ПРАКТИЧЕСКАЯ РАБОТА № 8

(4 академических часа)

Тема: Методы сортировки.

Цель задания:

Получение практических навыков по составлению алгоритмов и программ сортировки данных в массиве.

Методика выполнения работы.

Задание 1.

Разработать программу, выполняющую сортировку данных одномерного массива с использованием 3-х видов сортировки (сортировка выбором, сортировка вставкой и сортировка методом «пузырька»). Программа должна включать:

1) ввод исходных данных;
2) вывод массива исходных данных;
3) обработку данных (сортировка);
4) осуществление метода двоичного поиска;
5) вывод результатов выполнения программы;
6) для тестирования программы сформировать исходные данные таким образом, чтобы проверить каждый вариант альтернативы каждого разветвления алгоритма.

Задание 2. Разработать программу в соответствии с вариантом.

Теоретическая часть.

Сортировка – это расположение элементов множества данных в определенной последовательности: в порядке возрастания или убывания ключей – признаков сортировки. Множеством для сортировки могут быть элементы массивов или записи файлов. Ключом может быть элемент записи, по которому производится сортировка или любое сочетание поисковых признаков, представленное логическим выражением.

Сортировка нужна для того, чтобы:

1) обеспечить более эффективную обработку в больших наборах данных (например, поиск);
2) представить человеку массивы данных в форме, удобной для анализа;
3) построить гистограммы распределения данных.

Критериями оценки различных методов сортировки могут быть:

1) количество сравнений и пересылок записей;
2) время сортировки заданного объема данных;
3) требуемый объем ОП для сортировки;
4) сложность алгоритмов и программ сортировки.

Методы сортировки можно разделить на следующие группы:

1) сортировка вставкой (включением);
2) сортировка выбором (выделением);
3) сортировка обменом (“пузырьковая сортировка”);
4) сортировка распределением;
5) сортировка подсчетом;
6) сортировка слиянием.

1. Сортировка вставкой.

Принцип метода:

Массив разделяется на две части: отсортированную и не отсортированную. Элементы из не отсортированной части поочередно выбираются и вставляются в отсортированную часть так, чтобы не нарушить в ней упорядоченность элементов. В начале работы алгоритма в качестве отсортированной части массива принимают только один первый элемент, а в качестве не отсортированной части - все остальные элементы.
Таким образом, алгоритм будет состоять из n-1 прохода (n –размерность массива), каждый из которых будет включать четыре действия:

- взятие очередного i-го не отсортированного элемента и сохранение его в дополнительной переменной;
- поиск позиции j в отсортированной части массива, в которой присутствие взятого элемента не нарушит упорядоченности элементов;
- сдвиг элементов массива от (i-1)-го до j-го впрво, чтобы освободить найденную позицию вставки;
- вставка взятого элемента в найденную j-ю позицию.

```
…
  int cur;   // текущий элемент массива
   int j;      // текущий индекс
   for (int i = 1; i < array.Length; i++)
     {
        cur = array[i];
        j = i;
        while ((j > 0) && (cur < array[j - 1]))
        {
           array[j] = array[j - 1];
           j--;
        }
        array[j] = cur;
     }
  ...
```

2. Сортировка выбором.

Принцип метода:
Находим (выбираем) в массиве элемент с минимальным значением на интервале от 1-го до n-го (последнего) элемента и меняем его местами с первым элементом. На втором шаге находим элемент с минимальным значением на интервале от 2-го до n-го элемента и меняем его местами со вторым элементом. И так далее для всех элементов до (n-1)-го.

```
...
int indx; //переменная для хранения индекса минимального элемента массива
for (int i = 0; i < intArray.Length; i++) //проходим по массиву с начала и до конца
{
   indx = i; //считаем, что минимальный элемент имеет текущий индекс
   for (int j = i; j < intArray.Length; j++)       //ищем минимальный элемент
 {
      if (intArray[j] < intArray[indx])
      {
         indx = j;           //нашли в массиве число меньше, чем intArray[indx]
       }
   }
    if (intArray[indx] == intArray[i]) //если минимальный элемент равен текущему
значению - ничего не меняем
      continue;
   //меняем местами минимальный элемент и первый в неотсортированной части
   int temp = intArray[i];    //временная переменная
```

```
    intArray[i] = intArray[indx];
    intArray[indx] = temp;
  }
  ...
```

3. Сортировка обменом (“пузырьковая сортировка”)

Обменные сортировки предусматривают систематический обмен местами между элементами пар значений, в которых нарушена упорядоченность, до тех пор, пока таких элементов не останется. То есть если два элемента расположены неупорядоченно, то они меняются местами. К обменным сортировкам относят:

- метод пузырькового всплытия;
- быструю сортировку (обменную сортировку с разделением);
- обменную сортировку со слиянием;
- поразрядную обменную сортировку.

Принцип метода пузырьковой сортировки:

Слева направо поочередно сравниваются два соседних элемента, и если их взаиморасположение не соответствует заданному условию упорядоченности, то они меняются местами. Далее берутся два следующих соседних элемента и так далее до конца массива.

После одного такого прохода на последней n-ой позиции массива будет стоять максимальный элемент (“всплыл” первый “пузырек”). Поскольку максимальный элемент уже стоит на своей последней позиции, то второй проход обмена выполняется до (n-1)-го элемента. И так далее. Всего требуется (n-1) проход.

```
…
int temp;
        for (int i = 0; i < mas.Length; i++)
        {
           for (int j = i + 1; j < mas.Length; j++)
           {
              if (mas[i] > mas[j])
              {
                 temp = mas[i];
                 mas[i] = mas[j];
                 mas[j] = temp;
              }
           }
        }
...
```

4. Двоичный поиск (бинарный поиск, поиск делением пополам)

Алгоритм двоичного поиска допустимо использовать для нахождения заданного элемента массива, упорядоченного по неубыванию.

Принцип двоичного поиска:

1) Исходный массив делится пополам и для сравнения выбирается средний элемент. Если он совпадает с искомым, то поиск заканчивается. Если же средний элемент меньше искомого, то все элементы левее его также будут меньше искомого. Следовательно, их можно исключить из зоны дальнейшего поиска, оставив только правую часть массива. Аналогично, если средний элемент больше искомого, то отбрасывается правая часть, а левая остается.
2) На втором этапе выполняются аналогичные действия над оставшейся половиной массива. В результате после второго этапа остается ¼ часть массива.
3) И так далее, пока или элемент будет найден, или длина зоны поиска станет равной нулю. В последнем случае искомый элемент найден не будет.

Один из вариантов процедуры, реализующей метод двоичного поиска, будет иметь следующий вид:

```
...
b=Convert.ToInt32.Console.ReadLine ();       // критерий поиска
int L = 0;                          // левая граница
int R = n - 1;                      // правая граница (n – размерность массива)
int k = (R + L) / 2;                //середина
Int F =0;                          //флаг – результат поиска
While (L<=R)
    {
      k = (R + L) / 2;
      if (a[k] == b)
       {
          Console.WriteLine ("Элемент найден на позиции - "+k);
           F=1;
          Break;
        }
else
      if (a[k] < b)
        L = k+1;        // поиск в правой половине
      else
        R = k-1;         // поиск в левой половине
    }
    if (F==0)
      {
         Console.WriteLine("Элемент не найден");
      }
...
```

Варианты задания2: разработка программ с использованием различных методов сортировки данных.

Вариант 1.

Создать целочисленную матрицу A(n,n) и выполнить сортировку элементов главной диагонали методом вставки. Отсортированную последовательность вывести на экран.

Вариант 2.

Создать матрицу вещественных элементов A(n,n) и выполнить сортировку отрицательных элементов методом выбора. Отсортированную последовательность вывести на экран.

Вариант 3.

Создать матрицу вещественных элементов A(n,n) и выполнить сортировку элементов, расположенных под главной диагональю методом «пузырька». Отсортированную последовательность вывести на экран.

Вариант 4.

Создать целочисленный массив A(m) и выполнить сортировку элементов, находящихся на четных местах методом «пузырька». Отсортированную последовательность вывести на экран.

Вариант 5.

Создать массив строкового типа A(n) и выполнить сортировку элементов, больших заданного s методом «пузырька». Отсортированную последовательность вывести на экран.

Вариант 6.
Создать матрицу A(n,m) и выполнить сортировку элементов, меньших заданного b методом выбора. Отсортированную последовательность вывести на экран.

Вариант 7.
Создать массив целочисленного типа A(n) и выполнить поиск заданного элемента s методом двоичного поиска. На экран вывести найденный элемент, либо запись, что элемент не найден.

Вариант 8.
Создать матрицу A(n,m) вещественных значений и выполнить сортировку четных элементов методом выбора. Отсортированную последовательность вывести на экран.

Вариант 9.
Создать целочисленный массив A(m) и выполнить сортировку элементов, находящихся на нечетных местах методом выбора. Отсортированную последовательность вывести на экран.

Вариант 10.
Создать матрицу вещественных элементов A(n,n) и выполнить сортировку положительных элементов методом «пузырька». Отсортированную последовательность вывести на экран.

Вариант 11.
Создать целочисленный массив A(m) и выполнить сортировку элементов с третьего по пятнадцатый методом вставкой. Отсортированную последовательность вывести на экран.

Вариант 12.
Создать матрицу вещественных элементов A(n,n) и выполнить сортировку элементов, расположенных над главной диагональю методом «пузырька». Отсортированную последовательность вывести на экран.

Вариант 13.
Создать матрицу A(n,m) вещественных значений и выполнить поиск заданного элемента s методом двоичного поиска. На экран вывести найденный элемент, либо запись, что элемент не найден.

Вариант 14.
Создать матрицу вещественных элементов A(n,m) и выполнить сортировку элементов, больше заданного a и меньше заданного b методом «пузырька». Отсортированную последовательность вывести на экран.

Вариант 15.
Создать целочисленный массив A(m) и выполнить проверку: является ли введенный массив отсортированным. Отсортированную последовательность вывести на экран.