N E W S
OF THE NATIONAL ACADEMY OF SCIENCES OF THE REPUBLIC OF KAZAKHSTAN
PHYSICO-MATHEMATICAL SERIES
ISSN 1991-346X **https://doi.org/10.32014/2019.2518-1726.17**
Volume 2, Number 324 (2019), 99 – 107

UDC 373.1.02:519.71:002
МРНТИ 20.01.07

L.A. Smagulova, A.U.Yelepbergenova, G.A.Mursakimova, A.Nurbekova

Zhetysu state university named after I. Zhansugurov, Taldykorgan, Kazakhstan
jgu_laura@mail.ru, aigul_eu@mail.ru, gmursakimova@mail.ru, ainven_87@mail.ru

SORTING ALGORITHMS AND COMPARISON OF THEIR EFFECTIVENESS

Abstract. The present work is dedicated to the methods of sorting data and analysis of their complexity. There are several reasons for analysis of algorithms. One of them is necessity to evaluate the boundary values for the amount of memory or time required by some algorithm for successful data processing. The sorting process can implemented by various algorithms. The choice of algorithm depends on the structure of the data being processed. In practice two classes of sorting are used: external and internal. If the amount of input data fits within the range of available internal RAM they say about the algorithms for internal sorting. But if the input data are stored in files, i.e. external memory, they say about external sorting.

This work demonstrates the fundamental algorithms of internal soritng with quadratic time and quick algorithms with *O(n*logn)* complexity. Quick sorting algorithms such as merge sorting and Hoare"s quicksort algorithms are given. Also simpler methods of internal sorting such as exchange sort, Shell"s method, insertion and selection algorithms are discussed as well. The article describes the idea behind these methods, algorithms on which they are based, complexity of these algorithms and provides concrete examples of programs.

Keywords: array, data, sorting, ordering, exchange sorting, insertion, selection, merge, quicksort, algorithm complexity.

Introduction. Regardless of whether you are a student or a professional programmer and in which sphere of activity you work it is neccessary for you to obtain a knowledge of algorithms and data structures. They are crucial building blocks for solving problems.

The concept of an algorithm is not something completely new to us as we meet them on every step in our everyday live. This can be an algorithm for computing a mathematical function, an algorithm of a technological process, an algorithm of designing a computer or some engineering construction, etc.

In all areas of its activities, in particular, in the field ofinformation processing, a person is faced with various methods of solving problems. They determine theorder of actions to obtain the desired result - we can interpret this as the initial or intuitive definition of thealgorithm.

An algorithm is a finite prescription given in a language, defining a finite sequence of executable elementary operations for solving a problem, common to a class of possible input data [1].

The main algorithms of processing data structures are sorting and search algorithms.

Sorting is one of the most important procedures for processing structured data. Sorting is the process of rearrangement of a given sequence of objects in some predefined order [2]. A certain order, e.g. increasing or decreasing in alphabetic order, in the sequence of objects is necessary for convenience of using this sequence.

It is much easier to work with ordered objects than with those arranged randomly. It becomes much simpler to search for an existing element or delete or insert a new one.

The goal of sorting is to facilitate the search for the next element in a pre-sorted sequence.

The process of sorting data can be implemented by various algorithms. The choice of algorithm depends on the structure of input data. There are two classes of sorting — sorting of arrays and sorting of files[3,PP 43].These classes are also called internal and external respectively. If the amount of input data allows to use internal RAM then we say about algorithms of internal sorting. If the data are stored in extrenal memory, that is files, we say about external sorting. This work puts emphasis on fundamental algorithms of internal sorting. A great diversity of sorting algorithms leads to the necessity of their analysis to achieve their maximum efficiency.

Methods. To solve this problem, we used method of comparative analysis, methods of theoretical and research and also methods of the analysis of data.

Results. Before continuing it is necessary to introduce some terms and definitions.

Let us consider the sequence of *n* elements: $a_1, a_2, \dots, a_n$. Each record $a_j$has a key k_j, which manages the sorting process. The goal of sorting is to rearrange these elements in such a way that all keys are in a non-decreasing order:

$$k_1 \le k_2 \le \dots \le k_n$$

A sorting algorithm is called stable if during the sorting process the relative order of the elements with equals keys do not change[3, PP 45].

The most important characteristic of a sorting algorithm is speed of its work which is determined by functional relation between average time of sorting the sequences of data elements with predefined length and the length itself. The sorting time is proportional to the number of comparison and reshuffling of data elements in the process of their sorting.

To evaluate the quality of algorithm it is necessary to define the term complexity (or effectiveness) of algorithm. The more time and the memory amount it takes to implement the algorithm the greater is its complexity and effectivenss[4]. The algorithm complexity are divided into capacity and time complexities. The time capacity is determined by the time taken by the implementation of algorithm. Capacity complexity is criterium indicating the memory overload for implementation of algorithm.

Solid algorithms of internal sorting require the number of comparisons to be equal to $n*\log n$ order.The following quick algorithms can serve as examples of such solid algorithms:

- merge sorting;
- partition sorting.

We will start the analysis with direct methods which are called simple ordering methods with time complexity of n^2 *order.*This group of algorithms is presented by the following simple algorithm of pair sorting:

- selectionsorting;
- exchangesorting;
- insertion sorting.

Though these algorithms are relatively slow, nevertheless they are convenient for use in describing the characteristic features of main principles of the majority of sorting methods.

Let us start discussing the quadratic time sorting with the exchange sort algorithms. Here the sorting is based on comparison of two elements. If the order of elements does not fit the required regularity then their exchange takes place. The process repeats until all the elements are arranged.Implementation of this algorithm in C++:

```
#include <iostream>
using namespace std;
int main()
    {
int *arr;
int size;
cout<<" Enter the number of elements in the array n =";
cin>> size;
arr = new int[size];
for (intI = 0; I< size; i++) {
cout<<"arr["<<I<<"] = ";
cin>>arr[i];
```

```
    }
int temp;
for (intI = 0; I< size - 1; i++) {
for (int j = 0; j < size -I- 1; j++) {
if (arr[j] >arr[j + 1]) {
temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
            }
        }
    }
for (intI = 0; I< size; i++) {
cout<<arr[i] <<"";
    }
cout<<endl;
delete [] arr;
return 0;
}
```

Bubble sort is a specific case of exchange sort. Its idea is reflected by its name. The heaviest elements go to the top of the sequence, while the lightest are placed at the bottom. The sequence of*n* data elements is compared from the very beginning to the very end so that adjacent elements are swapped if the first of them is lesser (or lighter) than the second. After the comparison ends the lightest element is dragged to the bottom of the sequence [5,6].

The next algorithm with quadratic time is the selection sort. The idea of the method is that in the beginning the smallest element is selected and separated from others. Then it changes places with the very first elements. After that this operation is repeated with the remaining*n-1* elements.The whole process is repeated until all elements are put in their appropriate places.Its algorithmlooks as follows:

```
#include <iostream>
using namespace std;
int main()
    {
int *arr;
int size;
inttemp;
intj;
cout<<" Enter the number of elements in the array n =";
cin>> size;
arr = new int[size];
for (intI = 0; I< size; i++) {
cout<<"arr["<<I<<"] = ";
cin>>arr[i];
    }
for (inti=0; i<size-1;i++)
{
int min=arr[i];
intprs=I;
for (j=i+1; j<size; j++)
if (arr[j]<min)
             {
min= arr[j];
prs=j;
temp=arr[i];
arr[i]=arr[prs];
arr[prs]=temp;

}
}
for (intI = 0; I< size; i++) {
cout<<arr[i] <<"";
    }

return 0;
}
```

The insertion sort selects sequentially each element from unordered sequence of elements, compares it to a pre-ordered element and then places it in an appropriate place.

The algorithm of this method is as follows:

- at the first stage two initial elements are compared. If the next element is lesser than the first then we swap their places, i.e. the next element is moved to the place of the previous element and this previous element is shifted to the next position to the right;
- at the second stage we select an element from unordered sequence and compare it to the two previously ordered elements. If is greater than these previous elements then it retains its place. Else if it is lesser then it is shifted to the appropriate place;
- all remaining elements are analysed the similar way until the whole sequence is ordered.

The code of this method is as follows:

```
#include <iostream>
using namespace std;
int main()
    {
int *arr;
int size;
int temp;
intI;
int j;
cout<<"Enter the number of elements in the array n =";
cin>> size;
arr = new int[size];
for (I = 0; I< size; i++) {
cout<<"arr["<<I<<"] = ";
cin>>arr[i];
    }
for (i=1; i<size;i++)
    {
for ( j=i-1; j>=0;j--)
if (arr[j]>arr[j+1]) {
temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
}
}
for (I = 0; I< size; i++) {
cout<<arr[i] <<"";
    }
return 0;
}
```

3	2	5	6	23	4	17	7	1	9
2	3	5	6	23	4	17	7	1	9
2	3	5	6	23	4	17	7	1	9
2	3	5	6	23	4	17	7	1	9
2	3	5	6	23					
2	3	4	5	6	23	17	7	1	9
2	3	4	5	6	17	23			
2	3	4	5	6	7	17	23		
1	2	3	4	5	6	7	17	23	
1	2	3	4	5	6	7	9	17	23

The method of direct insertion was improved by D.Shell. The Shell"s method does not compare neighbouring elements. Instead it compares elements located at the distance, where d — is number of steps between compared elements. If the sequence consists of *n* elements the initial value of *d=[n/2]*.After each comparison the d is decreased double times. At the last comparison it is increased to *d=1*. In the end such method outputs an ordered sequence. Its implementation looks as follows:

```
#include <iostream>
using namespace std;
int main()
    {
int *arr;
intsize,d;
int temp;
intI;
int j;
cout<<"Enter the number of elements in the array n =";
cin>> size;
arr = new int[size];
for (I = 0; I< size; i++) {
cout<<"arr["<<I<<"] = ";
cin>>arr[i];
    }
{
d=size;
d=d/2;
while (d>0)
{
for (i=0; i<size-d; i++)
{
j=I;
while (j>=0 &&arr[j]>arr[j+d])
{
temp=arr[j];
arr[j]=arr[j+d];
arr[j+d]=temp;
j--;
}
}
d=d/2;
}
for (i=0; i<size ; i++)
cout<<arr[i]<<”“;
}
}
```

The average time of the algorithm“s complexity depends on length of intervals — *d* which contain the sorted elements of source array of capacity *N* on each step of algorithm.

Now let us cover the quicksort algorithms. One version of the quicksort algorithms is merge sort. It works the following way:

- the sequence is divided to two equal parts;
- each part is sorted separately;
- separately sorted parts of the source sequence are merged.

Now let us provide the program implementation of this algorithm:

```
#include <iostream>
using namespace std;
intarr[100];
int size;
void merge(int l, int r) {
if (r == l)        return;
if (r - l == 1) {
if (arr[r] <arr[l])
swap(arr[r], arr[l]);return;
    }
int m = (r + l) / 2;
merge(l, m);
merge(m + 1, r);
intbuf[100];
int xl = l;
intxr = m + 1;
int cur = 0;
while (r - l + 1 != cur) {
if (xl > m)
buf[cur++] = arr[xr++];
else if (xr> r)
buf[cur++] = arr[xl++];
else if (arr[xl] >arr[xr])
buf[cur++] = arr[xr++];
elsebuf[cur++] = arr[xl++];
    }
for (intI = 0; I< cur; i++)
arr[I + l] = buf[i];
}
int main() {
cout<<"Enter the number of elements in the array n =”;cin>> size;
for (intI = 0; I< size; i++)
cin>>arr[i];
merge(0, size - 1);
for (intI = 0; I< size; i++)
cout<<arr[i] <<"“;
return 0;
}
```

The next algorithm was invented by T.Hoare. In practice it is generally considered to be on of the most effective quicksort algorithms. This algorithm is known as quicksort algorithm and its complexity equals *O(n*logn).*The quicksort algorithm belongs to the group of divide-and-conquer algorithms.

The essense of this algorithm is as follows. A key element is selected and fixed. With respect to this element all other elements with larger weight are shifted right and the element with lesser weight are shifted left. Also with respect to the selected key the whole sequence is divided into two parts and for each part the process is repeated. Let us provide the code of the quicksort algorithm where the role of the key element is played by the central element of the sorted sequence:

```
#include <iostream>
using namespace std;
int first, last;
// sort function
void sort(int* arr, int first, int last)
{
intI = first, j = last;
doubletmp, x = arr[(first + last) / 2];

do {
while (arr[i] < x)
i++;
while (arr[j] > x)
j--;

if (I<= j)
   {
if (I< j)
      {
tmp=arr[i];
arr[i]=arr[j];
arr[j]=tmp;
      }
i++;
j--;
   }
} while (I<= j);

if (I< last)
sort(arr, I, last);
if (first < j)
sort(arr, first,j);
}
//main function
int  main()
{
intsize;
cout<<"Enter the number of elements in the array n =";    cin>> size;
int *arr=new int[size];
for (inti=0; i<size; i++)
{
cout<<"arr["<<I<<"] = ";
cin>>arr[i];

   }
first=0; last=size-1;
sort (arr, first, last);
for (inti=0; i<size; i++)
cout<<arr[i]<<"";
}
```

The merge sort algorithm based on division of the source sequence into separate parts was pretty simple, while the process of merging the sorted parts was much more complicated. On the contrary, in the partition sort algorithm the most complicated part was dividng array into parts, while the process of mergin these parts was much simpler.

Conclusions. Finishing our review and anlysis of sorting methods we attempted to compare their effectiveness.In our opinion the essence of each method is to provide effective means for rearranging given sequence in increasing or decreasing order.It is safe to say that algorithms with quadratic time are easier to understand and use while the quicksort algorithms are harder to undersand but at the same time more efficient.We think that it is due to user itself to select the appropriate method for his specific case.

Л.А. Смағұлова, А.Ө. Елепбергенова, Г.А. Мурсакимова, А. Нұрбекова

І. Жансүгіров атындағы Жетісу мемлекеттік университеті, Талдықорған, Қазақстан

СҰРЫПТАУ АЛГОРИТМДЕРІ ЖӘНЕ ОЛАРДЫҢ ТИІМДІЛІЛІКТЕРІН САЛЫСТЫРУ

Аннотация. Бұл мақала деректерді сұрыптау әдістеріне және олардың өнімділігін талдауға арналған. Алгоритмдерді талдаудың бірқатар маңызды себептері бар. Олардың бірі – деректерді өңдеу үшін алгоритмге қажет болатын бағалау, жады көлемі үшін шекаралар немесе жұмыс уақытын алу қажеттілігі болып табылады. Деректерді сұрыптау процесі түрлі алгоритмдер арқылы жүзеге асырылуы мүмкін. Алгоритмді таңдау өңделетін деректер құрылымына тәуелді болады. Іс жүзінде екі сұрыптау класы қолданылады: ішкі және сыртқы. Егер кіріс деректерінің көлемі жедел, ішкі жадымен шектелетін болса, онда ішкі сұрыптау алгоритмдері туралы, ал егер деректер файлдарда орналастырылса, яғни, сыртқы жадыда, онда сыртқы сұрыптау туралы айтылады.

Бұл жұмыста біз ішкі сұрыптаудың негізгі: күрделілігі квадраттық уақытқа тең және *O (n * log n)* күрделілікке тең жылдам сұрыптау алгоритмі деп аталатын алгоритмдерді қарастырамыз. Сұрыптаудың жылдам алгоритмдері: біріктіру арқылы сұрыптау, Хоара жылдам сұрыптауы және негұрлым қарапайым ішкі сұрыптау әдістері: алмастыру көмегімен, тікелей кірістіру арқылы сұрыптау, Шелл әдісі, таңдау алгоритмдерінің жұмыстары келтіріледі. Мақалада бұл әдістердің негізгі идеясы мен мәнісі, жұмыс алгоритмі, алгоритмдердің күрделілігі ескеріледі, бағдарламалар мысалдары келтіріледі.

Түйін сөздер: массив, деректер, сұрыптау, реттілік, алмастыру, қою, біріктіру арқылы сұрыптау, жылдам сұрыптау, алгоритмнің күрделілігі.

Л.А. Смагулова, А.У. Елепбергенова, Г.А. Мурсакимова, А.Нурбекова

Жетысуский государственный университет им.И. Жансугурова, Талдыкорган, Казахстан

АЛГОРИТМЫ СОРТИРОВКИ И СРАВНЕНИЕ ИХ ЭФФЕКТИВНОСТИ

Аннотация. Данная статья посвящена методам сортировки данных и их анализа трудоемкости. Существует ряд важных причин для анализа алгоритмов. Одной из них является необходимость получения оценок или границ для объема памяти или времни работы, которое потребуется алгоритму для успешной обработки данных. Процесс сортировки данных может быть осуществлен различными алгоритмами. Выбор алгоритма зависит от структуры обрабатываемых данных. На практике применяется два класса сортировки: внутренней и внешней. Если объем входных данных позволяет обходиться оперативной, внутренней памятью, то говорят об алгоритмах внутренней сортировки, а если данные размещаются в файлы, т.е. внешней памяти, то речь идет о внешней сортировке.

В данной работе мы продемонстрируем основные алгоритмы внутренней сортировки: с квадратичным временем и алгоритмы сортировки которые называются быстрыми и имеют трудоемкость *O(n*logn)*.

Приводятся быстрые алгоритмы сортировки, такие как сортировка слиянием, быстрая сортировка Хоара. Более простые методы внутренней сортировки, такие как сортировка с помощью обмена, с помощью прямого включения, метод Шелла, алгоритмы выбора. В статье рассматривается идея и суть этих методов, алгоритм работы, трудоемкость этих алгоритмов, приводятся примеры программ.

Ключевые слова: массив, данные, сортировка, упорядочивание, сортировка обмена, вставка, выбор, слияние, быстрая сортировка, трудоемкость алгоритма.

Information about authors:

Smagulova L.A. - Zhetysu state university named after I. Zhansugurov, Taldykorgan, Kazakhstan; jgu_laura@mail.ru; https://orcid.org/0000-0002-1359-2119

Yelepbergenova A.U. - Zhetysu state university named after I. Zhansugurov, Taldykorgan, Kazakhstan; aigul_eu@mail.ru; https://orcid.org/0000-0002-3525-1825

Mursakimova G.A. - Zhetysu state university named after I. Zhansugurov, Taldykorgan, Kazakhstan; gmursakimova@mail.ru; https://orcid.org/0000-0001-8608-3561

Nurbekova A. - Zhetysu state university named after I. Zhansugurov, Taldykorgan, Kazakhstan; ainven_87@mail.ru; https://orcid.org/0000-0002-4588-1222

REFERENCES

[1] Kenzhebaeva Zh.E., Baynazarova R.M. Mathematical and algorithmic models of information processing and management systems. Reports of the national academy of sciences of the republic of kazakhstan.Volume 1, Number 323 (2019).PP. 117 – 121. ISSN 2224-5227 https://doi.org/10.32014/2019.2518-1483.18

[2] Вирт Н. Алгоритмы и структуры данных.:Пер с анг.-2-е изд. испр. СПб.: Невский Диалект, 2008, 352с.

[3] Котов В.М. Алгоритмы и структуры данных: учеб. пособие /В.М.Котов, Е.П.Соболевская, А.А. Толстиков.- Минск: БГУ, 2011.-267с.

[4] Гагарина Л.Г. Алгоритмы и структуры данных : Учебное пособие / Л.Г. Гагарина, В.Д. Колдаев. М.: Финансы и статистика, 2009.- 304 с.

[5] Кнут, Д.Э. Искусство программирования. Том 3. Сортировка и поиск / Дональд Эрвин Кнут; Под общ.ред. Ю.Козаченко. 2-е изд. / Пер. с англ. М.: Вильямс, 2007. 824с

[6] Макконнелл, Дж. Анализ алгоритмов. Активный обучающий подход : Пер. с англ. /Дж. Макконнелл. 3-е доп. изд.- М.: Техносфера, 2009. 416 с.