Все материалы контрольной можно найти здесь:
https://github.com/vladislovshilov/ProgrammingTechnology

1. Теоретические вопросы

1. Алгоритм бинарного поиска:

Алгоритм бинарного поиска используется для поиска нужного элемента в отсортированном массиве.
Главная суть, и собственно причина такого названия, - при поиске элемента, исходный массив делится надвое до тех пор, пока не будет найден индекс нужного элемента, при условии, что он есть.
Поясню: берется индекс середины исходного массива. Если элемент массивы больше нужного нам элементы, мы в следующих итерациях используем только те индексы, что остались по левую сторону от текущего(меньше). Если же элемент больше, то соответсвенно правую сторону.
Следовательно дальше мы снова ищем середину из оставшегося отрезка, и повторяем итерации до тех пор, пока не будет найдет нужный элемент, либо невозможно будет найти середину отрезка(его длинна станет равно 1).

2. Алгоритм сортировка Хоара:

Улучшенная версия сортировки пузырьком.
В массиве выбирается опорный элемент, зачастую это рандомный элемент, но для большей эффективности лучше брать либо средний элемент, либо медианный.
Далее каждый элемент сравнивается с опорным и определяется в одну из трех(или двух, в зависимости от типа сортировки) групп: меньше, равны и больше опорного.
После этого алгоритм рекурсивно повторяется для левой(меньше) и правой(больше) групп.

2.Практическое задание

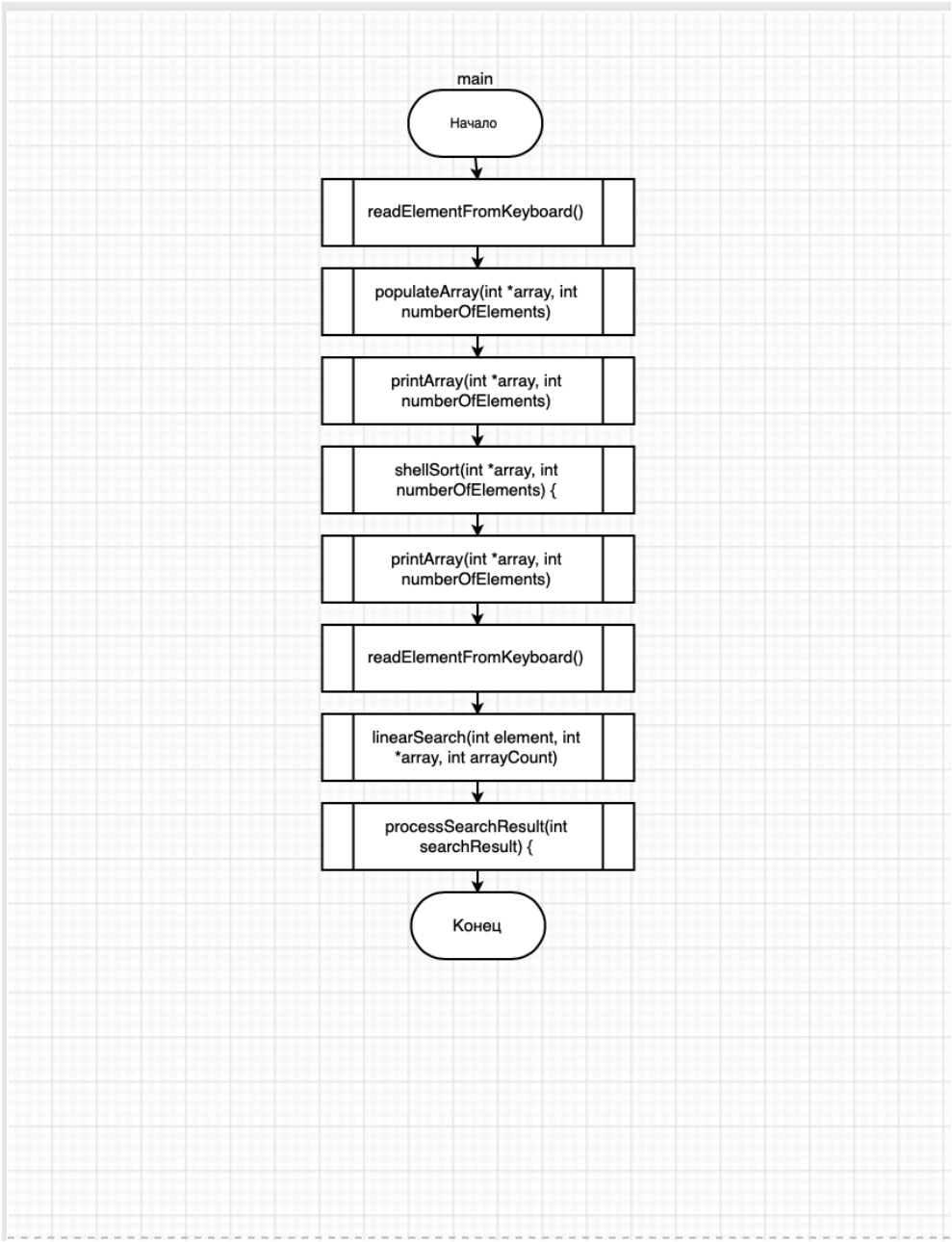
```c
//
//  main.c
//  Sort
//
//  Created by Vlados iOS on 2/2/19.
//  Copyright © 2019 Vladislav Shilov. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

/// Returns: read integer number from keboard

int readElementFromKeyboard() {
    char inputElement[100];
    int element = -1;
    bool shouldShowMessage = false;
    do {
        if(shouldShowMessage == true) {
            printf("Try to input positive integer number:\n");
        }
        scanf("%s",inputElement);
        element = (int)atol(inputElement) Go Forward
        shouldShowMessage = true;
    } while(element <= 0);
    return element;
}

/// Fills input array with random numbers
/// Params: array - array which we send via refernce on memory
///         numberOfElements - number of elements on array

void populateArray(int *array, int numberOfElements) {
    for (int i = 0; i < numberOfElements; i++) {
        array[i] = rand() % 100 + 1;
    }
}
/// Prints array on console in a convenient format
/// Params: array - array which we send via refernce on memory
///         numberOfElements - number of elements on array

void printArray(int *array, int numberOfElements) {
    for (int i = 0; i < numberOfElements; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
}

/// Sort array via Shell's method
/// Params: array - array which we send via refernce on memory
///         numberOfElements - number of elements on array

void shellSort(int *array, int numberOfElements) {
    unsigned i, j, step;
    int temp;
    for(step = numberOfElements / 2; step > 0; step /= 2)
        for(i = step; i < numberOfElements; i++) {
            temp = array[i];
            for(j = i; j >= step; j -= step) {
                if(temp < array[j - step])
                    array[j] = array[j - step];
                else
                    break;
            }
            array[j] = temp;
        }
}
```

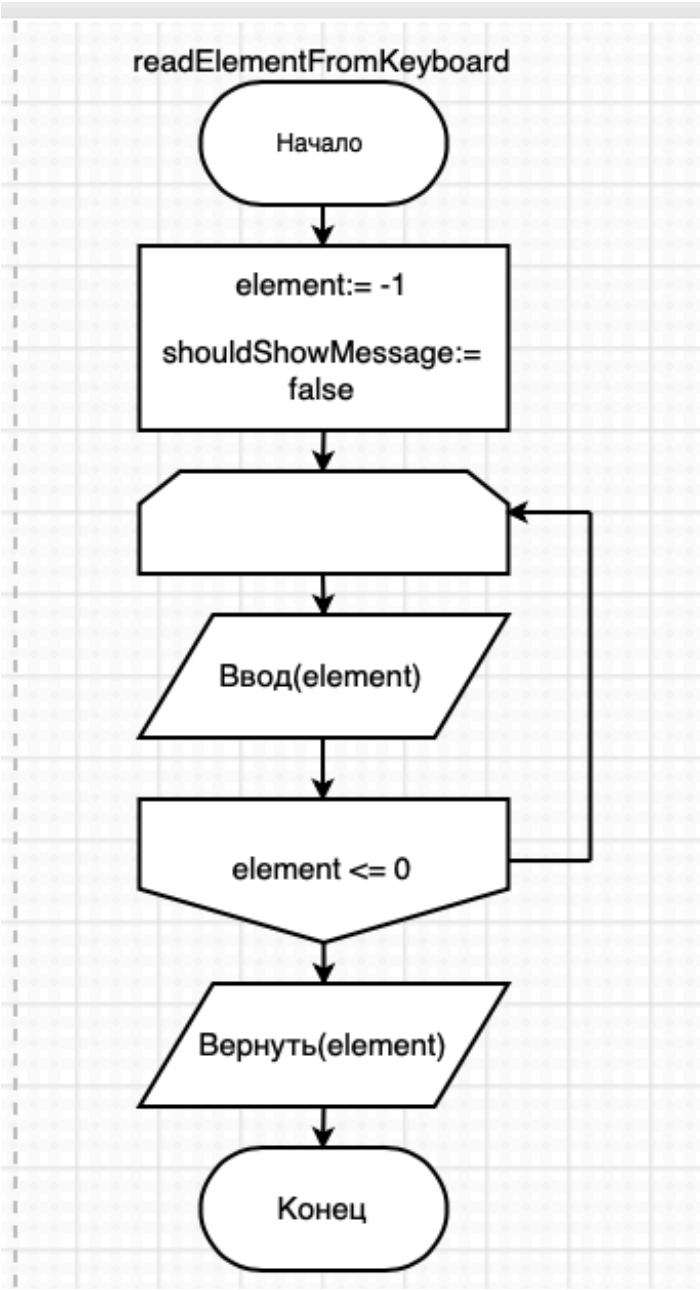
```c
/// Find needed element in array via linear search
/// Params: element - element to find
///         array - array which we send via refernce on memory
///         numberOfElements - number of elements on array
/// Returns: index of found element or -1 if this element was not found

int linearSearch(int element, int *array, int arrayCount) {
    for (int i = 0; i < arrayCount; i++) {
        if (element == array[i]) {
            return i;
        }
    }
    return -1;
}

/// Prints to the screen result of linear search
/// Params: Result of liners search

void processSearchResult(int searchResult) {
    if (searchResult == -1) {
        printf("There is no this element in array\n");
    }
    else {
        printf("Finded element index: %d\n", searchResult);
    }
}
int main(int argc, const char * argv[]) {

    int numberOfElements;
    int elementToFind;

    printf("Input number of elements \n");
    numberOfElements = readElementFromKeyboard();

    int array[numberOfElements];
    populateArray(array, numberOfElements);

    printf("Generated array: \n");
    printArray(array, numberOfElements);

    // Shell's sort
    shellSort(array, numberOfElements);

    printf("Sorted array: \n");
    printArray(array, numberOfElements);

    printf("Input element to find:\n");
    elementToFind = readElementFromKeyboard();

    // Linear search
    int foundElementIndex = linearSearch(elementToFind, array, numberOfElements);
    processSearchResult(foundElementIndex);

    return 0;
}
```

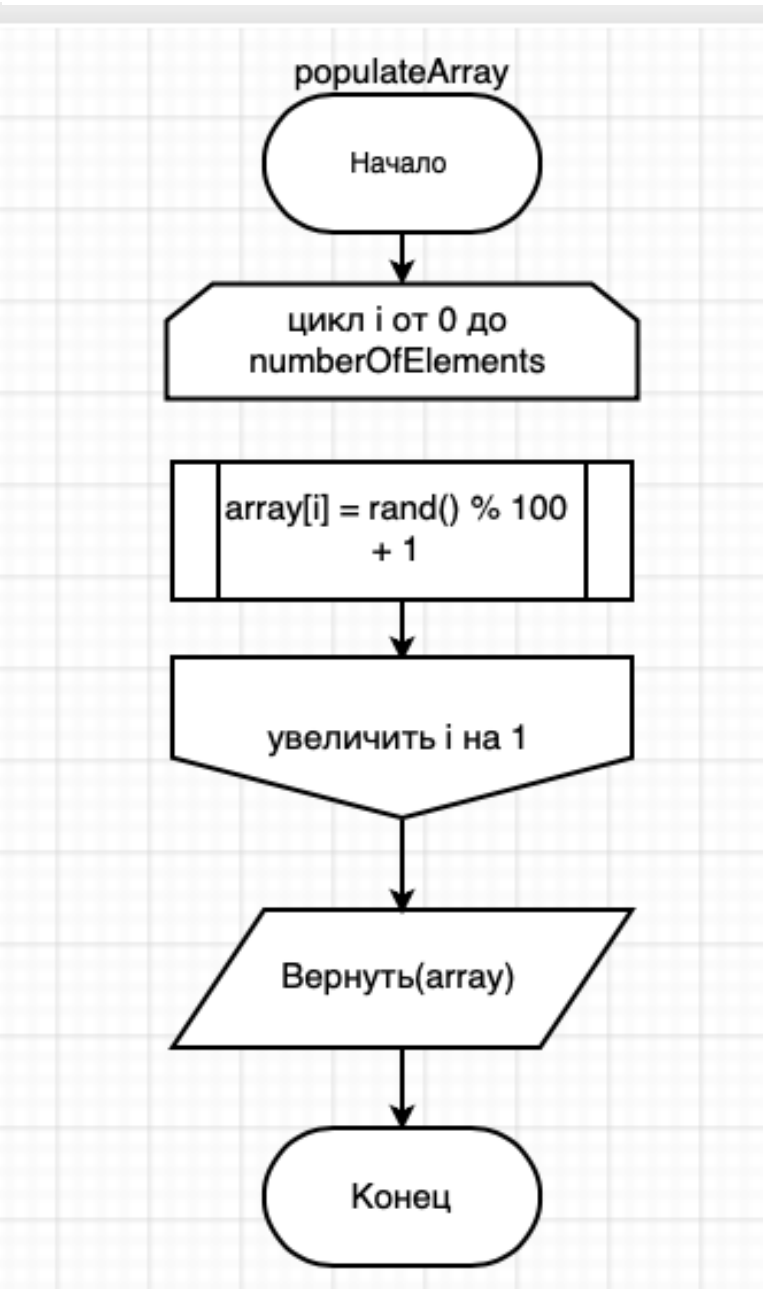
Результат работы алгоритмов:

```
Input number of elements
s
Try to input positive integer number:
d
Try to input positive integer number:
-123
Try to input positive integer number:
20
Generated array:
8 50 74 59 31 73 45 79 24 10 41 66 93 43 88 4 28 30 41 13
Sorted array:
4 8 10 13 24 28 30 31 41 41 43 45 50 59 66 73 74 79 88 93
Input element to find:
24
Finded element index: 4
Program ended with exit code: 0
```

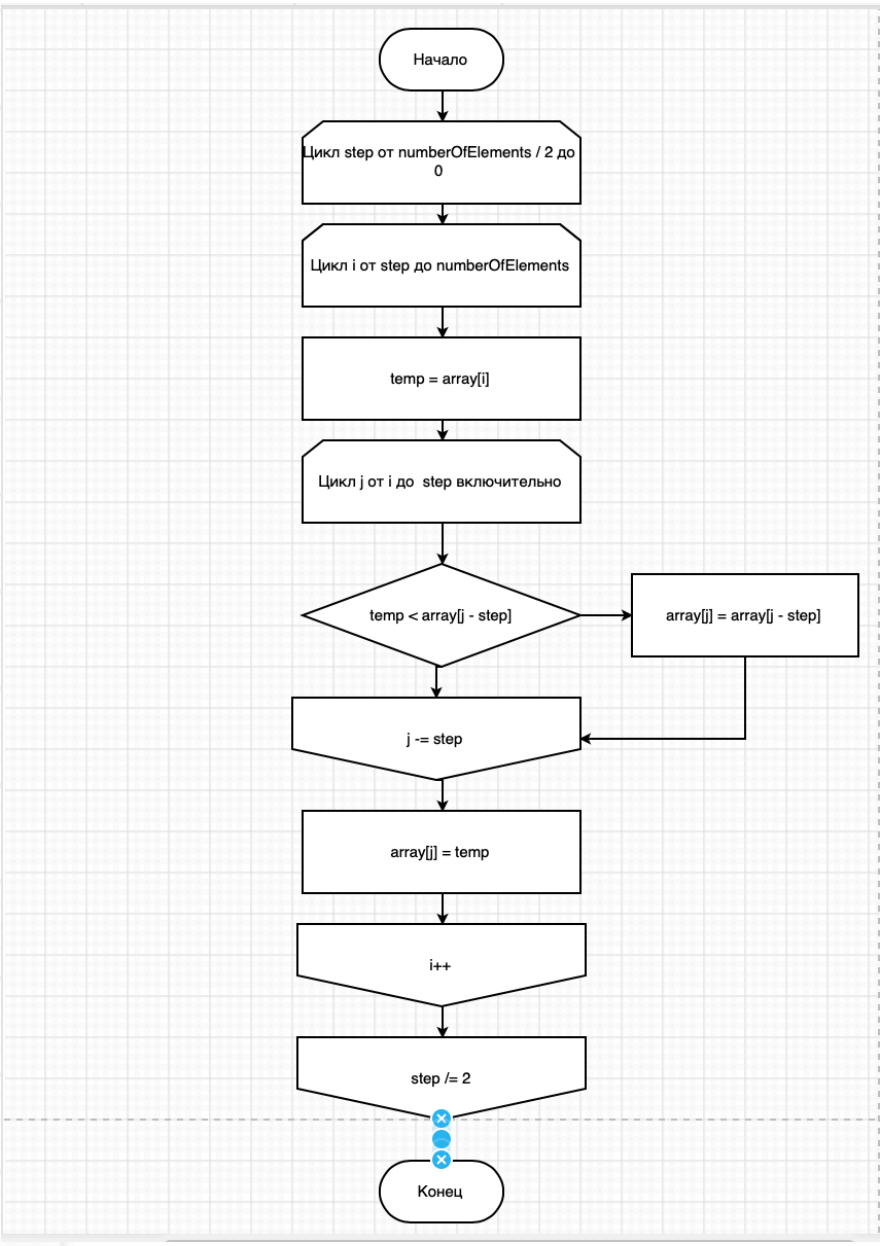
Блок схемы:

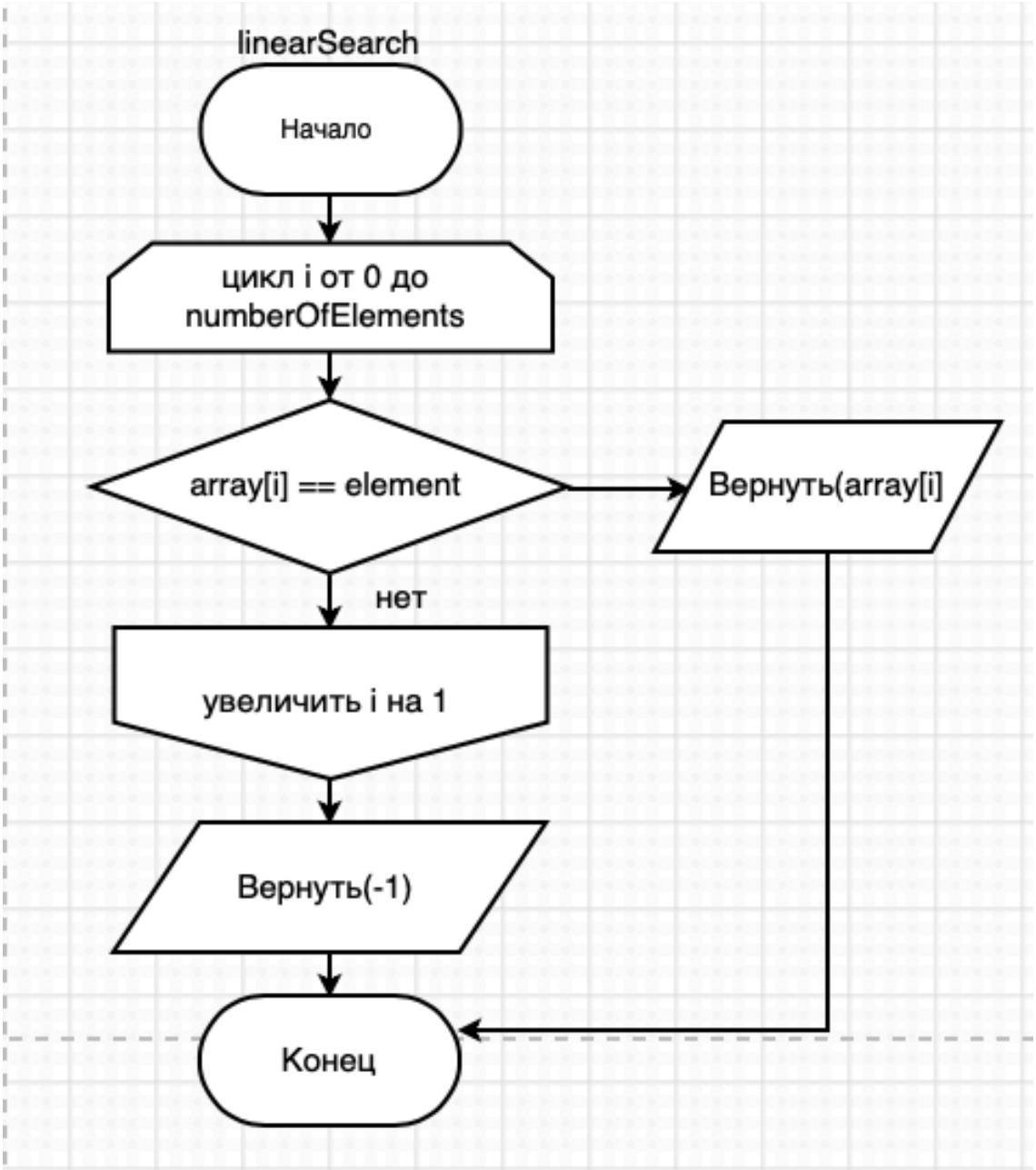

main
Начало
readElementFromKeyboard()
populateArray(int *array, int numberOfElements)
printArray(int *array, int numberOfElements)
shellSort(int *array, int numberOfElements) {
printArray(int *array, int numberOfElements)
readElementFromKeyboard()
linearSearch(int element, int *array, int arrayCount)
processSearchResult(int searchResult) {
Конец


readElementFromKeyboard
Начало
element:= -1
shouldShowMessage:= false
Ввод(element)
element <= 0
Вернуть(element)
Конец


populateArray
Начало
цикл i от 0 до numberOfElements
array[i] = rand() % 100 + 1
увеличить i на 1
Вернуть(array)
Конец


printArray
Начало
цикл i от 0 до numberOfElements
Вывести array[i]
увеличить i на 1
Конец


Начало
Цикл step от numberOfElements / 2 до 0
Цикл i от step до numberOfElements
temp = array[i]
Цикл j от i до step включительно
temp < array[j - step]
array[j] = array[j - step]
j -= step
array[j] = temp
i++
step /= 2
Конец


linearSearch
Начало
цикл i от 0 до numberOfElements
array[i] == element
Вернуть(array[i]
нет
увеличить i на 1
Вернуть(-1)
Конец