**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# Лабораторная работа №3
# по дисциплине «Анализ алгоритмов»

**Тема** Поиск в массиве
**Студент** Звягин Д.О.
**Группа** ИУ7-53Б
**Преподаватель** Волкова Л.Л., Строганов Д.В.

Москва, 2024 г.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

**Поиск в массиве** — это одна из задач, часто встречающихся в программировании, поэтому выбор правильного алгоритма может сильно влиять на время выполнения, а следовательно и эффективность программы.

Целью данной работы является анализ трудоёмкости алгоритмов поиска в массиве.

Для достижения этой цели нужно выполнить следующие задачи:

— рассмотреть существующие алгоритмы поиска элемента в массиве;

— разработать рассмотренные алгоритмы поиска в массиве;

— реализовать разработанные алгоритмы;

— проанализировать полученные реализации по трудоёмкости.

# 1 Аналитическая часть

Существует два алгоритма поиска элемента в массиве [4]:

— алгоритм полного перебора;

— алгоритм бинарного поиска.

## 1.1 Алгоритм полного перебора

*Алгоритм полного перебора* (также называемый простым алгоритмом поиска или линейного поиска) — это алгоритм поиска элемента в массиве, подразумевающий последовательное итерирование по всем элементам в массиве. Во время каждой итерации очередной элемент массива сравнивается с искомым. Если значения совпали, итерирование прекращается и алгоритм возвращает индекс текущего элемента.

### 1.1.1 Преимущества

Преимуществом этого алгоритма является полное отсутствие каких-либо ограничений на содержимое массива. (не важен порядок элементов или тип данных, если можно определить равенство объектов этого типа)

### 1.1.2 Недостатки

Недостатком этого алгоритма является скорость его выполнения — в зависимости от того, на какой позиции расположен элемент в массиве, алгоритму может потребоваться от 1 до $N$ итераций, где $N$ — количество элементов в массиве.

## 1.2 Алгоритм бинарного поиска

*Алгоритм бинарного поиска* (также называемый алгоритмом двоичного поиска или поиском делением пополам) — это алгоритм, выполняющий поиск элемента в отсортированном массиве. Во время каждой итерации, размер той части массива, в который выполняется поиск, делится пополам. Это достигается выбором элемента, стоящего в середине этой части и сравнением его с искомым. Если искомый элемент найден, возвращается индекс этого элемента. Иначе, если искомый элемент больше, отбрасывается “левая” часть массива (та, что находится левее текущего элемента), иначе — “правая” (та, что находится правее)

### 1.2.1 Преимущества

Главным преимуществом этого алгоритма является скорость его выполнения.

### 1.2.2 Недостатки

Недостатками этого алгоритма являются:

1) необходимость в сортировке исходного массива;
2) необходимость наличия отношения порядка между элементами исходного массива.

## 1.3 Вывод

Были рассмотрены существующие алгоритмы поиска элемента в массиве.

# 2 Конструкторская часть

## 2.1 Требования к программному обеспечению

К разрабатываемой программе предъявлен ряд требований:

— на вход подаются массив arr и искомый элемент x;

— на выход подаётся целое число или сообщение об ошибке;

— индексация в массиве начинается с нуля;

— над элементами массива определено отношение порядка;

— если искомый элемент не находится в массиве, алгоритмы возвращают -1 в качестве индекса;

— должна быть возможность вывода графиков количества сравнений в зависимости от положения элемента в массиве.

## 2.2 Разработка алгоритмов

Алгоритм простого поиска изображён на рисунке 2.1

Алгоритм двоичного поиска изображён на рисунке 2.2

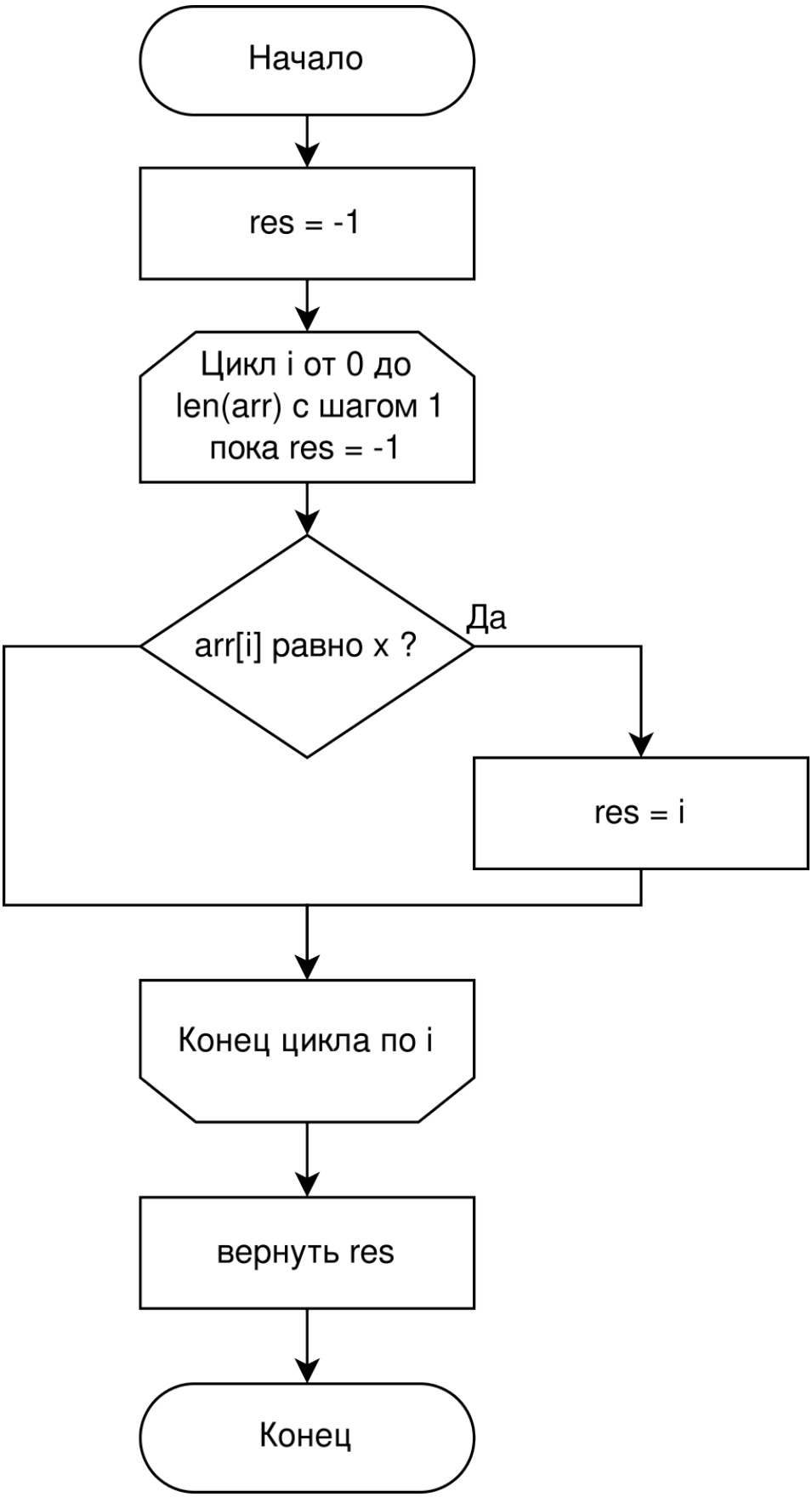


Рисунок 2.1 — Алгоритм простого поиска элемента в массиве

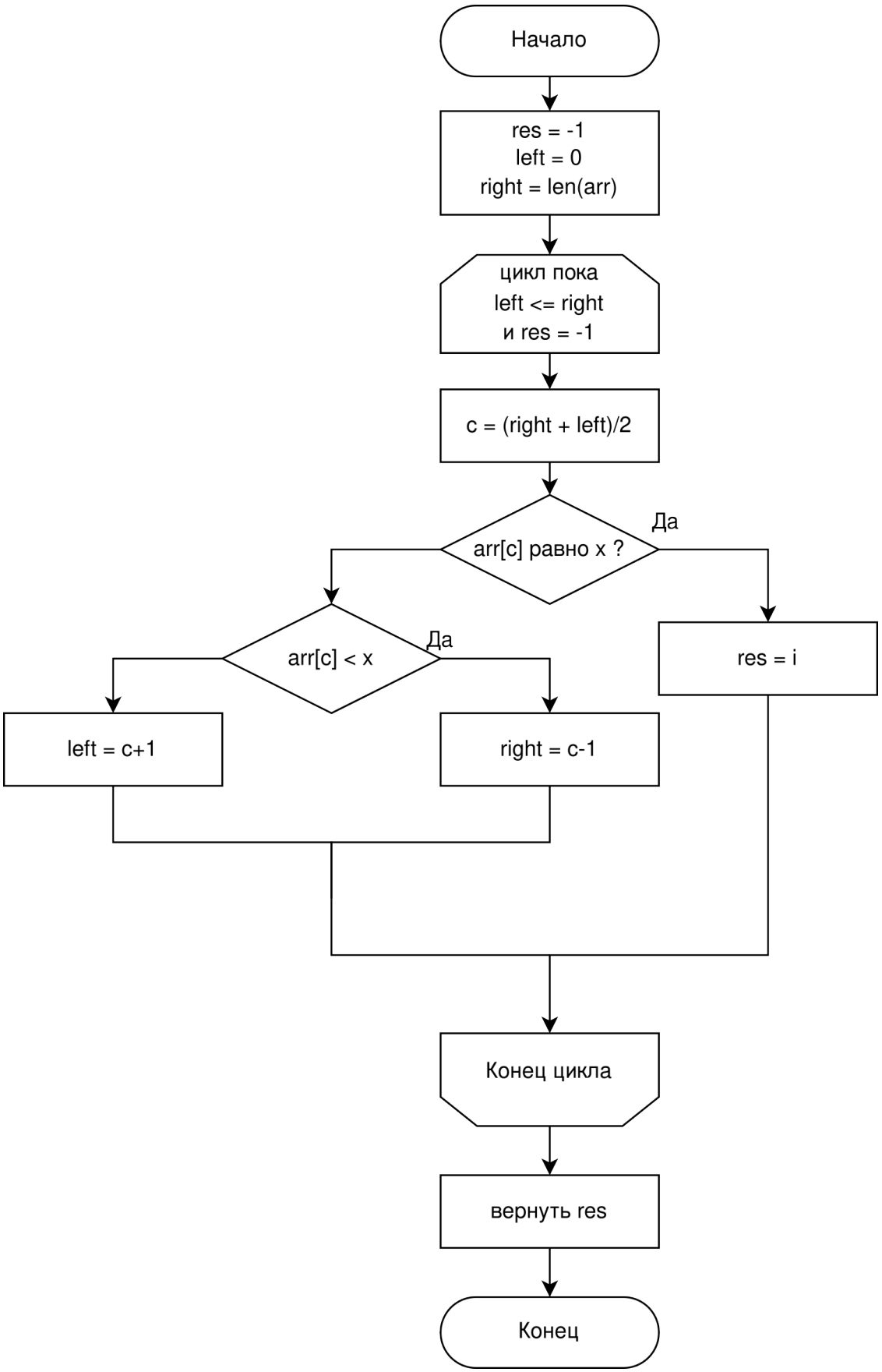


Рисунок 2.2 — Алгоритм двоичного поиска элемента в массиве

## 2.3 Вывод

В результате конструкторской части были определены требования к ПО, а также построены схемы алгоритмов простого и двоичного поиска элементов в массиве.

# 3 Технологическая часть

## 3.1 Средства разработки

В качестве языка программирования был выбран python3 [1], так как его стандартная библиотека достаточна для реализации данных алгоритмов, а также данный язык обладает множеством инструментов для визуализации и работы с данными и таблицами.

В качестве основного файла был выбран инструмент jupyter notebook [2], так как он позволяет организовать код в виде блоков, а также выводить данные и графики прямо в нём, что позволяет наглядно продемонстрировать все замеры.

Для построения графиков использовалась библиотека matplotlib [3].

## 3.2 Реализация алгоритмов

Листинг 3.1 — Простой алгоритм поиска элемента в массиве

```
def search(arr: list[int], x: int) -> int:
  for i in range(len(arr)):
      if arr[i] == x:
          return i
  return -1
```

Листинг 3.2 — Бинарный алгоритм поиска элемента в массиве

```
def binsearch(arr: list[int], x: int) -> int:
  left = 0
  right = len(arr) - 1

  while left <= right:
      c = (right + left) // 2
      if arr[c] == x:
          return c
      elif arr[c] < x:
          left = c + 1
      else:
          right = c - 1

  return -1
```

Для замеров количества сравнений, были использованы модифицированные версии алгоритмов, которые подсчитывают сравнения в процессе работы.

## 3.3 Функциональные тесты

Для функционального тестирования были написаны специальные программы, генерирующие массивы и проверяющие результат работы алгоритма с настоящим положением элемента в массиве.

Эти тесты включают в себя:

— тест с элементом, не принадлежащим массиву (листинг 3.3);

— тест с поиском элемента в пустом массиве (листинг 3.4);

— тесты с поиском существующего в массиве элемента (листинг 3.5).

Листинг 3.3 — Тест поиска несуществующего элемента в массиве

```
arr = [i for i in range(N)]

res = simple.search(arr, -1)
if (res == -1):
    print("Nonexistent element test passed")
else:
    print("Nonexistent element test failed")

res = binary.binsearch(arr, -1)
if (res == -1):
    print("Nonexistent element test passed")
else:
    print("Nonexistent element test failed")
```

Листинг 3.4 — Тест поиска элемента в пустом массиве

```
res = simple.search([], 0)
if (res == -1):
    print("Empty array test passed")
else:
    print("Empty array test failed")

res = binary.binsearch([], 0)
if (res == -1):
    print("Empty array test passed")
else:
    print("Empty array test failed")
```

Листинг 3.5 — Тест поиска элементов в массиве

```
arr = [i for i in range(N)]

for i in range(N):
```

```
    print(f"{i=}", end="\r")
    res = simple.search(arr, i)
    if res != i:
        print(f"Simple search fail {i=}")

for i in range(N):
    print(f"{i=}", end="\r")
    res = binary.binsearch(arr, i)
    if res != i:
        print(f"Binary search fail {i=}")
```

Все тесты пройдены успешно

## Вывод

В ходе работы были разработаны алгоритмы простого и бинарного поиска элемента в массиве, а также было проведено их тестирование.

# 4 Исследовательская часть

## 4.1 Технические характеристики

Технические характеристики устройства, на котором проводились замеры:

— операционная система: EndeavourOS x86_64;

— процессор: 13th Gen Intel(R) Core(TM) i53500H (16) С частотой 4.70 ГГц;

— оперативная память: 16 ГБ с частотой 5200 МГц.

## 4.2 Исследование количества сравнений

В ходе исследования проводились замеры количества количества сравнений, необходимого для нахождения элемента в массиве.

Для измерения используются отсортированные массивы из 1022 элементов, где индекс элемента в массиве совпадает с его значением.

На рисунке 4.1 изображён график зависимости количества сравнений при поиске элемента в массиве от индекса этого элемента с помощью алгоритма простого поиска

На рисунке 4.2 изображена гистограмма, отображающая аналогичную зависимость для алгоритма бинарного поиска

На рисунке 4.3 изображена гистограмма из рисунка 4.2, отсортированная по количеству сравнений

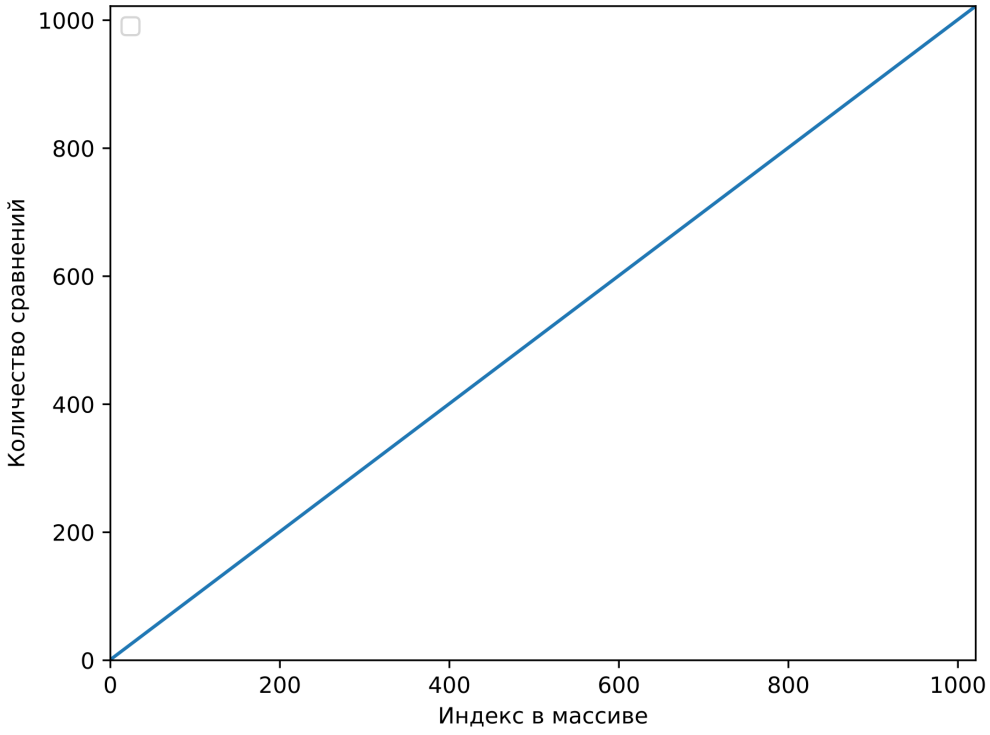


Рисунок 4.1 — График зависимости количества сравнений от индекса элемента с помощью алгоритма простого поиска

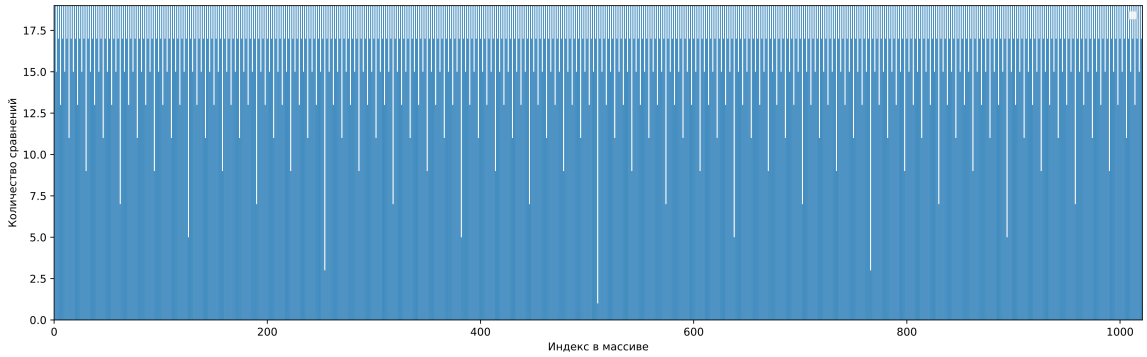


Рисунок 4.2 — Гистограмма зависимости количества сравнений от индекса элемента с помощью алгоритма двоичного поиска

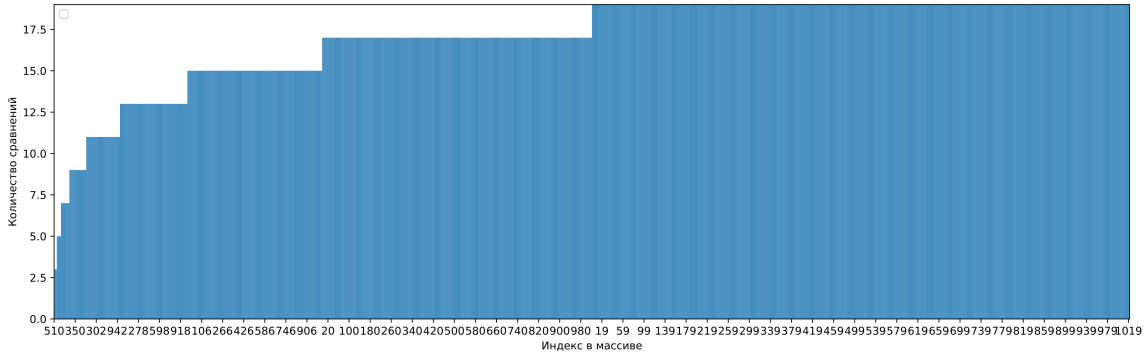


Рисунок 4.3 — Гистограмма зависимости количества сравнений от индекса элемента с помощью алгоритма двоичного поиска, отсортированная по количеству сравнений

## 4.3 Вывод

В результате измерений было обнаружено, что при размере массива равном 1022 элементам, среднее количество сравнений для простого и двоичного алгоритмов поиска равно 511 и 17 сравнениям соответственно.

Максимальное количество сравнений для простого и двоичного алгоритмов поиска равно 1022 и 19 сравнениям соответственно.

В результате исследования установлено, что алгоритм двоичного поиска является более эффективным, чем алгоритм простого поиска для отсортированных массивов длины равной 1022 элементам.

# ЗАКЛЮЧЕНИЕ

Была проанализирована трудоёмкость алгоритмов поиска в массиве.

В результате исследования установлено, что алгоритм двоичного поиска является более эффективным, чем алгоритм простого поиска для отсортированных массивов длины равной 1022 элементам.

Среднее количество сравнений для простого и двоичного алгоритмов поиска равно 511 и 17 сравнениям соответственно.

Максимальное количество сравнений для простого и двоичного алгоритмов поиска равно 1022 и 19 сравнениям соответственно.

Были выполнены следующие задачи:

— были рассмотрены существующие алгоритмы поиска элемента в массиве;

— рассмотренные алгоритмы поиска в массиве были разработаны;

— разработанные алгоритмы были реализованы;

— полученные реализации были проанализированы по трудоёмкости.

Цели и задачи лабораторной работы выполнены.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Документация Языка программирования Python / [Электронный ресурс] // Python : [сайт]. — URL: https://www.python.org/ (дата обращения: 25.09.2024).

2. Документация Jupyter Notebook: The Classic Notebook Interface / [Электронный ресурс] // Jupyter : [сайт]. — URL: https://jupyter.org/ (дата обращения: 25.09.2024).

3. Документация библиотеки Matplotlib: Visualization with Python / [Электронный ресурс] // Matplotlib : [сайт]. — URL: https://matplotlib.org/ (дата обращения: 25.09.2024).

4. Д. К. Искусство программирования для ЭВМ. Том 3. Сортировка и поиск. // ООО “И. Д. Вильямс” // 2014 год // 824 страницы