) = O(f(n))$$
$$o(f(n)) + o(f(n)) = o(f(n))$$
$$o(f(n)) + O(f(n)) = O(f(n)) + O(f(n)) = O(f(n))$$
$$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$$
$$o(f(n)) \cdot O(g(n)) = o(f(n)) \cdot o(g(n)) = O(f(n) \cdot g(n))$$
$$O(O(f(n))) = O(f(n))$$
$$o(o(f(n))) = o(O(f(n))) = O(o(f(n))) = o(f(n))$$

Асимптотическая сложность алгоритмов



O(1)

Порядок роста O(1) означает, что вычислительная сложность не зависит от размера входных данных.

O(n)

O(n) - линейный рост сложности алгоритма в зависимости от входных данных.

```
for i in range(N):
    # do some useful stuff
    ...
```

$O(n^2)$

$O(n^2)$ - квадратичная зависимость от входных данных

```
for i in range(N):
    for j in range(N)
        # do some useful stuff
        ...
```

O(log(n))

Порядок роста $O(\log n)$ означает, что время выполнения алгоритма растет логарифмически с увеличением размера входного массива.

$O(e^n)$

Временная сложность алгоритма экспоненциально зависит от данных.

O(n!)

Задачи связанные с полным перебором обычно обычно решаются за факториальное время. Например задача коммивояжёра решаемая метом полного перебора.

Поиск

Алгоритм	Структура данных	Временная сложность		Сложность по памяти
		В среднем	В худшем	В худшем
Поиск в глубину (DFS)	Граф с |V| вершинами и |E| ребрами	-	O(|E| + |V|)	O(|V|)
Поиск в ширину (BFS)	Граф с |V| вершинами и |E| ребрами	-	O(|E| + |V|)	O(|V|)
Бинарный поиск	Отсортированный массив из n элементов	O(log(n))	O(log(n))	O(1)
Линейный поиск	Массив	O(n)	O(n)	O(1)
Кратчайшее расстояние по алгоритму Дейкстры используя двоичную кучу как очередь с приоритетом	Граф с |V| вершинами и |E| ребрами	O((|V| + |E|) log |V|)	O((|V| + |E|) log |V|)	O(|V|)
Кратчайшее расстояние по алгоритму Дейкстры используя массив как очередь с приоритетом	Граф с |V| вершинами и |E| ребрами	O(|V|^2)	O(|V|^2)	O(|V|)
Кратчайшее расстояние используя алгоритм Беллмана–Форда	Граф с |V| вершинами и |E| ребрами	O(|V||E|)	O(|V||E|)	O(|V|)

https://habrahabr.ru/post/188010/

Сортировка

Алгоритм	Структура данных	Временная сложность			Вспомогательные данные
		Лучшее	В среднем	В худшем	В худшем
Быстрая сортировка	Массив	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Сортировка слиянием	Массив	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Пирамидальная сортировка	Массив	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Пузырьковая сортировка	Массив	O(n)	O(n^2)	O(n^2)	O(1)
Сортировка вставками	Массив	O(n)	O(n^2)	O(n^2)	O(1)
Сортировка выбором	Массив	O(n^2)	O(n^2)	O(n^2)	O(1)
Блочная сортировка	Массив	O(n+k)	O(n+k)	O(n^2)	O(nk)
Поразрядная сортировка	Массив	O(nk)	O(nk)	O(nk)	O(n+k)

https://habrahabr.ru/post/188010/

Структуры данных

Структура данных	Временная сложность								Сложность по памяти
	В среднем				В худшем				В худшем
	Индексация	Поиск	Вставка	Удаление	Индексация	Поиск	Вставка	Удаление	
Обычный массив	O(1)	O(n)	-	-	O(1)	O(n)	-	-	O(n)
Динамический массив	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Односвязный список	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Двусвязный список	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Список с пропусками	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Хеш таблица	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Бинарное дерево поиска	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Декартово дерево	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
B-дерево	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Красно-черное дерево	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Расширяющееся дерево	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
АВЛ-дерево	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

https://habrahabr.ru/post/188010/

Указания для задач

Для замера работы функции нужно использовать метод now() класса datetime модуля datetime

Указания для задач

```
array = [0] * N
array.insert(N,0)
```

```
import datetime


def main():
    t1 = datetime.datetime.now()
    #You'r code here
    ...
    print(datetime.datetime.now() - t1)

if __name__ == '__main__':
    main()
```

Указания для задач

```
import numpy as np


...
# Generate numpy Array with N random numbers
array = np.random.rand(N)
#Sort Array by quick sort
sorted(array)
...
```

Указания для задач

jupyter MyTests Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Help Python 2

Code CellToolbar

In [6]:
```
import datetime
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

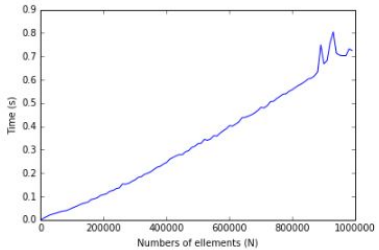
In [7]:
```
times = []
for N in xrange(1000, 1000000, 10000):
    array = np.random.rand(N)
    t1 = datetime.datetime.now()
    sorted(array)
    t_res = datetime.datetime.now() - t1
    times.append([N,t_res])
y = [t[1].total_seconds() for t in times]
x = [t[0] for t in times]
```

In [8]:
```
%matplotlib inline
```

In [9]:
```
plt.plot(x, y)

plt.xlabel('Numbers of ellements (N)')
plt.ylabel('Time (s)')

plt.savefig("test.png")
plt.show()
```



Задачи

- Реализовать алгоритм перемножения квадратных матриц. Матрицы могут задаваться как список списков. Считывать можно из файла потока ввода, или задавать случайным образом (используя функцию np.random.rand(N)). Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Найти все пифагоровы тройки ($c^2 = a^2 + b^2$) для заданного интервала. Интервал задается парой чисел через пробел считанных из входного потока (например: 10 100) помните, что верхняя грань отрезка должна быть больше нижней. Если задано одно число, то считаем, что ограничение снизу равно по умолчанию 1. Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Реализовать алгоритм факторизации числа (разложение числа как произведение двух других чисел). Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Реализовать алгоритм рассчитывающий сочетания и размещения.
- Факториал довольно емкостная функция, при расчете которого для больших значений может случится переполнение (т.е. полученное число будет больше чем максимально возможное число в вашей системе). Подумайте как преодолеть эту проблему.

Спасибо за внимание!