Московский авиационный институт

(Национальный исследовательский университет)

Институт №3:

Информатика и вычислительная техника

Кафедра 304.

Отчёт по лабораторной работе

По учебной дисциплине «Программирование»

На тему

«*Алгоритмы поиска*»

Группа: М3О-210Б-23

Выполнил:

Миронов А.Д.

Приняли:

Чечиков Ю.Б.

2024 год Москва

Задание

Лабораторная работа № 2
Алгоритмы поиска

Цель работы: изучить основные принципы работы алгоритмов поиска, исследовать их свойства:

- алгоритм **BLS** - последовательный поиск (Better_Linear_Search);
- алгоритм **SLS** - быстрый последовательный поиск (Sentinel_ Linear_Search);
- алгоритм **OAS** - последовательный поиск в упорядоченном массиве (Ordered_Array_Search);
- алгоритм **BS** - бинарный поиск (Binary Search).

Задание

Для алгоритмов **BLS** и **SLS** в качестве входного массива использовать одну и ту же последовательность значений (функция **rand**( ), можно использовать соответствующую функцию из первой лабораторной работы).

Для алгоритмов **OAS** и **BS** – значения массива должны быть отсортированы по неубыванию, одна и та же последовательность чисел (можно использовать соответствующую функцию из первой лабораторной работы).

Оценить длительность поиска для различных значений размеров последовательностей (начиная с 10000 до 200000 элементов массива, провести измерения не менее, чем для 10 разных размерностей).

Для каждой размерности рассматриваются случаи нахождения ключа поиска в начале, в середине и в конце массива, случай отсутствия ключа в массиве.

Для алгоритмов **BLS** и **SLS** кроме подсчета **времени,** необходимого для поиска, требуется определить сколько раз выполняются операции **сравнения** (сравнение ключа с элементом массива, а также в **BLS** добавляется подсчет сравнений при анализе индекса элемента массива в цикле…).

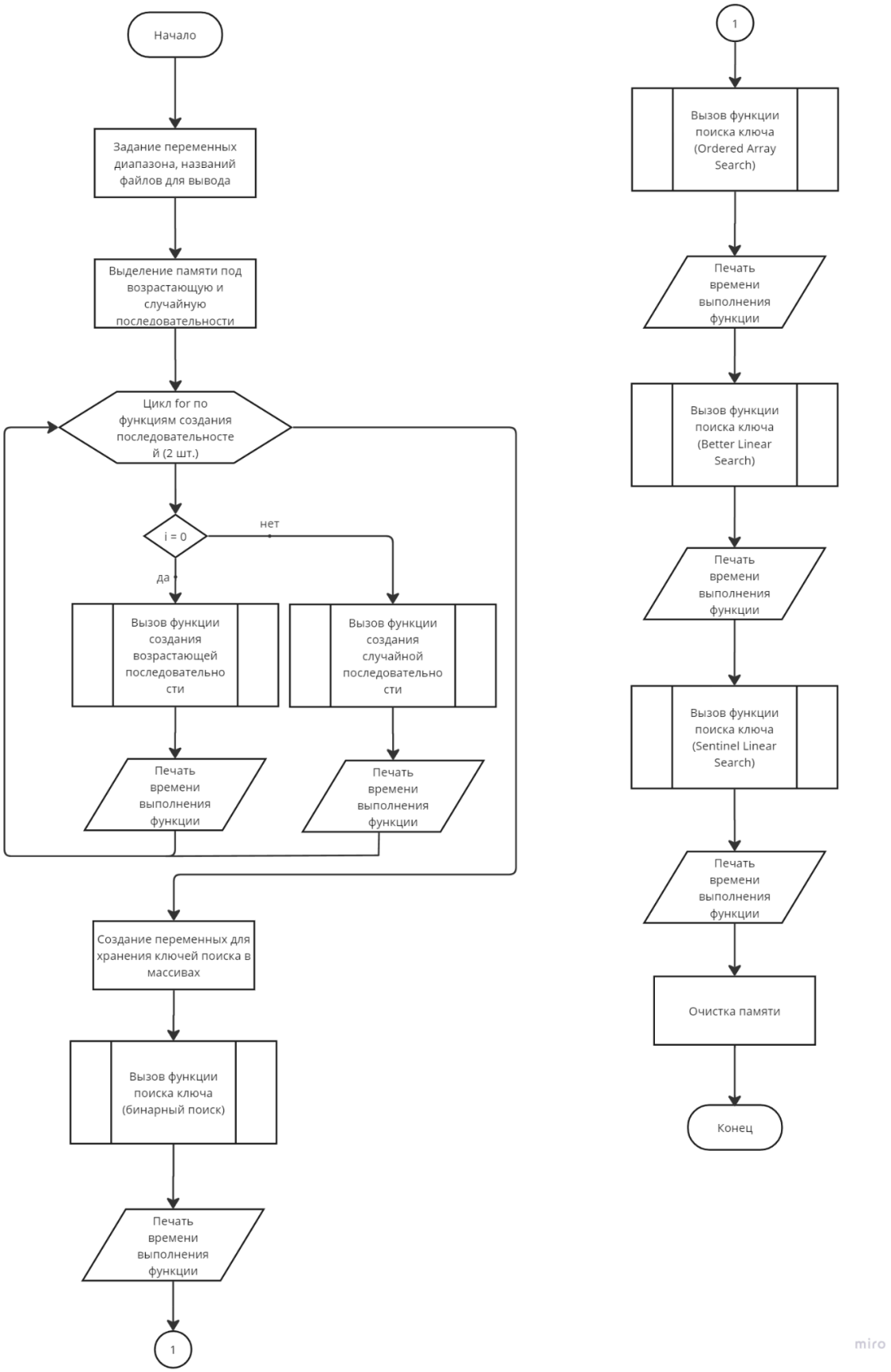
Все результаты оформить в виде таблиц и графиков. На графиках **- только временные характеристики** поиска.

По результатам сделать выводы об эффективности того или иного алгоритма поиска.

Отчет по лабораторной работе должен содержать:

- задание;
- структурные схемы главной функции, всех алгоритмов поиска;
- тексты программ;
- результаты работы программ и результаты сопоставительного анализа – в виде таблиц, графиков;
- выводы по работе.

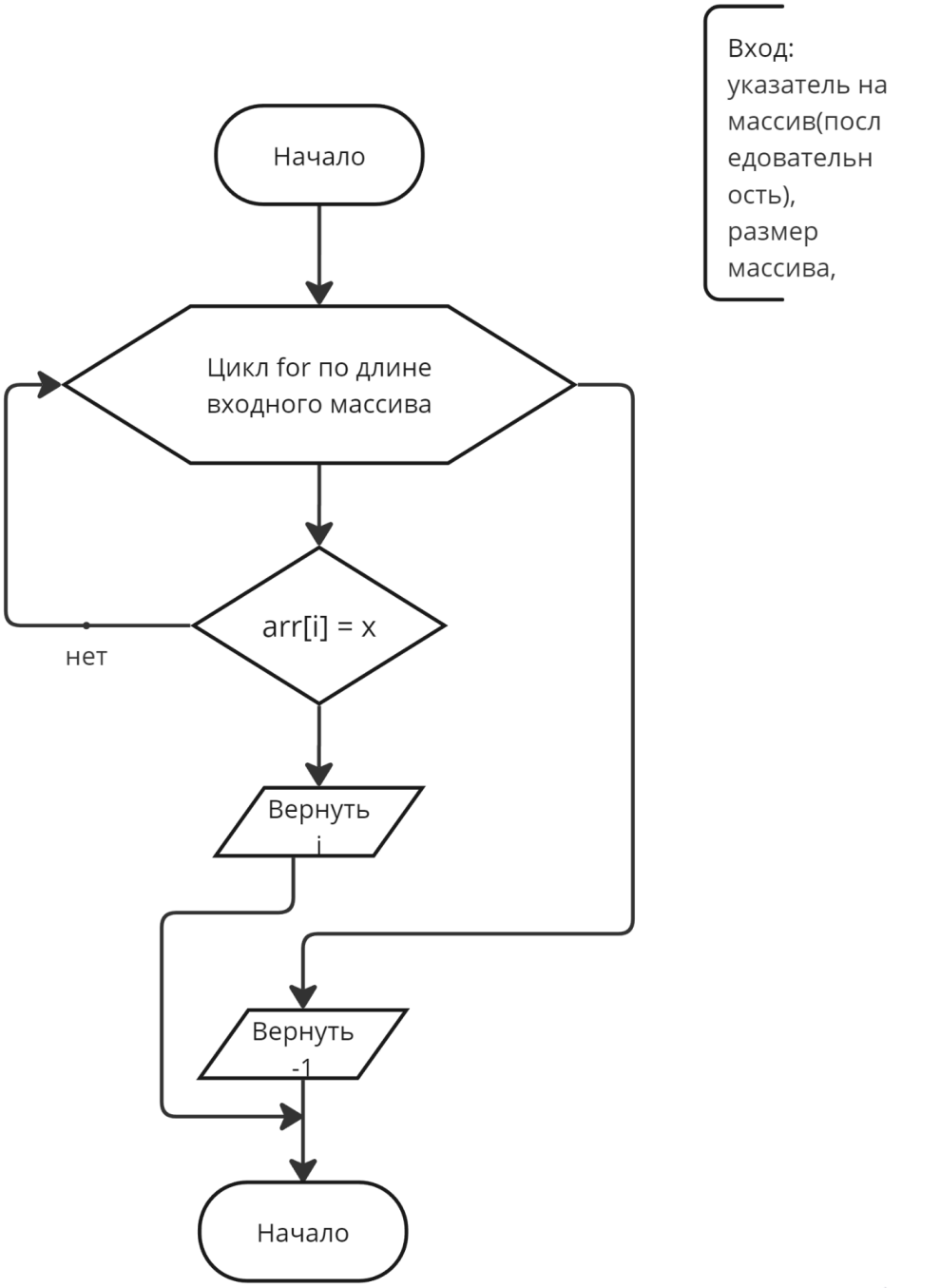
Структурная схема главной функции



Структурные схемы алгоритмов поиска

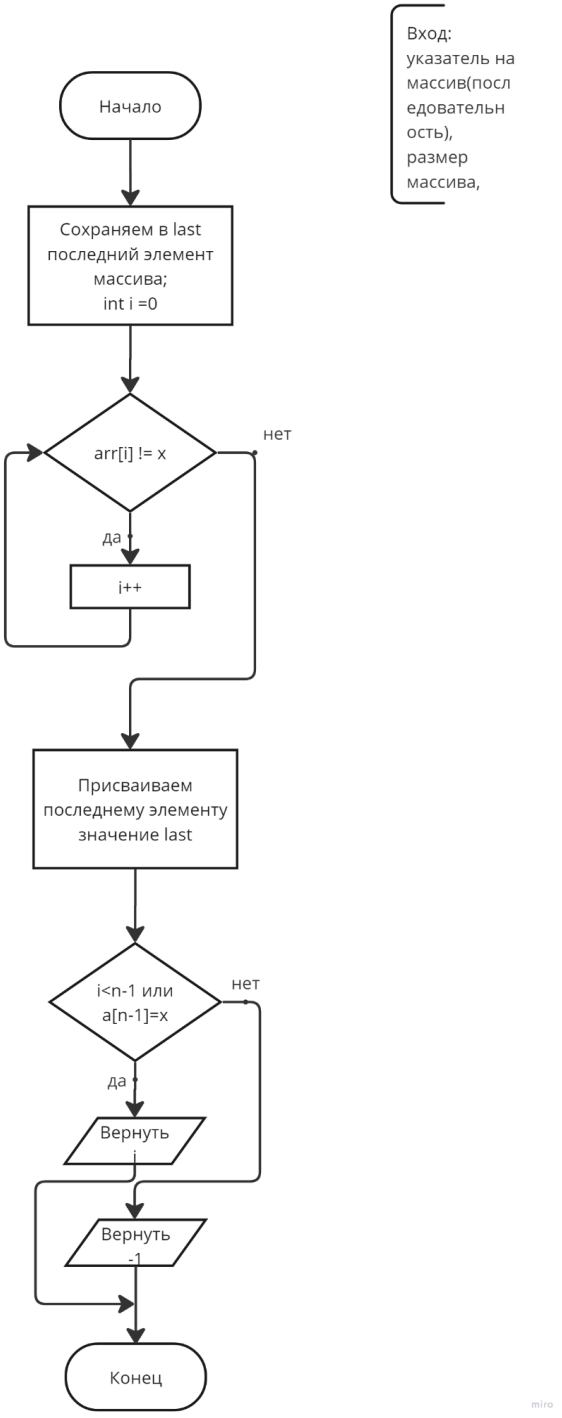
Линейный поиск (Better Linear Search)

Функция выполняет линейный поиск элемента в заданном массиве. На вход поступают следующие параметры: массив целых чисел для поиска, количество элементов в массиве (n) и искомое значение (x). При нахождении элемента функция возвращает его индекс, в противном случае — -1, что указывает на отсутствие элемента в массиве



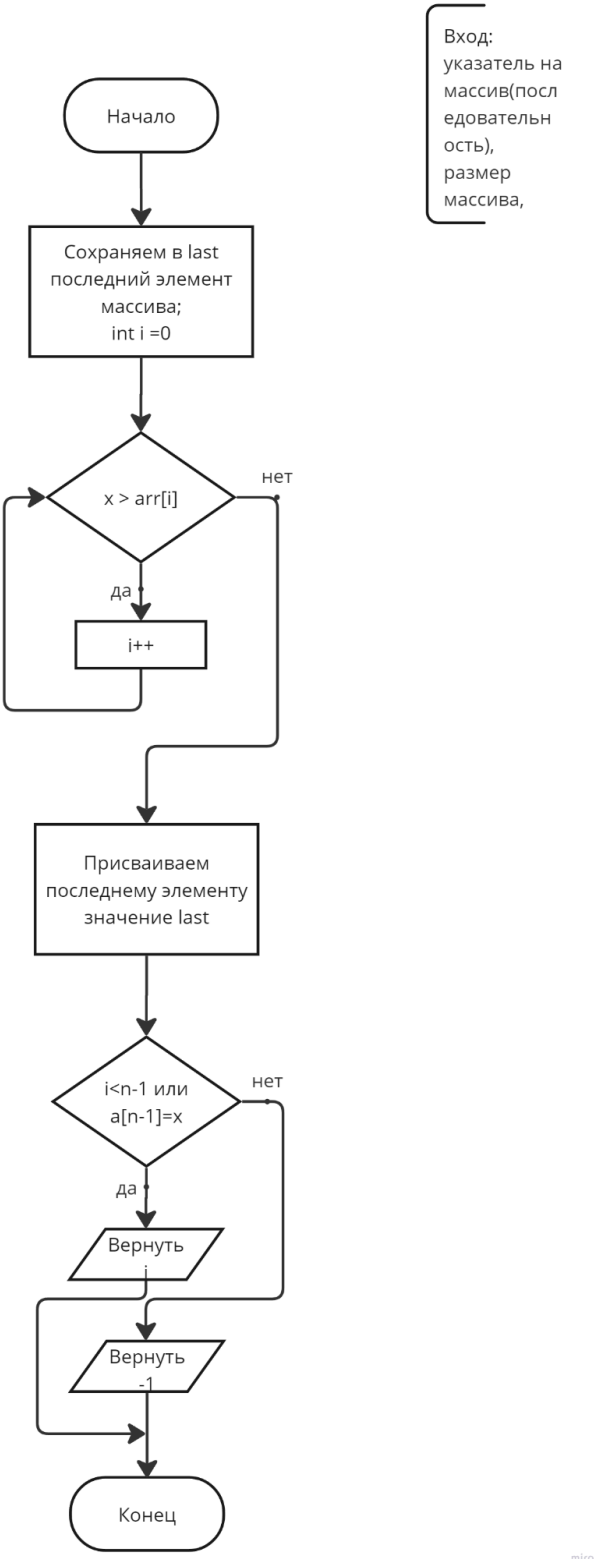
Sentinel Linear Search

Функция выполняет поиск элемента в заданном массиве с использованием метода линейного "сентинельного" поиска. На вход поступают следующие параметры: массив целых чисел для поиска, количество элементов в массиве (n) и искомое значение (x).
Сначала функция сохраняет последний элемент массива, затем перемещается по массиву, пока не найдет искомое значение. После этого последний элемент восстанавливается в свою позицию. Если элемент найден, функция возвращает его индекс; если элемент отсутствует, возвращается -1.



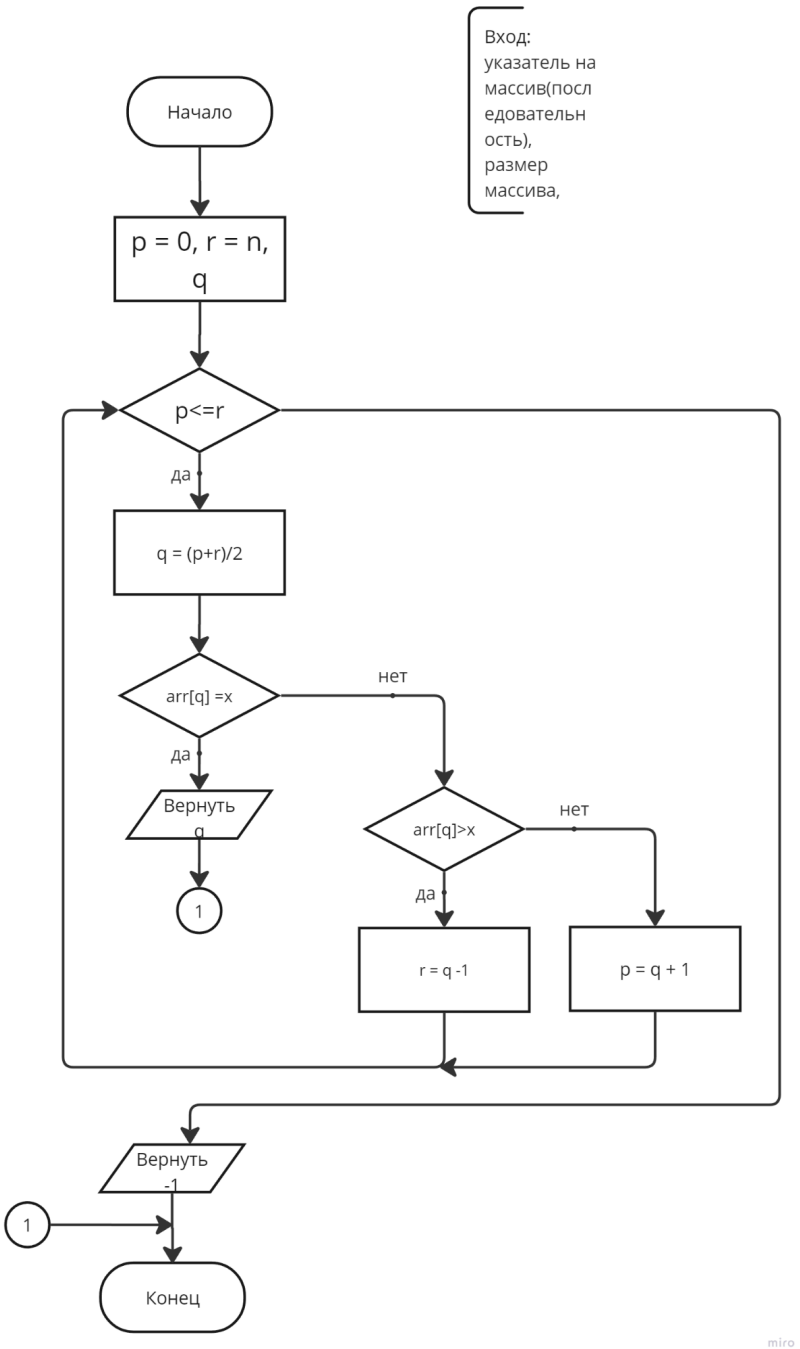
Ordered Array Search

Вход: массив целых чисел для поиска, количество элементов в массиве (n) и искомое значение (x).
Алгоритм аналогичен алгоритму *Sentinel Linear Search* с измененным циклом while (x > arr[i]) для работы с отсортированным массивом



Бинарный поиск (Binary Search)

Функция осуществляет бинарный поиск элемента в отсортированном массиве. На вход принимаются следующие параметры: указатель на массив целых чисел для поиска, количество элементов в массиве (n) и искомое значение (x). В процессе выполнения функция последовательно делит массив пополам, сравнивая искомый элемент с центральным элементом. Если элемент найден, возвращается его индекс; если элемент меньше центрального, поиск продолжается в левой половине массива, в противном случае — в правой. Если элемент отсутствует, функция возвращает -1, что свидетельствует о его недоступности в массиве.



Код программы

```
/*****************************************************************************
*                КУРС КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ                                *
*----------------------------------------------------------------------------*
* Project Type  : Win32 Console Application                                  *
* Project Name  : lab2                                                       *
* File Name     : lab2.cpp                                                   *
* Programmer(s) : Миронов А.Д., Фомин В.А.,  М3О-210Б-23                     *
* Modified By   :                                                            *
* Created       : 06/10/24                                                   *
* Last Revision :                                                            *
* Comment(s)    :                                                            *
*****************************************************************************/

 #include <iostream>
 #include <chrono>
 #include <fstream>
 #include <math.h>

 using namespace std;

 void increasing(int* arr,int size, int min, int max);
 void random(int* arr, int size, int min, int max);

 int BLS(int* arr, int n, int x, int* count);      // Better_Linear_Search
 int SLS(int* arr, int n, int x, int* count);      // Sentinel_Linear_Search
 int OAS(int* arr, int n, int x, int* count);      // Ordered_Array_Search
 int BS(int* arr, int n, int x, int* count);   // Binary_Search

 void file_output(const char* filename, int* arr, int size);

 time_t time_count(void (*func)(int*, int, int, int), int* arr, int size, int min,
 int max, const char* filename);
 time_t algo_time(int (*func)(int*, int, int, int*), int* arr, int size, int key,
 int* count);

 void (*functions[])(int*, int, int, int) = {
        increasing,
        random
 };

 int main(){
    srand(time(NULL));

    const char fnames[2][30]{
        "increasing.txt",
        "random.txt"
    };

    int count = 0;

    int size = 145000;
```

```
    int min = 0;
    int max = 200000;

    int* Randoms = new int[size];
    int* Ordered = new int[size];

    // Формирование файлов
    for (int i = 0; i < 2; i++){
        if(i == 0) {
            cout << fnames[i] << " time taken to generate: " <<
time_count(functions[i], Ordered, size, min, max, fnames[i]) << "mks" << endl;
        } else{
            cout << fnames[i] << " time taken to generate: " <<
time_count(functions[i], Randoms, size, min, max, fnames[i]) << "mks" << endl;
        }
    }

    cout << endl;

    int wantedInRandBegin = Randoms[10];
    int wantedInRandMid = Randoms[size / 2];
    int wantedInRandEnd = Randoms[size - 10];
    int wantedInRandNo = -1;

    int wantedInOrderedBegin = 10;
    int wantedInOrderedMid = size / 2;
    int wantedInOrderedEnd =  size - 10;
    int wantedInOrderedNo = -1;

    /// В начале списка
    cout << "\t\tSorted Beginning" << endl;
    cout << "Binary_Search\t key: " << wantedInOrderedBegin << "\t index: " <<
BS(Ordered, size, wantedInOrderedBegin, &count)
    << "\t time to find: " << algo_time(BS, Ordered, size, wantedInOrderedBegin,
&count) << " ns" << endl;
    cout << "Ordered_Array_Search\t key: " << wantedInOrderedBegin << "\t index: " <<
OAS(Ordered, size, wantedInOrderedBegin, &count)
    << "\t time to find: " << algo_time(OAS, Ordered, size, wantedInOrderedBegin,
&count) << " ns" << endl;

    cout << endl;

    cout << "\t\tRand Beginning" << endl;
    cout << "Better_Linear_Search\t key: " << wantedInRandBegin << "\t index: " <<
BLS(Randoms, size, wantedInRandBegin, &count)
    << "\t time to find: " << algo_time(BLS, Randoms, size, wantedInRandBegin,
&count) << " ns" << "\tNumber of comparisons " << count << endl;

    count = 0;

    cout << "Sentinel_Array_Search\t key: " << wantedInRandBegin << "\t index: " <<
SLS(Randoms, size, wantedInRandBegin, &count)
    << "\t time to find: " << algo_time(SLS, Randoms, size, wantedInRandBegin,
```

```
&count) << " ns" << "\tNumber of comparisons " << count << endl;
   cout << "###################################" << endl;

   ////////////////////////
   ///
   /// В середине
   cout << "\t\tSorted Middle" << endl;
   cout << "Binary_Search\t key: " << wantedInOrderedMid << "\t index: " <<
BS(Ordered, size, wantedInOrderedMid, &count)
   << "\t time to find: " << algo_time(BS, Ordered, size, wantedInOrderedMid,
&count) << " ns" << endl;
   cout << "Ordered_Array_Search\t key: " << wantedInOrderedMid << "\t index: " <<
OAS(Ordered, size, wantedInOrderedMid, &count)
   << "\t time to find: " << algo_time(OAS, Ordered, size, wantedInOrderedMid,
&count) << " ns" << endl;

   cout << endl;

   cout << "\t\tRand Middle" << endl;
   cout << "Better_Linear_Search\t key: " << wantedInRandMid << "\t index: " <<
BLS(Randoms, size, wantedInRandMid, &count)
   << "\t time to find: " << algo_time(BLS, Randoms, size, wantedInRandMid, &count)
<< " ns" << "\tNumber of comparisons " << count << endl;

   count = 0;

   cout << "Sentinel_Array_Search\t key: " << wantedInRandMid << "\t index: " <<
SLS(Randoms, size, wantedInRandMid, &count)
   << "\t time to find: " << algo_time(SLS, Randoms, size, wantedInRandMid, &count)
<< " ns" << "\tNumber of comparisons " << count << endl;
   cout << "###################################" << endl;
   ////////////////////////////////////
   ///
   /// В Конце
   cout << "\t\tSorted End" << endl;
   cout << "Binary_Search\t key: " << wantedInOrderedEnd << "\t index: " <<
BS(Ordered, size, wantedInOrderedEnd, &count)
   << "\t time to find: " << algo_time(BS, Ordered, size, wantedInOrderedEnd,
&count) << " ns" << endl;
   cout << "Ordered_Array_Search\t key: " << wantedInOrderedEnd << "\t index: " <<
OAS(Ordered, size, wantedInOrderedEnd, &count)
   << "\t time to find: " << algo_time(OAS, Ordered, size, wantedInOrderedEnd,
&count) << " ns" << endl;

   cout << endl;

   cout << "\t\tRand End" << endl;
   cout << "Better_Linear_Search\t key: " << wantedInRandEnd << "\t index: " <<
BLS(Randoms, size, wantedInRandEnd, &count)
   << "\t time to find: " << algo_time(BLS, Randoms, size, wantedInRandEnd, &count)
<< " ns" << "\tNumber of comparisons " << count << endl;

   count = 0;
```

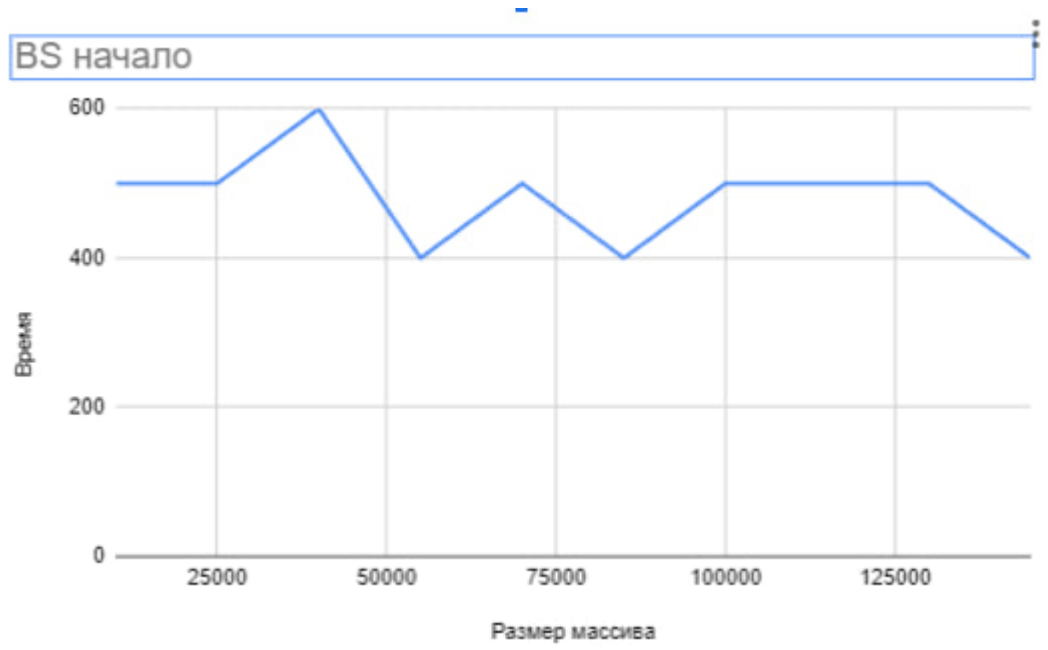
```
    cout << "Sentinel_Array_Search\t key: " << wantedInRandBegin << "\t index: " <<
SLS(Randoms, size, wantedInRandBegin, &count)
    << "\t time to find: " << algo_time(SLS, Randoms, size, wantedInRandBegin,
&count) << " ns" << "\tNumber of comparisons " << count << endl;
    cout << "###################################" << endl;

    ////////////////////////////////////////
    ///
    /// Нет

    cout << "\t\tSorted No" << endl;
    cout << "Binary_Search\t key: " << wantedInOrderedNo << "\t index: " <<
BS(Ordered, size, wantedInOrderedNo, &count)
    << "\t time to find: " << algo_time(BS, Ordered, size, wantedInOrderedNo, &count)
<< " ns" << endl;
    cout << "Ordered_Array_Search\t key: " << wantedInOrderedNo << "\t index: " <<
OAS(Ordered, size, wantedInOrderedNo, &count)
    << "\t time to find: " << algo_time(OAS, Ordered, size, wantedInOrderedNo,
&count) << " ns" << endl;

    cout << endl;

    cout << "\t\tRand No" << endl;
    cout << "Better_Linear_Search\t key: " << wantedInRandNo << "\t index: " <<
BLS(Randoms, size, wantedInRandNo, &count)
    << "\t time to find: " << algo_time(BLS, Randoms, size, wantedInRandNo, &count)
<< " ns" << "\tNumber of comparisons " << count << endl;

    count = 0;

    cout << "Sentinel_Array_Search\t key: " << wantedInRandNo << "\t index: " <<
SLS(Randoms, size, wantedInRandNo, &count)
    << "\t time to find: " << algo_time(SLS, Randoms, size, wantedInRandNo, &count)
<< " ns" << "\tNumber of comparisons " << count << endl;

    delete[] Randoms;
    delete[] Ordered;

    return 0;
}

void increasing(int* arr,int size, int min, int max){
    for(int i = 0; i < size; i++){
        // cout << min << " " << max << " " << i << " " << size << endl;
        arr[i] = i;
    }
}

void random(int* arr, int size, int min, int max){
    for (int i = 0; i < size; i++) {
        arr[i] = min + rand() % (max - min + 1); // Генерация случайного числа в
диапазоне [min, max]
```

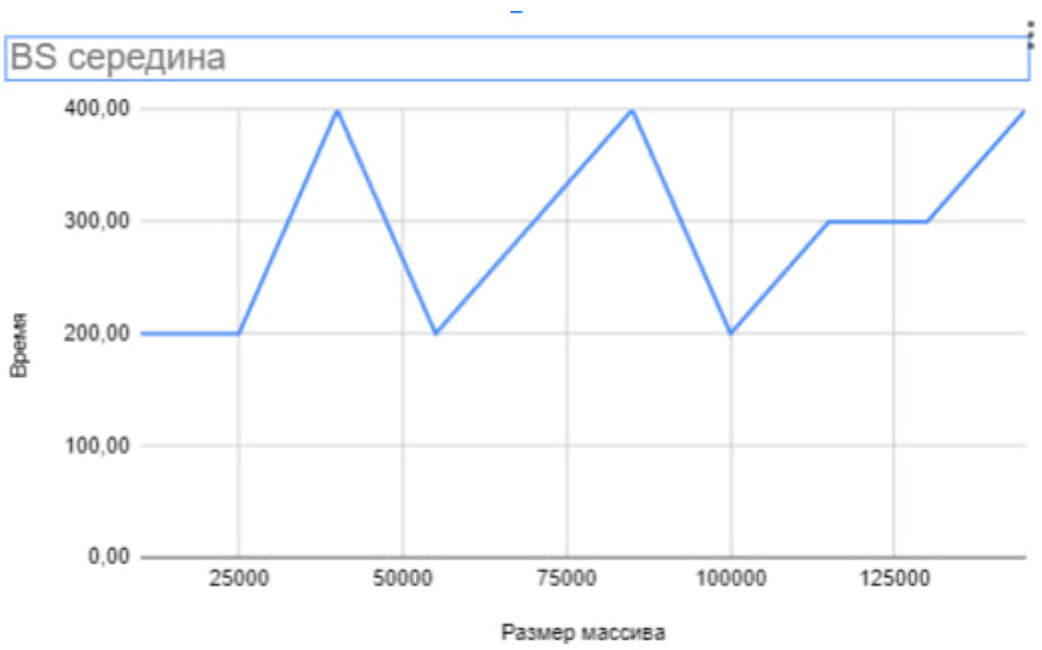
```
        if(i == (size - 10)){ // Позиция искомого числа
            arr[i] = -2; // Искомое число
        }
    }
}

int BLS(int* arr, int n, int x, int* count){
    int c = 0; // Счетчик сравнений
    for(int i = 0; i < n; i++) {
        c++; // Увеличиваем счетчик при каждом сравнении
        if (arr[i] == x) {
            *count = c; // Сохраняем количество сравнений
            return i; // Возвращаем индекс найденного элемента
        }
    }
    *count = c; // Если элемент не найден, сохраняем количество сравнений
    return -1; // Возвращаем -1, если элемент не найден
}

int SLS(int* arr, int n, int x, int* count){
    int c = 0; // Счетчик сравнений
    int last = arr[n-1];
    arr[n-1] = x; // Устанавливаем последний элемент в искомое значение
    int i = 0;

    while(arr[i] != x){
        c++; // Увеличиваем счетчик при каждом сравнении
        i++;
    }

    arr[n-1] = last; // Восстанавливаем последний элемент
    c++; // Увеличиваем счетчик для последнего сравнения

    // Сохраняем количество сравнений
    *count = c;

    if(i < n - 1 || arr[n-1] == x){
        return i; // Возвращаем индекс найденного элемента
    }
    return -1; // Возвращаем -1, если элемент не найден
}

int OAS(int* arr, int n, int x, int* count){
    int last = arr[n-1];
    int i = 0;
    while(x > arr[i]){
        i++;
    }
    arr[n-1] = last;
    *count = i;
    if(i < n - 1 || arr[n - 1] == x){
        return i;
    }
```

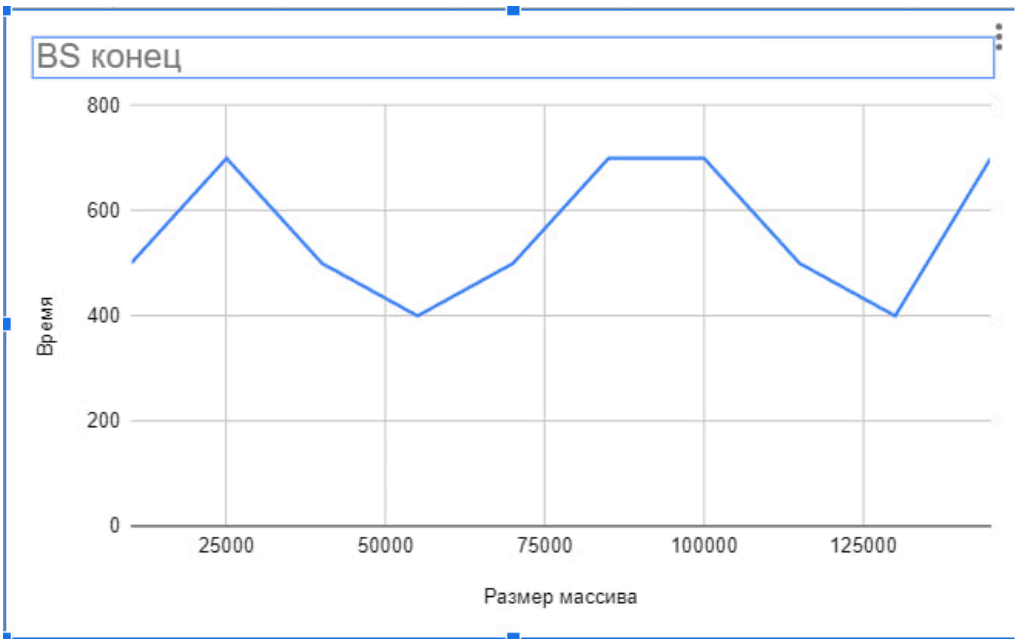
```
    return -1;
}

int BS(int* arr, int n, int x, int* count){
    int p = 0, r = n;
    int q;
    while (p <= r){
        q = (p+r)/2;
        if(arr[q] == x){
            return q;
        }
        else if (arr[q]>x){
            r = q - 1;
        }
        else {
            p = q + 1;
        }
    }
    return -1;
}

void file_output(const char* filename, int* arr, int size){
    ofstream file(filename);
    for (int i = 0; i < size; i++) {
        file << arr[i] << endl;
    }
    file.close();
}

time_t time_count(void (*func)(int*, int, int, int), int* arr, int size, int min,
int max, const char* filename){
    auto start = chrono::steady_clock::now();
    func(arr, size, min, max);
    auto end = chrono::steady_clock::now();
    auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
    file_output(filename, arr, size);
    return duration.count();
}

time_t algo_time(int (*func)(int*, int, int, int*), int* arr, int size, int key,
int* count){
    auto start = chrono::steady_clock::now();
    func(arr, size, key, count);
    auto end = chrono::steady_clock::now();
    auto duration = chrono::duration_cast<chrono::nanoseconds>(end - start);
    return duration.count();
}
```

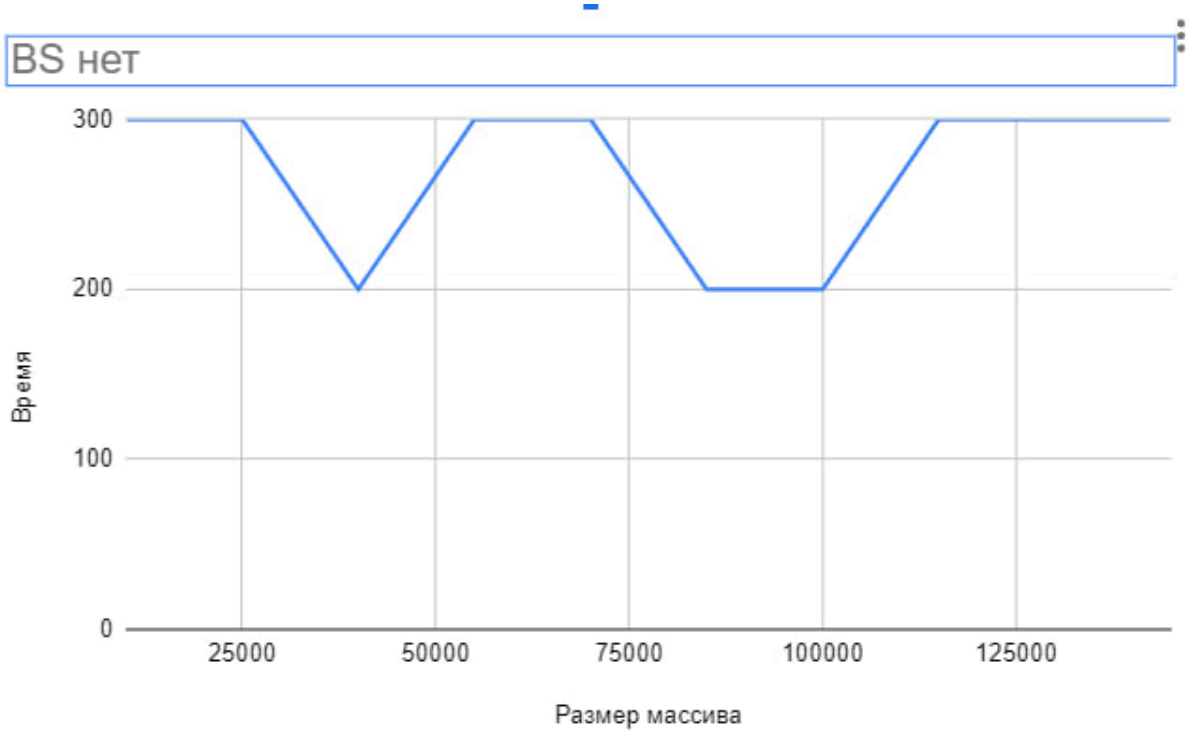
Графики

BS

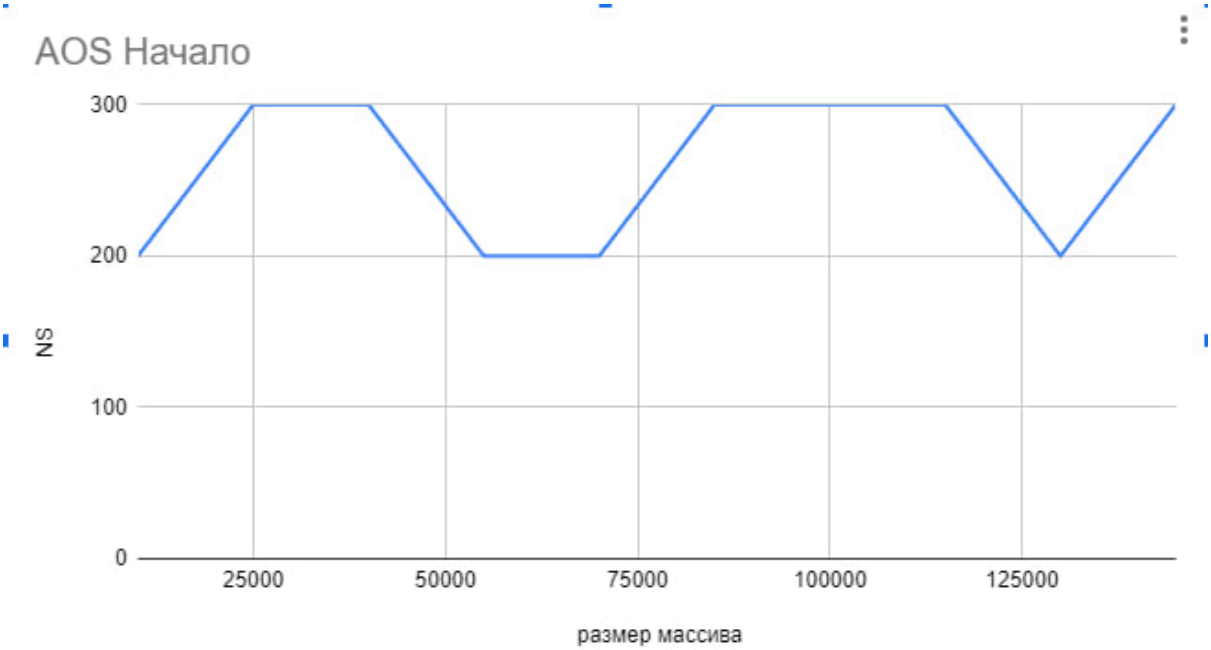

BS начало
600
400
200
0
Время
25000
50000
75000
100000
125000
Размер массива

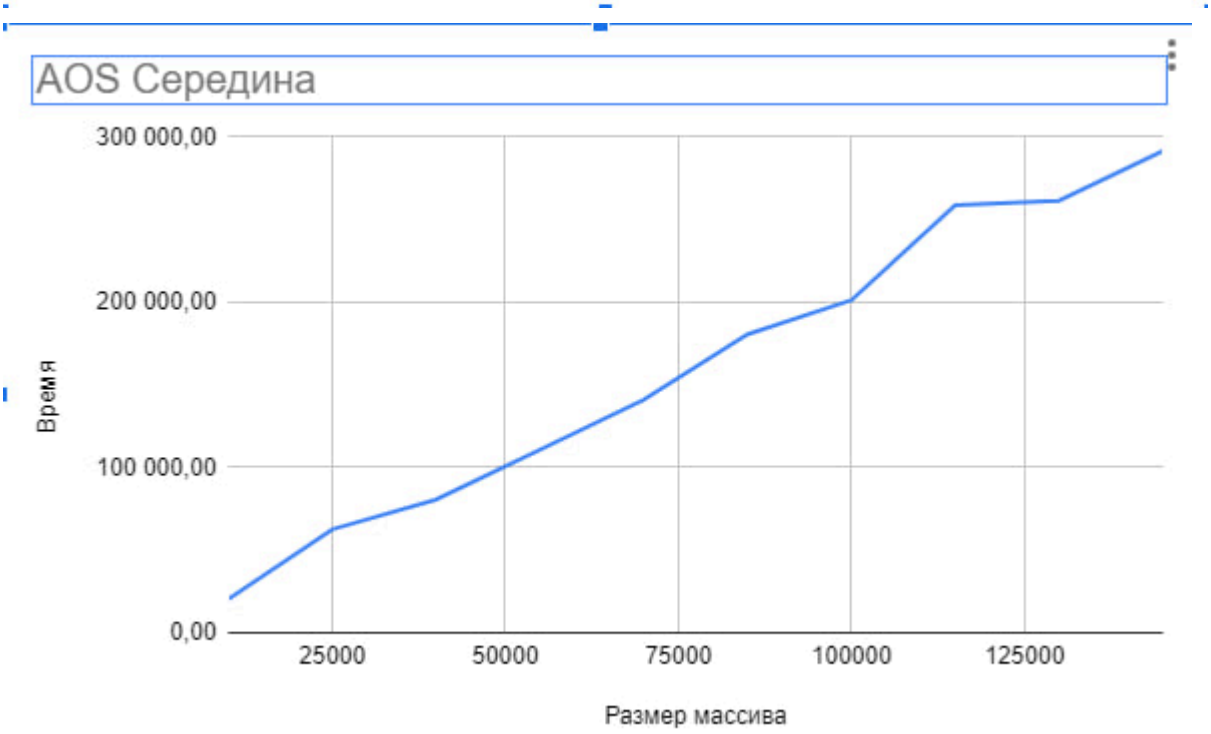

BS середина
400,00
300,00
200,00
100,00
0,00
Время
25000
50000
75000
100000
125000
Размер массива

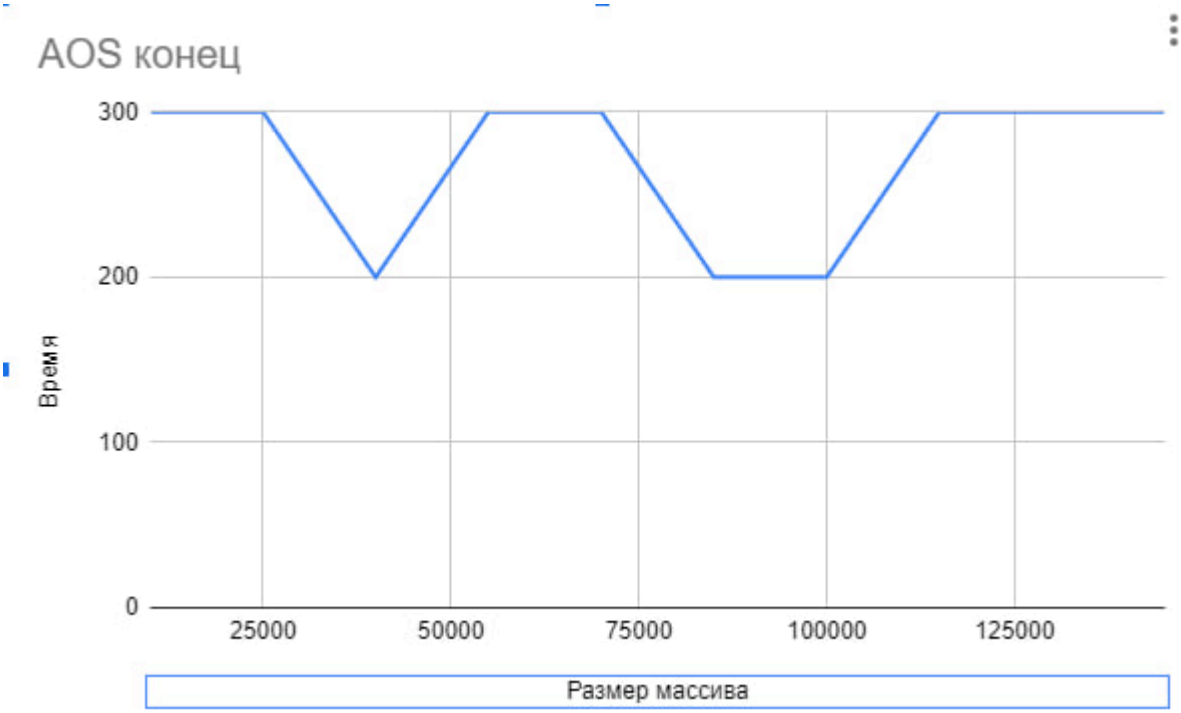

BS конец
800
600
400
200
0
Время
25000
50000
75000
100000
125000
Размер массива

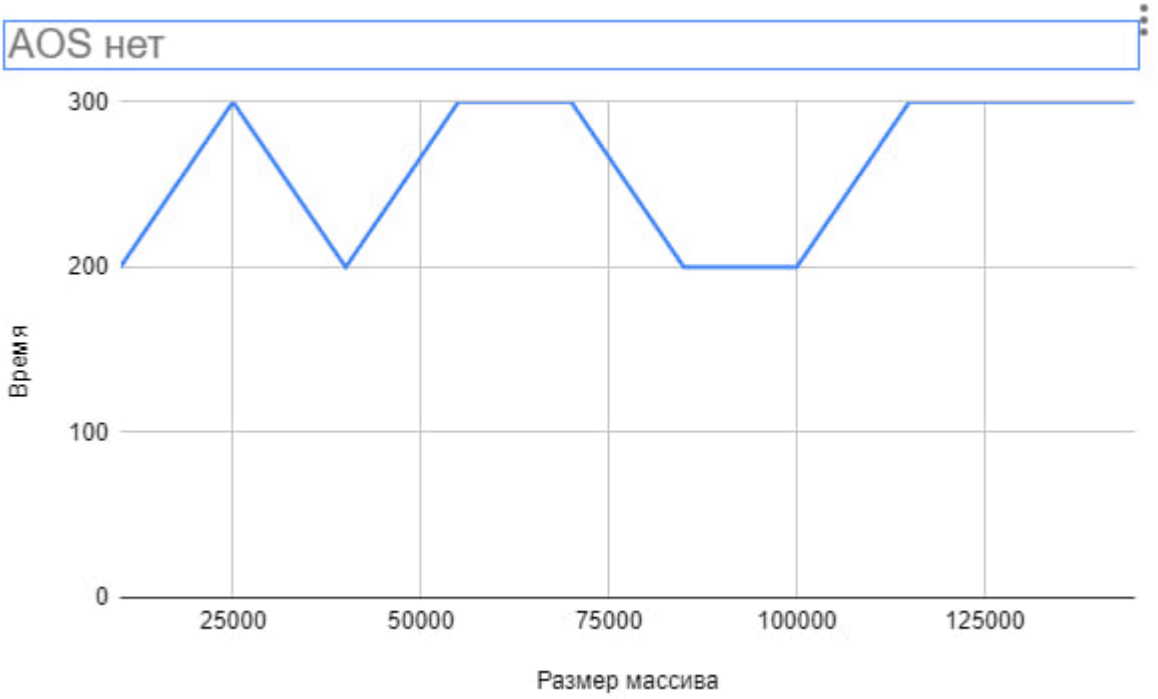

BS нет
300
200
100
0
Время
25000
50000
75000
100000
125000
Размер массива

AOS

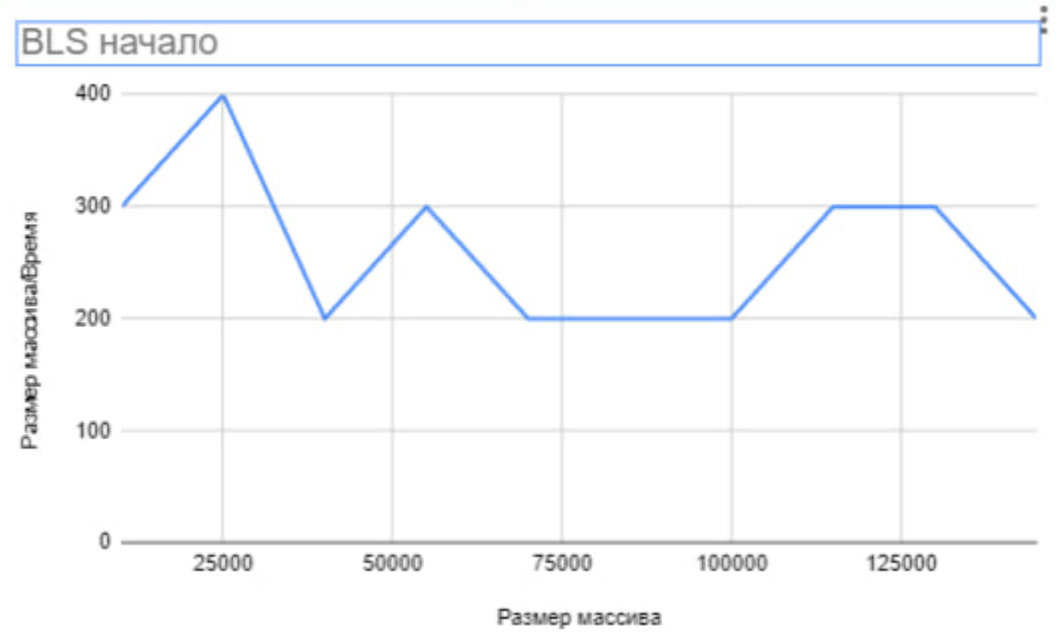

AOS Начало
300
200
100
0
NS
25000
50000
75000
100000
125000
размер массива

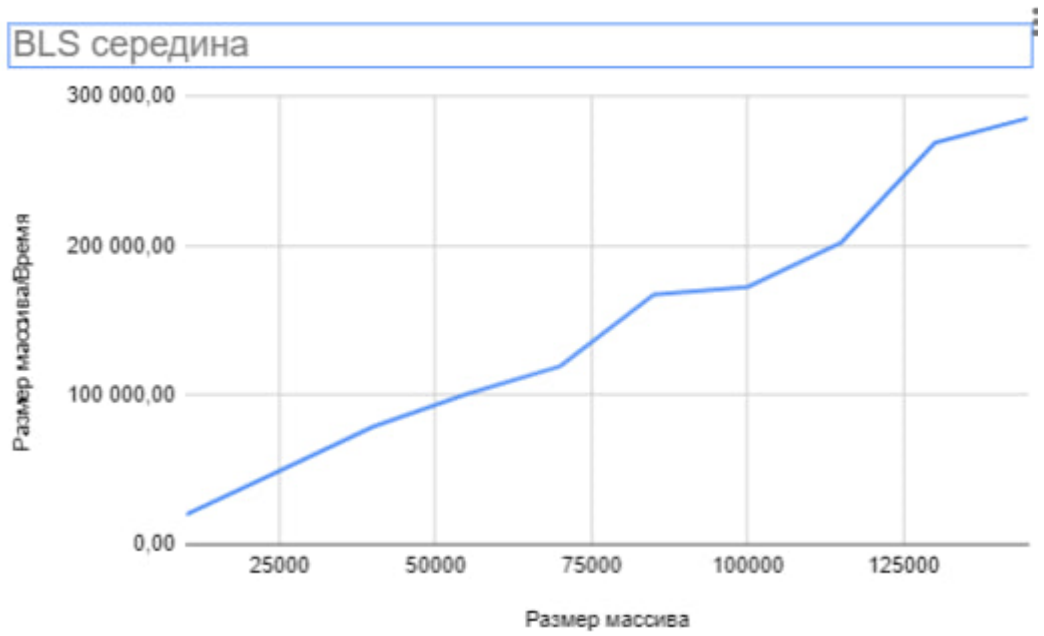

AOS Середина
300 000,00
200 000,00
100 000,00
0,00
Время
25000
50000
75000
100000
125000
Размер массива

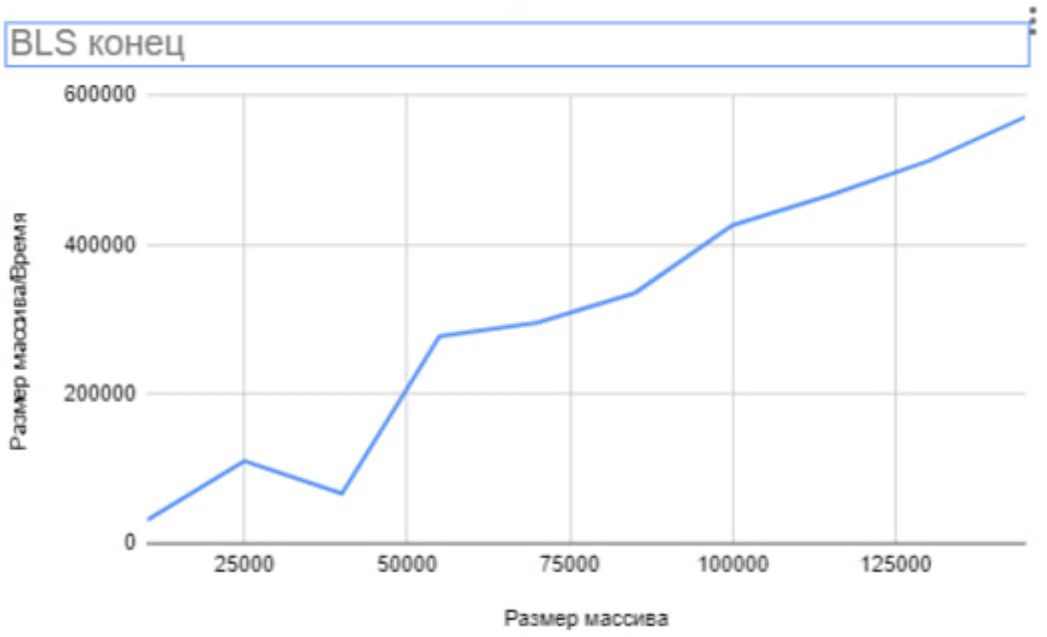

AOS конец
300
200
100
0
Время
25000
50000
75000
100000
125000
Размер массива

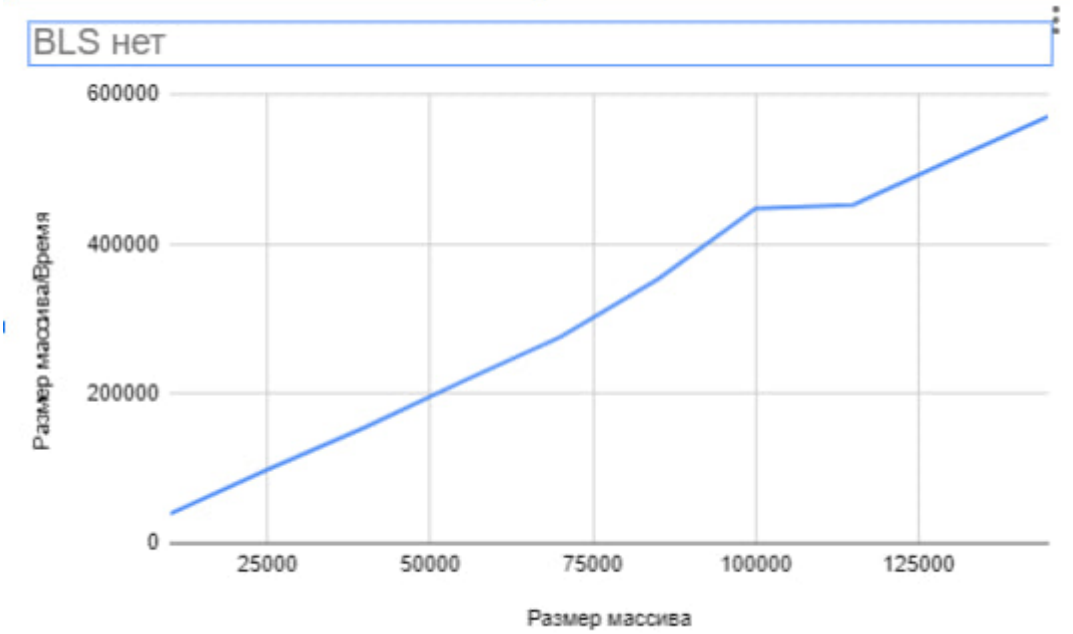

AOS нет
300
200
100
0
Время
25000
50000
75000
100000
125000
Размер массива

BLS

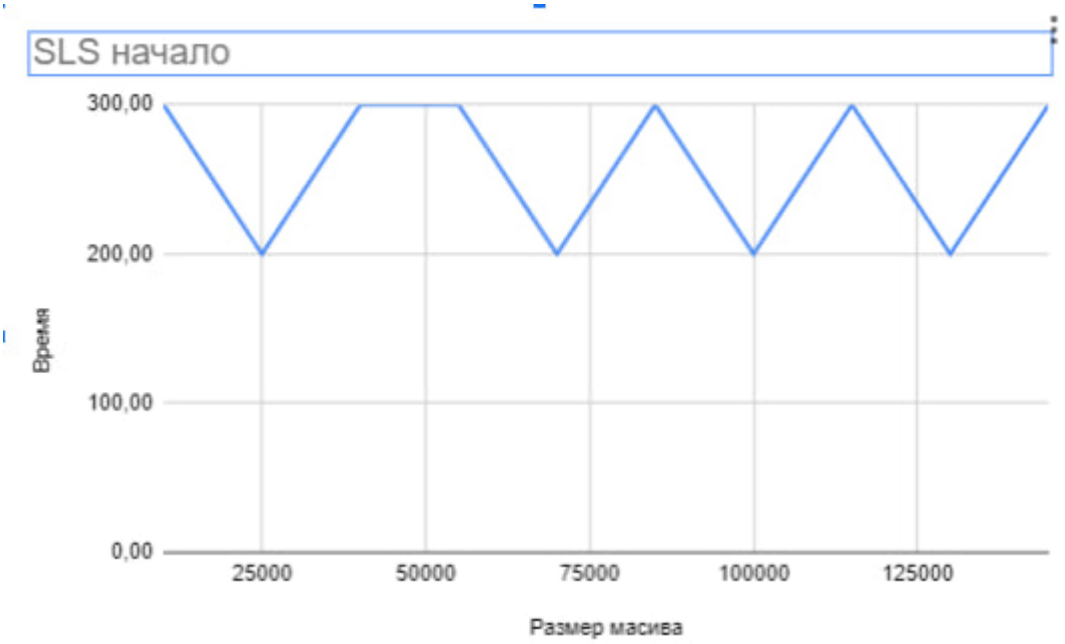

BLS начало
400
300
200
100
0
Размер массива/Время
25000
50000
75000
100000
125000
Размер массива

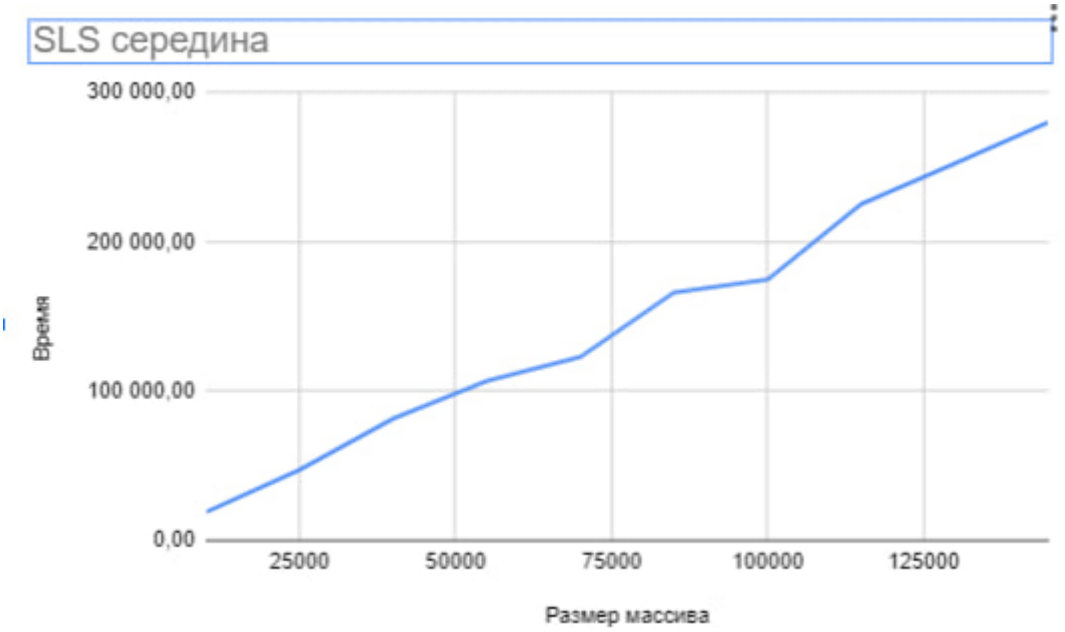

BLS середина
300 000,00
200 000,00
100 000,00
0,00
Размер массива/Время
25000
50000
75000
100000
125000
Размер массива

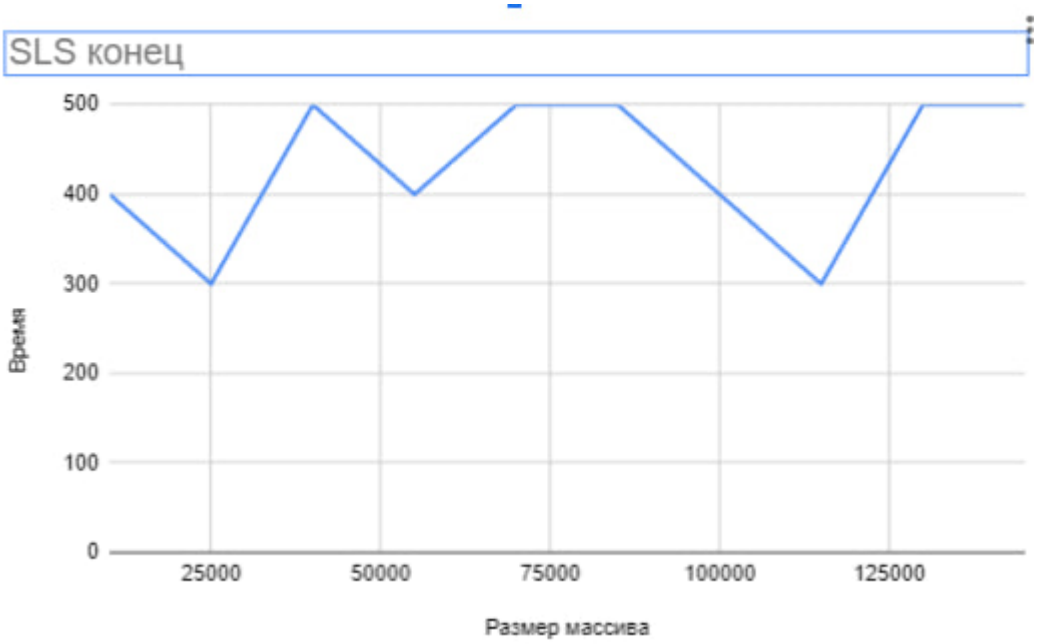

BLS конец
600000
400000
200000
0
Размер массива/Время
25000
50000
75000
100000
125000
Размер массива

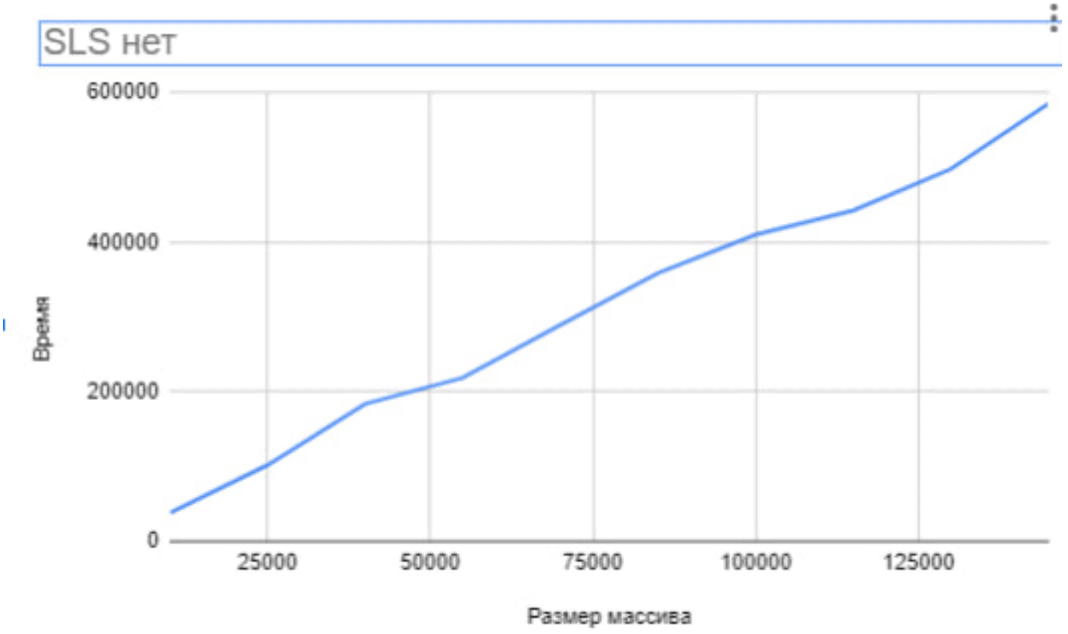

BLS нет
600000
400000
200000
0
Размер массива/Время
25000
50000
75000
100000
125000
Размер массива

SLS


SLS начало
300,00
200,00
100,00
0,00
Время
25000
50000
75000
100000
125000
Размер масива


SLS середина
300 000,00
200 000,00
100 000,00
0,00
Время
25000
50000
75000
100000
125000
Размер массива





Количество сравнений для BLS и SLS

BLS						SLS				
	начало	середина	конец	нет			начало	середина	конец	нет
10000	11	5 001,00	9991	10000		10000	11,00	5 001,00	11	10000
25000	11	12 501,00	24990	25000		25000	11,00	12 501,00	11	25000
40000	11	20 001,00	39990	40000		40000	11,00	20 001,00	11	40000
55000	11	27 501,00	54990	55000		55000	11,00	27 501,00	11	55000
70000	11	35 001,00	69990	70000		70000	11,00	35 001,00	11	70000
85000	11	42 501,00	84990	85000		85000	11,00	42 501,00	11	85000
100000	11	50 001,00	99990	100000		100000	11,00	50 001,00	11	100000
115000	11	57 501,00	114990	115000		115000	11,00	57 501,00	11	115000
130000	11	65 001,00	129990	130000		130000	11,00	65 001,00	11	130000
145000	11	72 501,00	144990	145000		145000	11,00	72 501,00	11	145000

Результаты работы программ и результаты сопоставительного анализа – в виде таблиц, графиков