### Семинар 2. Одномерные массивы. Сортировка массивов. Оценка вычислительной сложности алгоритма

*Сортировка* – это процесс упорядочивания информации по определенному признаку. Цель сортировки – облегчение последующего поиска элементов. Это почти универсальный вид обработки информации, с которым мы встречаемся в жизни повсеместно.

Существует огромное количество методов сортировки и, соответственно, алгоритмов их реализации. Часть из этих алгоритмов в некотором смысле оптимальна, другие имеют свои достоинства и недостатки. Поэтому, прежде чем использовать алгоритм, реализующий какой либо метод, следует выполнить анализ его производительности в конкретных условиях.

В качестве оценки производительности методов обычно используют *функциональную зависимость* времени работы программы от размерности исходного массива t(n). При анализе алгоритмов в первую очередь интерес представляет характер зависимости при достаточно больших значениях размерности задачи (n→∞).

В математике характер зависимости часто определяют ее порядком. *Порядком* некоторой функции t(n) при достаточно больших n называют другую функцию g(n), такую что

t(n)

lim -------- = const≠ 0 .

 n→∞ g(n)

Это обозначается как t(n)=O[g(n)].

Например, для полинома f(n) = 2n4-3n3+5n-6, порядком является полином n4, или f(n)=O(n4), так как

2n4-3n3+5n-6

lim-------------- = 2 .

n→∞n4

В программировании порядок зависимости времени работы программы, реализующей некоторый метод, от размерности исходных данных n называют *вычислительной сложностью данного метода*. Так вычислительная сложность O(const) означает, что время решения задачи с использованием данного метода не зависит от размерности задачи, O(n) – означает, что время работы пропорционально размерности задачи, O(n2) – означает, что время работы пропорционально квадрату размерности задачи и т. д.

*Примечание*. В некоторых случаях целесообразно различать вычислительную сложность метода и его конкретной реализации, так как неудачная реализация может существенно ухудшить предполагаемую вычислительную сложность метода.

Временную сложность можно оценить, используя в качестве единиц измерения временные единицы (мкс., сек. и т. д.). Но можно – используя *время выполнения* основных, характеризующих процесс *операций*, количество которых соответствует количеству итераций (повторений) цикла, например, операций сравнения, операций пересылки. Время выполнения этих операций можно считать постоянным. Следовательно, функциональная зависимость количества выполняемых операций от размерности задачи по характеру будет совпадать с временной зависимостью.

На практике интересны методы сортировки, которые позволяют экономно использовать оперативную память, поэтому целесообразно рассмотреть только методы, не требующие использования дополнительных массивов. Такие методы в практике программирования называют *прямыми*. Самыми простыми из прямых методов являются:

1) метод выбора;

2) метод вставки;

3) метод обменов (метод пузырька).

Рассмотрим эти методы на конкретном примере.

**Задание.** Разработать программу сортировки элементов массива A(n), где n≤50, используя метод выбора, метод вставки и метод обменов. Оценить эффективность применения указанных методов.

1. **Метод выбора.**

Сортировка посредством выбора представляет собой один из самых простых методов сортировки. Он предполагает следующую последовательность действий.

Сначала находим минимальный элемент массива. Найденный элемент меняем местами с первым элементом. Затем повторяем процесс с n-1 элементами, начиная со второго, потом с n-2 элементами, начиная с третьего и т.д. до тех пор, пока не останется один, самый большой элемент массива (рисунок 1).



**Рисунок 1.** Сортировка выбором

Алгоритм сортировки выбором приведен на рисунке 2.



**Рисунок 2 –** Схема алгоритма сортировки выбором

Алгоритм приведен для общего случая. При решении задачи на языке С/С++ нужно учесть особенности индексации массивов ( индекс массива начинается от 0).

Оценим временную сложность данного метода, используя в качестве основной операции операцию сравнения.

Для поиска минимального элемента в каждом проходе потребуется выполнить: n-1, n-2, ..., 1 операций сравнения, т.е. всего n(n-1)/2 операций сравнения. Следовательно, вычислительная сложность данного метода O(n2). Причем время сортировки не зависит от исходного порядка элементов.

Пример программы сортировки обменом приведен ниже

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

**inta[50];**

**inti,j,imin,min,n;**

**intmain()**

**{**

**srand((unsigned)time(NULL));//Настройка датчика случайных чисел**

**puts("Input n<=50");**

**scanf("%d",&n);**

**printf("input %3d elem. massiva\n",n);**

 **for(i=0;i<n;i++) // scanf("%d",&a[i]);// вводсклавиатуры**

**a[i]=rand()%30-12; // формирование датчиком случайных чисел**

**printf("\n");**

**puts("InputedMassiv");**

 **for(i=0;i<n;i++)printf("%4d",a[i]);**

**printf("\n");**

**for(i=0;i<n-1;i++)**

**{ min=a[i];**

**imin=i;**

 **for(j=i+1;j<n;j++)**

 **if (a[j]<min)**

 **{min=a[j];**

 **imin=j;**

 **}**

 **a[imin]=a[i];**

 **a[i]=min;**

 **}**

**puts("Sorted Massiv");**

**for(i=0;i<n;i++)printf("%4d",a[i]);**

**printf("\n");**

 **return 0;**

**}**

**Пример работы программы**

1. **Метод вставки*.***

Сортировку вставками можно описать следующим образом. В исходном состоянии считают, что сортируемая последовательность состоит из двух последовательностей: уже сортированной (она на первом шаге состоит из единственного – первого элемента) и последовательности элементов, которые еще необходимо сортировать. На каждом шаге из сортируемой последовательности извлекается элемент и вставляется в первую последовательность так, чтобы она оставалась сортированной. Поиск места вставки осуществляют с конца, сравнивая вставляемый элемент ai с очередным элементом сортированной последовательности aj. Если элемент aiбольше aj, его вставляют вместо aj+1, иначе сдвигают aj вправо и уменьшают j на единицу. Поиск места вставки завершают, если элемент вставлен или достигнут левый конец массива. В последнем случае элемент ai вставляют на первое место (рисунок 3).



**Рисунок 3-**Сортировка вставками

Разрабатывая алгоритм, избавимся от проверки достижения начала массива. Прием, который позволяет отменить эту проверку, носит название «проверки с барьером». С использованием этого приема проверка организуется так, чтобы из цикла поиска места вставки в любом случае происходил выход по первому условию. Для этого достаточно поместить вставляемый элемент перед первым элементом массива, как элемент с индексом 0. Этот элемент и станет естественным барьером для ограничения выхода за левую границу массива.

Алгоритм сортировки вставками приведен на рисунке 4.



**Рисунок 4-** Схема алгоритма сортировки вставками

**ВариантforC\C++**

**Рисунок 5 -** Схема алгоритма сортировки вставкамив С/С++

**#include<iostream>**

**#include<time.h>**

**usingnamespacestd;**

**intmain()**

**{**

**setlocale(LC\_ALL,"Russian");**

**doublea[50];**

**doubletemp=0;**

**cout<<"Inputn<=50"<<endl;**

**intn;**

**cin>>n;**

**cout<<"Inputedmassiv"<<endl;**

**srand(time(NULL));**

**for(intw=0;w<n;w++){**

**a[w]=rand()%50-20;**

**cout<<a[w]<<" ";**

**}**

**cout<<endl;**

**for(inti=1;i<n;i++){**

**temp=a[i];**

**for(intj=i-1;(j>=0&&a[j]>temp);j--)**

**{**

**a[j+1]=a[j];**

**a[j]=temp;**

**}**

**}**

**cout<<"Sortedmassiv"<<endl;**

**for(inti=0;i<n;i++){**

**cout<<a[i]<<" ";**

**}**

**return0;**

**}**

**Примеры работы программы**







**Вариантwhile**

**Программа**

**#include<iostream>**

**#include<time.h>**

**usingnamespacestd;**

**intmain()**

**{**

**setlocale(LC\_ALL,"Russian");**

**doublea[50];**

**doubletemp=0;**

**cout<<"Inputn<=50"<<endl;**

**intn;**

**cin>>n;**

**cout<<"Inputedmassiv"<<endl;**

**srand(time(NULL));**

**for(intw=0;w<n;w++){**

**a[w]=rand()%50-20;**

**cout<<a[w]<<" ";**

**}**

**cout<<endl;**

**for(inti=1;i<n;i++)**

**{**

**temp=a[i];**

**intj=i-1;**

**while(j>=0&&a[j]>temp)**

**{**

**a[j+1]=a[j];**

**a[j]=temp;**

**j--;**

**}**

**}**

**cout<<"Sorted massiv"<<endl;**

**for(inti=0;i<n;i++){**

**cout<<a[i]<<" ";**

**}**

**return0;**

**}**

**Примерработы**

****

Оценим временную сложность данного метода, также определив количество операций сравнения.

Для поиска места i-го элемента каждый раз потребуется выполнить от одной до i-1 операций сравнения, т.е. в среднем i/2 операций сравнения. Значение i изменяется от 2 до n, т.е. выполняется n-1 проход, в каждом из которых происходит в среднем от 1 до n/2 сравнения. Таким образом, суммарно в среднем для решения задачи требуется выполнить (n-1)(n/2+1)/2 = (n2+n-2)/4 операций сравнения. Откуда вычислительная сложность метода в среднем также равна Oср.(n2), хотя время выполнения примерно в два раза меньше, чем у предыдущего метода. Интересно, что в данном случае вычислительная сложность зависит от исходного расположения элементов массива.

Так в лучшем случае, когда массив уже упорядочен, поиск места вставки требует одного сравнения для каждого элемента, и количество сравнений равно n-1. Соответственно, вычислительная сложность равна Oл.(n).

В худшем случае, если элементы массива в исходном состоянии расположены в обратном порядке, поиск места вставки для каждого элемента потребует: 1, 2, 3, ..., n-1 сравнения, следовательно, всего потребуется n(n-1)/2 операций сравнения, т. е. время выполнения программы примерно совпадет со временем программы, реализующей метод выбора. Значит вычислительная сложность в худшем также, как в среднем, равна Ox.(n2)

Таким образом, за счет ускорения сортировки в лучших случаях данный метод имеет лучшие временные характеристики, чем предыдущий.

**Сортировка с помощью обмена.**

Алгоритм прямого обмена основывается на сравнении пары соседних элементов. Если расположение элементов не удовлетворяет условиям сортировки, то их меняют местами. Сравнения и перестановки продолжают до тех пор, пока не будут упорядочены все элементы. Определить, что элементы упорядочены можно считая количество выполненных перестановок: если количество перестановок равно нулю, то массив отсортирован (рисунок 6).



**Рисунок 6 -** Сортировка обменом

В своем простейшем виде алгоритм сортировки с помощью обмена представлен на рисунке 7.



**Рисунок 7 –** Схема алгоритма сортировки обменами

**Программа**

#include<stdlib.h>

#include<time.h>

#include<stdio.h>

#include<conio.h>

int**main**()

{

inta[100],n,i,j,m;

puts("input n< = 100");

scanf("%d",&n);

srand((unsigned)time(NULL));

puts("Input array is forming");

for(i=0;i<n;i++)

{a[i]=rand()%51-20;

printf("%4d",a[i]);

}printf("\n");

i=0;intk=1;

while((i<n-1)&&(k>0))

{

k=0;

for(j=0;j<n-i-1;j++)

{

if(a[j]>a[j+1])

{

intb=a[j];;

a[j]=a[j+1];

a[j+1]=b;

k++;

}

}

i++;

}

puts("Sorted array");

for(i=0;i<n;i++)

printf("%4d",a[i]);

printf("\n");

getch();

return0;

}

**Пример работы программы**

****

****

Оценим вычислительную сложность данного метода. Очевидно, что она сильно зависит от исходного расположения элементов массива. Так в лучшем случае, если массив был уже отсортирован, потребуется выполнить n-1 сравнение для проверки правильности расположения элементов массива, т. е. вычислительная сложность в лучшем Oл.(n).

В худшем случае, если массив отсортирован в обратном порядке, будет выполнено: n-1, n-2, ...1 операций сравнения, т. е. всего (n2-n)/2 операций сравнения, следовательно, вычислительная сложность в худшем определяется как Ox.(n2).

Выполнить усредненную оценку данного метода достаточно сложно, так как в отличие от предыдущих случаев, зависимость времени выполнения от количества неправильно стоящих элементов не является линейной. Например, два элемента, стоящие на своем месте в начале массива практически не повлияют на время сортировки, а два элемента, стоящих на своем месте в конце массива, вызовут ее досрочное завершение. По результатам тестирования можно считать, что в среднем этот метод сортировки обеспечивает время примерно в 2 раза лучше, чем предыдущий. Вычислительная сложность в среднем данного метода Oср.(n2).

*Примечание*. По материалам данного раздела можно сделать ошибочный вывод, что все методы сортировки в среднем имеют вычислительную сложность O(n2). Методы сортировки, рассмотренные в данном разделе, не являются самыми быстрыми, они просто самые простые и потому используются чаще всего. Существуют методы быстрой сортировки, которые обеспечивают в среднем и даже в худшем вычислительную сложность O(nlog2n). Разработаны также методы, обеспечивающие для специальных данных вычислительную сложность Oср.(n). Один из методов быстрой сортировки будет рассмотрен в разделе разработка рекурсивных программ.