Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Нижегородский Государственный Университет им.

Н.И.Лобачевского» (ННГУ)

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

Сравнение быстродействия алгоритмов сортировки массивов

Выполнил:

студент группы 3821Б1ПМ3

Сучков В.Н.

Проверил:

заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г

Нижний Новгород

2021г.

# Содержание

[Содержание 1](#_Toc87710077)

[Введение 3](#_Toc87710078)

[Постановка задачи 4](#_Toc87710079)

[Руководство пользователя 5](#_Toc87710080)

[Первая программа 5](#_Toc87710081)

[Вторая программа 6](#_Toc87710082)

[Руководство программиста 9](#_Toc87710083)

[Описание структуры кода программ 9](#_Toc87710084)

[Первая программа 9](#_Toc87710085)

[Вторая программа 10](#_Toc87710086)

[Описание структур данных 14](#_Toc87710087)

[Описание алгоритмов 14](#_Toc87710088)

[Алгоритмы для работы с данными из вне 14](#_Toc87710089)

[Сортировка «пузырьком» 15](#_Toc87710090)

[Сортировка «вставками» 16](#_Toc87710091)

[Быстрая сортировка 17](#_Toc87710092)

[Эксперименты 19](#_Toc87710093)

[Заключение 21](#_Toc87710094)

[Список литературы 22](#_Toc87710095)

[Приложение 1 23](#_Toc87710096)

[Приложение 1.1 23](#_Toc87710097)

[Приложение 1.2 25](#_Toc87710098)

# Введение

Массивы один из самых распространённых типов данных, который позволяет удобно работать с большим количеством чисел одно типа. Изучение, построение и оптимизация алгоритмов работы с массивами — это задача, которая естественным образом встает перед программистом, работающим с массивами. Работая с последовательностью чисел, записанной в массив, для оптимизации поиска нужных элементов нередко используются алгоритмы, позволяющее упорядочить, то есть отсортировать, массив и тем самым упростить поиск нужных элементов внутри него.

В настоящее время существует множество различных сортировок (т.е. алгоритмов сортировки), которые работаю с разной эффективностью и применяются в различных ситуациях. В настоящей работе на языке программирования «С» будет реализовано три вида наиболее известных сортировок: пузырьковая (Bubble Sort), сортировка вставками (Insertion Sort) и быстрая сортировка (Quick Sort), а также будет произведено сравнение их быстродействия при одинаковых входных данных.

# Постановка задачи

Реализовать сортировки массивов данных (тип данных определяется преподавателем) задаваемых: обязательно случайно, дополнительно с клавиатуры или из файла.

Реализовать сортировки: пузырьком, вставкой, быстрая. Сравнить время работы, сделать выводы.

Первая программа создает текстовый файл с записанными в него числами. Программа принимает количество чисел n, максимальное и минимальное значение.

Вторая программа читает текстовый файл с набором чисел, выводит консольный интерфейс (печать, сортировка, сброс, выход), выполняет выбранные действия.

# Руководство пользователя

## Первая программа

Программа встречает пользователя сообщением с просьбой указать способ заполнения текстового документа (см. рис. 1): набором случайных чисел или вводом чисел пользователем вручную с клавиатуры.

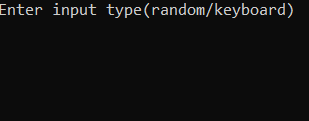


Рис. 1 Программа после запуска

Если пользователь напишет в консоли команду «random», то будет создан текстовый документ, который будет заполнен случайными числами, количество этих чисел, а также диапазон генерации чисел так же задаются пользователем (см. рис. 2).

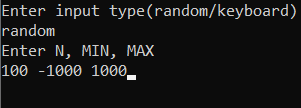


Рис. 2 Ввод параметров генерации чисел

После нажатия отправки параметров генерации программа завершает свою работу.

Если же пользователь напишет в консоли команду «keyboard», то после этого указатель в консоли переместиться на новую строку и пользователь сможет начать ввод чисел с клавиатуры. Числа передаются программе по одному и как только пользователь введет все нужные числа, чтобы прекратит ввод ему нужно отправить любой символ, не являющийся числом. Пример такой работы пользователя с интерфейсом программы приведен на рисунке 3.

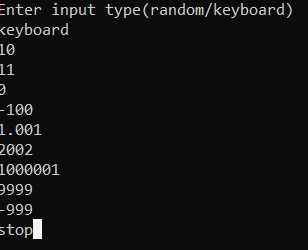


Рис. 3 Пример ввода с клавиатуры

После ввода слова stop программа прекратит работу.

При любом способе ввода результатом работы программы будет текстовый файл, который наполнен некоторым количеством чисел. Например, для набора из 10 случайных чисел, сгенерированных в диапазоне от -10 до 10, файл будет выглядеть как показано на рисунке 4.

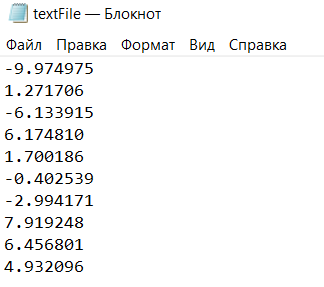


Рис. 4 Текстовый файл с сгенерированными числами

## Вторая программа

При запуске программы пользователь видит сообщение о том, сколько чисел удалось распознать в текстовом файле, а следующей строкой ему сообщается набор команд, которые может выполнить программа: печать, сортировка, сброс, выход (см. рис. 5).



Рис. 5 Интерфейс программы

При вводе команды «print» в консоли будет напечатан текущий набор чисел из файла (см. рис. 6).

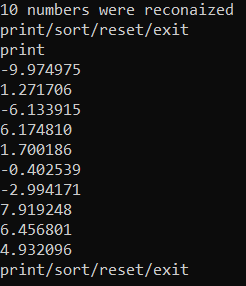


Рис. 6 Вывод чисел на экран

При вводе команды «sort» программа предложит пользователю выбрать тип сортировки из тех, что были реализованы в программе. После выбора типа сортировки программа отсортирует числа в файле и сообщит пользователю о завершении операции, а также укажет время, затраченное на сортировку в миллисекундах. (см. рис. 7).

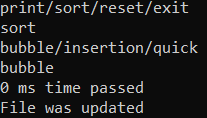


Рис. 7 Пример запуска сортировки

После чего, если напечатать числа из файла в консоли, можно будет убедиться в том, что программы выполнила сортировку (см. рис. 8).

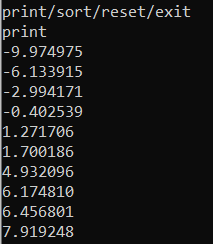


Рис. 8 Числа из файла после сортировки

При вводе команды «reset», числа из файла вернуться в те позиции, на которых они были при запуске программы (см. рис. 9)

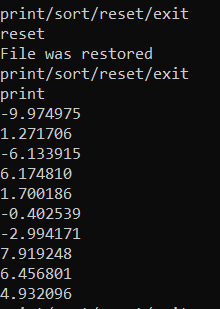


Рис. 9 Работа команды сброса

Для завершения работы программы пользователю достаточно ввести команду «exit».

# Руководство программиста

## Описание структуры кода программ

### Первая программа

1. Объявление функции, которая будет генерировать и записывать числа в текстовый файл

|  |
| --- |
| void WriteRand(FILE\* file,int n, double min, double max)  {  double num = 0.0;  *for*(int k = 0; k < n; k++)  {  num = ((double)rand())/RAND\_MAX \*(max - min) + min;  fprintf(file, "%lf\n", num);  }  } |

1. Объявление функции для сравнения массивов символов

|  |
| --- |
| int StrComp(char \*str\_1, char \*str\_2) *//сравнивает 1-е со вторым*  {    int i = 0;    int m =0 ;  *while*(str\_1[m] != 0) m++;  *while*((str\_1[i] == str\_2[i]) && str\_1[i] != 0)    i++;  *if*(m == i) *return* 1;  *return* 0;  } |

1. Объявление функции main(). В этом блоке кода происходит получение и проверка данных от пользователя на корректность, а также заполнение текстового файла

|  |
| --- |
| int main()  {  FILE \*fp = 0;  fp = fopen("textFile.txt", "w");  int n = 0;  double min, max = 0.0;  printf("Enter input type(random/keyboard)\n");  char str[9];    scanf("%s", str);  *if* (StrComp(str, "keyboard"))    {      double num = 0.0;  *while*(scanf("%lf\n", &num) != 0)      {        fprintf(fp, "%lf\n", num);      }      fclose(fp);  *return* 0;    }    printf("Enter N, MIN, MAX\n");    scanf("%d %lf %lf", &n, &min, &max);  *if*(n<=0 || (max - min)<0)    {      printf("Error: invalid input!");  *return* 0;    }    WriteRand(fp, n, min, max);    fclose(fp);  *return* 0;  } |

### Вторая программа

1. Объявление функции для сравнения массивов символов, а также объявление функций для сортировок трех видов

|  |
| --- |
| int StrComp(char \*str\_1, char \*str\_2)  {    int i = 0;  int m = 0 ;  *while*(str\_1[m] != 0) m++;  *while*((str\_1[i] == str\_2[i]) && str\_1[i] != 0)      i++;  *if*(m == i) *return* 1;  *return* 0;  }  void BubbleSort(int N, double \*arr)  {    double s = 0.0  *for*(int i = 0; i < N - 1; i++)    {  *for*(int j = 0; j < N-1; j++)      {  *if* (arr[j] > arr[j + 1])        {          s = arr[j];          arr[j] = arr[j + 1];          arr[j + 1] = s;        }      }    }  }  void InserionSort(int N, double \*arr)  {    double s = 0.0;    int j = 0;  *for*(int i = 1; i < N; i++)    {      j = i - 1;  *while*(j >= 0 && (arr[j] > arr[j + 1]))      {        s = arr[j];        arr[j] = arr[j + 1];        arr[j + 1] = s;        j--;      }    }  }  void QuickSort(int N, double \*arr)  {    long i =0, j = N - 1;  double key = 0.0, s = 0.0;    key = arr[(int)(N/2)];  *while*(i <= j)    {  *while*(arr[i] < key) i++;  *while*(arr[j] > key) j--;  *if* (i <= j)      {        s = arr[i];        arr[i] = arr[j];        arr[j] = s;        i++;        j--;      }    }  *if* (j > 0) QuickSort(j, arr);  *if* (i < N) QuickSort(N - i, arr + i);  } |

1. Объявление функции main(), в которой будет проходить инициализация чисел их текстового файла, определение их количества, их запись в массив для работы с ними, а также реализован основной интерфейс программы

|  |
| --- |
| int main()  {    char str[32];    int N = 0, l = 0;    double \*fl,\*num, ch = 0;    FILE \*fs = 0;    fs = fopen("textFile.txt", "r");  *if* (fs == NULL)    {     printf("Error: cannot open file\n");  *return* 0;    }  *while*(fscanf(fs, "%lf\n", &ch) != EOF) N++;    num = (double\*)malloc(sizeof(double)\*N);    fl = (double\*)malloc(sizeof(double)\*N);    freopen("textFile.txt","r", fs);  *for*(int k = 0; k < N; k++)    {      fscanf(fs, "%lf\n", &num[k]);      fl[k] = num[k];    }    printf("%d numbers were reconaized\n", N);  *while*(1)    {      printf("print/sort/reset/exit\n");      scanf("%s", str);  *if* (StrComp(str, "print"))      {        double temp = 0.0;        freopen("textFile.txt", "r", fs);  *for*(int k = 0; k < N; k++)        {          fscanf(fs, "%lf\n", &temp);          printf("%lf\n", temp);  *continue*;       }      }  *else* *if* (StrComp(str, "sort"))      {        printf("bubble/insertion/quick\n");        scanf("%s", str);        unsigned time = clock();  *if* (StrComp(str, "bubble"))         {           BubbleSort(N, num);        }  *else* *if* (StrComp(str, "insertion"))        {          InserionSort(N, num);        }  *else* *if* (StrComp(str, "quick"))        {          QuickSort(N, num);        }  *else*        {          printf("Error: invalid input!\n");  *continue*;        }        time = clock() - time;        printf("%d ms time passed\n", time);        freopen("textFile.txt", "w", fs);  *for*(int i = 0; i < N; i++)      {          fprintf(fs, "%lf\n", num[i]);        }        printf("File was updated\n");  *continue*;      }  *else* *if* (StrComp(str, "reset"))      {        freopen("textFile.txt", "w", fs);  *for*(int i = 0; i < N; i++)        {          fprintf(fs, "%lf\n", fl[i]);          num[i] = fl[i];        }        printf("File was restored\n");  *continue*;      }  *else* *if* (StrComp(str, "exit"))      {  *break*;      }  }    fclose(fs);  *return* 0;  } |

## Описание структур данных

Одним из ключевых элементов второй программы является динамический массив, хранящий переменные типа double, в которой записываются числа, полученные из файла и с которым по ходу программы, происходит основная доля работы. Именно элементы массива подвергаются сортировке, а затем записываются в текстовый файл. Помимо этого, присутствуют различные счетчики типа long int или int, без которых невозможна работа сортировок. Важнейшим элементом интерфейса является массив символов, в который записываются команды, вводимые пользователем во время работы программы.

При написании программы были использованы стандартные библиотеки языка «C»: stdlib.h, stdio.h, time.h, conio.h.

## Описание алгоритмов

### Алгоритмы для работы с данными из вне

Алгоритм сравнения строк. На вход алгоритму подаются две строки – массивы, состоящие из символов. После этого для одного из них подсчитывается количество символов до нулевой отметки, которая помещается в строку во время считывания сразу после последнего считаного символа. Затем производится посимвольное сравнения двух строк до тех пор, пока одна из строк не закончится. Если число совпавших символов и число символов в одной из строк совпали, значит это одинаковые строки и в таком случае алгоритм возвращает 1, иначе 0.

|  |
| --- |
| int StrComp(char \*str\_1, char \*str\_2)  {    int i = 0;    int m = 0 ;  *while*(str\_1[m] != 0) m++;  *while*((str\_1[i] == str\_2[i]) && str\_1[i] != 0)      i++;  *if*(m == i) *return* 1;  *return* 0;  } |

Алгоритм записи чисел из файла в массив. Изначальное количество символов в массиве полагается неизвестным и потому, сначала делается первый прогон по текстовому файлу. В результате мы получаем количество чисел типа double, которые удалось получить из файла. Затем возврат к началу файла и запись чисел в массив. Также исходный набор чисел из файла сохраняется во втором массиве, чтобы была возможность вернуть числа на свои места.

|  |
| --- |
| *while*(fscanf(fs, "%lf\n", &ch) != EOF) N++;  num = (double\*)malloc(sizeof(double)\*N);  fl = (double\*)malloc(sizeof(double)\*N);  freopen("textFile.txt","r", fs);  *for*(int k = 0; k < N; k++)  {    fscanf(fs, "%lf\n", &num[k]);    fl[k] = num[k];  } |

### Сортировка «пузырьком»

На вход алгоритму подается число элементов в сортируемом массиве, а также указатель на начало массива. На первой итерации внешнего цикла рассматривается нулевой элемент массива: он сравнивается с последующим элементом и, если оказывается больше него, то меняется с ним местами, а если оказывается меньше, то рассматривается уже следующий элемент. Эта процедура повторяется для каждого элемента массива, который в начале i-ой итерации внешнего цикла оказывается на месте j в массиве до тех пор, пока он будет отсортирован.

|  |
| --- |
| void BubbleSort(int N, double \*arr)  {    double s = 0.0;  *for*(int i = 0; i < N - 1; i++)    {  *for*(int j = 0; j < N-1; j++)      {  *if* (arr[j] > arr[j + 1])        {          s = arr[j];          arr[j] = arr[j + 1];          arr[j + 1] = s;        }      }    }  } |

Сортировка Bubble Sort считается наиболее простой и в тоже время наиболее время затратной. Её эффективность оценивается как от размера входных данных, то есть количества элементов в исходном массиве. И в самом деле это можно легко заметить, ведь внешний цикл повторяется n раз и в каждом из таких циклов есть вложенный цикл, который повторяется n раз, отсюда можно сделать вывод, что всего совершается проверок.

### Сортировка «вставками»

На вход алгоритма подается число элементов в массиве и указатель на начало массива. Данный алгоритм помещает как бы разделяет исходный массив на отсортированную и неотсортированную часть, а затем по очереди помещает в отсортированную часть массива оставшиеся элементы массива на соответствующие им места уже в упорядоченном виде, то есть вставляет каждый новый элемент на свое место в уже отсортированном массиве по очереди сравнивая его с упорядоченной последовательностью до тех пор, пока он не займет нужное место.

|  |
| --- |
| void InserionSort(int N, double \*arr)  {    double s = 0.0;    int j = 0;  *for*(int i = 1; i < N; i++)    {      j = i - 1;  *while*(j >= 0 && (arr[j] > arr[j + 1]))      {        s = arr[j];        arr[j] = arr[j + 1];        arr[j + 1] = s;        j--;      }    }  } |

В худшем случае, как и в среднем, сортировка вставками затрачивает времени на свою работу. Данный алгоритм по совей сложности схож с сортировкой пузырьком, ведь в нем производятся схожие по структуре процессы: сравнение элементов и их сдвиги. Однако на практике такой алгоритм нередко оказывается быстрее сортировки пузырьком, так как в ней вне зависимости от того какой был исходный массив выполняется n проверок для каждого элемента с первого до( n – 1)-ого, в то время как сортировки вставкой выполняют проверки для новых элементов до тех пор, пока они не займут своего места и в изначально частично упорядоченном массиве могу работать значительно быстрее.

### Быстрая сортировка

Как и в других случаях, на вход этому алгоритму подается число элементов и указатель на начало массива. Однако в отличии от двух других сортировок он работает по совершенно другому принципу. Для начала выбирается средний по положению элемент входного массива – ключ. Затем просматриваются элементы массива до ключа и после него. Как только слева находится элемент больше ключа, и справа находится элемент меньше ключа, то они меняются местами. Это происходит до тек пор, пока все элементы слева от ключа не будут меньше него, а все справа не станут больше. Таким образом мы имеем два массива: массив с числами меньше ключа и массив с числами больше ключа. Затем происходит рекурсивный вызов сортировки для каждого из этих массивов. Этот процесс происходит до тех пор, пока все элементы массива не окажутся отсортированными, то есть пока один из массивов на некоторой итерации не станет одноэлементным.

|  |
| --- |
| void QuickSort(int N, double \*arr)  {    long i =0, j = N - 1;    double key = 0.0, s = 0.0;    key = arr[(int)(N/2)];  *while*(i <= j)    {  *while*(arr[i] < key) i++;  *while*(arr[j] > key) j--;  *if* (i <= j)      {        s = arr[i];        arr[i] = arr[j];        arr[j] = s;        i++;        j--;      }    }  *if* (j > 0) QuickSort(j, arr);  *if* (i < N) QuickSort(N - i, arr + i);  } |

Данный алгоритм принципиально отличается от двух других. Это касается и его сложности. В сравнении с квадратичными сортировками (пузырьковая, вставками), быстрая сортировка более чем оправдывает свое название, ведь её худшее время работы , в то время как среднее время работы составляет , что быстрее чем в случае с квадратичными сортировками.

# Эксперименты

В качестве эксперимента посмотрим на практике какой из алгоритмов сортировки работает быстрее с набором случайной сгенерированных чисел.

Для этого с помощью первой программы создадим текстовый файл с большим количеством чисел (см. рис. 10).

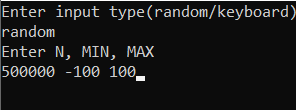


Рис. 10 Параметры генерации чисел

Затем запустим программу два и с её помощью отсортируем элементы в текстовом файле с помощью сортировки пузырьком (см. рис. 11)

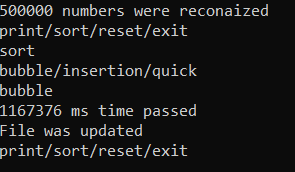


Рис. 11 Запуск и завершение сортировки

Сортировка пузырьком продлилась примерно 1167 секунд. После её завершения возвращаем файл к первоначальному виду и сортируем его с помощью сортировки вставкой. Затем повторяем это процедуру для быстрой сортировки.

Полученные в экспериментах данные приведены в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип сортировки** | **Сложность** | **Размер входных данных** | **Время работы, с** |
| Пузырьком |  | 500000 | ≈1167 |
| Вставками |  | 500000 | ≈264 |
| Быстрая |  | 500000 | ≈0,062 |
| Пузырьком |  | 5000 | ≈0,063 |
| Вставками |  | 5000 | ≈0,031 |
| Быстрая |  | 5000 |  |

Таблица 1. Сравнение времени работы сортировок

Все эти данные были получены на одной и той же машине, при одном и том же диапазоне генерации случайных чисел.

# Заключение

В ходе данной лабораторной работы, на языке программирования «С» были написаны программы позволяющие работать с текстовыми файлами: заполнять их числами и получать числа из них. Реализованы три алгоритма сортировки числовых массивов: сортировка пузырьком, вставками и быстрая, а также был проведен эксперимент по сравнению их быстродействия.

По результатам экспериментов можно сделать вывод о том, что самая быстрая сортировка, как при больших, так и при маленьких входных данных, это «быстрая» сортировка, что соответствует теоретическим данным. Далее по быстродействию идет сортировка «вставками», которая выполнилась вдвое быстрее пузырьковой при небольшом размере входных данных, и почти в пять раз быстрее при значительном количестве обрабатываемых данных. Ну и наиболее медленной оказалась сортировка «пузырьком», которая выполнялась дольше всех других сортировок при одинаковом характере и количестве входных данных.

# Список литературы

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ — 2-е. — М.: Вильямс, 2005. — 1296 с.
2. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск / под ред. В. Т. Тертышного (гл. 5) и. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с.
3. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО, 2014. –– 320 с.
4. Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с.

# Приложение 1

## Приложение 1.1

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  void WriteRand(FILE\* file,int n, double min, double max)  {    double num = 0.0;  *for*(int k = 0; k < n; k++)    {      num = ((double)rand())/RAND\_MAX \*(max - min) + min;      fprintf(file, "%lf\n", num);    }  }  int StrComp(char \*str\_1, char \*str\_2) *//сравнивает 1-е со вторым*  {    int i = 0;    int m =0 ;  *while*(str\_1[m] != 0) m++;  *while*((str\_1[i] == str\_2[i]) && str\_1[i] != 0)      i++;  *if*(m == i) *return* 1;  *return* 0;  }  int main()  {    FILE \*fp = 0;    fp = fopen("textFile.txt", "w");    int n = 0;    double min, max = 0.0;    printf("Enter input type(random/keyboard)\n");    char str[9];    scanf("%s", str);  *if* (StrComp(str, "keyboard"))    {      double num = 0.0;  *while*(scanf("%lf\n", &num) != 0)      {        fprintf(fp, "%lf\n", num);      }      fclose(fp);  *return* 0;    }    printf("Enter N, MIN, MAX\n");    scanf("%d %lf %lf", &n, &min, &max);  *if*(n<=0 || (max - min)<0)    {      printf("Error: invalid input!");  *return* 0;    }    WriteRand(fp, n, min, max);    fclose(fp);  *return* 0;  } |

## Приложение 1.2

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#include* <conio.h>  *#include* <time.h>  int StrComp(char \*str\_1, char \*str\_2)  {    int i = 0;    int m = 0 ;  *while*(str\_1[m] != 0) m++;  *while*((str\_1[i] == str\_2[i]) && str\_1[i] != 0)      i++;  *if*(m == i) *return* 1;  *return* 0;  }  void BubbleSort(int N, double \*arr)  {    double s = 0.0;  *for*(int i = 0; i < N - 1; i++)    {  *for*(int j = 0; j < N-1; j++)      {  *if* (arr[j] > arr[j + 1])        {          s = arr[j];          arr[j] = arr[j + 1];          arr[j + 1] = s;        }      }    }  }  void InserionSort(int N, double \*arr)  {    double s = 0.0;    int j = 0;  *for*(int i = 1; i < N; i++)    {      j = i - 1;  *while*(j >= 0 && (arr[j] > arr[j + 1]))      {        s = arr[j];        arr[j] = arr[j + 1];        arr[j + 1] = s;        j--;      }    }  }  void QuickSort(int N, double \*arr)  {    long i =0, j = N - 1;    double key = 0.0, s = 0.0;    key = arr[(int)(N/2)];  *while*(i <= j)    {  *while*(arr[i] < key) i++;  *while*(arr[j] > key) j--;  *if* (i <= j)      {        s = arr[i];        arr[i] = arr[j];        arr[j] = s;        i++;        j--;      }    }  *if* (j > 0) QuickSort(j, arr);  *if* (i < N) QuickSort(N - i, arr + i);  }  int main()  {    char str[32];    int N = 0, l = 0;    double \*fl,\*num, ch = 0;    FILE \*fs = 0;    fs = fopen("textFile.txt", "r");  *if* (fs == NULL)    {     printf("Error: cannot open file\n");  *return* 0;    }  *while*(fscanf(fs, "%lf\n", &ch) != EOF) N++;    num = (double\*)malloc(sizeof(double)\*N);    fl = (double\*)malloc(sizeof(double)\*N);    freopen("textFile.txt","r", fs);  *for*(int k = 0; k < N; k++)    {      fscanf(fs, "%lf\n", &num[k]);      fl[k] = num[k];    }    printf("%d numbers were reconaized\n", N);  *while*(1)    {      printf("print/sort/reset/exit\n");      scanf("%s", str);  *if* (StrComp(str, "print"))      {        double temp = 0.0;        freopen("textFile.txt", "r", fs);  *for*(int k = 0; k < N; k++)        {          fscanf(fs, "%lf\n", &temp);          printf("%lf\n", temp);  *continue*;       }      }  *else* *if* (StrComp(str, "sort"))      {        printf("bubble/insertion/quick\n");        scanf("%s", str);        unsigned time = clock();  *if* (StrComp(str, "bubble"))         {           BubbleSort(N, num);        }  *else* *if* (StrComp(str, "insertion"))        {          InserionSort(N, num);        }  *else* *if* (StrComp(str, "quick"))        {          QuickSort(N, num);        }  *else*        {          printf("Error: invalid input!\n");  *continue*;        }        time = clock() - time;        printf("%d ms time passed\n", time);        freopen("textFile.txt", "w", fs);  *for*(int i = 0; i < N; i++)        {          fprintf(fs, "%lf\n", num[i]);        }        printf("File was updated\n");  *continue*;      }  *else* *if* (StrComp(str, "reset"))      {        freopen("textFile.txt", "w", fs);  *for*(int i = 0; i < N; i++)        {          fprintf(fs, "%lf\n", fl[i]);          num[i] = fl[i];        }        printf("File was restored\n");  *continue*;      }  *else* *if* (StrComp(str, "exit"))      {  *break*;      }    }      fclose(fs);  *return* 0;  } |