Векторизация by Kirill Suglobov, 317 gr.

Три реализации задачи "E. Кодирование длин серий. 2021"

Ниже приведены три реализации рассматриваемой задачи.

1. Полностью невекторизованная реализация (без использования NumPy)

In [1]:
```
def encode_rle(x):
    l = len(x)
    keys = []
    quant = []
    for i in range(l):
        if (i == 0):
            cur = x[0]
            count = 1
        elif (i != 0 and x[i] == cur):
            count += 1
        else:
            keys.append(cur)
            quant.append(count)
            cur = x[i]
            count = 1
        if (i == l - 1):
            keys.append(cur)
            quant.append(count)
    return (keys, quant)
```

In [2]:
```
print(encode_rle([0, 0, 1, 1, 1, 2, 1]))
```

```
([0, 1, 2, 1], [2, 3, 1, 1])
```

2. Частично векторизованная реализация

In [3]:
```
import numpy as np


def encode_rle(x):
    mask = np.concatenate(([True], x[1:] != x[:-1]))
    (keys, quant) = (np.array([]), np.array([]))
    l = np.shape(x)[0]
    prev_i = 0
    for (i, j) in zip(range(l), mask):
        if (j):
            keys = np.append(keys, x[i])
            if i != 0:
                quant = np.append(quant, i - prev_i)
                prev_i = i
        if (i == l - 1):
            quant = np.append(quant, i - prev_i + 1)
    return (keys, quant)
```

In [4]:
```
print(encode_rle(np.array([0, 0, 1, 1, 1, 2, 1])))
```

```
(array([0., 1., 2., 1.]), array([2., 3., 1., 1.]))
```

3. Полностью векторизованная реализация

In [5]:
```
import numpy as np


def encode_rle(x):
    if np.shape(x)[0] == 0:
        return ([], [])
    mask = np.concatenate(([True], x[1:] != x[:-1], [True]))
    return (x[mask[:-1]], np.diff(np.where(mask)[0]))
```

In [6]:
```
print(encode_rle(np.array([0, 0, 1, 1, 1, 2, 1])))
```

```
(array([0, 1, 2, 1]), array([2, 3, 1, 1], dtype=int64))
```

Сравнение времени работы реализаций на данных разного размера

1. Генерация входных данных

Будем генерировать входные данные как массив (array для реализации 1 и numpy.array для реализации 2 и 3) длины l, состоящий из целых чисел "0", "1", "2". Этого хватит для сравнения времени работы программы, так как достаточно, чтобы чередующиеся серии одинаковых чисел были различны (серии чисел "1", разделённые серией чисел "2", различаются), то есть для задачи хватило бы всего двух различных чисел, например "0" и "1".

In [7]:
```
import numpy as np


n = 10
k = 3
for l in range(1, n + 1):
    input = np.random.randint(0, 3, l)
    print(input)
```

```
[2]
[0 0]
[0 2 2]
[2 1 2 0]
[1 2 1 2 0]
[1 1 0 1 0 1]
[0 2 0 2 2 2 2]
[1 1 2 2 0 2 0 0]
[1 0 0 2 2 1 1 2 0]
[1 0 2 2 0 1 1 2 1 0]
```

2. Сбор времени выполнения

Ниже представлена программа, состоящая из 3-х рассматриваемых соответсвующих трём реализациям функций. И функции тестирования, куда подаются числа n и k. Она возвращает кортеж из массивов значений для построения графика.

n - максимальная длина входного массива.

k - количество запусков одной функции на массиве фиксированной длины l.

Будем изменять размер входных данных (размер l массива) от 0 до n каждые k запусков программы (итераций), во время которых будем измерять время работы программы с помощью модуля timeit. Массив заполняется случайными числами из промежутка [0, 2] на каждой длине l (l из промежутка [0, n]). И этот случайный массив k раз подаётся на вход всем трём функциям. Время выполнения с k запусков усредняется на каждой функции для данной длины массива l. Далее по этим данным будут построены графики зависимости времени исполнения кода от размера входных данных.

In [8]:

```
from timeit import default_timer as timer
import numpy as np


def encode_rle_1(x):
    l = len(x)
    keys = []
    quant = []
    for i in range(l):
        if (i == 0):
            cur = x[0]
            count = 1
        elif (i != 0 and x[i] == cur):
            count += 1
        else:
            keys.append(cur)
            quant.append(count)
            cur = x[i]
            count = 1
        if (i == l - 1):
            keys.append(cur)
            quant.append(count)
    return (keys, quant)


def encode_rle_2(x):
    mask = np.concatenate(([True], x[1:] != x[:-1]))
    (keys, quant) = (np.array([]), np.array([]))
    l = np.shape(x)[0]
    prev_i = 0
    for (i, j) in zip(range(l), mask):
        if (j):
            keys = np.append(keys, x[i])
            if i != 0:
                quant = np.append(quant, i - prev_i)
                prev_i = i
        if (i == l - 1):
            quant = np.append(quant, i - prev_i + 1)
    return (keys, quant)


def encode_rle_3(x):
    if np.shape(x)[0] == 0:
        return ([], [])
    mask = np.concatenate(([True], x[1:] != x[:-1], [True]))
    return (x[mask[:-1]], np.diff(np.where(mask)[0]))


def test_implementations(n, k):
    size_vals = np.arange(0, n + 1)
    time_vals = np.zeros((n + 1, 3))
    for l in range(0, n + 1):
        input = np.random.randint(0, 3, l)
        time_arr = np.zeros((3, k))
        for i in range(k):
            start = timer()
            encode_rle_1(input)
            end = timer()
            time_arr[0][i] = end - start
        input = np.array(input)
        for i in range(k):
            start = timer()
            encode_rle_2(input)
            end = timer()
            time_arr[1][i] = end - start
        for i in range(k):
            start = timer()
            encode_rle_3(input)
            end = timer()
            time_arr[2][i] = end - start
```

```
        time_vals[l] = np.mean(time_arr, axis=1)
        if l % 10 == 0:
            print(l)
    print("=====Done!=====")
    return (size_vals, time_vals)
```

Получим данные для таких заданных n и k:

1) n = 100, k = 10

2) n = 100, k = 100

3) n = 1000, k = 10

4) n = 1000, k = 100

Печать каждой десятой длины l входных данных производится для контроля хода вычислений.

In [9]:
```
(size_vals_1, time_vals_1) = test_implementations(100, 10)
```

```
0
10
20
30
40
50
60
70
80
90
100
=====Done!=====
```

In [10]:
```
(size_vals_2, time_vals_2) = test_implementations(100, 100)
```

```
0
10
20
30
40
50
60
70
80
90
100
=====Done!=====
```

In [11]:
```
(size_vals_3, time_vals_3) = test_implementations(1000, 10)
```

```
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
```

180
190
200
210
220
230
240
250
260
270
280
290
300
310
320
330
340
350
360
370
380
390
400
410
420
430
440
450
460
470
480
490
500
510
520
530
540
550
560
570
580
590
600
610
620
630
640
650
660
670
680
690
700
710
720
730
740
750
760
770
780
790
800
810
820
830
840
850
860
870
880
890
900
910

```
920
930
940
950
960
970
980
990
1000
=====Done!=====
```

In [12]:

```
(size_vals_4, time_vals_4) = test_implementations(1000, 100)
```

```
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
310
320
330
340
350
360
370
380
390
400
410
420
430
440
450
460
470
480
490
500
510
520
530
540
550
560
570
580
590
```

```
600
610
620
630
640
650
660
670
680
690
700
710
720
730
740
750
760
770
780
790
800
810
820
830
840
850
860
870
880
890
900
910
920
930
940
950
960
970
980
990
1000
=====Done!=====
```

Сохраняем вычисленные данные:

In [13]:

```
np.savetxt('100_10.txt', time_vals_1)
np.savetxt('100_100.txt', time_vals_2)
np.savetxt('1000_10.txt', time_vals_3)
np.savetxt('1000_100.txt', time_vals_4)
```

3. Построение графиков

Пояснение к легенде карты: 1st, 2nd, 3rd - это графики среднего времени работы в зависимости от длины входных данных первой, второй и третей реализаций соответственно. Для каждого из 4-х случаев приведено 2 графика: 1, 2, 3 реализации и 1, 3 реализации (для более детального рассмотрения).

In [14]:

```
import math
import matplotlib
import matplotlib.pyplot as plt

#отображение графиков в ноутбуке
%matplotlib inline

#для четкой прорисовки графиков
%config InlineBackend.figure_format = 'svg'
```

График 1 (1, 2, 3 реализации): (n = 100, k = 10):

In [15]:
```
plt.figure(figsize=(10,5))
plt.title('Implementation lead time (n = 100, k = 10)',fontsize=16)
plt.plot(size_vals_1, time_vals_1[:,0], linestyle='solid', color='r', label='1st')
plt.plot(size_vals_1, time_vals_1[:,1], linestyle='dotted', color='g', label='2nd')
plt.plot(size_vals_1, time_vals_1[:,2], linestyle='dashed', color='b', label='3rd')
plt.grid(True)
plt.ylabel('Time, s',  fontsize=14)
plt.xlabel('Size, count',  fontsize=14)
plt.legend(fontsize=12)
plt.savefig('123_100_10.pdf')
plt.show()
```

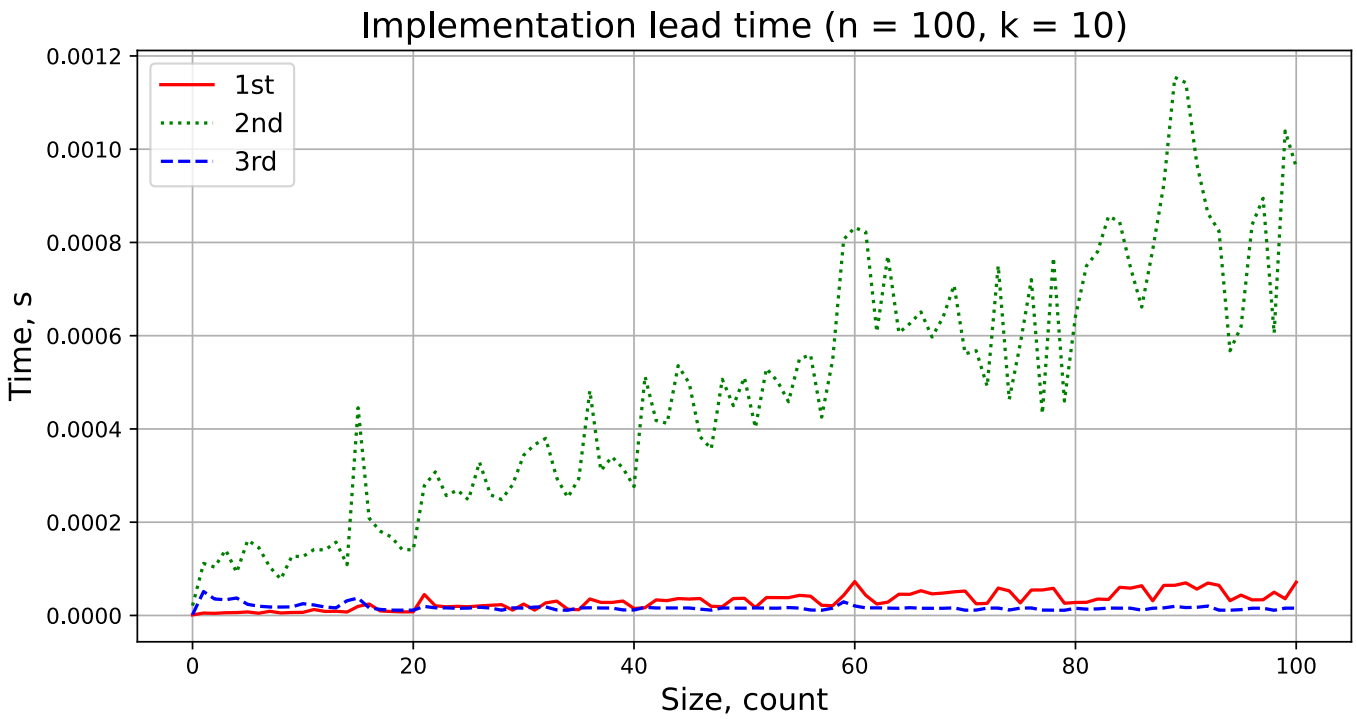


График 2 (1, 3 реализации): (n = 100, k = 10):

In [16]:
```
plt.figure(figsize=(10,5))
plt.title('Implementation lead time (n = 100, k = 10)',fontsize=16)
plt.plot(size_vals_1, time_vals_1[:,0], linestyle='solid', color='r', label='1st')
plt.plot(size_vals_1, time_vals_1[:,2], linestyle='dashed', color='b', label='3rd')
plt.grid(True)
plt.ylabel('Time, s',  fontsize=14)
plt.xlabel('Size, count',  fontsize=14)
plt.legend(fontsize=12)
plt.savefig('13_100_10.pdf')
plt.show()
```

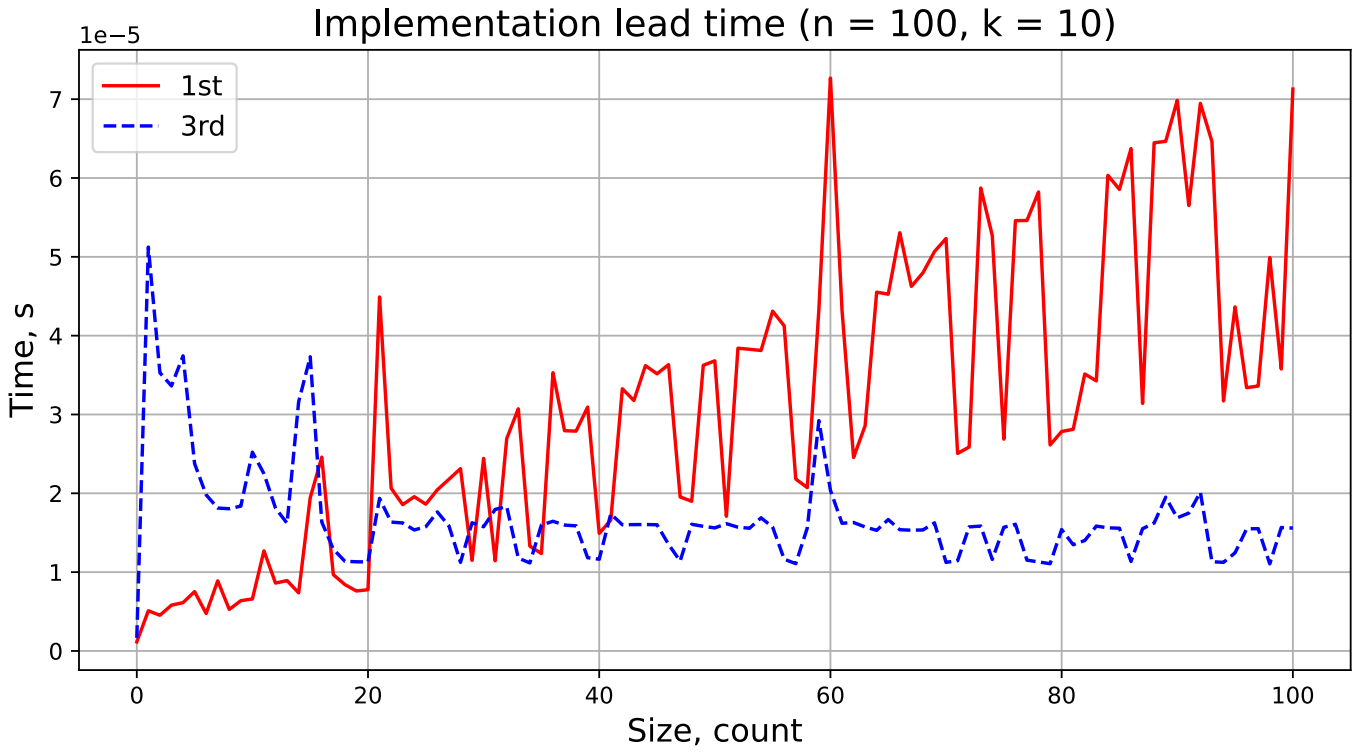


График 3 (1, 2, 3 реализации): (n = 100, k = 100):

In [17]:

```
plt.figure(figsize=(10,5))
plt.title('Implementation lead time (n = 100, k = 100)',fontsize=16)
plt.plot(size_vals_2, time_vals_2[:,0], linestyle='solid', color='r', label='1st')
plt.plot(size_vals_2, time_vals_2[:,1], linestyle='dotted', color='g', label='2nd')
plt.plot(size_vals_2, time_vals_2[:,2], linestyle='dashed', color='b', label='3rd')
plt.grid(True)
plt.ylabel('Time, s',  fontsize=14)
plt.xlabel('Size, count',  fontsize=14)
plt.legend(fontsize=12)
plt.savefig('123_100_100.pdf')
plt.show()
```

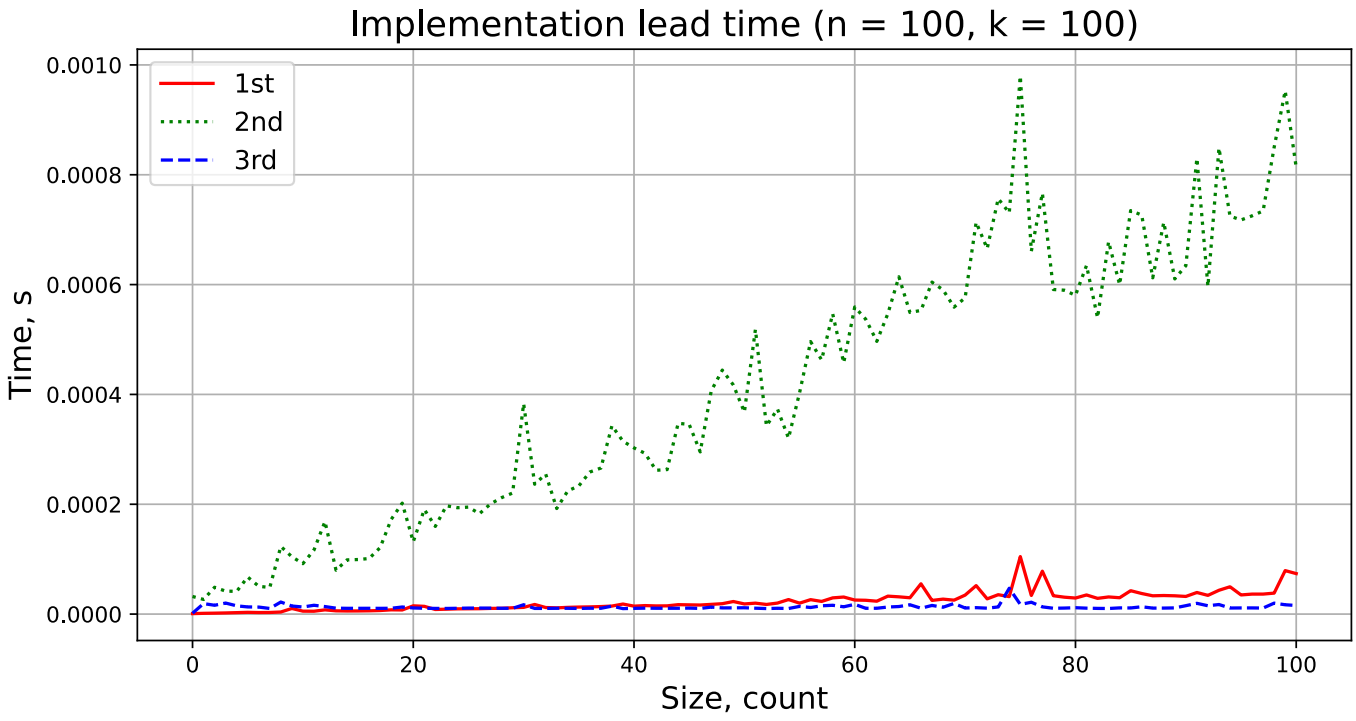


График 4 (1, 3 реализации): (n = 100, k = 100):

In [18]:

```
plt.figure(figsize=(10,5))
plt.title('Implementation lead time (n = 100, k = 100)',fontsize=16)
plt.plot(size_vals_2, time_vals_2[:,0], linestyle='solid', color='r', label='1st')
```

```
plt.plot(size_vals_2, time_vals_2[:,2], linestyle='dashed', color='b', label='3rd')
plt.grid(True)
plt.ylabel('Time, s',  fontsize=14)
plt.xlabel('Size, count',  fontsize=14)
plt.legend(fontsize=12)
plt.savefig('13_100_100.pdf')
plt.show()
```

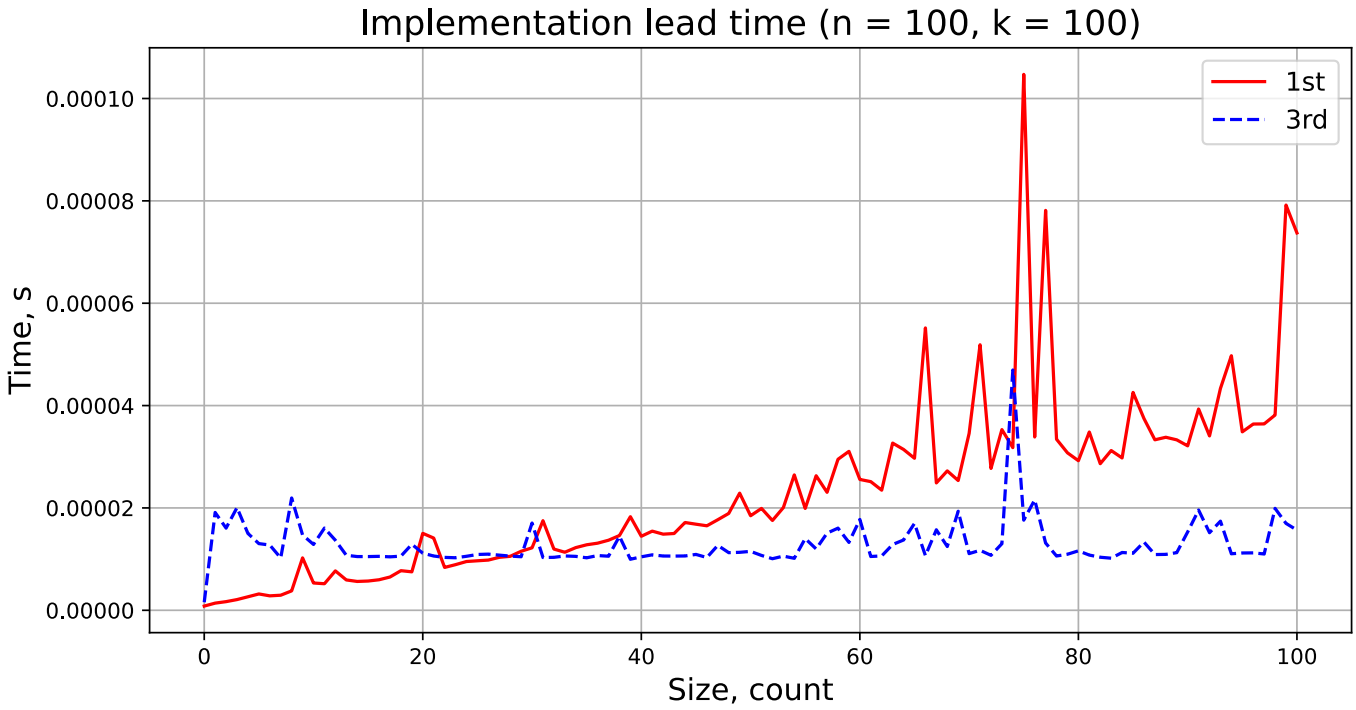


График 5 (1, 2, 3 реализации): (n = 1000, k = 10):

In [19]:

```
plt.figure(figsize=(10,5))
plt.title('Implementation lead time (n = 1000, k = 10)',fontsize=16)
plt.plot(size_vals_3, time_vals_3[:,0], linestyle='solid', color='r', label='1st')
plt.plot(size_vals_3, time_vals_3[:,1], linestyle='dotted', color='g', label='2nd')
plt.plot(size_vals_3, time_vals_3[:,2], linestyle='dashed', color='b', label='3rd')
plt.grid(True)
plt.ylabel('Time, s',  fontsize=14)
plt.xlabel('Size, count',  fontsize=14)
plt.legend(fontsize=12)
plt.savefig('123_1000_10.pdf')
plt.show()
```

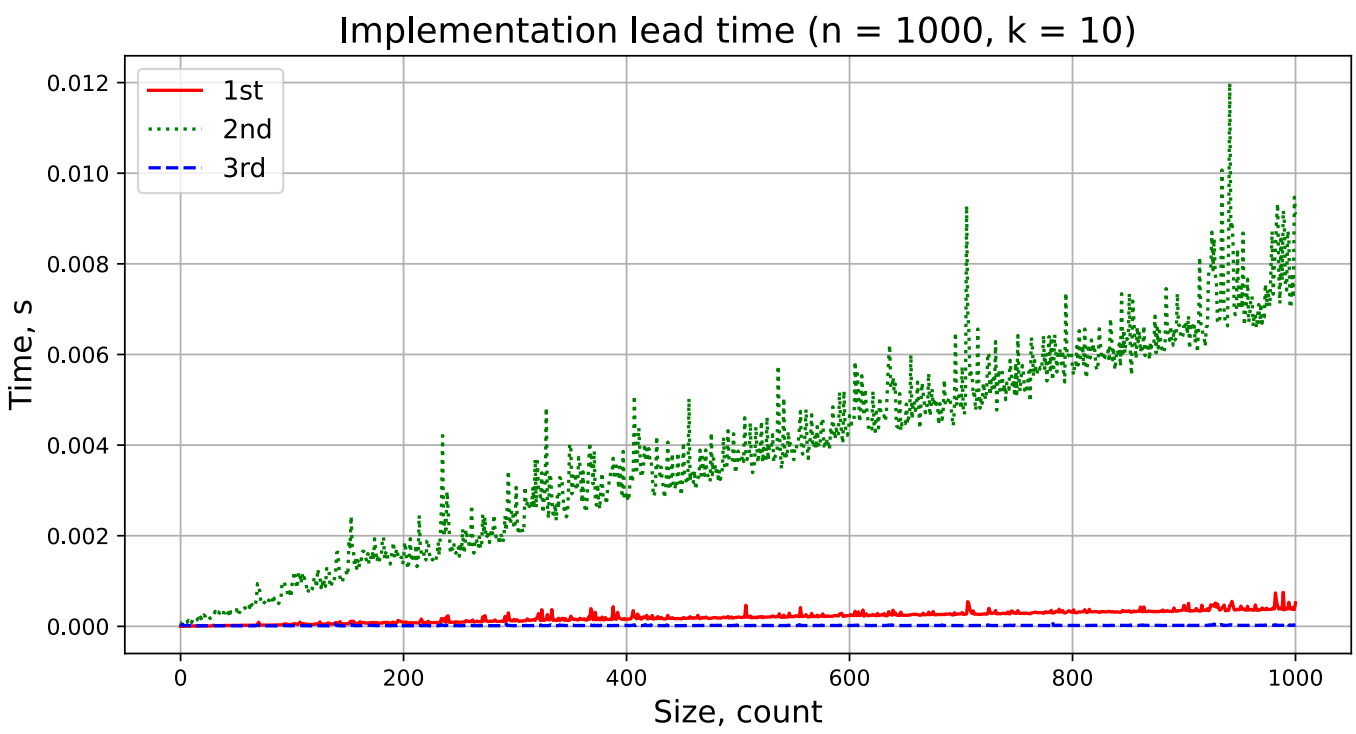


График 6 (1, 3 реализации): (n = 1000, k = 10):

In [20]:

```
plt.figure(figsize=(10,5))
plt.title('Implementation lead time (n = 1000, k = 10)',fontsize=16)
plt.plot(size_vals_3, time_vals_3[:,0], linestyle='solid', color='r', label='1st')
plt.plot(size_vals_3, time_vals_3[:,2], linestyle='dashed', color='b', label='3rd')
plt.grid(True)
plt.ylabel('Time, s',  fontsize=14)
plt.xlabel('Size, count',  fontsize=14)
plt.legend(fontsize=12)
plt.savefig('13_1000_10.pdf')
plt.show()
```

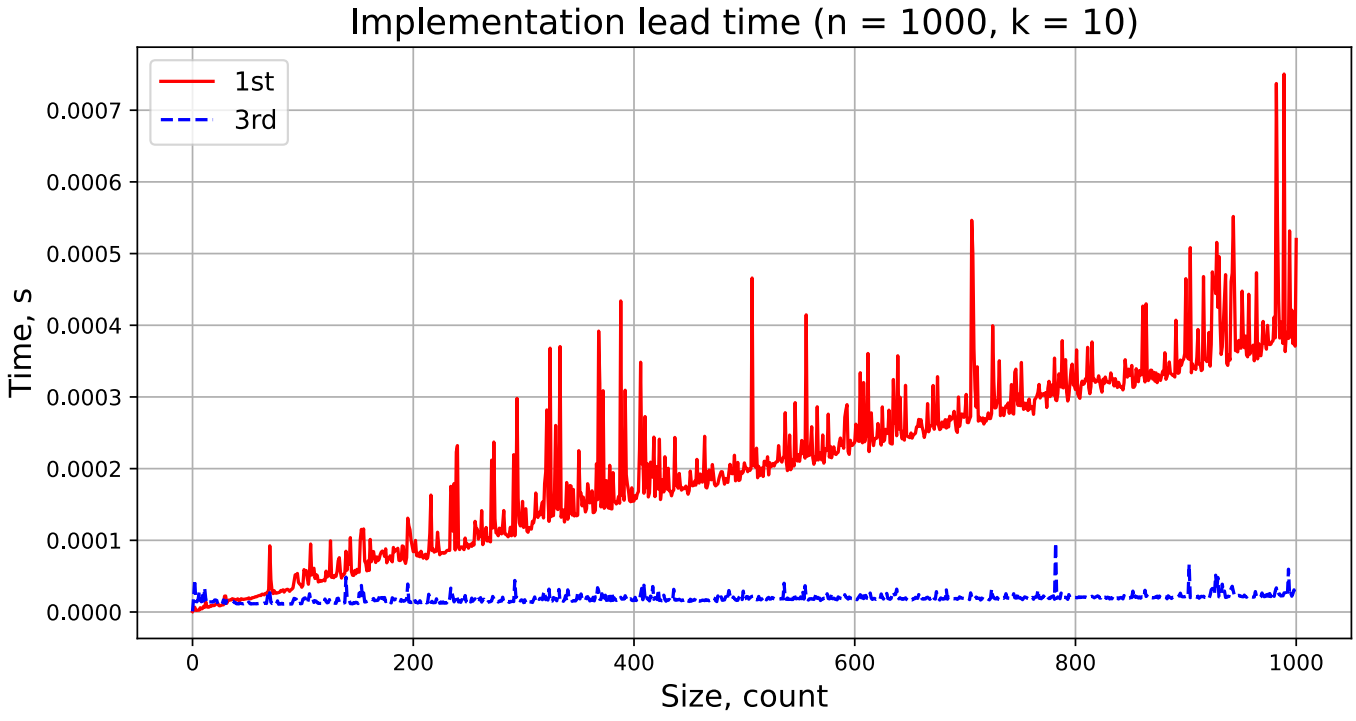


График 7 (1, 2, 3 реализации): (n = 1000, k = 100):

In [21]:

```
plt.figure(figsize=(10,5))
plt.title('Implementation lead time (n = 1000, k = 100)',fontsize=16)
plt.plot(size_vals_4, time_vals_4[:,0], linestyle='solid', color='r', label='1st')
plt.plot(size_vals_4, time_vals_4[:,1], linestyle='dotted', color='g', label='2nd')
plt.plot(size_vals_4, time_vals_4[:,2], linestyle='dashed', color='b', label='3rd')
```

```
plt.grid(True)
plt.ylabel('Time, s',  fontsize=14)
plt.xlabel('Size, count',  fontsize=14)
plt.legend(fontsize=12)
plt.savefig('123_100_100.pdf')
plt.show()
```

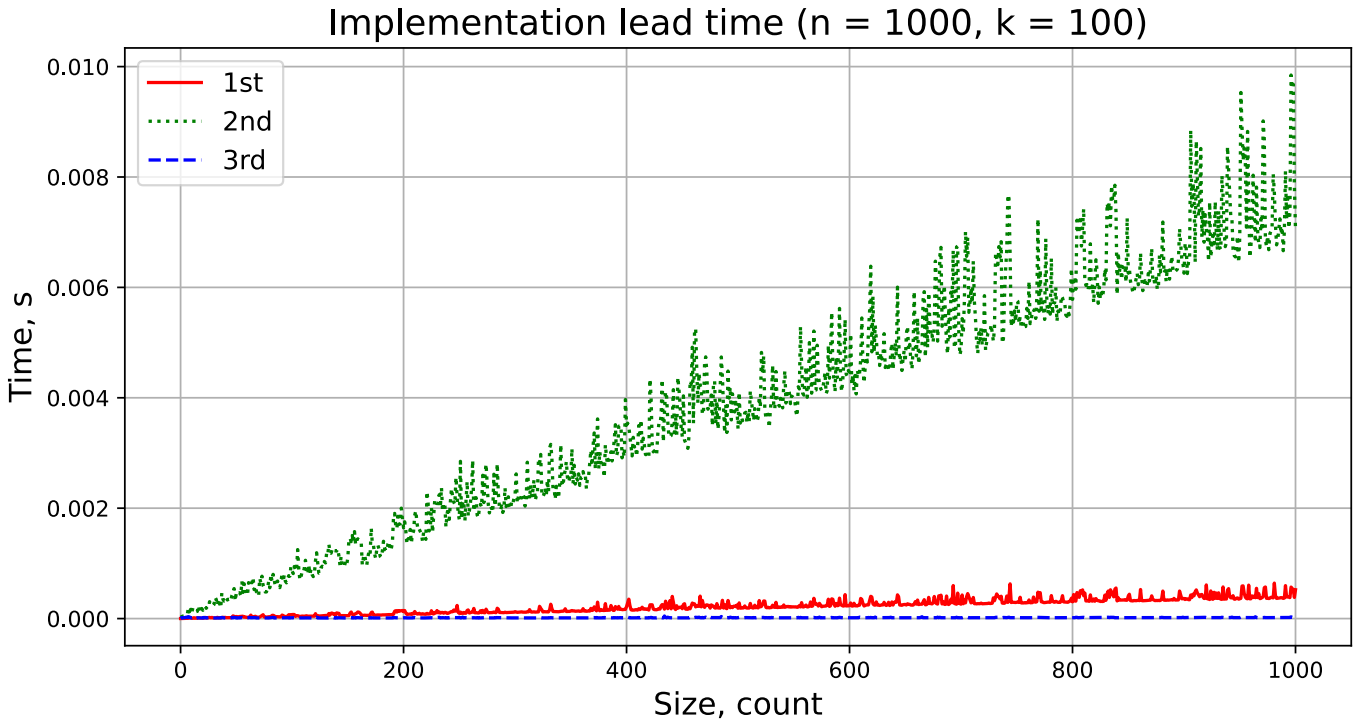
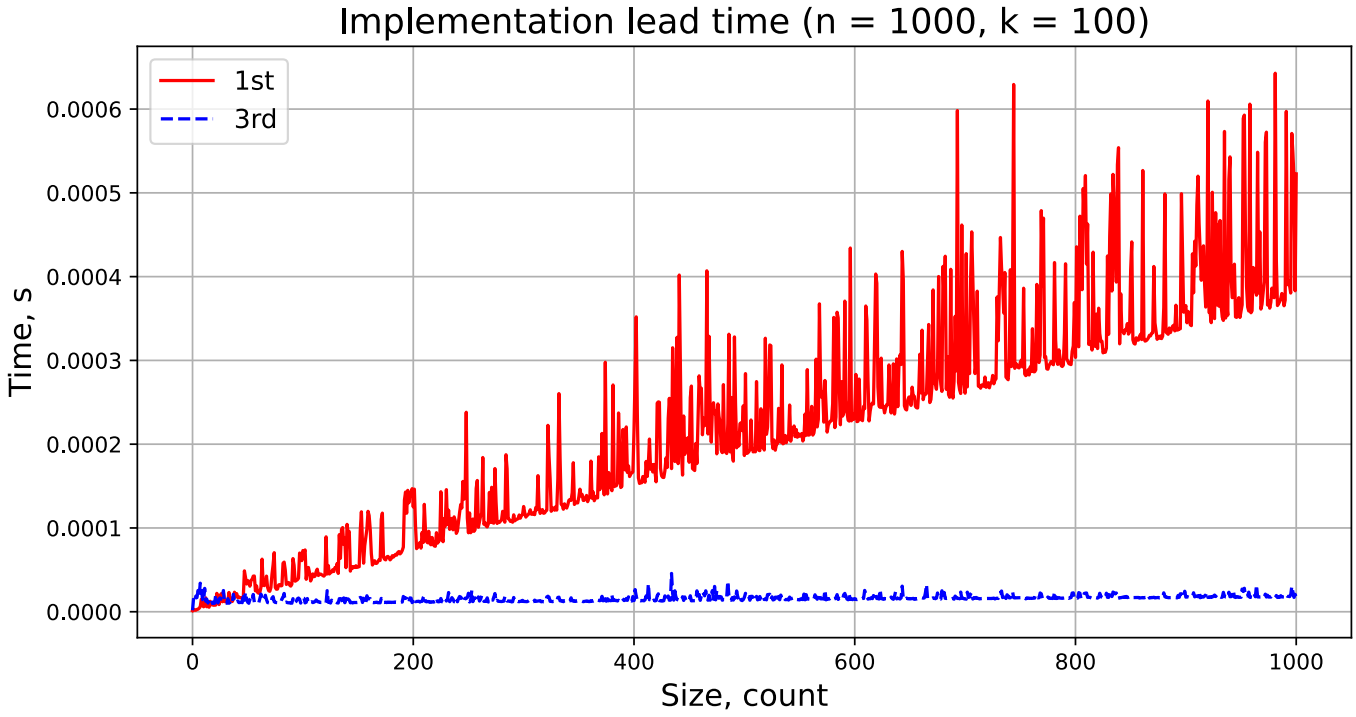


График 8 (1, 3 реализации): (n = 1000, k = 100):

In [22]:

```
plt.figure(figsize=(10,5))
plt.title('Implementation lead time (n = 1000, k = 100)',fontsize=16)
plt.plot(size_vals_4, time_vals_4[:,0], linestyle='solid', color='r', label='1st')
plt.plot(size_vals_4, time_vals_4[:,2], linestyle='dashed', color='b', label='3rd')
plt.grid(True)
plt.ylabel('Time, s',  fontsize=14)
plt.xlabel('Size, count',  fontsize=14)
plt.legend(fontsize=12)
plt.savefig('13_1000_100.pdf')
plt.show()
```



4. Вывод

Эмпирически на примере данной задачи было показано, что полностью векторизованная реализация с использованием NumPy (3rd) работает быстрее полностью невекторизованной реализации на чистом Python (1st), которая, в свою очередь, работает быстрее реализации с частичной векторизацией (2nd).