**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

**«Сибирский государственный университет науки и технологий   
имени академика М.Ф. Решетнева»**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Алгоритмы и структуры данных

|  |
| --- |
| Сравнительный анализ эффективности методов сортировки массивов данных в оперативной памяти. |

Руководитель В. В. Тынченко

подпись, дата инициалы, фамилия

Обучающийся БПИ22-02, 221219040 К.В. Трифонов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2023 г.

# Цель работы:

Изучение алгоритмов сортировки массивов данных в ОП, особенностей их программной реализации и эффективности работы на различных наборах исходных данных.

# Общая постановка задачи.

Разработать и реализовать программу, предназначенную для исследования времени работы и поведения методов сортировки, указанных в варианте задания.

Для решения поставленной задачи применить объектно-ориентированный подход. Разработать шаблон класса «Одномерный массив», включающий в себя необходимый минимум методов, обеспечивающий полноценное функционирование объектов указанного класса в рамках данной программы: конструкторы (по умолчанию, с параметрами, копирования), деструктор, методы ввода данных в массив с клавиатуры, заполнения случайными значениями из заданного диапазона, вывода массива на экран, вывода массива в файл, сортировки алгоритмами, указанными в варианте задания.

Провести исследование параметров работы, указанных в варианте задания методов сортировки на различных наборах данных. Представить отчет, содержащий результаты исследования и полученные выводы.

# Требования к функциональным возможностям программы.

Программа должна содержать меню, позволяющее выбирать один из двух режимов работы программы:

1. сортировка одного массива, сформированного случайным образом;
2. режим накопления статистических данных

В **первом режиме** требуется предоставить пользователю следующие возможности:

* задавать размер массива;
* указывать диапазон значений элементов массива;
* выбирать метод сортировки массива.

Результаты работы программы в данном режиме:

* вывести на экран количество сравнений и перестановок элементов массива;
* исходный и отсортированный массивы вывести в файл с указанным именем.

Во **втором режиме** пользователь должен иметь возможность:

* выбирать способ формирования элементов массива (случайные значения, упорядоченная последовательность значений, значения расположены в обратном порядке);
* задавать диапазон и шаг изменения размеров массива;
* выбирать метод сортировки массива.

Результаты работы программы в данном режиме:

Для каждого значения размера формируется новый массив и сортируется выбранным методом. В файл с указанным именем выводятся значения времени сортировки для каждого количества элементов в массиве.

# Вариант №21.

Программно реализовать следующие методы сортировки данных в оперативной памяти: сортировка прямыми обменами; сортировка прямыми включениями.

Оценить быстродействие указанных методов и степень естественности их поведения.

## Код **программы:**

# Описание класса *array:*

#ifndef ARRAY\_H

#define ARRAY\_H

*template*<*typename* T>

*class* array{

*int* size;

        T \*arr;

*public:*

        array();

        array(*int* *n*);

        array(*const* array *&A*);

        ~array(){delete arr;}

*void* setS(*int* *S*); *//Задать длину*

*int* getS(); *//Вернуть длину*

*void* fill(); *//Ввод с клавиатуры*

*void* fillR(*int* *n1*, *int* *n2*); *//Рандомные в диапазоне*

*void* fillU(); *//Упорядоченные*

*void* fillUr(); *//Упорядоченные в обратном порядке*

*void* get(); *//Вывод на экран*

*void* get\_file(ofstream *&fin*); *//Вывод в поток*

*void* shakerSort(); *//Сортировка объекта прямыми обменами*

*void* insertSort(); *//Сортировка объекта прямыми включениями*

};

#endif *//ARRAY\_H*

# Конструкторы:

По умолчанию:

*template*<*typename* T>

array<T>::array(){

    size = 10;

    arr = new T[size];

}

С параметрами (параметр длины массива):

*template*<*typename* T>

array<T>::array(*int* *n*){

    size = *n*;

    arr = new T[size];

}

Копий:

*template*<*typename* T>

array<T>::array(*const* array &*A*){

    size = *A*.size;

    arr = new T[size];

    for (*int* i = 0; i < size; i++){

        arr[i] = *A*.arr[i];

    }

}

# Методы ввода-вывода:

Заполнение клавиатуры:

*template*<*typename* T>

*void* array<T>::fill(){

    for (*int* i = 0; i < size; i++){

        cout<<"Введите элемент №"<<i+1<<": \n";

        cin>>arr[i];

    }

}

Заполнение с помощью *time.h:*

*template*<*typename* T>

*void* array<T>::fillR(*int* *n1*, *int* *n2*){

    srand(time(0));

    for (*int* i = 0; i < size; i++){

        arr[i] = *n1* + rand()%(*n2*-*n1*+1);

    }

}

Упорядоченные значения:

*template*<*typename* T>

*void* array<T>::fillU(){

    for (*int* i = 0; i < size; i++){

        arr[i] = i+1;

    }

}

*//Упорядоченные в обратном порядке*

*template*<*typename* T>

*void* array<T>::fillUr(){

    for (*int* i = 0; i < size; i++){

        arr[i] = size-i;

    }

}

Вывод на экран:

*template*<*typename* T>

*void* array<T>::get(){

    cout<<"Массив длинной "<<size<<":\n";

    for (*int* i = 0; i < size; i++){

        cout<<arr[i]<<" ";

    }

    cout<<"\n";

}

Вывод в поток:

*template*<*typename* T>

*void* array<T>::get\_file(ofstream *&fin*){

    for (*int* i = 0; i < size; i++){

*fin*<<arr[i]<<" ";

    }

*fin*<<"\n";

}

Задание поля длины:

*template*<*typename* T>

*void* array<T>::setS(*int* *S*){

    size = *S*;

    arr = new T[size];

}

*При этом, массив создаётся заново, во избежание проблем с потерей данных.*

Возврат поля длины:

*template*<*typename* T>

*int* array<T>::getS(){

    return size;

}

*Используется для быстрого обращения к полю в массиве объектов.*

# Методы сортировки (согласно варианту задания):

Сортировка прямыми обменами:

*template*<*class* T>

*void* array<T>::shakerSort() {

*long* j, k = size-1;

*long* lb=1, ub = size-1; *// границы неотсортированной части массива*

  T x;

  do {

*// проход снизу вверх*

    for( j=ub; j>0; j-- ) {

      if ( arr[j-1] > arr[j] ) {

        x=arr[j-1]; arr[j-1]=arr[j]; arr[j]=x;

        k=j;

      }

    }

    lb = k+1;

*// проход сверху вниз*

    for (j=1; j<=ub; j++) {

      if ( arr[j-1] > arr[j] ) {

        x=arr[j-1]; arr[j-1]=arr[j]; arr[j]=x;

        k=j;

      }

    }

    ub = k-1;

  } while ( lb < ub );

}

Сортировка прямыми включениями:

*template*<*class* T>

*void* array<T>::insertSort() {

  T x;

*long* i, j;

  for ( i=0; i < size; i++) { *// цикл проходов, i - номер прохода*

    x = arr[i];

*// поиск места элемента в готовой последовательности*

    for ( j=i-1; j>=0 && arr[j] > x; j--)

      arr[j+1] = arr[j]; *// сдвигаем элемент направо, пока не дошли*

*// место найдено, вставить элемент*

    arr[j+1] = x;

  }

}

# Меню файла main.cpp:



Первый режим работы программы:

case 1:{

*int* n;

                cout<<"Задайте размер массива: ";

                cin>>n;

                array<*int*> A(n);

                cout<<"Введите диапозон генерации от n1 до n2 (n1\_n2):";

*int* n1, n2;

                cin>>n1>>n2;

                A.fillR(n1,n2);

                A.get();

                array<*int*> B(A);

                B.shakerSort();

                cout<<"Отсортированный массив: \n";

                B.get();

                string name;

                cout<<"Укажите имя файла вывода: \n";

                cin>>name;

                ofstream fout(name);

                if (!fout.is\_open()){

                    cout<<"Ошибка открытия!";

                }

                else{

                    A.get\_file(fout);

                    B.get\_file(fout);

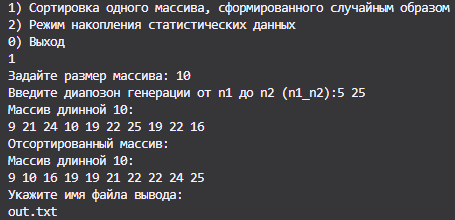
                }

                break;

            }

Создаётся массив длинной n со случайными значениями от n1 до n2, его копия сортируется и выводится на экран, далее в пользовательский файл выводится прежний массив и отсортированная копия.

Пример работы режима с заполнением массива длинной 10 эл случайными числами от 5 до 25 (вывод данных в консоль и пользовательский файл out.txt):



**

Второй режим работы программы:

 case 2:{

*int* in2,in3;

*int* a1,b1,l;

                cout<<"1) Заполнение случайными значениями\n"

                <<"2) Заполнение упорядоченными значениями\n"

                <<"3) Заполнение обратноупорядоченными значениями\n";

                cin>>in2;

                cout<<"Введите диапазон (от \_ до \_) и шаг через пробел:\n";

                cin>>a1>>b1>>l;

                array<*int*> \*A;

*int* N = (b1-a1)/l+1;

                A = new array<*int*> [N];

                switch (in2){

                    case 1:{

                        for (*int* i = 0; i<N-1; i++){

                            A[i].setS(((i)\*l)+a1);

                            A[i].fillR(1,20);

                        }

                        break;

                    }

                    case 2:{

                        for (*int* i = 0; i<N-1; i++){

                            A[i].setS(((i)\*l)+a1);

                            A[i].fillU();

                        }

                        break;

                    }

                    case 3:{

                        for (*int* i = 0; i<N-1; i++){

                            A[i].setS(((i)\*l)+a1);

                            A[i].fillUr();

                        }

                        break;

                    }

                }

                cout<<"Всего заполнено "<<N-1<<" массивов.\n";

Создание массива объектов пользовательской длинны N = (b1-a1)/L+1, где b1 и a1 границы диапазона длин массивов и L – шаг. Последующее заполнение в зависимости от выбора пользователя, используя методы класса.

Далее пользователь выбирает метод сортировки:

cout<<"1) Cортировка прямыми обменами shaker sort\n"

                <<"2) Cортировка прямыми включениями insert sort\n";

                cin>>in2;

                ofstream fout("results.txt");

*int* t1 = clock();

                switch (in2){

                    case 1:{

                        for (*int* i = 0; i<N-1; i++){

*int* t\_1 = clock();

                            A[i].shakerSort();

*int* t\_2 = clock();

*int* ans = t\_2 - t\_1;

                            cout<<A[i].getS()<<" эл отсортированы за "<<ans<<"ms\n";

                            fout<<A[i].getS()<<" "<<ans<<"\n";

                        }

                        break;

                    }

                    case 2:{

                        for (*int* i = 0; i<N-1; i++){

*int* t\_1 = clock();

                            A[i].insertSort();

*int* t\_2 = clock();

*int* ans = t\_2 - t\_1;

                            cout<<A[i].getS()<<" эл отсортированы за "<<ans<<"ms\n";

                            fout<<A[i].getS()<<" "<<ans<<"\n";

                        }

                        break;

                    }

                }

*int* t2 = clock();

*int* answ = (t2-t1);

                cout<<"Общее время: "<<answ<<"ms\n";

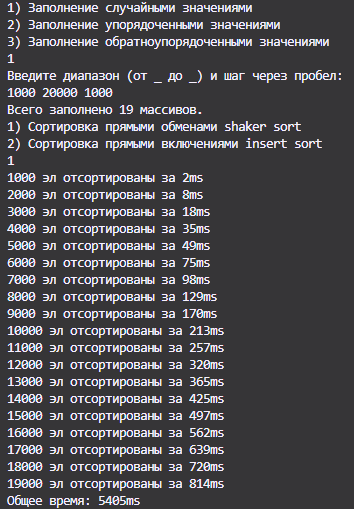
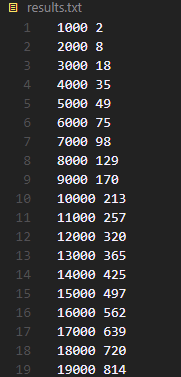
                fout.close();

                break;

            }

Записывается время t1, затем каждый объект массива сортируется методом класса, в зависимости от выбора пользователя, и выводится время на экран и в файл *results.txt*, за которое он был отсортирован, далее записывается время t2 и выводится на экран общее время, за которое были отсортированы все объекты массива.

Пример работы режима в диапазоне от 1000 до 20000 эл с шагом 1000 эл (вывод данных в консоль и в файл results.txt):

# Исследование работы алгоритмов shaker sort и insert sort

График зависимости времени от длины массива, заполненного случайными значениями:

Асимптотика у shaker sort как и у insert sort в среднем и худшем случае – O(n2), в лучшем случае – O(n). Однако insert sort быстрее shaker sort в среднем в три раза. Это достигается за счёт меньшего числа перестановок, т.к. алгоритм сразу вставляет значение в нужное место, а не перетягивает его через весь массив.

График зависимости времени сортировки shaker sort от длины массива, заполненного случайными, упорядоченными и обратно упорядоченными значениями:

Время на сортировку упорядоченных в обратном порядке значений увеличено, по сравнению со случайными значениями, а сортировка упорядоченных массивов, наоборот занимает 1-2ms, затраченные на 1 проход по массиву. Следовательно алгоритм ведет себя естественно.

График зависимости времени сортировки insert sort от длины массива, заполненного случайными, упорядоченными и обратно упорядоченными значениями:

Расхождение с временем сортировки случайных и обратно упорядоченных значений более стремительно, по сравнению с shaker sort (почти в 2 раза), а сортировка упорядоченных массивов, как и у shaker sort занимает 1-2ms, затраченные на 1 проход по массиву. Следовательно алгоритм ведет себя естественно.

# Вывод

Несмотря на родство shaker sort и insert sort, второй является более эффективным, за счёт меньшего числа перестановок элементов массива (на длине 20000 эл у shaker sort около 150 миллионов сравнений, у insert sort около 85 миллионов). Также insert sort имеет большую степень естественности по сравнению с shaker sort.