

Цель работы.

Разработать алгоритм и написать программу на языке C++, которая позволяет: создать целочисленный массив и выполнить различные виды сортировки.

Основные теоретические положения.

Во многих практических ситуациях приходится работать с большим объемом данных. Для этого удобно использовать массивы, которые являются достаточно простой в плане программирования структурой данных.

Использование массива приводит к выделению в памяти набора ячеек под определенным именем. Формально определение массива таково: совокупность однотипных данных, хранящихся в последовательных ячейках памяти и имеющих общее имя. Эти ячейки называются элементами массива. Все элементы массива пронумерованы по порядку, а номер называется Индексом элемента массива. Важно отметить, что все элементы массива имеют один и тот же тип данных. Для обращения к конкретному элементу массива необходимо указать имя массива и далее в квадратных скобках индекс элемента. Массивы могут быть одномерными и многомерными. Здесь мы рассмотрим технологию работы с одномерными массивами.

Для объявления одномерного массива используется следующая форма записи.

ТИП имя массива[размер];

Здесь с помощью элемента записи ТИП объявляется базовый тип массива. Базовый тип определяет тип данных каждого элемента, составляющего массив. Количество элементов, которые будут храниться в массиве, определяется параметром размер. Например, при выполнении приведенной ниже инструкции объявляется int- массив (состоящий из 10 элементов) с именем A.

int A [10];

Доступ к отдельному элементу массива осуществляется с помощью индекса. Индекс описывает позицию элемента внутри массива. В C++ первый элемент массива имеет нулевой индекс. Поскольку массив A содержит 10 элементов, его индексы изменяются от 0 до 9. Чтобы получить доступ к элементу массива по индексу, достаточно указать нужный номер элемента в квадратных скобках. Так, первым элементом массива A является A [0], а последним — A [9].

Существуют разные алгоритмы сортировки массива данных. Было подсчитано, что до четверти времени централизованных компьютеров уделяется сортировке данных. Это потому, что намного легче найти значение в массиве, который был заранее отсортирован. В противном случае поиск немного похожит на поиск иголки в стоге сена. Рассмотрим варианты сортировки.

Сортировка выбором (Selection sort)

Для того, чтобы отсортировать массив в порядке возрастания, следует на каждой итерации найти элемент с наибольшим значением. С ним нужно поменять местами последний элемент. Следующий элемент с наибольшим значением становится уже на предпоследнее место. Так должно происходить, пока элементы, находящиеся на первых местах в массиве, не окажутся в надлежащем порядке.

Пример кода на C++

```
int j = 0;
int tmp = 0;
for (int i = 0; i < a; i++)
{
    j = i;
    for (int k = i; k < a; k++)
    {
        if (A[j] > A[k])
        {
            j = k;
        }
    }
    tmp = A[i];
    A[i] = A[j];
    A[j] = tmp;
}
```

Сортировка 1000 и 10000 элементов массива рис1.

73	73	73	73	73	73	73	96	96	96	96
77	77	77	78	78	78	78	97	97	97	97
82	83	83	83	83	83	83	97	97	97	97
88	88	88	88	88	88	89	97	97	98	98
92	92	92	93	93	93	93	98	98	98	98
96	96	96	96	96	96	96	98	98	98	98
Time: 0.0011807s							Time: 0.100165s			

Рисунок 1. Алгоритм сортировки выбором.

Пузырьковая сортировка (Bubble sort)

При пузырьковой сортировке сравниваются соседние элементы и меняются местами, если следующий элемент меньше предыдущего. Требуется несколько проходов по данным. Во время первого прохода сравниваются первые два элемента в массиве. Если они не в порядке, они меняются местами и затем сравниваются элементы в следующей паре. При том же условии они так же меняются местами. Таким образом сортировка происходит в каждом цикле пока не будет достигнут конец массива.

Пример кода на C++

```
int tmp = 0;
for (int i = 0; i < a; i++)
{
    for (int j = (a - 1); j >= (i + 1); j--)
    {
        if (A[j] < A[j - 1])
        {
            tmp = A[j];
            A[j] = A[j - 1];
            A[j - 1] = tmp;
        }
    }
}
```

Сортировка 1000 и 10000 элементов массива рис2.

82	83	83	83	83	83	97	97	97	97	97
88	88	88	88	88	88	97	97	98	98	98
92	92	92	93	93	93	98	98	98	98	98
96	96	96	96	96	96	98	98	98	98	98
Time: 0.0032368s						Time: 0.305205s				

Рисунок 2. Алгоритм сортировки пузырьком.

Сортировка вставками (Insertion sort)

При сортировке вставками массив разбивается на две области: упорядоченную и неупорядоченную. Изначально весь массив является неупорядоченной областью. При первом проходе первый элемент из неупорядоченной области изымается и помещается в правильном положении

в упорядоченной области. На каждом проходе размер упорядоченной области возрастает на 1, а размер неупорядоченной области сокращается на 1. Основной цикл работает в интервале от 1 до N-1. На j-й итерации элемент [i] вставлен в правильное положение в упорядоченной области. Это сделано путем сдвига всех элементов упорядоченной области, которые больше, чем [i], на одну позицию вправо. [i] вставляется в интервал между теми элементами, которые меньше [i], и теми, которые больше [i].

Пример кода C++

```
int key = 0;
int i = 0;
for (int j = 1; j < a; j++)
{
    key = A[j];
    i = j - 1;
    while (i >= 0 && A[i] > key)
    {
        A[i + 1] = A[i];
        i = i - 1; A[i + 1] = key;
    }
}
```

Сортировка 1000 и 10000 элементов массива рис3.

82	83	83	83	83	83	97	97	97	97	97
88	88	88	88	88	88	97	97	98	98	98
92	92	92	93	93	93	98	98	98	98	98
96	96	96	96	96	96	98	98	98	98	98
Time: 0.0009715s						Time: 0.0973716s				

Рисунок 3. Алгоритм сортировки вставками.

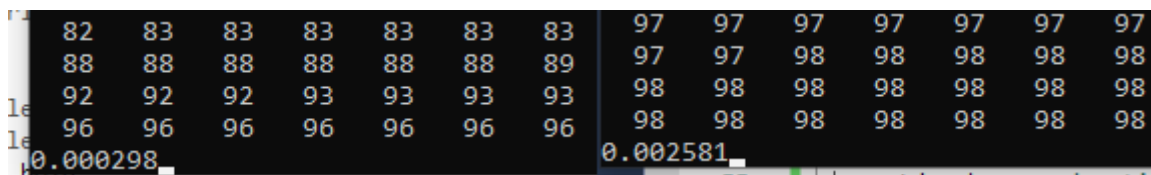
Быстрая сортировка (Quick sort)

Быстрая сортировка использует алгоритм "разделяй и властвуй". Она начинается с разбиения исходного массива на две области. Эти части находятся слева и справа от отмеченного элемента, называемого опорным. В конце процесса одна часть будет содержать элементы меньшие, чем опорный, а другая часть будет содержать элементы больше опорного.

Пример кода C++

```
// Функция быстрой сортировки
void quickSort(int *numbers, int left, int right)
{
    int pivot; // опорный элемент
    int l_hold = left; //левая граница
    int r_hold = right; // правая граница
    pivot = numbers[left];
    while (left < right) // пока границы не сомкнутся
    {
        while ((numbers[right] >= pivot) && (left < right))
            right--; // сдвигаем правую границу пока элемент [right] больше
        [pivot]
        if (left != right) // если границы не сомкнулись
        {
            numbers[left] = numbers[right]; // перемещаем элемент [right] на
место разрезающего
            left++; // сдвигаем левую границу вправо
        }
        while ((numbers[left] <= pivot) && (left < right))
            left++; // сдвигаем левую границу пока элемент [left] меньше [pivot]
        if (left != right) // если границы не сомкнулись
        {
            numbers[right] = numbers[left]; // перемещаем элемент [left] на место
[right]
            right--; // сдвигаем правую границу вправо
        }
    }
    numbers[left] = pivot; // ставим опорный элемент на место
    pivot = left;
    left = l_hold;
    right = r_hold;
    if (left < right) // Рекурсивно вызываем сортировку для левой и правой части
массива
    {
        quickSort(numbers, left, pivot - 1);
        if (right > pivot)
            quickSort(numbers, pivot + 1, right);
    }
}
```

Сортировка 1000 и 10000 элементов массива рис4.



82	83	83	83	83	83	83	97	97	97	97	97	97	97
88	88	88	88	88	88	89	97	97	98	98	98	98	98
92	92	92	93	93	93	93	98	98	98	98	98	98	98
96	96	96	96	96	96	96	98	98	98	98	98	98	98
0.000298							0.002581						

Рисунок 4. Алгоритм быстрой сортировки.

Постановка задачи.

Необходимо написать программу, которая:

- 1) Создает целочисленный массив размерности $N = 100$. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.
- 2) Отсортировать заданный в пункте 1 элементы массива [...] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.
- 3) Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.
- 4) Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения. Выводит индексы всех элементов, которые равны этому значению, и их количество.
- 5) Выводит количество элементов в отсортированном массиве, которые меньше числа a , которое инициализируется пользователем.
- 6) Выводит количество элементов в отсортированном массиве, которые больше числа b , которое инициализируется пользователем.
- 7) Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором. (*)
- 8) Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

Должна присутствовать возможность запуска каждого пункта многократно.

Выполнение работы.

Создаём целочисленный массив размерности $N = 100$. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

```

#include <iostream>
using namespace std;
int main()
{
    const int a = 100;
    short A[a];
    srand(time(0));
    for (int i = 0; i < a; i++)
        A[i] = rand() % (199 - 1) - 99;
    for(int i = 0; i < a; i++)
        cout <<"A["<< i <<"]="<< A[i] <<"\n";
    return 0;
}

```

Работа программы представлена на рис 5.

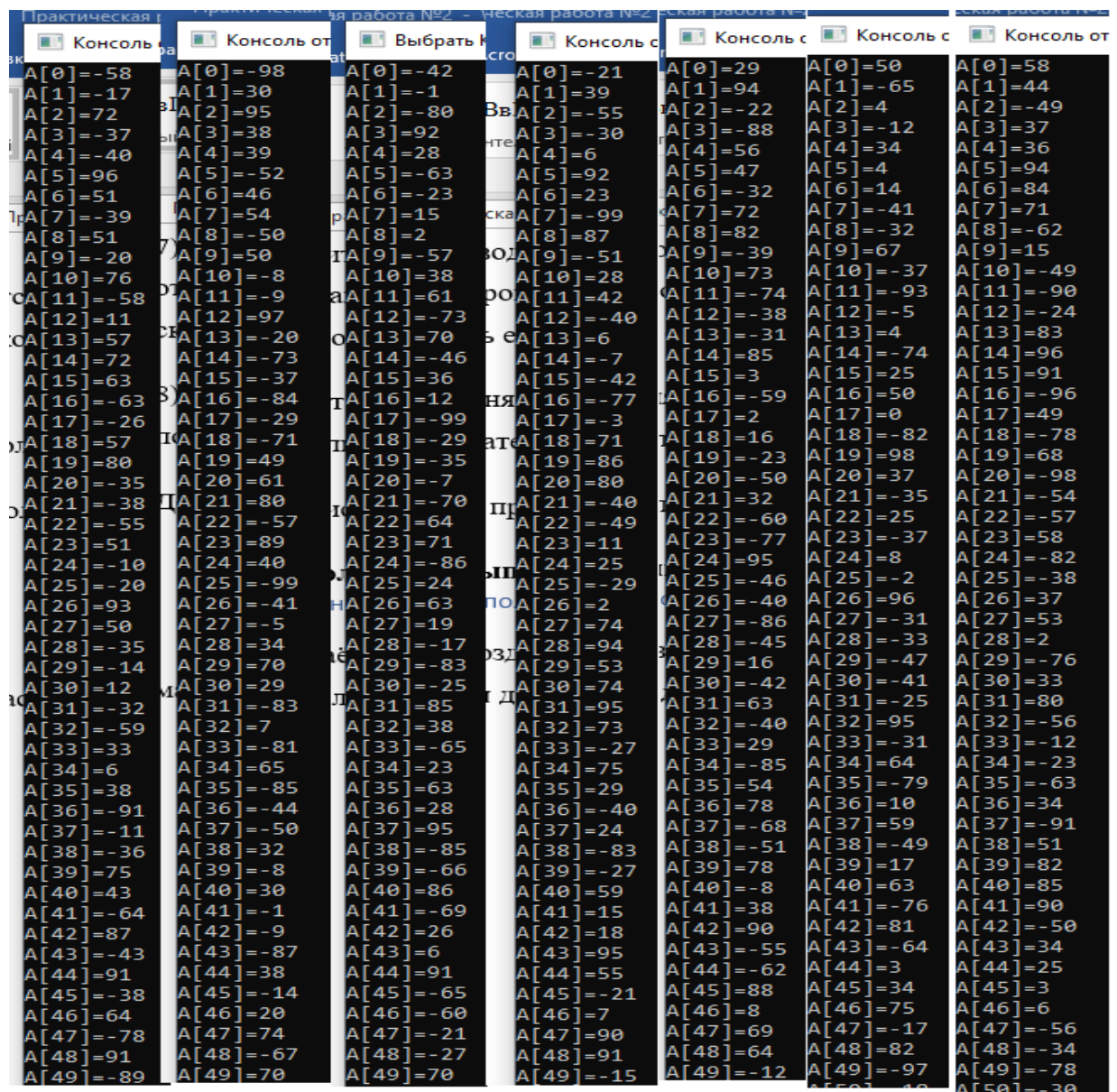


Рисунок 5. Работа программы заполнения массива.

Сортируем элементы массива сортировкой вставками (insert sort) сортировкой (от меньшего к большему). Также определяем время, затраченное на сортировку, используя библиотеку chrono.

Работа программы сортировки представлена на рис.6

Рисунок 6. Работа программы сортировки.

Ищем максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в неотсортированном массиве рис7.

Рисунок 7. Работа программы поиска максимального и минимального элемента в неотсортированном массиве данных.

Ищем максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве рис8.

Min: -97	Min: -98
Max: 95	Max: 95
Time sorted data array: 1.1e-06s	Time sorted data array: 1.1e-06s
Min: -98	Min: -74
Max: 97	Max: 96
Time sorted data array: 1.2e-06s	Time sorted data array: 5e-07s

Рисунок 8. Работа программы поиска максимального и минимального элемента в отсортированном массиве данных.

Выводим среднее значение максимального и минимального значения. Выводим индексы всех элементов, которые равны этому значению, и их количество. Для того чтобы показать работу сократим диапазон значений элементов -9 до 9 Рис 9.

<pre> 0 Mean A[40] number A[41] number A[42] number A[43] number A[44] number A[45] number number of array elements = 0= 6 -9 -9 -9 -9 -8 -8 -8 -7 -7 -7 -7 -7 -7 -6 -6 -6 -5 -5 -4 -4 -4 -3 -3 -3 -3 -3 -2 -2 -2 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 4 4 4 4 4 4 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 8 8 8 8 8 8 8 8 8 Time: 1.2e-06s Create data array: 1 </pre>	<pre> Min: -9 Max: 8 Time sorted data array: 1.1e-06s </pre>
---	--

Рисунок 9. Работа программы.

Выводим количество элементов в отсортированном массиве, которые меньше числа a , которое инициализируется пользователем. Вернем диапазон значений элементов от -99 до 99. Рис.10. для примера выбраны значения элементов 22,2,80.

<pre> Enter the number 22 The number of elements is less 22= 62 </pre>	<pre> Enter the number 2 The number of elements is less 2= 53 </pre>	<pre> Enter the number 80 The number of elements is less 80= 90 </pre>
<pre> -96 -91 -91 -90 -90 -89 -88 -84 -84 -83 -79 -76 -73 -73 -73 -72 -72 -72 -71 -71 -69 -68 -68 -67 -64 -64 -63 -61 -60 -58 -55 -47 -46 -46 -45 -42 -39 -39 -34 -25 -25 -21 -15 -15 -14 -13 -12 -12 -8 -8 -8 -7 -1 2 2 3 6 17 19 19 20 21 22 23 24 25 26 26 31 33 35 35 36 38 40 40 43 45 47 49 56 56 61 62 63 65 70 73 73 75 80 86 87 88 88 92 93 95 96 98 Time: 1.3e-06s </pre>		

Рисунок 10. Работа программы.

Выводим количество элементов в отсортированном массиве, которые больше числа b , которое инициализируется пользователем. Рис.11 в качестве примера выбраны элементы 22,2,80.

```
Enter the number 22
The number of elements is greater 22= 37
Enter the number 2
The number of elements is greater 2= 45
Enter the number 80
The number of elements is greater 80= 9
-96 -91 -91 -90 -90 -89 -88 -84 -84 -83 -79 -76 -73 -73 -73 -72 -72 -72 -71 -71 -69 -68 -68 -67
-64 -64 -63 -61 -60 -58 -55 -47 -46 -46 -45 -42 -39 -39 -34 -25 -25 -21 -15 -15 -14 -13 -12 -12
-8 -8 -8 -7 -1 2 2 3 6 17 19 19 20 21 22 23 24 25 26 26 31 33 35 35
36 38 40 40 43 45 47 49 56 56 61 62 63 65 70 73 73 75 80 86 87 88 88 92
93 95 96 98
Time: 1.3e-06s
```

Рисунок 11. Работа программы.

Выводим информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором. Рис.12

```
Enter the number 11
Yes
Time sorted data array: 6e-07s
Search by brute force 11 coincidences = 2
Time sorted data array: 4e-07s
Difference:Brute force - binary search -2e-07 s

Enter the number 2
Yes
Time sorted data array: 4e-07s
Search by brute force 2 coincidences = 1
Time sorted data array: 5e-07s
Difference:Brute force - binary search 1e-07 s

Enter the number 4
The element not found!
Time sorted data array: 5e-07s
Search by brute force 4 coincidences = 0
Time sorted data array: 4e-07s
Difference:Brute force - binary search -1e-07 s

Enter the number 5
Yes
Time sorted data array: 5e-07s
Search by brute force 5 coincidences = 2
Time sorted data array: 5e-07s
Difference:Brute force - binary search 0 s
```

Рисунок 12. Работа программы

Меняет местами элементы массива, индексы которых вводит пользователь. Для того что чтобы сократить рисунок с выводом результата уменьшим массив данных до 15 элементов. Рис 13

A[0]=-92 WAS	2 11	13 1
A[1]=-61 WAS	A[0]=-92 Became	A[0]=-92 Became
A[2]=-60 WAS	A[1]=-61 Became	A[1]=92 Became
A[3]=-48 WAS	A[2]=45 Became	A[2]=45 Became
A[4]=-48 WAS	A[3]=-48 Became	A[3]=-48 Became
A[5]=-45 WAS	A[4]=-48 Became	A[4]=-48 Became
A[6]=-12 WAS	A[5]=-45 Became	A[5]=-45 Became
A[7]=2 WAS	A[6]=-12 Became	A[6]=-12 Became
A[8]=13 WAS	A[7]=2 Became	A[7]=2 Became
A[9]=20 WAS	A[8]=13 Became	A[8]=13 Became
A[10]=26 WAS	A[9]=20 Became	A[9]=20 Became
A[11]=45 WAS	A[10]=26 Became	A[10]=26 Became
A[12]=64 WAS	A[11]=-60 Became	A[11]=-60 Became
A[13]=92 WAS	A[12]=64 Became	A[12]=64 Became
A[14]=93 WAS	A[13]=92 Became	A[13]=-61 Became
Write 0 < A & B < 99 :	A[14]=93 Became	A[14]=93 Became
	Time sorted data array: 4e-07s	Time sorted data array: 4e-07s

Рисунок 13. Работа программы смены элементов.

Выводы.

В данной практической были рассмотрены (линейные) одномерные массивы вывод ввод элементов, сортировка массивов и работа с элементами массива. Это одно из основополагающих понятий в программировании так как массив представляет собой набор однотипных данных, а для работы любой программы, необходимы данные (желательно отсортированные), которые она будет обрабатывать. Есть программисты, которые всё рабочее время проводят в изучении и внедрении алгоритмов сортировки. Это потому, что подавляющее большинство программ в бизнесе включает в себя управление базами данных. Люди ищут информацию в базах данных всё время. Это означает, что основы алгоритмов работы с массивами данных и их сортировкой очень востребованы.

ПРИЛОЖЕНИЕ А

ПОЛНЫЙ КОД ПРОГРАММЫ

```
// Практическая работа №2
#include <iostream>
#include <iomanip>
#include <chrono>
using namespace std;
int main()
```

```

{
    const int a = 100;
    short A[a];
    M1:
    auto start = std::chrono::system_clock::now();
    int q, max = A[0], min = A[0], ser = 0, o = 0, n;
    auto srand(time(0));
    for (int i = 0; i < a; i++)
        A[i] = rand() % (199 - 1) - 99;
    auto end = std::chrono::system_clock::now();
    for (int i = 0; i < a; i++)
        cout << "A[" << i << "]=" << A[i] << "\n";
    std::chrono::duration<double> diff = end - start;
    cout << "\n" << "Time: " << diff.count() << "s";
    cout << "\n" << "Create data array: 1 " << "\n" << "Start sorting: 2"
    << "\n" << "Start searching for max and min element: 3" << "\n";
    cin >> q;
    cin.sync();
    switch (q)
    {
        case 1: goto M1;
        case 2: goto M2;
        case 3: goto M3;
        default: cout << "ERROR \n";
    }
    M2:
    start = chrono::system_clock::now();
    for (int i = 1; i < a; i++)
    {
        int x = A[i];
        int j = i;
        while ((j > 0) && (x < A[j - 1]))
        {
            A[j] = A[j - 1];
            j--;
        }
        A[j] = x;
    }
    end = chrono::system_clock::now();
    for (int i = 0; i < a; i++)
    {
        cout << setw(5) << left << A[i];

    }
    chrono::duration<double> set = end - start;
    cout << "\n" << "Time: " << set.count() << "s";
    cout << "\n" << "Create data array: 1 " << "\n" << "Start
    sorting: 2" << "\n" << "Start searching for max and min element: 3" <<
    "\n Mean: 4" << "\n Item: 5" << "\n";
    cin >> q;
    cin.sync();
}

```

```

        switch (q)
        {
            case 1: goto M1;
            case 2: goto M2;
            case 3: goto M3;
            case 4: goto M4;
            case 5: goto M5;
            default: cout << "ERROR \n";
        }
        M3:
            start = chrono::system_clock::now();
        for (int i = 0; i < a; i++)
            if (A[i] > max)
                max = A[i];
        for (int i = 0; i < a; i++)
            if (A[i] < min)
                min = A[i];
            end = chrono::system_clock::now();
            cout << "Min: " << min << "\n" << "Max: " << max << "\n";
            std::chrono::duration<double> tide = end - start;
            cout << "\n" << "Time sorted data array: " << tide.count()
<< "s";
            cout << "\n" << "\n Create data array: 1 " << "\n Start
sorting: 2" << "\n Start searching for max and min element: 3" << "\n
Mean: 4" << "\n";
            cin >> q;
            cin.sync();
            switch (q)
            {
                case 1: goto M1;
                case 2: goto M2;
                case 3: goto M3;
                case 4: goto M4;
                default: cout << "ERROR \n";
            }
            M4:
                ser = (max + min) / 2;
                cout << "\n" << ser << " Mean" << "\n";
                for (int i = 0; i < a; i++)
                {
                    if (A[i] == ser)
                    {
                        cout << "A[" << i << "]" << " number " << "\n";
                        ++o;
                    }
                }
                cout << "number of array elements = " << ser << "= " << o;
                cout << "\n" << "\n Create data array: 1 " << "\n Start
sorting: 2" << "\n Start searching for max and min element: 3" << "\n
Mean: 4" << "\n Next: 5" << "\n";
                cin >> q;

```

```

cin.sync();
switch (q)
{
case 1: goto M1;
case 2: goto M2;
case 3: goto M3;
case 4: goto M4;
case 5: goto M5;
default: cout << "ERROR \n";
}
M5:
int e = 0;
cout << "\n Enter the number ";
cin >> n;
cin.sync();
for (int i = 0; i < a; i++)
{
    if (A[i] < n)
    {
        ++e;
    }
}
cout << "The number of elements is less " << n << "=
" << e;

cout << "\n" << "\n Create data array: 1 " << "\n
Start sorting: 2" << "\n Start searching for max and min element: 3"
<< "\n Mean: 4" << "\n Back: 5" << "\n Next: 6" << "\n";
cin >> q;
cin.sync();
switch (q)
{
case 1: goto M1;
case 2: goto M2;
case 3: goto M3;
case 4: goto M4;
case 5: goto M5;
case 6: goto M6;
default: cout << "ERROR \n";
}
M6:
int p = 0;
cout << "\n Enter the number ";
cin >> n;
cin.sync();
for (int i = 0; i < a; i++)
{
    if (A[i] > n)
    {
        ++p;
    }
}

```

```

        cout << "The number of elements is greater " << n << " = " << p;
        cout << "\n" << "\n Create data array: 1 " << "\n Start
sorting: 2" << "\n Start searching for max and min element: 3" << "\n
Mean: 4" << "\n Item: 5" << "\n Repeat: 6" << "\n Next: 7" << "\n";
        cin >> q;
        cin.sync();
        switch (q)
        {
        case 1: goto M1;
        case 2: goto M2;
        case 3: goto M3;
        case 4: goto M4;
        case 5: goto M5;
        case 6: goto M6;
        case 7: goto M7;
        default:  cout << "ERROR \n";
        }
M7:
        cout << "\n Enter the number ";
        cin >> n;
        cin.sync();
        int h = 0;
        int s = -1;
        int l = a;
        start = chrono::system_clock::now();
        while (h <= l)
        {
                int mid = (h + l) / 2;
                if (n == A[mid])
                {
                        s = mid;
                        break;
                }
                if (n < A[mid])
                        l = mid - 1;
                else
                        h = mid + 1;
        }
        end = chrono::system_clock::now();
        if (s == -1)
                cout << "The element not found!\n";
        else
                cout << "Yes\n";
        std::chrono::duration<double> xz = end - start;
        cout << "\n" << "Time sorted data array: " << xz.count() << "s";
        start = chrono::system_clock::now();
        int r = 0;
        for (int i = 0; i < a; i++)
        {
                if (A[i] == n)
                {

```

```

        ++r;
    }
}
end = chrono::system_clock::now();
cout << "\n Search by brute force " << n << " coincidences = " << r;
std::chrono::duration<double> acdc = end - start;
cout << "\n" << "Time sorted data array: " << acdc.count() << "s";
cout << "\n Difference:" << "Brute force -" << " binary search " <<
acdc.count() - xz.count() << " s";
cout << "\n" << "\n Create data array: 1 " << "\n Start sorting: 2" <<
"\n Start searching for max and min element: 3" << "\n Mean: 4" << "\n
Item: 5" << "\n Item: 6" << "\n Repeat: 7"<< "\n Next: 8" << "\n";
cin >> q;
cin.sync();
switch (q)
{
case 1: goto M1;
case 2: goto M2;
case 3: goto M3;
case 4: goto M4;
case 5: goto M5;
case 6: goto M6;
case 7: goto M7;
case 8: goto M8;
default: cout << "ERROR \n";
}

M8:
for (int i = 0; i < a; i++)
cout << "A[" << i << "]= " << A[i] << " WAS" << "\n";
int w;
cout << "Write 0 < A & B < 99 : \n";
cin >> n >> w;
cin.sync();
start = chrono::system_clock::now();
int tmp;
tmp = A[n];
A[n] = A[w];
A[w] = tmp;
end = chrono::system_clock::now();
for (int i = 0; i < a; i++)
cout << "A[" << i << "]= " << A[i] << " Became" << "\n";
std::chrono::duration<double> yo = end - start;
cout << "\n" << "Time sorted data array: " << yo.count() << "s";
cout << "\n" << "\n Create data array: 1 " << "\n Start sorting: 2" <<
"\n Start searching for max and min element: 3" << "\n Mean: 4" << "\n
Item: 5 " << "\n Item: 6" << "\n Item: 7" << "\n Repeat: 8" <<
"\n";

cin >> q;
cin.sync();
switch (q)
{

```



```
        case 1: goto M1;
        case 2: goto M2;
        case 3: goto M3;
        case 4: goto M4;
        case 5: goto M5;
        case 6: goto M6;
        case 7: goto M7;
        case 8: goto M8;
        default:  cout << "\n the project is over \n";
    }
    return 0;
}
```