**Министерство науки и высшего образования РФ**

**ФГБОУ ВО «РХТУ им. Д. И. Менделеева»**

**ОТЧЕТ**

ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

**ВЫПОЛНИЛ:** студент группы МК-20 Краснов М.В.

**ПРОВЕРИЛ:** к.т.н. Красильников И.В.

**Москва**

**2021**

Постановка задачи

Для изучения было представлено два алгоритма возведения числа в степень:

1)

*2)*

Для реализации каждого алгоритма было предложено 2 варианта реализации:

1) Рекурсивный алгоритм

2) Итерационный алгоритм

Где,

expt\_bi\_for(b,n) – итерационная функция . Сложность O(log n)  
expt\_bi\_rec(b,n) - рекурсионная функция . Сложность O(log n)  
expt\_line\_rec(b,n) – рекурсионная функция . Сложность О(n)  
expt\_line\_for(b,n) - итерационная функция . Сложность О(n)

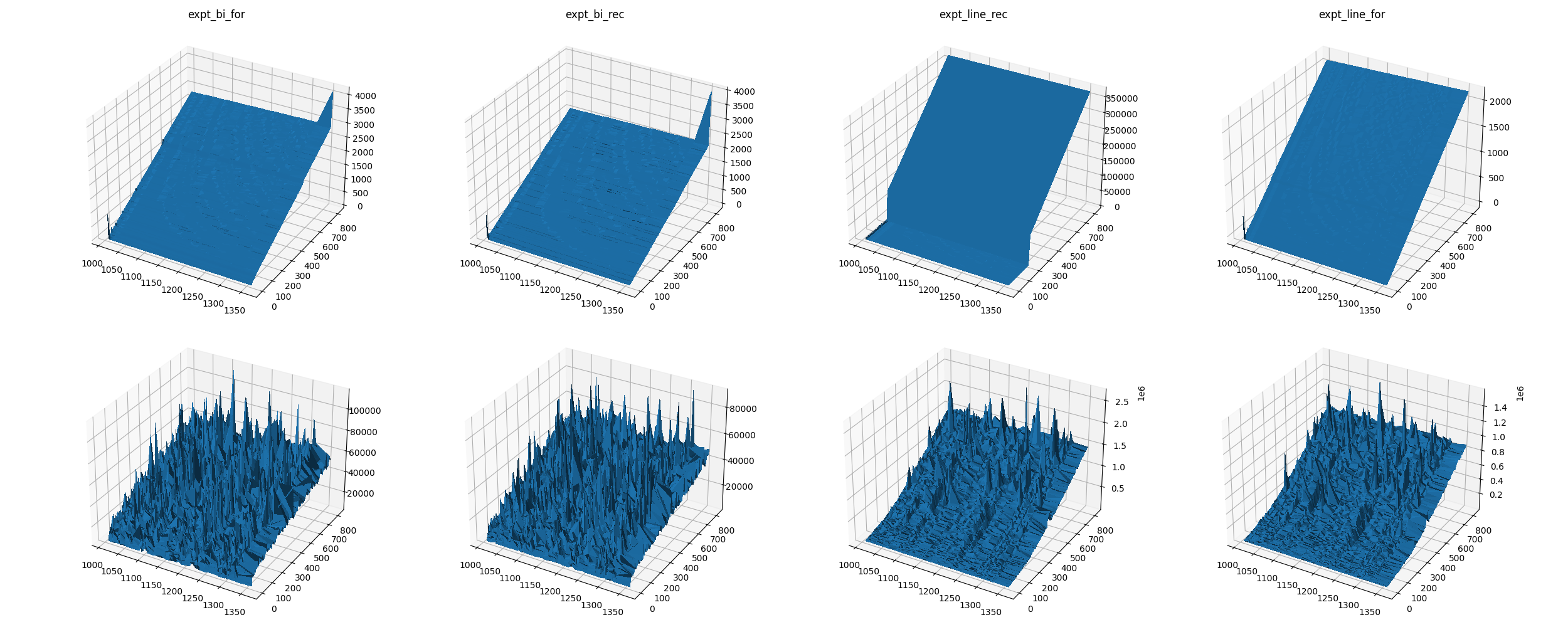
Методика исследования.

Исследование проводилась на диапазоне b простых чисел, найденных в диапазоне от 100000, в количестве 50; и диапазоне простых чисел n от 100 до 800. Выбор простых чисел для связан с попытками нивелировать, по возможности, внутреннюю оптимизацию и кэширование математических вычислений. Выбор верхнего края для диапазона n=800 связан с ограничением количества рекурсий для данного языка программирования.

Изучались трассировки времени в наносекундах (time) для выполнения функции и пиковые значения затрат памяти (peak). При этом, из каждого значения peak на каждой итерации вычитались затраты памяти на хранение списка с результатами (results), который, естественно, рос с каждой итерацией. Каждый алгоритм запускался не менее 100 раз с последующим усреднением результатов измерений.

Результаты исследования.

Ниже представлены графики полученных результатов измерения,



Где верхний график – зависимость затрат памяти от b и n, а нижний – временные затраты в наносекундах.

1)Как видно из графиков, для всех реализаций существует прямая зависимость затрат памяти от увеличения степени числа n, при этом самый «выгодный» по памяти алгоритм – линейный итерационный expt\_line\_for(b,n) , с пиковыми затратами памяти 2000 при максимальных n.

2) Самым же жадным по памяти (а также – по времени) оказался рекурсионный линейный алгоритм expt\_line\_rec(b,n) , с затратами более при максимальных n.

3) При этом, наблюдается независимость затрат памяти и времени от b, что на графике выглядит как поверхность, параллельная оси b.

4)Группа алгоритмов expt\_bi показала наименьшие затраты по памяти (4000)и по времени (не менее 40000, но не более 100000 в пике) на максимальных n. При этом графики обоих реализаций бинарного алгоритма практически идентичны, хотя при малых b, график expt\_bi\_rec(b,n) более пологий в проекции оси z.

5)Затраты памяти для всех реализаций имеют линейную зависимость от n, и практически независимы от b.

Такие же зависимости наблюдаются и по затратам времени, однако здесь поверхность графика имеет более сложную структуру. Так, мы можем говорить лишь о том, что при неких показателях b или n, линия среза поверхности z(b,n)по соответствующей оси для алгоритма не меньше некого числа, так как такой срез представляет собой ломаную линию. На графике это заметно как наличие пиков на поверхности графика. При этом, визуально заметна некоторая упорядоченность этих пиков, образующих, в свою очередь, гребни и конгломераты.

Выводы:

1) Бинарный алгоритм показал себя как более экономный по временным затратам. При этом его рекурсионный вариант expt\_bi\_rec(b,n) практически на всех значениях b, кроме замыкающих диапазон, показал затраты памяти, идентичные линейной итерационной вариации expt\_line\_for(b,n) . Таким образом, данную реализацию алгоритма на приведенных диапазонах значений b и n можно признать самым эффективным.

2) Рекурсионный линейный алгоритм expt\_line\_rec(b,n)показал себя как самый «жадный» по памяти и временным затратам.

3) Итерационный линейный алгоритм expt\_line\_for(b,n)показал наиболее скромные запросы по памяти, со стабильной линейной зависимостью.

4)Показатели затрат ресурсов при исполнении алгоритмов практически не зависели от величины b.

5) Затраты ресурсов при исполнении алгоритмов вполне коррелировали с их сложностью O(F).

6)Наличие сложной поверхности графиков зависимости временных затрат от b и n, можно связать со множеством факторов:

А) наличие внутренней оптимизации вычислений компилятора;

Б) оптимизация и кэширование вычислений процессором

В) Регулировка тактовой частоты процессора в режиме многозадачности со стороны операционной системы

Код программы

*import* tracemalloc

*import* time

*import* pandas *as* pd

*import* matplotlib.pyplot *as* plt

*import* sys

*import* sympy

b\_primes = []

n\_primes = []

tracemalloc.start()

n = 1000

*for* i *in* range(0, 50):

    b = sympy.nextprime(n)

    b\_primes.append(b)

    n = b

m = 0

*while* m<800:

    n = sympy.nextprime(m)

    n\_primes.append(n)

    m = n

def devide(*x*, *devider*):

    o = int(x % devider)

*return* int((x - o) / devider), o

def expt\_bi\_for(*b*, *n*):

*if* b == 0: *return* 0

*if* b == 1: *return* 1

*if* n == 0: *return* 1

*if* n == 1: *return* b

    result = b

    nns = []

*while* True:

        n, o = devide(n, 2)

*if* n == 0: *break*

        nns.append((n, o))

*for* np *in* range(len(nns) - 1, -1, -1):

        result = ((result) \*\* 2) \* (b) \*\* nns[np][1]

*return* result

def expt\_bi\_rec(*b*, *n*):

*if* b == 0: *return* 0

*if* b == 1: *return* 1

*if* n == 0: *return* 1

*if* n == 1: *return* b

    n, o = devide(n, 2)

*if* o == 0: *return* expt\_bi\_rec(b, n) \*\* 2

*if* o == 1: *return* (expt\_bi\_rec(b, n) \*\* 2) \* b

def expt\_line\_rec(*b*, *n*):

*if* b == 0: *return* 0

*if* b == 1: *return* 1

*if* n == 0: *return* 1

*if* n == 1: *return* b

*return* b \* expt\_line\_rec(b, n - 1)

def expt\_line\_for(*b*, *n*):

*if* b == 0: *return* 0

*if* b == 1: *return* 1

*if* n == 0: *return* 1

*if* n == 1: *return* b

    result = b

*for* i *in* range(0, n - 1):

        result = result \* b

*return* result

def runAlg(*alg*, *tracer*, *b*, *n*, *repeats*):

    tracer.clear\_traces()

    t1 = time.perf\_counter\_ns()

*for* i *in* range(0, repeats):

        alg(b, n)

    t2 = time.perf\_counter\_ns()

    size, peak = tracemalloc.get\_traced\_memory()

*return* {"time": (t2 - t1) / repeats, "size": size, "peak": peak, }

def main():

    results = {'expt\_bi\_for': {'n': [], 'b': [], 'time': [], 'peak': []},

               'expt\_bi\_rec': {'n': [], 'b': [], 'time': [], 'peak': []},

               'expt\_line\_rec': {'n': [], 'b': [], 'time': [], 'peak': []},

               'expt\_line\_for': {'n': [], 'b': [], 'time': [], 'peak': []}}

    repeats = 100

    nlimit = 800;

    nstep = int(round(nlimit / 50));

*for* b *in* list(b\_primes):

*for* n *in* n\_primes:

            listSize = sys.getsizeof(results)

            a = runAlg(expt\_bi\_for, tracemalloc, b, n, repeats)

            results['expt\_bi\_for']['n'].append(n)

            results['expt\_bi\_for']['b'].append(b)

            results['expt\_bi\_for']['time'].append(a['time'])

            results['expt\_bi\_for']['peak'].append(a['peak']-listSize)

            a = runAlg(expt\_bi\_rec, tracemalloc, b, n, repeats)

            results['expt\_bi\_rec']['n'].append(n)

            results['expt\_bi\_rec']['b'].append(b)

            results['expt\_bi\_rec']['time'].append(a['time'])

            results['expt\_bi\_rec']['peak'].append(a['peak']-listSize)

            a = runAlg(expt\_line\_rec, tracemalloc, b, n, repeats)

            results['expt\_line\_rec']['n'].append(n)

            results['expt\_line\_rec']['b'].append(b)

            results['expt\_line\_rec']['time'].append(a['time'])

            results['expt\_line\_rec']['peak'].append(a['peak']-listSize)

            a = runAlg(expt\_line\_for, tracemalloc, b, n, repeats)

            results['expt\_line\_for']['n'].append(n)

            results['expt\_line\_for']['b'].append(b)

            results['expt\_line\_for']['time'].append(a['time'])

            results['expt\_line\_for']['peak'].append(a['peak']-listSize)

    expt\_bi\_for\_g = pd.DataFrame(results['expt\_bi\_for'])

    expt\_bi\_rec\_g = pd.DataFrame(results['expt\_bi\_rec'])

    expt\_line\_rec\_g = pd.DataFrame(results['expt\_line\_rec'])

    expt\_line\_for\_g = pd.DataFrame(results['expt\_line\_for'])

    fig = plt.figure(*figsize*=(25, 10))

    ax1 = fig.add\_subplot(241, *projection*='3d')

    ax1.plot\_trisurf(expt\_bi\_for\_g['b'], expt\_bi\_for\_g['n'], expt\_bi\_for\_g['peak'], *linewidth*=0, *antialiased*=False)

    ax2 = fig.add\_subplot(242, *projection*='3d')

    ax2.plot\_trisurf(expt\_bi\_rec\_g['b'], expt\_bi\_rec\_g['n'], expt\_bi\_rec\_g['peak'], *linewidth*=0, *antialiased*=False)

    ax3 = fig.add\_subplot(243, *projection*='3d')

    ax3.plot\_trisurf(expt\_line\_rec\_g['b'], expt\_line\_rec\_g['n'], expt\_line\_rec\_g['peak'], *linewidth*=0,

*antialiased*=False)

    ax4 = fig.add\_subplot(244, *projection*='3d')

    ax4.plot\_trisurf(expt\_line\_for\_g['b'], expt\_line\_for\_g['n'], expt\_line\_for\_g['peak'], *linewidth*=0,

*antialiased*=False)

    ax1.set\_title('expt\_bi\_for')

    ax2.set\_title('expt\_bi\_rec')

    ax3.set\_title('expt\_line\_rec')

    ax4.set\_title('expt\_line\_for')

    ax5 = fig.add\_subplot(245, *projection*='3d')

    ax5.plot\_trisurf(expt\_bi\_for\_g['b'], expt\_bi\_for\_g['n'], expt\_bi\_for\_g['time'], *linewidth*=0, *antialiased*=False)

    ax6 = fig.add\_subplot(246, *projection*='3d')

    ax6.plot\_trisurf(expt\_bi\_rec\_g['b'], expt\_bi\_rec\_g['n'], expt\_bi\_rec\_g['time'], *linewidth*=0, *antialiased*=False)

    ax7 = fig.add\_subplot(247, *projection*='3d')

    ax7.plot\_trisurf(expt\_line\_rec\_g['b'], expt\_line\_rec\_g['n'], expt\_line\_rec\_g['time'], *linewidth*=0,

*antialiased*=False)

    ax8 = fig.add\_subplot(248, *projection*='3d')

    ax8.plot\_trisurf(expt\_line\_for\_g['b'], expt\_line\_for\_g['n'], expt\_line\_for\_g['time'], *linewidth*=0,

*antialiased*=False)

    fig.show()

    df=pd.DataFrame(results)

    print(df)

    df.to\_html('out.html')

    df.to\_excel('out.xlsx')

    expt\_bi\_for\_g.to\_excel('expt\_bi\_for\_g.xlsx')

    expt\_bi\_rec\_g.to\_excel('expt\_bi\_rec\_g.xlsx')

    expt\_line\_rec\_g.to\_excel('expt\_line\_rec\_g.xlsx')

    expt\_line\_for\_g.to\_excel('expt\_line\_for\_g.xlsx')

    print(expt\_line\_for\_g)

*if* \_\_name\_\_ == '\_\_main\_\_':

    main()