МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г.
ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Лабораторная работа №4

по дисциплине: Алгоритмы и структуры данных тема:
«Сравнительный анализ алгоритмов поиска(Pascal/C)»

Выполнил: ст. группы ПВ-211

Чувилко Илья Романович

Проверил:

Синюк Василий Григорьевич

Белгород 2022 г.

Цель работы: изучение алгоритмов поиска элемента в массиве и закрепление навыков в проведении сравнительного анализа алгоритмов.

1. Изучить алгоритмы поиска:
   1) в неупорядоченном массиве:
      - линейный;
      - быстрый линейный;
   2) в упорядоченном массиве:
      - быстрый линейный;
      - бинарный;
      - блочный.

2. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов поиска.

3. Провести эксперименты по определению временных характеристик алгоритмов поиска. Результаты экспериментов представить в виде таблиц 12 и 13. Клетки таблицы 12 содержат максимальное количество операций сравнения при выполнении алгоритма поиска, а клетки таблицы 13 — среднее число операций сравнения.

4. Построить графики зависимости количества операций сравнения от количества элементов в массиве.

5. Определить аналитическое выражение функции зависимости количества операций сравнения от количества элементов в массиве.

6. Определить порядок функций временной сложности алгоритмов поиска.

Задание 2

```
#include <stdio.h>
#include "stdbool.h"
#include <time.h>
#include <stdlib.h>
#include <math.h>

unsigned numCompares = 0;

bool linearSearch(int *a, const int n, const int x) {
  int *ptr = a;
  for (int i = 0; i < n; i++) {
    numCompares++;
    if (*ptr == x)
      return true;
  }
  return false;
}

bool linearSearchFast(int *a, const int n, const int x) {
  int i = 0;
  int t = a[n];
  a[n] = x;
  while (a[i] != x) {
```

```
    numCompares++;
    i++;
  }
  a[n] = t;
  return a[i] == x;
}

bool linearSearchOrdered(int *a, const int n, const int x) {
  int i = 0;
  int t = a[n];
  a[n] = x;
  while (a[i] < x) {
    numCompares++;
    i++;
  }
  a[n] = t;
  return i < n;
}

bool binarySearch(int *a, const int n, const int x) {
  int left = -1;
  int right = n;
  while (right - left > 1) {
    numCompares++;
    int middle = left + (right - left) / 2;

    numCompares++;
    if (a[middle] < x)
      left = middle;
    else
      right = middle;
  }
  return a[right] == x;
}

int min(const int x, const int y) {
  return x < y ? x : y;
}

bool blockSearch(int *a, const int n, const int x) {
  const int blockSize = ceil(sqrt(n));
  const int nBlocks = ceil(n / blockSize);
  for (int beginningOfBlock = blockSize * (nBlocks - 1);
       beginningOfBlock >= 0; beginningOfBlock -= blockSize)
    if (++numCompares && a[beginningOfBlock] <= x)
      return linearSearchOrdered(a + beginningOfBlock,
                                 min(blockSize, n - beginningOfBlock), x);
  return false;
}



void getSortedArray(int *a, int n) {
  for (int i = 0; i < n; i++) {
    a[i] = i;
  }
}

void getReversedArray(int *a, int n) {
  for (int i = 0; i < n; i++) {
    a[i] = n - 1 - i;
  }
```

```
}

void getRandomArray(int *a, int n) {
  for (int i = 0; i < n; i++) {
    a[i] = rand() % n;
  }
}

#define SIZE_ARRAY 50
#define ACCURACY 1000

void getTimesForOrdered(bool (*SearchFunc)(int*, int, int)) {
  int array[SIZE_ARRAY];

  for (int size = 5; size <= SIZE_ARRAY; size += 5) {
    getSortedArray(array, size);

    unsigned sum = 0;
    for (int i = 0; i < ACCURACY; i++) {
      numCompares = 0;
      SearchFunc(array, size, rand() % size);
      sum += numCompares;
    }

    printf("%d ", (int)ceil((float)sum / (float)ACCURACY));
  }
  printf("\n");
}

void getTimesForUnordered(bool (*SearchFunc)(int*, int, int)) {
  int array[100];

  getTimesForOrdered(SearchFunc);

  for (int size = 5; size <= SIZE_ARRAY; size += 5) {
    getReversedArray(array, size);

    unsigned sum = 0;
    for (int j = 0; j < ACCURACY; j++) {
      numCompares = 0;
      SearchFunc(array, size, rand() % size);
      sum += numCompares;
    }

    printf("%d ", sum / ACCURACY);
  }
  printf("\n");

  for (int size = 5; size <= SIZE_ARRAY; size += 5) {
    getRandomArray(array, size);

    unsigned sum = 0;
    for (int i = 0; i < ACCURACY; i++) {
      numCompares = 0;
      SearchFunc(array, size, rand() % size);
      sum += numCompares;
    }

    printf("%d ", sum / ACCURACY);
  }
```

```
  printf("\n");
}

int main() {
  printf("linearSearch:\n");
  getTimesForUnordered(&linearSearch);

  printf("\nlinearSearchFast:\n");
  getTimesForUnordered(&linearSearchFast);

  printf("\nlinearSearchOrdered:\n");
  getTimesForOrdered(&linearSearchOrdered);

  printf("\nbinarySearch:\n");
  getTimesForOrdered(&binarySearch);

  printf("\nblockSearch:\n");
  getTimesForOrdered(&blockSearch);
}
```

Задание 3

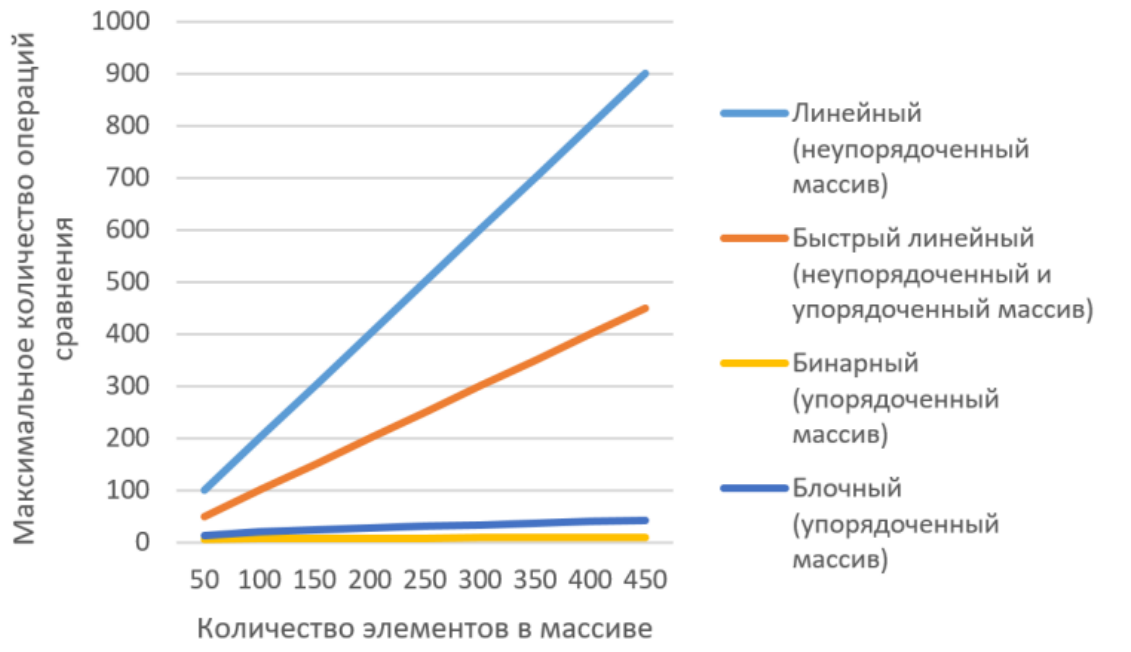
Максимальное количество операций сравнения

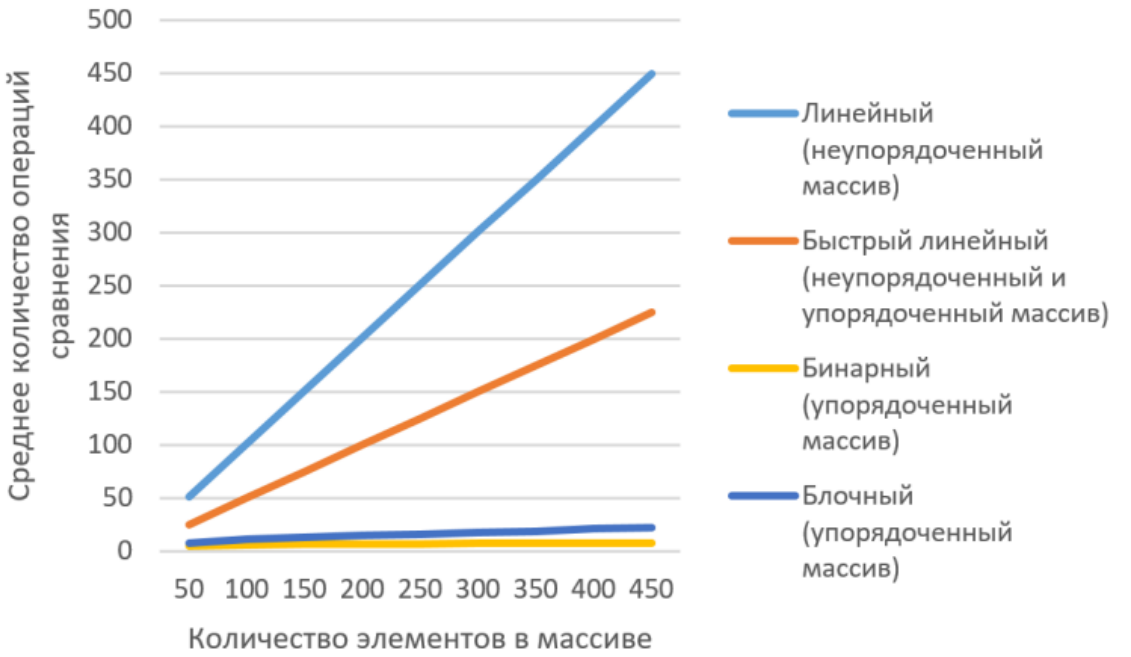
Алгоритмы поиска	Количество элементов в массиве								
	50	100	150	200	250	300	350	400	450
Линейный (неупорядоченный массив)	100	200	300	400	500	600	700	800	900
Быстрый линейный (неупорядоченный массив)	50	100	150	200	250	300	350	400	450
Быстрый линейный (упорядоченный массив)	50	100	150	200	250	300	350	400	450
Бинарный (упорядоченный массив)	6	7	8	8	8	9	9	9	9
Блочный (упорядоченный массив)	14	20	24	28	31	34	37	40	42

Среднее количество операций сравнения

Алгоритмы поиска	Количество элементов в массиве								
	50	100	150	200	250	300	350	400	450
Линейный (неупорядоченный массив)	51	101	151	201	251	301	350	400	450
Быстрый линейный (неупорядоченный массив)	25	50	75	100	125	150	175	200	225
Быстрый линейный (упорядоченный массив)	25	50	75	100	125	150	175	200	225
Бинарный (упорядоченный массив)	5	6	7	7	7	8	8	8	8
Блочный (упорядоченный массив)	8	11	13	15	16	18	19	21	22

Задание 4





Задание 5:
1) Линейный поиск (неупорядоченный массив): $C_{max}(N) = N * 2$
2) Быстрый линейный поиск (неупорядоченный массив): $C_{max}(N) = N$
3) Быстрый линейный поиск (упорядоченный массив): $C_{max}(N) = N$
4) Бинарный поиск (упорядоченный массив): $Cmax(N) = \lceil \log N \rceil$
5) Блочный поиск (упорядоченный массив): $Cmax(N) = \lfloor \sqrt{N} * 2 \rfloor$

Задание 6:
6) Линейный поиск (неупорядоченный массив): $O(N) = N$
7) Быстрый линейный поиск (неупорядоченный массив): $O(N) = N$
8) Быстрый линейный поиск (упорядоченный массив): $O(N) = N$
9) Бинарный поиск (упорядоченный массив): $O(N) = \log N$
10) Блочный поиск (упорядоченный массив): $O(N) = \sqrt{N}$

Контрольные вопросы:

1. В чем заключается задача поиска?
2. Всегда ли быстрый линейный поиск быстрее линейного поиска?
3. От чего зависит время поиска в неупорядоченном массиве?
4. Чем алгоритм быстрого линейного поиска в упорядоченном массиве отличается от алгоритма быстрого линейного поиска в неупорядоченном массиве?
5. В чем заключается бинарный поиск?
6. Определите индексы элементов массива, бинарный поиск которых наиболее продолжителен.
7. Разработайте и реализуйте итеративный и рекурсивный алгоритмы бинарного поиска?
8. В чем заключается блочный поиск?
9. От чего зависит время блочного поиска?
10. Как правильно выбрать количество блоков в блочном поиске?
11. Определите максимальное количество элементов массива, которые могут быть обработаны при блочном поиске.
12. Пусть искомый элемент равен i-му элементу массива. Какой алгоритм рациональнее использовать в этом случае?
13. Выполните сравнительный анализ алгоритмов поиска для случая, когда искомого элемента нет в массиве.
14. Выполните сравнительный анализ алгоритмов поиска для случая, когда в массиве только один элемент.
15. Реализуйте алгоритмы поиска на языке программирования высокого уровня. Выполните трассировку при поиске в массиве из одного элемента.
16. От чего завист порядок функции временной сложности алгоритмов поиска. Каким он может быть для различных алгоритмов?

Вывод: в ходе выполнения лабораторной работы мы изучили методы поиска элемента в массиве и приобрели навыки в проведении сравнительного анализа различных методов поиска.