

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

**ОТЧЕТ**  
**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №9**  
**дисциплины «Алгоритмизация»**  
**Вариант 14**

Выполнил:  
Касимов Асхаб Арсенович  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р. А., канд. технических  
наук, доцент кафедры  
инфокоммуникаций

---

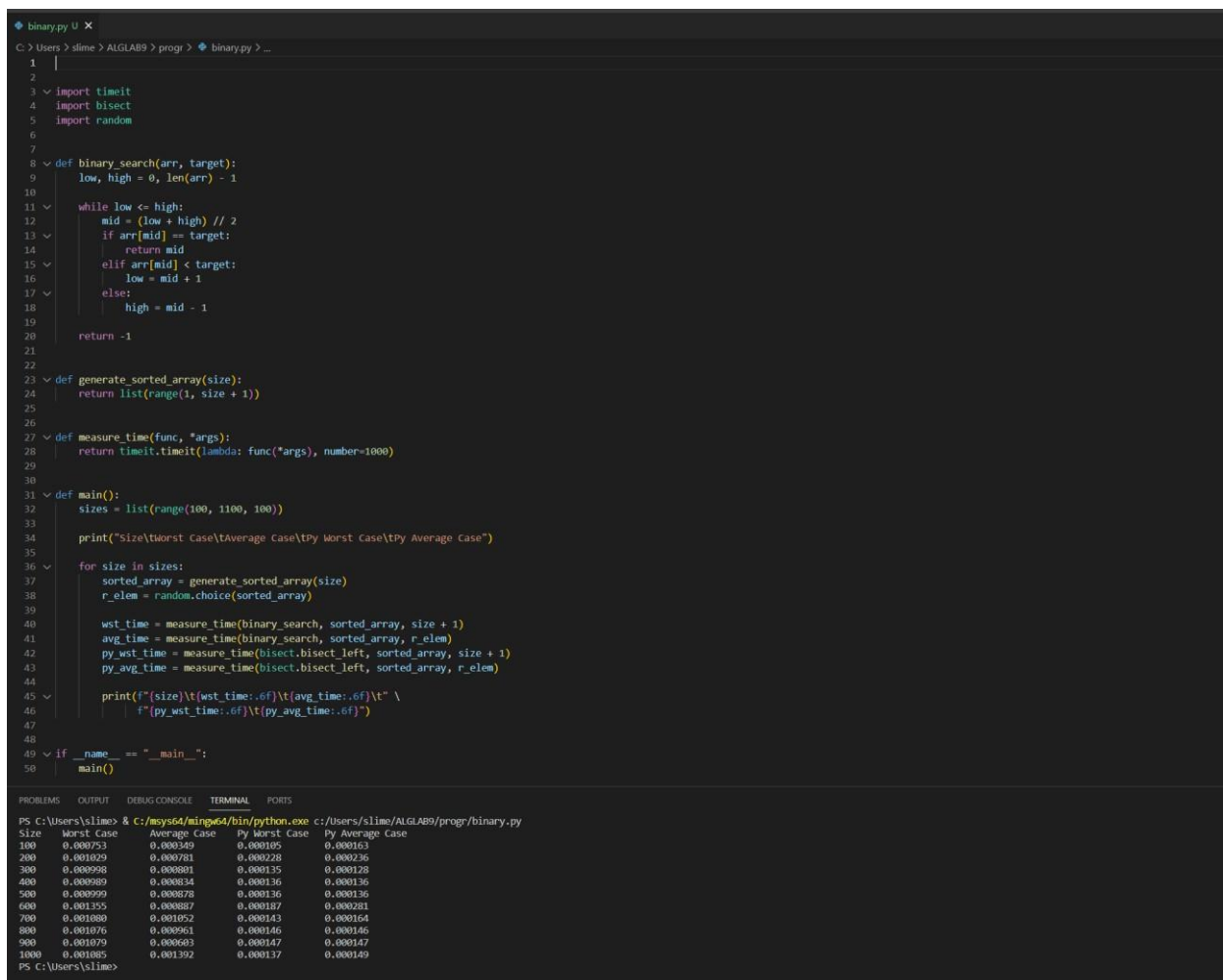
(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

## Ход работы

1. Написал программу, которая подсчитывает время, затрачиваемое на выполнение алгоритма бинарного поиска, а также бинарного поиска предоставляемого языком программирования Python, предусмотрел варианты среднего и худшего случая.



```
1 |
2 |
3 | import timeit
4 | import bisect
5 | import random
6 |
7 |
8 | def binary_search(arr, target):
9 |     low, high = 0, len(arr) - 1
10 |
11 |     while low <= high:
12 |         mid = (low + high) // 2
13 |         if arr[mid] == target:
14 |             return mid
15 |         elif arr[mid] < target:
16 |             low = mid + 1
17 |         else:
18 |             high = mid - 1
19 |
20 |     return -1
21 |
22 |
23 | def generate_sorted_array(size):
24 |     return list(range(1, size + 1))
25 |
26 |
27 | def measure_time(func, *args):
28 |     return timeit.timeit(lambda: func(*args), number=1000)
29 |
30 |
31 | def main():
32 |     sizes = list(range(100, 1100, 100))
33 |
34 |     print("Size\tWorst Case\tAverage Case\tPy Worst Case\tPy Average Case")
35 |
36 |     for size in sizes:
37 |         sorted_array = generate_sorted_array(size)
38 |         r_elem = random.choice(sorted_array)
39 |
40 |         wst_time = measure_time(binary_search, sorted_array, size + 1)
41 |         avg_time = measure_time(binary_search, sorted_array, r_elem)
42 |         py_wst_time = measure_time(bisect.bisect_left, sorted_array, size + 1)
43 |         py_avg_time = measure_time(bisect.bisect_left, sorted_array, r_elem)
44 |
45 |         print(f"{size}\t{wst_time:.6f}\t{avg_time:.6f}\t" \
46 |               f"{py_wst_time:.6f}\t{py_avg_time:.6f}")
47 |
48 |
49 | if __name__ == "__main__":
50 |     main()
```

| Size | Worst Case | Average Case | Py Worst Case | Py Average Case |
|------|------------|--------------|---------------|-----------------|
| 100  | 0.000753   | 0.000349     | 0.000105      | 0.000163        |
| 200  | 0.001029   | 0.000781     | 0.000228      | 0.000236        |
| 300  | 0.000998   | 0.000801     | 0.000135      | 0.000128        |
| 400  | 0.000980   | 0.000834     | 0.000136      | 0.000136        |
| 500  | 0.000999   | 0.000878     | 0.000136      | 0.000136        |
| 600  | 0.001355   | 0.000887     | 0.000187      | 0.000281        |
| 700  | 0.001080   | 0.001052     | 0.000143      | 0.000164        |
| 800  | 0.001076   | 0.000961     | 0.000146      | 0.000146        |
| 900  | 0.001079   | 0.000603     | 0.000147      | 0.000147        |
| 1000 | 0.001085   | 0.001392     | 0.000137      | 0.000149        |

Рисунок 1. Работа программы

Таблица 1. Время работы алгоритмов бинарного поиска

| Размер массива (n) | Время (худший случай) (сек) | Время (средний случай) (сек) | Время (худший случай, язык) (сек) | Время (средний случай, язык) |
|--------------------|-----------------------------|------------------------------|-----------------------------------|------------------------------|
| 100                | 0,000677                    | 0,000467                     | 0,000115                          | 0,000105                     |
| 200                | 0,000781                    | 0,000721                     | 0,000124                          | 0,000118                     |
| 300                | 0,001019                    | 0,000753                     | 0,000131                          | 0,000129                     |
| 400                | 0,000969                    | 0,000851                     | 0,000125                          | 0,000127                     |
| 500                | 0,001061                    | 0,00093                      | 0,000136                          | 0,000132                     |
| 600                | 0,001153                    | 0,000806                     | 0,000144                          | 0,000143                     |
| 700                | 0,00107                     | 0,001                        | 0,000144                          | 0,000149                     |
| 800                | 0,00111                     | 0,001054                     | 0,000141                          | 0,000138                     |
| 900                | 0,00105                     | 0,001032                     | 0,000147                          | 0,00014                      |
| 1000               | 0,001056                    | 0,001044                     | 0,000153                          | 0,000143                     |

2. Перенес данные по алгоритму бинарного поиска в таблицу Excel и произвел необходимые расчеты. Получил коэффициенты логарифмической зависимости из уравнений  $385000 \cdot a + 5500b = 5,2218$  и  $5500a + 10b = 0,008658$ . Получилось, что  $a = 0,0005592$ ,  $b = 0,00000557$ .

3. Построил график на основе полученных данных для среднего случая.

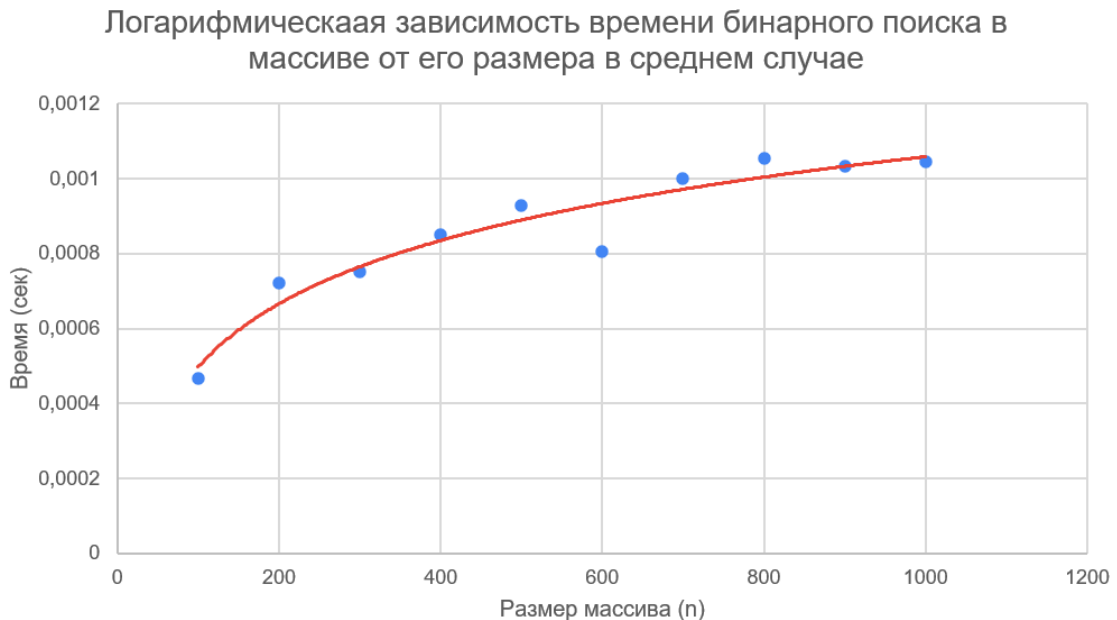


Рисунок 2. Средний случай

4. Произвел аналогичные действия для худшего случая, а также для среднего и худшего случая бинарного поиска предоставляемого Python.

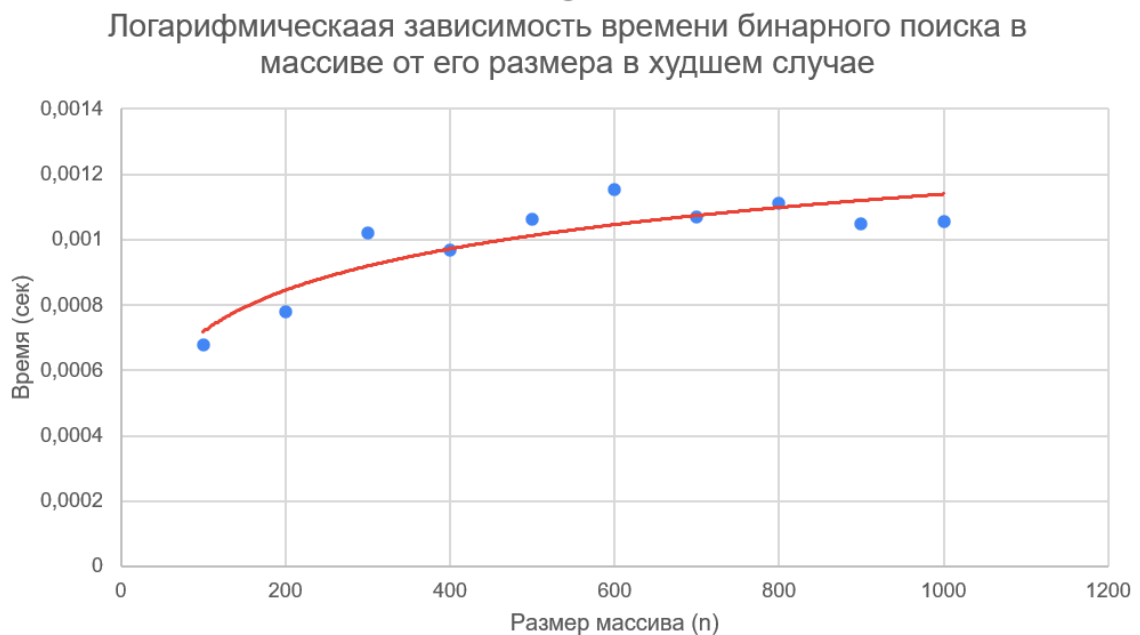


Рисунок 3. Законченный график для худшего случая

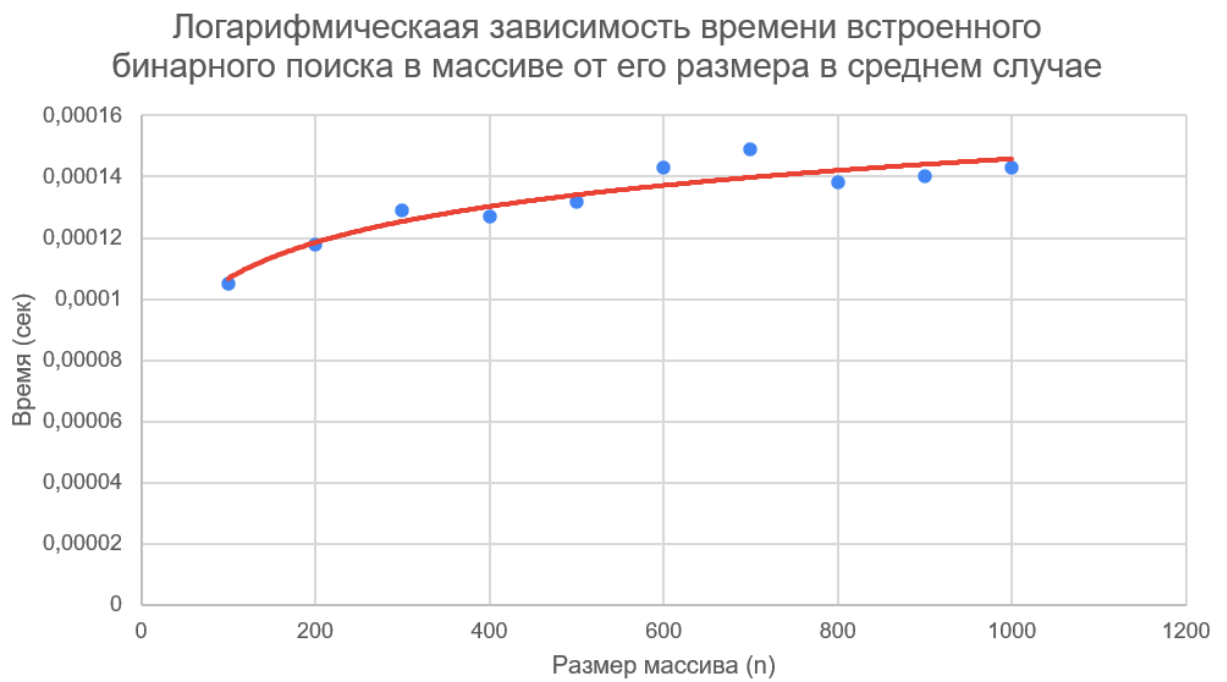


Рисунок 4. Средний случай для встроенного алгоритма

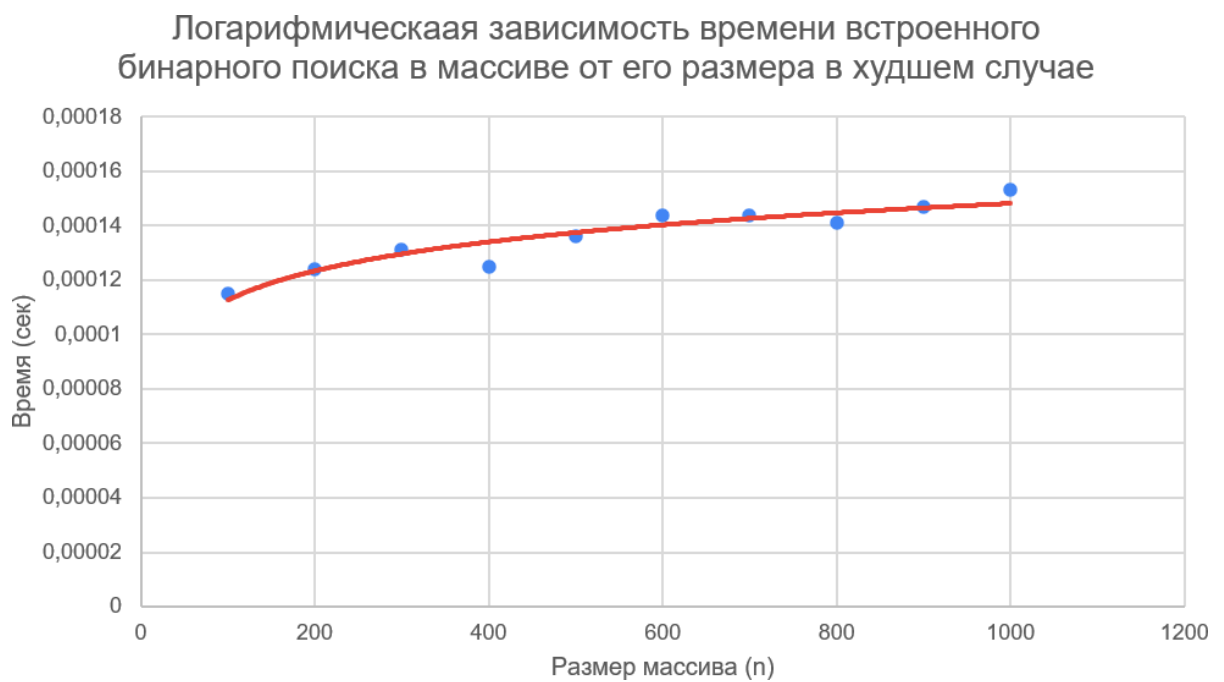


Рисунок 5. Худший случай для встроенного алгоритма

5. Привел для анализа графики зависимости времени работы алгоритма линейного поиска от размера массива.

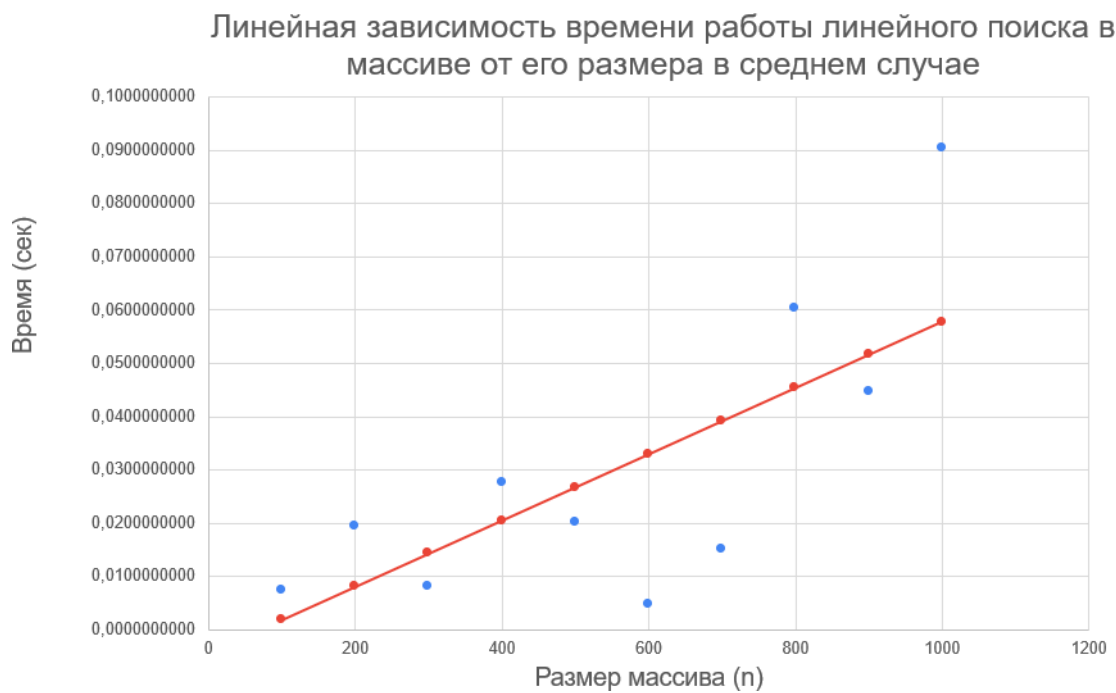


Рисунок 6. Средний случай

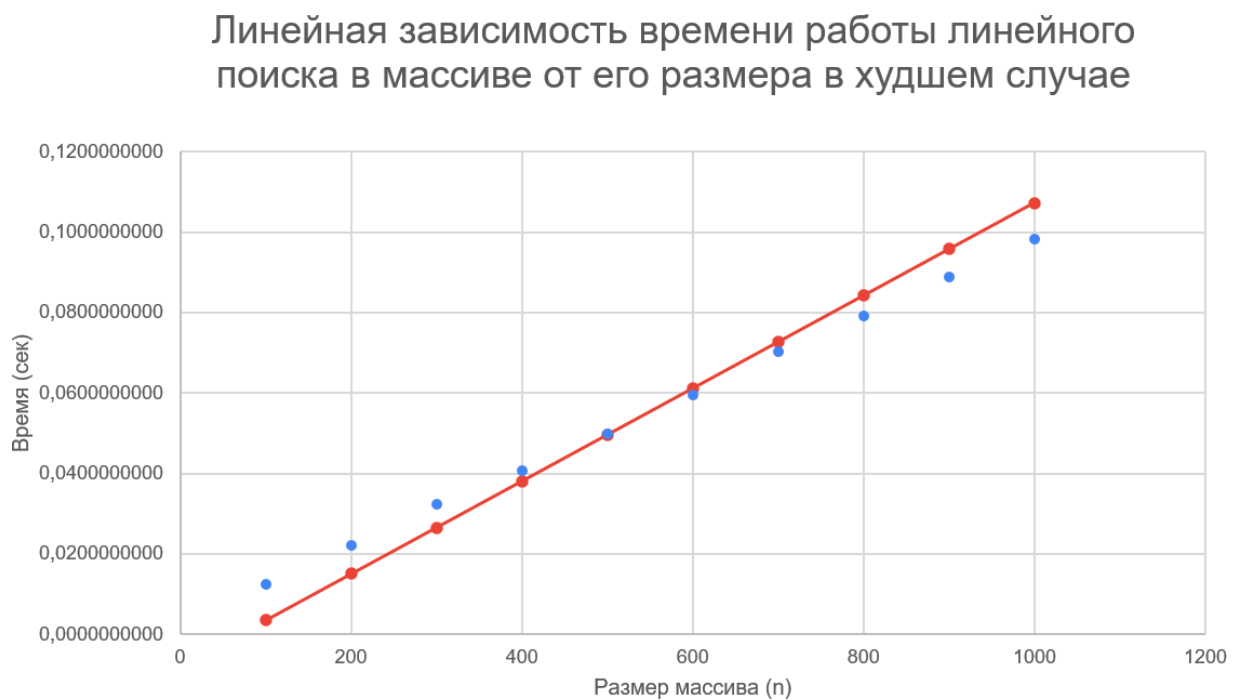


Рисунок 7. Худший случай

Вывод: в ходе выполнения лабораторной работы был проведен анализ алгоритма бинарного поиска в среднем и худшем случаях. Благодаря методу наименьших квадратов был построен график, из которого видно логарифмическую зависимость времени выполнения алгоритма от размера массива. Из полученных результатов, можно сделать вывод о том, что

алгоритм бинарного поиска значительно эффективнее линейного поиска. Кроме того была исследована функция встроенная в Python. Она показала большую эффективность, чем собственная реализация этого алгоритма, из чего можно сделать вывод о том что наиболее эффективным решением для поиска элемента в массиве является функция в Python (`bisect.bisect_left()`)ю.