Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Национальный исследовательский университет «МЭИ»

|  |  |  |  |
| --- | --- | --- | --- |
| Институт: | ИВТИ |  |  |
| Направление подготовки: | | 09.03.01 Информатика и вычислительная техника | |

**ОТЧЕТ по практике**

|  |  |
| --- | --- |
| **Наименование практики:** | Учебная практика: профилирующая практика |

**СТУДЕНТ**

|  |  |
| --- | --- |
|  | / / |
| *(подпись )* | (*Фамилия и инициалы*) |

|  |  |
| --- | --- |
| Группа | А-20-21 |
|  | *(номер учебной группы)* |

**ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ ПО ПРАКТИКЕ**

|  |
| --- |
| зачтено |
| *(зачтено, не зачтено)* |

|  |  |
| --- | --- |
|  | / Емельянов Д.М. / |
| *(подпись )* | (*Фамилия и инициалы члена комиссии*) |

|  |  |
| --- | --- |
|  | / / |
| *(подпись )* | (*Фамилия и инициалы члена комиссии*) |

**Москва**

**20\_22\_**

Документация бригады №2.

Тема №3: Поиск элемента в одномерном упорядоченном массиве.

Программа написана на языке C++ и реализована в виде одного файла, содержащего основной текст программы и процедуры, выполненные на языке C++. Задачу можно разделить на четыре задачи:

Подзадача 1. Ввести данные

Подзадача 2.Выбрать способ сортировки

Подзадача 3. Найти номер нужного элемента массива

Подзадача 4. Вывести номер этого элемента

**Условие** Вводим данные, после этого мы выбираем способ сортировки (пузырьком, выбором, вставкой). Дальше находим элемент и номер элемента, который нам нужен, проходя по массиву, а потом выводим индекс на экран.

**Утонченная постановка задачи**

Дан одномерный массив mas из n элементов. Найти элемент, равный el

**Пример**

Дан массив из 8 элементов

8.0 9.7 0.0 2.0 8.0 6.0 1.0 2.0

Сортируем его:

0.0 1.0 2.0 2.0 6.0 8.0 8.0 9.7

Нужно найти номер элемента равного 9.7

Это 8 номер

**Таблица данных**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| класс | имя | описание | тип | Структура | Формат в/в |
| Входные данные | mas | Заданный массив | Вещественный тип данных | Одномерный динамический массив | +XX.XX (:5:2) |
| Входные данные | el | Значение элемента номер которого нужно найти | Вещественный тип данных | простая переменная | +XX.XX (:5:2) |
| Входные данные | n | Кол-во элементов массива | Целый тип данных | простая переменная | XX (:2) |
| Выходные данные | i | Индекс элемента, который нашли, выводим (i+1) | Целый тип данных | простая переменная | XX (:2) |
| Промежуточные  данные | j | Индекс массива, используем для сортировки | Целый тип данных | простая переменная | XX (:2) |
| Промежуточные  данные | lenD | Длина массива, используем эту переменную в процедурах | Целый тип данных | простая переменная | XX (:2) |
| Промежуточные  данные | tmp | Записывается элемент массива, чтобы в дальнейшем поменять значения двух элементов массива, что и нужно для сортировки | Целый тип данных | простая переменная | XX (:2) |
| Промежуточные  данные | key | Записывается элемент массива, чтобы в дальнейшем поменять значения двух элементов массива, что и нужно для сортировки | Целый тип данных | простая переменная | XX (:2) |
| Промежуточные  данные | flag | Существует для контроля аномальной ситуации | Логический тип данных | простая переменная |  |
| Промежуточные  данные | ch | Используется для консольного приложения, то есть выбора способа сортировки, а так же поиска нужного элемента | Строковый тип данных | простая переменная |  |
| Промежуточные  данные | data | Массив, используем эту переменную в процедурах | Вещественный тип данных | Одномерный динамический массив | +XX.XX (:5:2) |

**Функциональные тесты.**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ теста** | **Входные данные** | **Ожидаемый  результат** | **Смысл теста** |
| 1 | N=6  7.00 8.00 -6.00 4.00 2.00 -7.00  Найти 4.00 | -7.00 -4.00 -2.00 4.00 7.00 8.00  4 | Различные значения массива |
| 2 | N=3  -8.00 -9.00 -6.80  Найти -6.80 | -9.00 -8.00 -6.80  3 | Все элементы отрицательные |
| 3 | N=3  -9.00 -9.00 -9.00  Найти -9.0 | -9.00 -9.00 -9.00  1 | Все элементы отрицательные и равны друг другу |
| 4 | N=3  9.00 9.00 9.00  Найти 9.00 | 9.00 9.00 9.00  1 | Все элементы положительные и равны друг другу |
| 5 | N=4  1.00 0.30 3.00 2.00  Найти 1.00 | 0.30 1.00 2.00 3.00  2 | Все элементы положительные |
| 6 | N=4  1.00 0.30 3.00 2.00  Найти 1.4 | 0.3 1,0 2.0 3.0  Нет такого элемента | Нет элемента равного 1.4 |
| 7 | N=5  1.20 1.70 0.60 3.50 2.70  Найти 2.70 | 0.60 1.20 1.70 2.70 3,50  4 | Все элементы десятичные и положительные |
| 8 | N=5  -1.20 -1.70 -0.60 -3.50 -2.70  Найти -2.70 | -3.50 -2.70 -1.70 -1.20 -0.60  2 | Все элементы десятичные и отрицательные |
| 9 | N=5  1.00 1.00 1.00 3.00 2.00  Найти 2.00 | 1.00 1.00 1.00 2.00 3.00  4 | Все элементы целые и положительные |
| 10 | N=5  -1.00 -1.00 -1.00 -3.00 -2.00  Найти 2.00 | -3.00 -2.00 -1.00 -1.00 -1.00  2 | Все элементы целые и отрицательные |

**Анализ сложности алгоритма:**

В данной программе используется **сортировка пузырьком**

|  |
| --- |
| void bubbleSort(double\*\* data, int lenD)  {  double tmp = 0;  for(int i = 0;i<lenD;i++){  for(int j = (lenD-1);j>=(i+1);j--){  if((\*data)[j]<(\*data)[j-1]){  tmp = (\*data)[j];  (\*data)[j]=(\*data)[j-1];  (\*data)[j-1]=tmp;  }  }  }  for(int i=0;i<lenD;i++){  printf(" %7.2lf ",(\*data)[i]);  }  printf("\n");  } |

Сортировка пузырьком – один из самых простых способом сортировки, часто применим в учебных целях. Данная сортировка хорошо применима для массива с небольшим количеством элементом, чем больше элементов, тем дольше он будет сортировать. К плюсам сортировки пузырьком относится простота реализации алгоритма.

Алгоритм сортировки пузырьком сводится к повторению проходов по элементам сортируемого массива. Проход по элементам массива выполняет внутренний цикл. За каждый проход сравниваются два соседних элемента, и если порядок неверный элементы меняются местами. Внешний цикл будет работать до тех пор, пока массив не будет отсортирован. Таким образом внешний цикл контролирует количество срабатываний внутреннего цикла.

В данной программе используется **сортировка выбором**

|  |
| --- |
| void SortAlgo(double \*\*data, int lenD)  {  int j = 0;  double tmp = 0;  for(int i=0; i<lenD; i++){  j = i;  for(int k = i; k<lenD; k++){  if(((\*data)[j])>((\*data)[k])){  j = k;  }  }  tmp = ((\*data)[i]);  ((\*data)[i]) = ((\*data)[j]);  ((\*data)[j]) = tmp;  }  printf("Sort massive\n");  for(int i=0;i<lenD;i++){  printf(" %7.2lf ",(\*data)[i]);  }  printf("\n");  } |

Проходим по массиву в поисках максимального элемента. Этот максимальный элемент меняем местами с последним элементом. Неотсортированная часть массива уменьшилась на один элемент. К этой неотсортированной части применяем те же действия — находим максимум и ставим его на последнее место в неотсортированной части массива. И так продолжаем до тех пор, пока не отсортируем массив.

Он также зависит от длины массива, чем больше элементов, тем больше времени он будет занимать.

В данной программе используется **сортировка вставкой**

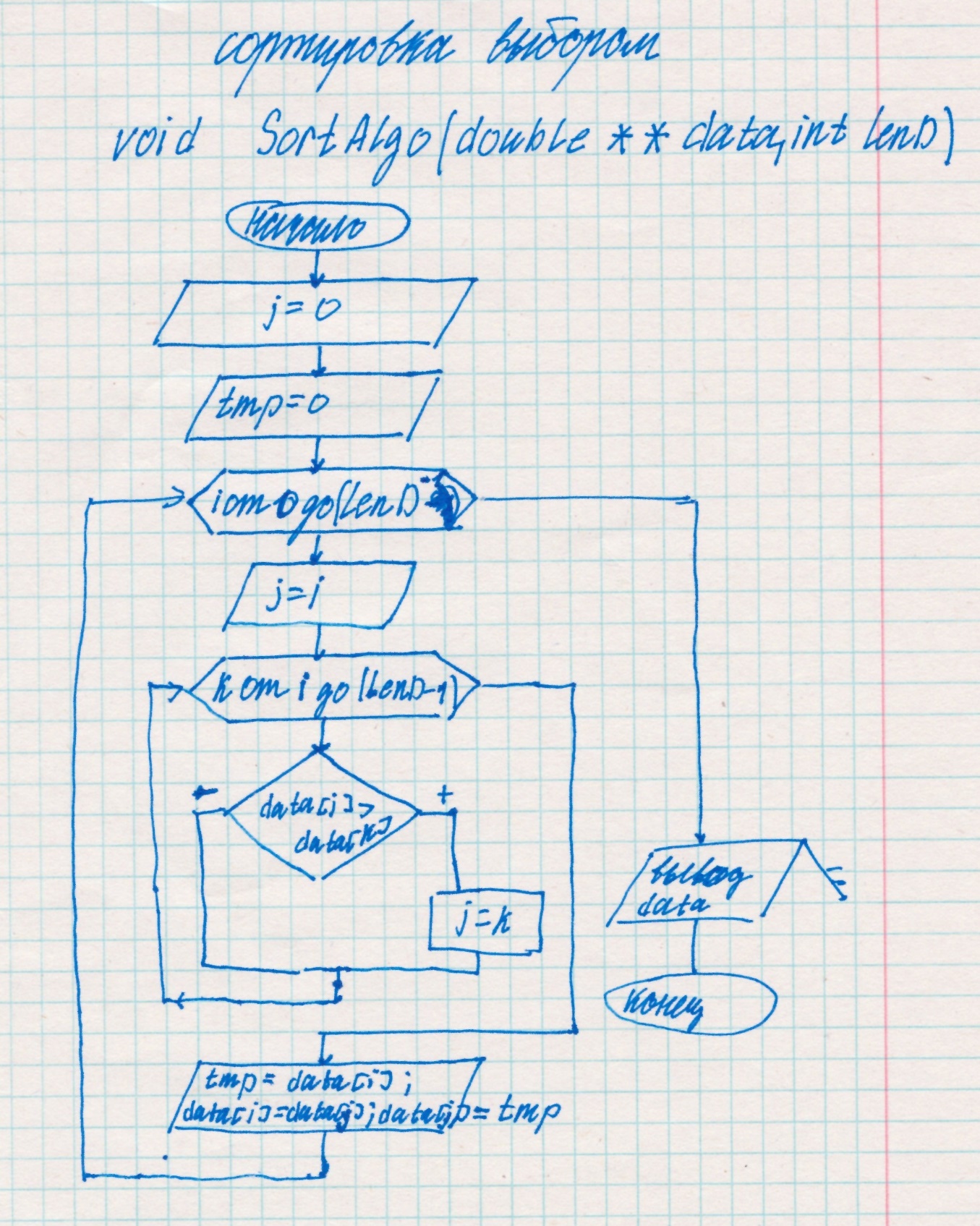
|  |
| --- |
| void insertionSort(double\*\* data, int lenD)  {  double key = 0;  int i = 0;  for(int j = 1;j<lenD;j++){  key = (\*data)[j];  i = j-1;  while(i>=0 && (\*data)[i]>key){  (\*data)[i+1] = (\*data)[i];  i = i-1;  (\*data)[i+1]=key;  }  }  for(int i=0;i<lenD;i++){  printf(" %7.2lf ",(\*data)[i]);  }  printf("\n");  } |

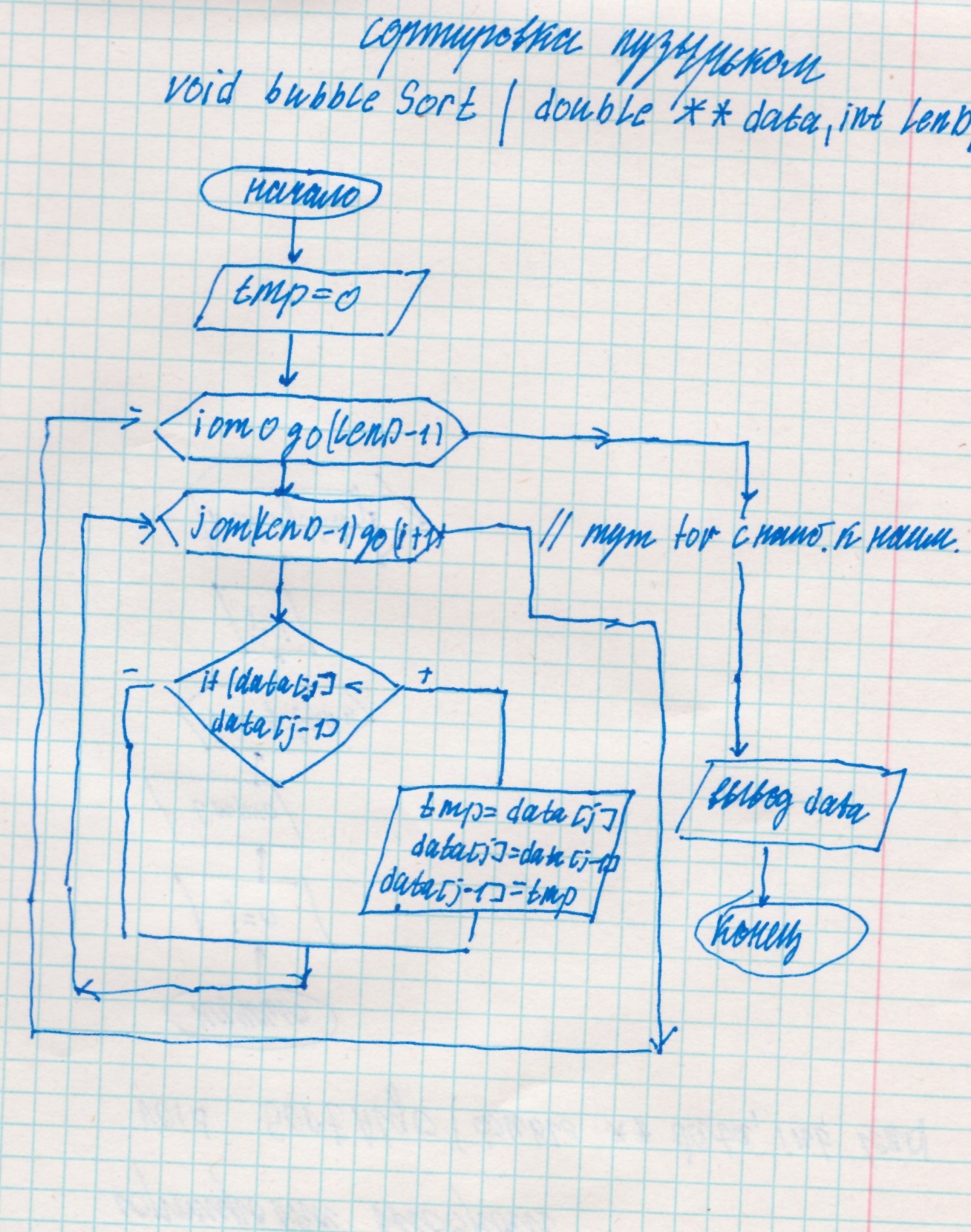
Сортировка вставками — достаточно простой алгоритм. Как в и любом другом алгоритме сортировки, с увеличением размера сортируемого массива увеличивается и время сортировки. Основным преимуществом алгоритма сортировки вставками является возможность сортировать массив по мере его получения. То есть имея часть массива, можно начинать его сортировать.

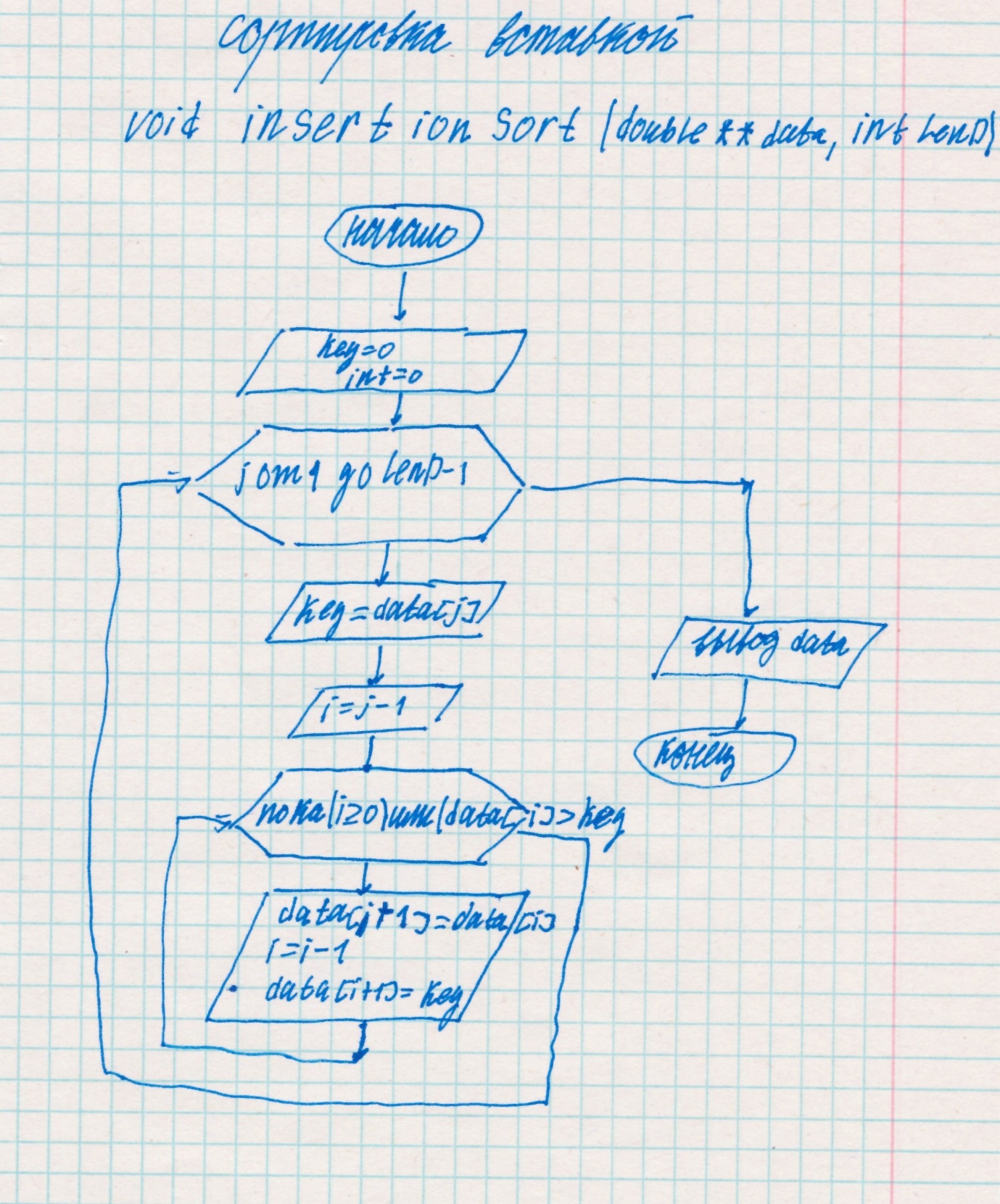
Преимущества: стабильно, быстрее пузырьковой;

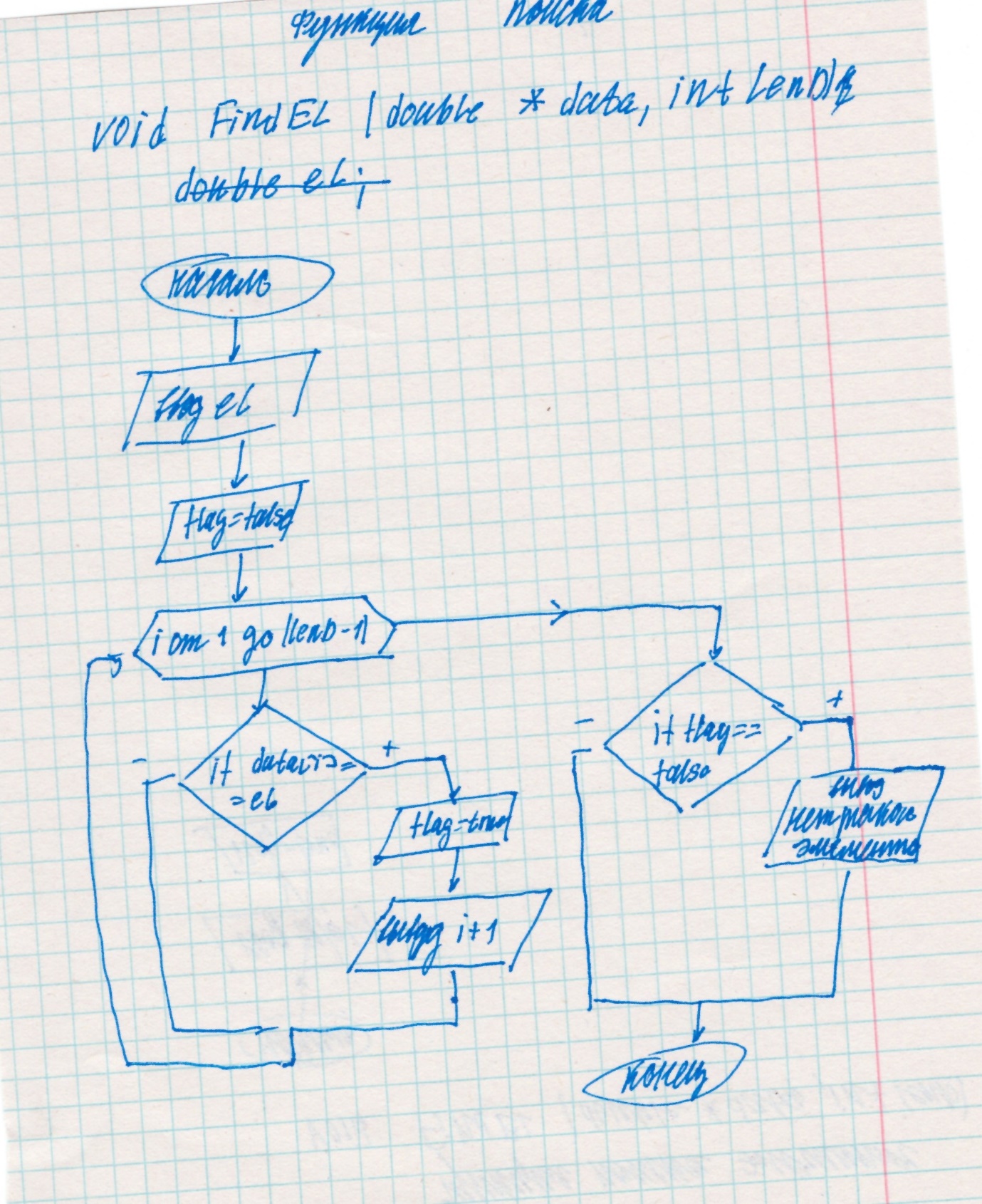
Недостатки: чем меньше число сравнений, тем больше данных перемещается после точки вставки, особенно когда большой объем;

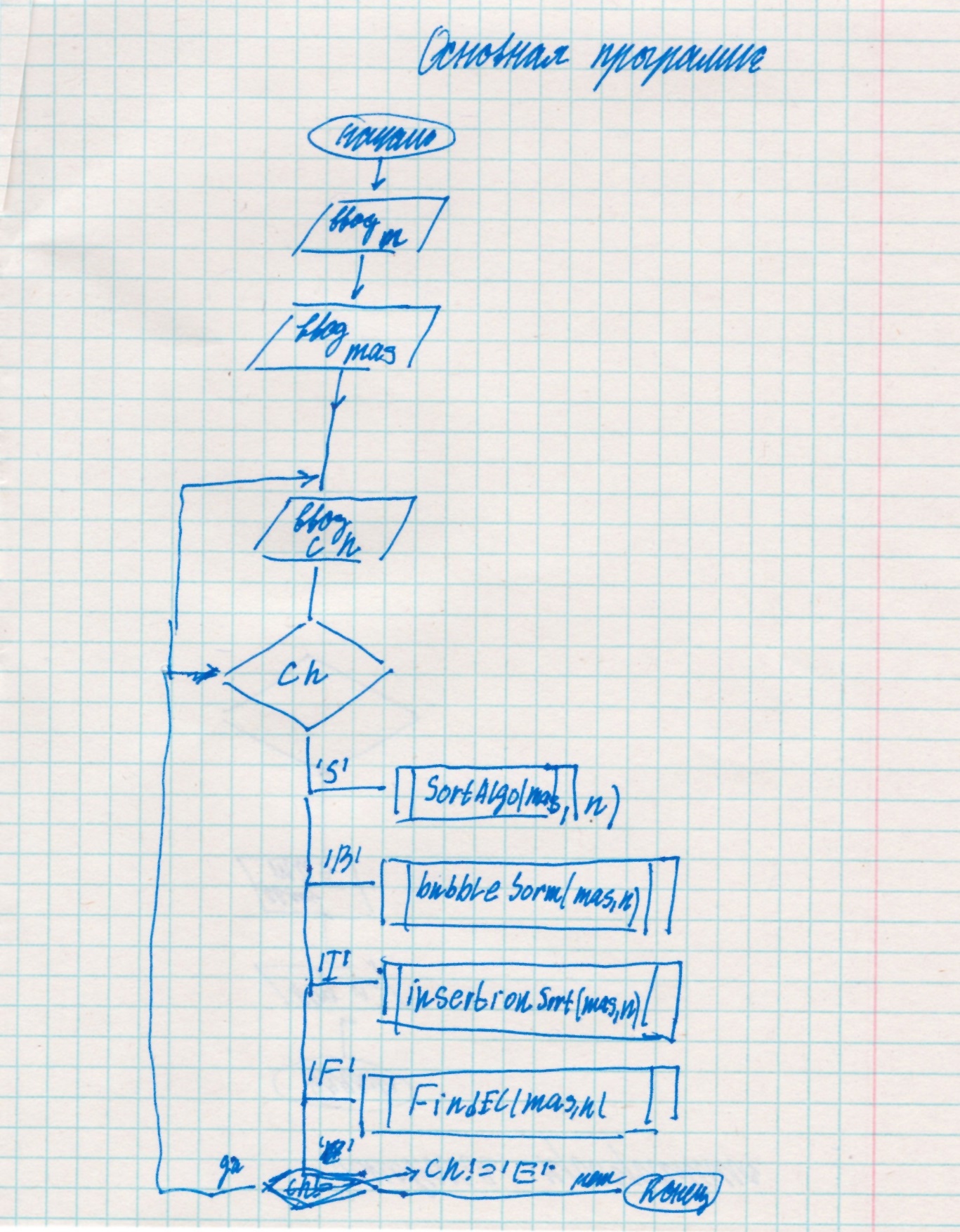
**Блок-схемы**











**Код программы:**

|  |
| --- |
| #include <stdio.h>  #include <cctype> //использование getch  #include <conio.h> //использование toupper  void SortAlgo(double \*\*data, int lenD)  {  int j = 0;  double tmp = 0;  for(int i=0; i<lenD; i++){  j = i;  for(int k = i; k<lenD; k++){  if(((\*data)[j])>((\*data)[k])){  j = k;  }  }  tmp = ((\*data)[i]);  ((\*data)[i]) = ((\*data)[j]);  ((\*data)[j]) = tmp;  }  printf("Sort massive\n");  for(int i=0;i<lenD;i++){  printf(" %7.2lf ",(\*data)[i]);  }  printf("\n");  }  void bubbleSort(double\*\* data, int lenD)  {  double tmp = 0;  for(int i = 0;i<lenD;i++){  for(int j = (lenD-1);j>=(i+1);j--){  if((\*data)[j]<(\*data)[j-1]){  tmp = (\*data)[j];  (\*data)[j]=(\*data)[j-1];  (\*data)[j-1]=tmp;  }  }  }  for(int i=0;i<lenD;i++){  printf(" %7.2lf ",(\*data)[i]);  }  printf("\n");  }  void insertionSort(double\*\* data, int lenD)  {  double key = 0;  int i = 0;  for(int j = 1;j<lenD;j++){  key = (\*data)[j];  i = j-1;  while(i>=0 && (\*data)[i]>key){  (\*data)[i+1] = (\*data)[i];  i = i-1;  (\*data)[i+1]=key;  }  }  for(int i=0;i<lenD;i++){  printf(" %7.2lf ",(\*data)[i]);  }  printf("\n");  }  void FindEl(double \*data,int lenD){  double el;  printf("enter the number you want to find");  scanf("%lf",&el);  printf("\n");  bool flag=false;  for(int i=0;i<lenD;i++)  {  if(data[i]==el){  flag=true;  printf("The desired item under the number %d\n",i+1);  break;  }  }  if (flag==false){  printf("There is no such element\n");  }  }  int main()  {  double \*mas;  int n;  printf("Enter the number of array elements");  scanf("%d",&n);  printf("Enter the elements of massive\n");  mas=new double[n];  for(int i=0;i<n;i++){  printf("mas[%d]=",i+1);  scanf(" %lf",&mas[i]);  }  char ch;  do{  printf("\  \nS - Selection sort;\  \nB - Bubble sort;\  \nI - Insertion sort;\  \nF - Find element;\  \nE - exit.\  \nYour choose \n");  scanf("%s",&ch);  ch=toupper(ch);  switch(ch){  case 'S':SortAlgo(&mas,n);break;  case 'B':bubbleSort(&mas,n);break;  case 'I':insertionSort(&mas,n);break;  case 'F':FindEl(mas,n);break;  case'E':return 0;  default:  printf("There is no such command\n");  getch();  }  }while(ch!='E');  } |