МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №1.1

Тема:

Оценка вычислительной сложности алгоритма

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент:	Враженко Д.О.
Группа:	ИКБО-10-23
Вариант:	7

Москва – 2024

ЦЕЛЬ РАБОТЫ

Приобретение практических навыков:

- эмпирическому определению вычислительной сложности алгоритмов на теоретическом и практическом уровнях;
- выбору эффективного алгоритма решения вычислительной задачи из нескольких.

ХОД РАБОТЫ

1. Задание 1

Выбрать эффективный алгоритм вычислительной задачи из двух предложенных, используя теоретическую и практическую оценку вычислительной сложности каждого из алгоритмов, а также его ёмкостную сложность.

1.1 Формулировка задачи:

Даны два алгоритма решения следующей задачи: дан массив размера от 10 до 100 элементов, удалить из массива все заданные числа. Реализовать два данных алгоритма в функциях.

1.2 Математическая модель решения задачи:

а) Устное описание и описание блок-схемой

Алгоритм 1:

Проходим по массиву, если замечаем элемент, который надо удалить, то удаляем его и смещаем все последующие элементы на 1 позицию влево. После сдвига уменьшаем значение переменной, хранящей длину массива, на 1.

На рис. 1 представлена блок-схема алгоритма 1.

Алгоритм 2:

Проходим по массиву, присваивая j-му элементу значение i-го элемента. Если i-ый элемент неравен удаляемому, то увеличиваем j на 1.

На рис. 2 представлена блок-схема алгоритма 2.

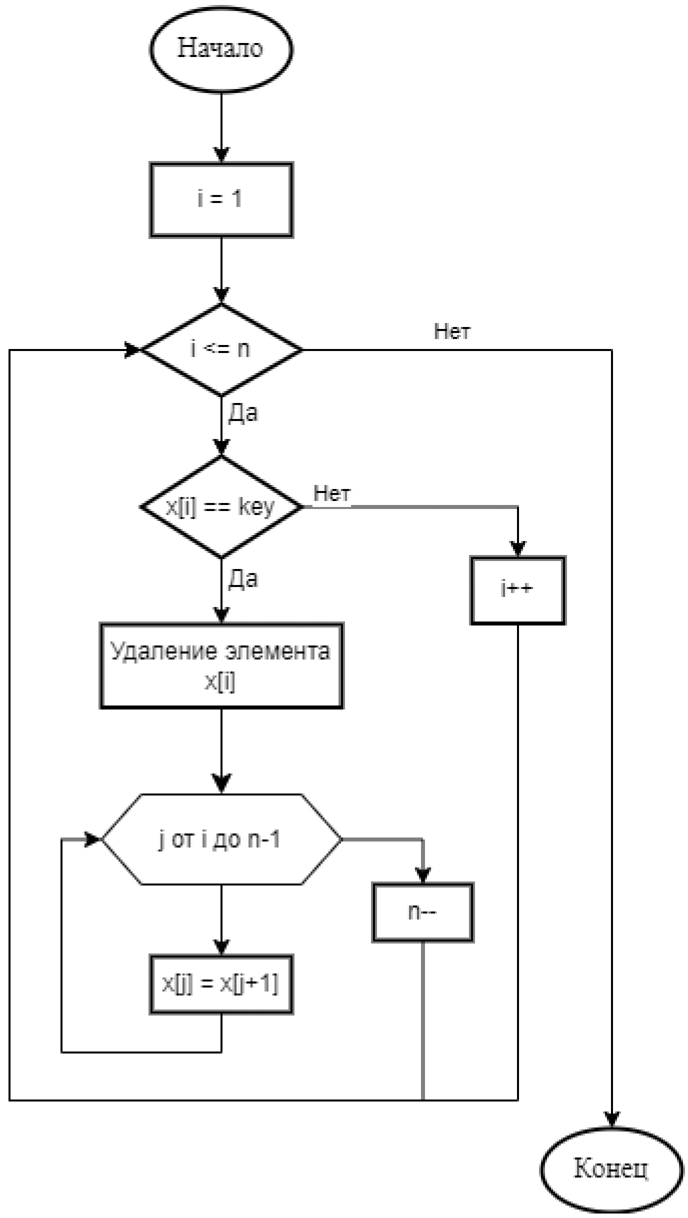

Рисунок 1 - Блок-схема алгоритма 1

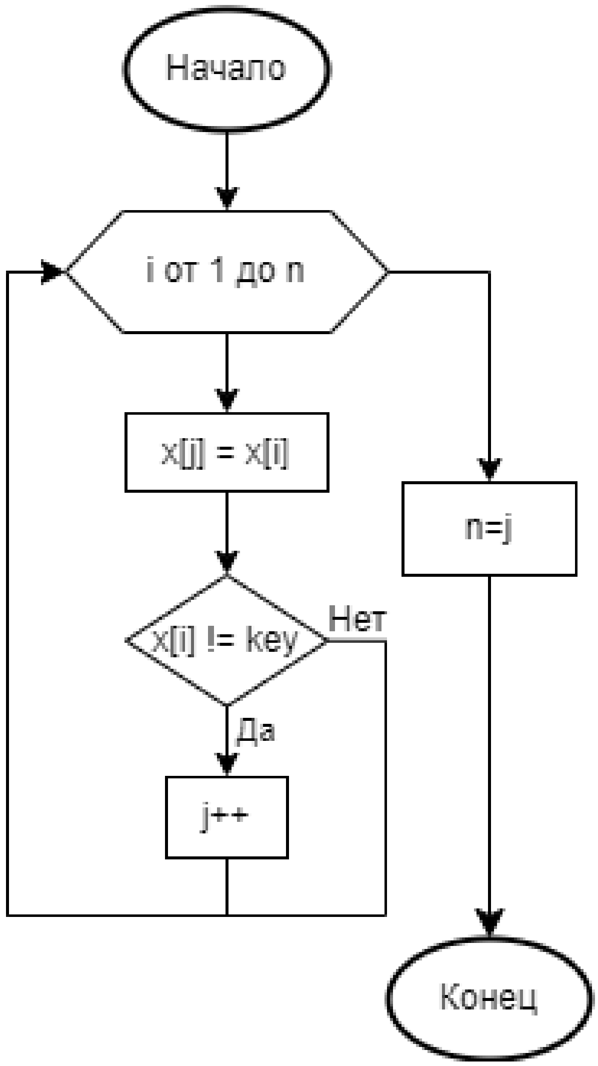

Рисунок 2 - Блок-схема алгоритма 2

b)

Алгоритм 1:

Внешним циклом является цикл while (i<=n).

Инвариант: i >= 1 && i <= n+1. Число n – конечное количество элементов в массиве, значит n >= 0, поэтому n+1 >= 1 при допустимых n, значит при инициализации предикат будет истинен. Цикл выполняется при условии i <= n. После каждого выполнения тела цикла либо i увеличивается на 1, либо n уменьшается на 1, следовательно условие i >= 1 выполняется после произвольного выполнения тела цикла, а также условие i <= n+1 не нарушится, так как цикл выполняется только при выполнении условия i <= n, а значит после произвольного выполнения тела цикла условие i <= n+1 будет истинно. При завершении цикла из-за изменения значений i и n будет выполняться равенство i = n+1, поэтому условия i >= 1 && i <= n+1 также не нарушатся. Также из-за того, что значение i не убывает и значение n не возрастает с каждым выполнением тела цикла уменьшается, то условие i <= n выполнится и цикл завершиться. Все вышесказанное доказывает корректность цикла.

Алгоритм 2:

Внешним циклом является цикл for i←1 to n do.

Инвариант: j - 1 <= i && i <= n + 1. Изначально переменная-счётчик i инициализируется значением 1, как и переменная j, так что условие j - 1 <= i выполняется при инициализации. Число n – конечное количество элементов в массиве, значит n >= 0. Поэтому i <= n + 1 при допустимых n, значит при инициализации предикат будет истинен. Цикл выполняется при условии i <= n. После каждого выполнения тела цикла либо i увеличивается на 1 условие i <= n+1 не нарушится, так как цикл выполняется только при выполнении условия i <= n, а значит после произвольного выполнения тела цикла условие i <= n+1 будет истинно. Переменная j увеличивается, только если текущий элемент по индексу i не равен заданному key, так что к концу итерации цикла j может быть больше i максимум на 1. При завершении цикла i = n + 1, так что условие i <= n + 1 выполняется. Условие j - 1 <= i также выполняется, так как наибольшее значение пер. j по завершении цикла равно n + 1.

c)

Алгоритм 1:

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
i←0	1	1
while(i<=n) do	n+1	n+1
if x[i] = key then	n	n
//удаление		

for j←i to n-1 do	0	n(n+1)/2
x[j]←x[j + 1]	0	n(n-1)/2
od		
n←n-1	0	n
else	0	0
i←i+1	n	0
endif		
od		

Для лучшего случая теоретическая сложность T(n) = 1+(n+1)+n+n = 3n+2 (линейная).

Для худшего случая теоретическая сложность T(n) = = 1+(n+1)+n+(n(n+1)/2)+(n(n-1)/2)+n = n^2+3n+2 (квадратичная).

Для среднего случая должно удовлетворяться условие: $T(n)_{лучший} < T(n)_{средний} < T(n)_{худший}$.

Алгоритм 2:

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
j←1	1	1
for i←1 to n do	n+1	n+1
x[j]←x[i]	n	n
if x[i] != key then	n	n
j++	0	n
endif		
od		
n←j	1	1

Для лучшего случая теоретическая сложность T(n) = 1+(n+1)+n+n+1 = 3n+3 (линейная).

Для худшего случая теоретическая сложность T(n) = 1+(n+1)+n+n+n+1 = 4n+3 (линейная).

Для среднего случая должно удовлетворяться условие: $T(n)_{лучший}<T(n)_{средний}<T(n)_{худший}$.

1.3 Реализация алгоритма в виде функции:

Алгоритм 1:

На рис. 3 и рис. 4 представлены коды программы в виде функции.

```
#include <iostream>
using namespace std;

int delFirstMetod(int* x, int n, int key);

int main() {
    int n; cin >> n;
    int* x = new int[n];
    for (int i = 0; i < n; i++)
        cin >> *(x + i);
    int key; cin >> key;
    delFirstMetod(x, n, key);
    delete[] x;
}
```

Рисунок 3 - Функция main алгоритма 1

```
int delFirstMetod(int* x, int n, int key) {
    int i = 0;
    int sum = 1;
    while (i < n) {
        sum++;
        if (x[i] == key) {
            sum++;
            //удаление
            for (int j = i; j < n - 1; j++) {
                sum++;
                x[j] = x[j + 1];
                sum++;
            }
            sum++;
            n = n - 1;
            sum++;
        }
        else {
            i = i + 1;
            sum++;
        }
    }
    sum++;
    return sum;
}
```

Рисунок 4 - Функция delFirstMetod алгоритма 1

Алгоритм 2:

На рис. 5 и рис. 6 представлены коды программы в виде функции.

```
#include <iostream>
using namespace std;

int delOtherMetod(int* x, int n, int key);

int main() {
    int n; cin >> n;
    int* x = new int[n];
    for (int i = 0; i < n; i++)
        cin >> *(x + i);
    int key; cin >> key;
    delOtherMetod(x, n, key);
    delete[] x;
}
```

Рисунок 5 - Функция main алгоритма 2

```
int delOtherMetod(int* x, int n, int key) {
    int j = 0;
    int sum = 1;
    for (int i = 0; i < n; i++) {
        sum++;
        x[j] = x[i];
        sum++;
        if (x[i] != key) {
            j++;
            sum++;
        }
        sum++;
    }
    sum++;
    n = j;
    sum++;
    return sum;
}
```

Рисунок 6 - Функция delOtherMetod алгоритма 2

1.4 Реализация функций:

Добавим в оба алгоритма датчик случайных чисел и вывод массивов на экран.

Алгоритм 1:

На рис. 7 и рис. 8 представлены коды программы.

```
#include <iostream>
using namespace std;

int delFirstMetod(int* x, int n, int key);

int main() {
    int n; cin >> n;
    int* x = new int[n];
    srand(time(0));
    for (int i = 0; i < n; i++)
        *(x + i) = rand() % 11;
    int key; cin >> key; cout << endl << endl;
    cout << "n = " << n << endl << "x = [ ";
    for (int i = 0; i < n; i++)
        cout << *(x + i) << " ";
    cout << "]" << endl << "key = " << key << endl << endl;
    cout << "sum = " << delFirstMetod(x, n, key);
    delete[] x;
}
```

Рисунок 7 - Функция main алгоритма 1

```
int delFirstMetod(int* x, int n, int key) {
    int i = 0;
    int sum = 1;
    while (i < n) {
        sum++;
        if (x[i] == key) {
            sum++;
            //удаление
            for (int j = i; j < n - 1; j++) {
                sum++;
                x[j] = x[j + 1];
                sum++;
            }
            sum++;
            n = n - 1;
            sum++;
        }
        else {
            i = i + 1;
            sum++;
        }
    }
    sum++;
    cout << "x = [ ";
    for (int i = 0; i < n; i++)
        cout << *(x + i) << " ";
    cout << "]" << endl;
    return sum;
}
```

Рисунок 8 - Функция defFirstMetod алгоритма 1

Алгоритм 2:

На рис. 9 и рис. 10 представлены коды программы.

```cpp
#include <iostream>
using namespace std;

int delOtherMetod(int* x, int n, int key);

int main() {
    int n; cin >> n;
    int* x = new int[n];
    srand(time(0));
    for (int i = 0; i < n; i++)
        *(x + i) = rand() % 11;
        //*(x + i) = 5;
    int key; cin >> key; cout << endl << endl;
    cout << "n = " << n << endl << "x = [ ";
    for (int i = 0; i < n; i++)
        cout << *(x + i) << " ";
    cout << "]" << endl << "key = " << key << endl << endl;
    cout << "sum = " << delOtherMetod(x, n, key);
    delete[] x;
}
```

Рисунок 9 - Функция main алгоритма 2

```cpp
int delOtherMetod(int* x, int n, int key) {
    int j = 0;
    int sum = 1;
    for (int i = 0; i < n; i++) {
        sum++;
        x[j] = x[i];
        sum++;
        if (x[i] != key) {
            j++;
            sum++;
        }
        sum++;
    }
    sum++;
    n = j;
    sum++;
    cout << "x = [ ";
    for (int i = 0; i < n; i++)
        cout << *(x + i) << " ";
    cout << "]" << endl;
    return sum;
}
```

Рисунок 10 - Функция delOtherMetod алгоритма 2

1.5 Тесты в разных ситуациях:

Алгоритмы		Случайный	Лучший	Худший
1	n = 10	41	32	132
	n = 100	1522	302	10302
2	n = 10	42	33	43
	n = 100	393	303	403

1.6 Сравнение тестов:

Будем рассматривать случайные случаи как средние.

Алгоритмы			Случайный	Лучший	Худший
1	n = 10	Теоретический	32<T(10)<132	32	132
		Практический	41	32	132
	n = 100	Теоретический	302<T(100)<10302	302	10302
		Практический	1522	302	10302
2	n = 10	Теоретический	33<T(n)<43	33	43
		Практический	42	33	43
	n = 100	Теоретический	303<T(n)<403	303	403
		Практический	393	303	403

1.7 Ёмкостная сложность:

Алгоритм 1:

Ёмкостная сложность равна n.

Алгоритм 2:

Ёмкостная сложность равна n.

Вывод о задании 1:

Наиболее эффективным является алгоритм 2 потому что он и в лучшем, и в худшем случаях имеет линейную сложность, содержит всего один цикл и скорость работы не зависит от значений элементов массива.

2. Задание 2

2.1 Формулировка задачи:

Реализовать алгоритм в соответствии с задачей варианта 7: найти минимальное чётное число в части матрицы - между главной и побочной диагоналями (диагонали образуют вертикальные песочные часы).

2.2 Математическая модель решения:

а) Устное описание и описание блок-схемой

На вход подаётся переменная N типа int, которая отвечает за размер квадратной матрицы. Затем создается сама квадратная матрица, заполняется случайными числами (в моём случае из диапазона [1, 100]) и выводится на экран. Так как максимальное значение в матрице может быть 100, то создадим новую переменную min_matrix типа int и инициализируем её значением 100+1. Теперь будем перебирать значения матрицы и искать те, которые удовлетворяют условию задачи, то есть лежат между главной и побочной диагоналями и образуют вертикальные песочные часы. Чтобы найти эти числа должно выполняться условие (i < j && i < (N - 1 - j)) || (i > j && i > (N - 1 - j)), где i - номер строки, j - номер столбца, N - длина матрицы. Далее следует стандартное условие, если значение элемента матрицы меньше min_matrix, то изменить значение min_matrix на значение элемента матрицы. В конце программы вывести значение min_matrix.

На рис. 11 представлена блок-схема данного алгоритма.

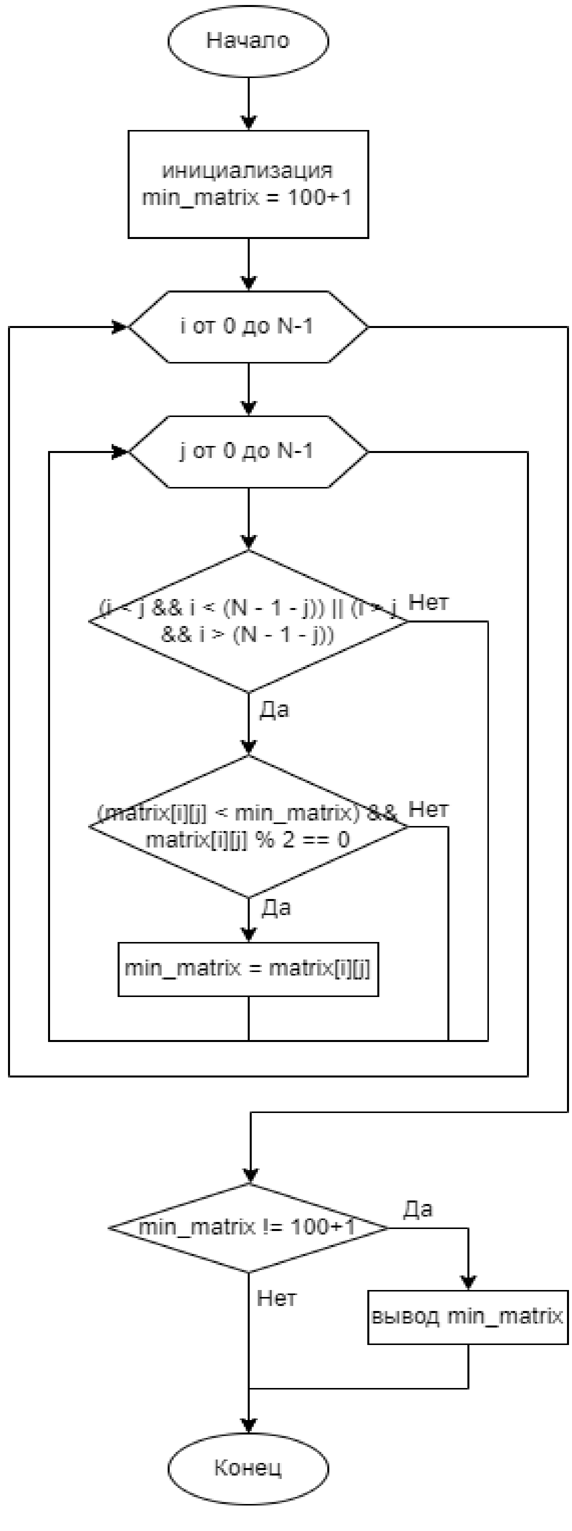

Рисунок 11 - блок-схема алгоритма

b)

Внешним циклом является цикл for (int i = 0; i < N; i++).

Инвариант: i < N && j < N. Изначально переменная-счётчик i инициализируется значением 0, как и переменная j, так что условие ((i < j && i < (N - 1 -

j)) || (i > j && i > (N - 1 - j))) выполняется при инициализации. Число N – конечное количество элементов в строке/столбце матрицы, значит N > 0. Всё время переменная j увеличивается на 1 и, когда она доходит до своего "максимального" значения, то есть N-1, то она сбрасывается обратно в 0, а переменная i увеличивается на 1. Такой цикл будет выполняться до тех пор, пока переменная i и переменная j вместе не дойдут до "максимального" значения, то есть N-1.

c)

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
min_matrix = 100+1	1	1
for (int i = 0; i < N; i++) {	N + 1	N + 1
for (int j = 0; j < N; j++) {	N + 1	N + 1
if ((i < j && i < (N - 1 - j)) || (i > j && i > (N - 1 - j))) {	N^2	N^2
if ((*(matrix + i * N + j) < min_matrix) && *(matrix + i * N + j) % 2 == 0) {	$\frac{N^2-([N/2]*2+N)}{2}$	$\frac{N^2-([N/2]*2+N)}{2}$
min_matrix = *(matrix + i * N + j)	0	$-\frac{N^2-([N/2]*2+N)}{2}$
} } } }		
if (min_matrix != 100 + 1) .. else	1	1

Для лучшего случая теоретическая сложность: (квадратичная)

$$T(N)=1+N(N+1)+1+N^2+\frac{N^2-(\lfloor N/2\rfloor*2+N)}{2}+1=\lceil\frac{5N^2+2N}{2}\rceil+3$$

Для худшего случая теоретическая сложность: (квадратичная)

$$T(N)=1+N(N+1)+1+N^2+\frac{N^2-(\lfloor N/2\rfloor*2+N)}{2}-\frac{N^2-(\lfloor N/2\rfloor*2+N)}{2}+1=3N^2+3$$

Для среднего случая должно удовлетворяться условие: $T(N)_{\text{лучший}}<T(N)_{\text{средний}}<T(N)_{\text{худший}}$.

2.4 Реализация алгоритма в виде одной функции

На рис. 12 представлен код программы в виде одной функции без декомпозиции на другие функции.

```cpp
#include <iostream>
using namespace std;

// Найти минимальное четное число в части матрицы – между главной и
// побочной диагоналями (диагонали образуют вертикальные песочные часы).

int main()
{
    cout << "N = "; int N; cin >> N;
    srand(time(0));
    int* matrix = new int[N * N];
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            *(matrix + i * N + j) = rand() % 100 + 1;
            cout << *(matrix + i * N + j);
            if (j < N - 1)
                cout << " ";
        }
        cout << endl;
    }
    int sum = 0;
    int min_matrix = 100 + 1;
    sum++;
    for (int i = 0; i < N; i++) {
        sum++;
        for (int j = 0; j < N; j++) {
            sum++;
            if ((i < j && i < (N - 1 - j)) || (i > j && i > (N - 1 - j))) {
                if ((*(matrix + i * N + j) < min_matrix) && *(matrix + i * N + j) % 2 == 0) {
                    min_matrix = *(matrix + i * N + j);
                    sum++;
                }
                sum++;
            }
            sum++;
        }
        sum++;
    }
    sum++;
    delete[] matrix;
    if (min_matrix != 100 + 1)
        cout << "min_matrix = " << min_matrix;
    else
        cout << "The required element was not found!";
    sum++;
    cout << endl << "sum = " << sum;
}
```

Рисунок 12 - Код программы без декомпозиции на другие функции

2.5 Тесты в разных ситуациях:

	Случайный	Лучший	Худший
N = 6	101	99	111
N = 10	270	263	303

2.6 Сравнение тестов:

Будем рассматривать случайные случаи как средние.

		Случайный	Лучший	Худший
n = 6	Теоретический	99<T(6)<111	99	111
	Практический	101	99	111
n = 10	Теоретический	263<T(10)<303	263	303
	Практический	270	263	303

2.7 Ёмкостная сложность:

Ёмкостная сложность равна N+4.

Вывод о задании 2:

```
#include <iostream>
using namespace std;

int main()
{
	cout << "N = "; int N; cin >> N;
	srand(time(0));
	/*int* mat = new int[N * N];
	long x = N * N * 5;
	for (int i = 0; i < N; i++) {
		for (int j = 0; j < N; j++) {
			*(mat + i * N + j) = x;
			x -= 2;
```

```
        }
    }*/
    int* matrix = new int[N * N];
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            *(matrix + i * N + j) = rand() % 100 + 1;
            //*(matrix + i * N+ j) = *(mat + i * N + j);
            cout << *(matrix + i * N + j);
            if (j < N - 1)
                cout << " ";
        }
        cout << endl;
    }
    //delete[] mat;
    int sum = 0;
    long min_matrix = N * N * 5 + 1;
    sum++;
    for (int i = 0; i < N; i++) {
        sum++;
        for (int j = 0; j < N; j++) {
            sum++;
            if ((i < j && i < (N - 1 - j)) || (i > j && i > (N - 1 - j))) {
                if ((*(matrix + i * N + j) < min_matrix) && *(matrix
+ i * N + j) % 2 == 0) {
                    min_matrix = *(matrix + i * N + j);
                    sum++;
                }
                sum++;
            }
```

```
                sum++;
            }
            sum++;
        }
        sum++;
        delete[] matrix;
        if (min_matrix != 100 + 1)
            cout << "min_matrix = " << min_matrix;
        else
            cout << "The required element was not found!";
        sum++;
        cout << endl << "sum = " << sum;
}
```