2. Алгоритми за сортиране на масиви.2ч.

С О Р Т И Р А Н Е И С М Е С В А Н Е

1. Същност на сортирането

СОРТИРАНЕТО е процес на пренареждане на дадено множество от обекти в определен ред.

Редът за сортиране обикновено се определя от приет критерий за качеството на сортировката, а крайният резултат от решението на задачата е преадресиране на мястото на обектите в компютъра.

При разработката на алгоритми и компютърни програми за сортиране на данни се появява много тясна връзка между избрания МЕТОД за сортиране и СТРУКТУРАТА на данните. Именно това е причината този проблем да бъде в основата на класификацията на методите за сортиране като: СОРТИРАНЕ НА МАСИВИ (т.нар. вътрешно сортиране) или СОРТИРАНЕ НА ПОСЛЕДОВАТЕЛНИ ФАЙЛОВЕ (т.нар. външно сортиране).

2. Сортиране на числови масиви

Преди началото на сортирането, ако числовите масиви са многомерни, те трябва да бъдат приведени по определен алгоритъм в редица (вектор-ред или –стълб). Методите за сортиране на числови масиви (множества) могат условно да се класифицират в три главни категории:

- сортиране чрез вмъкване;
- сортиране чрез селекция;
- сортиране чрез размяна.

2.1. СОРТИРАНЕ ЧРЕЗ ВМЪКВАНЕ

СОРТИРАНЕТО ЧРЕЗ ВМЪКВАНЕ се свежда до последователно сравняване на всеки елемент и вмъкването му на подходящо място в редицата по местоназначение на обектите. Този подход е много устойчив и неговата основна слабост е бавния избор на мястото за вмъкване. Различните алгоритми се отличават преди всичко по решенията, които предлагат за решаване на този проблем.

СОРТИРАНЕТО ЧРЕЗ ПРЯКО ВМЪКВАНЕ СЕ РЕАЛИЗИРА като елементите в множеството се разделят на две редици: по *местозначение* A_1, A_2, ..., A_{i-1} и *първоначална* редица. На всяка стъпка *i*-тият елемент от първоначалната редица се изважда и се премества в редицата по местозначение, като се вмъква на съответното му място.

Например:

***** Началната стойност на *i* е *2*.

Подходящото място за вмъкване на *i*-тия елемент се намира, като той се сравнява последователно отдясно-наляво с елементите от редицата по местозначение. Ако елементът от редицата е по-голям, той се измества с една позиция надясно, i-тия елемент заема неговото място и сравнението продължава със следващия елемент от редицата.

***** Процесът на сравнение приключва, когато се открие по-малък или равен елемент или се достигне до левия край на редицата по местоназначение.

Забележка: Алгоритъмът за пряко вмъкване лесно може да бъде подобрен, като се има предвид, че редицата по местоназначение е подредена.

Методът реализира следната идея.

Ако редицата

$\mathbf{a_0, a_1, ..., a_{i-1}}$

е сортирана във възходящ ред, елементът $\mathbf{a_i}$, се включва в редицата така, че да заеме "правилна" позиция, т.е. новата редица

$\mathbf{a_0, a_1, ..., a_i, ..., a_{i-1}}$

да е сортирана във възходящ ред.

Тези действия се повтарят за **i = 1, 2, ..., n-1**.

Методът **insertSort(Comparable[])** за решаване на задачата използва помощният метод **insert(Comparable[], int)**, който вмъква елемента **a[n-1]** в сортираната вече във възходящ ред редица $\mathbf{a_0, a_1, ..., a_{n-2}}$.

Реализация на метода:

```
void insertSort(double *a,int length){
        for(int i = 1; i  <  length; i++)
                insert(a, i);
}

void insert(double  a , int index){
        double x = a[index];
        int j;
        for(j = index-1; j  > = 0; j--)
                if(x < a[j])
                        a[j+1] = a[j];
                else
                        break;
        a[j+1] = x;
}
```

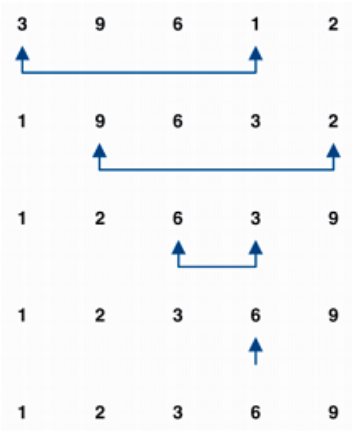
2.2. СОРТИРАНЕ ЧРЕЗ СЕЛЕКЦИЯ

СОРТИРАНЕТО ЧРЕЗ СЕЛЕКЦИЯ разглежда всички елементи в редицата по местоназначение на обектите и отделя този от тях, който отговаря на приетия критерий за качеството на сортировката. Отделеният елемент се разменя по място с първия (респ. последния) в редицата и се изключва по-нататък от сортировката. Различните алгоритми се отличават по бързината на селекцията (отделянето) на елемента с най-добро качество, подлежащ на поредната размяна.

СОРТИРАНЕТО ЧРЕЗ ПРЯКА СЕЛЕКЦИЯ се основава на следните основни принципи:

1. Намираме най-малката стойност в масива ;
2. Разменяме позицията и с първата несортирана стойност

3. Това се повтаря върху масив без сортираните стойности



Сортирането чрез пряка селекция в известен смисъл е противоположно на сортирането чрез пряко вмъкване. Пряката селекция разглежда всички елементи от първоначалната редица, за да намери такъв с най-малка стойност, който да се постави като следващ елемент в редицата по местоназначение. Докато прякото вмъкване разглежда при всяка стъпка само следващия елемент в първоначалната редица и всички елементи в редицата по местоназначение.

Пример:

За **i = 0, 1, ..., n-2** се разглежда редицата:

$\mathbf{a_i, a_{i+1}, ..., a_{n-1}}$

и се извършват следните действия:

- Намира се **k**, така че $\mathbf{a_k = min\{a_i, a_{i+1}, ..., a_{n-1}\}}$.
- Разменят се стойностите на $\mathbf{a_k}$ и $\mathbf{a_i}$.

Реализация на метода:

```
void selectSort(double *a,int length ){
        int k;
        double min;
        for(int i = 0; i  <  length-1; i++){
                min = a[i]; k = i;
                for(int j = i+1; j  <  length; j++)
                        if(a[j] < min){
                                k = j;
                                min = a[k];
                        }
                a[k] = a[i]; a[i] = min;
        }
}
```

2.3. СОРТИРАНЕ ЧРЕЗ РАЗМЯНА

СОРТИРАНЕТО ЧРЕЗ РАЗМЯНА се свежда до последователно сравняване и разменяне на двойки съседни елементи в редицата по местоназначение на обектите.

Алгоритмите, използващи този подход, много често включват различни подобрения за повишаване на неговата ефективност. Независимо от това самата РАЗМЯНА НА ДВА ЕЛЕМЕНТА е доста скъпа компютърна операция и всички алгоритми запазват по-ниска ефективност на сортиране от горните два метода.

Алгоритъмът за СОРТИРАНЕ ЧРЕЗ ПРЯКА РАЗМЯНА се основава на принципа на сравнение и размяна на двойки от съседни елементи, докато се сортират всички елементи. Както и при метода за сортиране чрез пряка селекция, така и тук се преминава известен брой пъти през масива, като всеки път най-големият елемент се премества в десния край на редицата.

Преминаването без размяна означава, че елементите са подредени, т.е. край на сортирането. За целта се използва флаг (променливата F), който се "запалва" само при извършване на размяна при поредното преминаване.

Алгоритъмът може да се подобри още, като се запомни мястото (индексът) на последната размяна. След този индекс елементите са в желания ред. Следователно следващото преминаване може да се извърши до този индекс.

Сортираме във възходящ ред редицата:

$a_0, a_1, ..., a_{n-1}$

Методът може да се опише по следния начин:

1. Нека **right = n-1**.
2. За редицата

$a_0, a_1, ..., a_{right}$

последователно се сравняват всеки два съседни елемента a_i и a_{i+1}.

Ако $a_i > a_{i+1}$, двата елемента си разменят местата и позицията **i** на размяна се запомня в променливата **k**.

Ако $a_i <= a_{i+1}$, двата елемента не си разменят местата.

Процесът продължава до изчерпване на редицата. Ако при това сканиране след размяната на елементите от позиции **i** и **i+1** не е извършена друга размяна, елементите от позициите: **k+1, k+2, ..., n-1** са правилно подредени.

3. **right = k**.
4. Ако **right > 0**, действията от т.2 и т.3 се повтарят.
5. Ако **right = 0**, редицата е сортирана.

Реализация на метода:

```
void bubbleSort(double a[],int length){
        int right = length-1, k;
        double temp;
        while(right  >  0){
                k = 0;
                for(int i = 0; i  <  right; i++)
                        if(a[i] > a[i+1]){
                                temp = a[i];
                                a[i] = a[i+1];
                                a[i+1] = temp;
                                k = i;
                        }
```

```
                    right = k;
                }
        }
```

2.4. НЯКОИ ПОДОБРЕНИ МЕТОДИ ЗА СОРТИРАНЕ

А. СОРТИРАНЕ ЧРЕЗ ВМЪКВАНЕ С НАМАЛЯВАЩА СТЪПКА (МЕТОД НА ШЕЛ)

Подобрение на сортирането чрез пряко вмъкване е предложено от Д. Л. Шел през 1959 г. В този алгоритъм няколко пъти се изпълнява прякото вмъкване като се преминава през елементите с различна и намаляваща стъпка.

Кнут показва, че един разумен избор на стъпки е редицата
1 , 4 , 13 , 40 , 121 , ... (използвана в обратен ред).
Той също препоръчва редицата за избор на стъпки
1 , 3 , 7 , 15 , 31 , ...

Първо, всички елементи, които са на разстояние седем позиции един от друг се групират и сортират по отделно. Този процес се нарича сортиране със стъпка 7. В пример с 10 елемента всяка група съдържа точно по два елемента. След това елементите се прегрупират така, че във всяка група те да са през три позиции и се сортират отново - сортиране със стъпка 3. Най-после при третото преминаване елементите се сортират чрез обикновено сортиране или сортиране със стъпка 1.

Този начин за сортиране е предложен от D.L.Shell през 1957 год. и реализира следната идея.

В зависимост от дължината **n** на редицата се намира "подходяща стойност" на стъпката **h**. Сортират се всички елементи на редицата, които са на разстояние **h** един от друг. След това **h** се намалява чрез целочислено делене на дадено естествено число и отново се сортират всички елементи на редицата, които са на разстояние **h** един от друг. Тези стъпки се изпълняват докато **h > 0**.

По предложение на Кнут и в съответствие с общата практика, стойностите на **h** се избират да принадлежат на редицата **1, 4, 13, 40, 121, 364, ...** и са по-малки от някаква гранична стойност, зависеща от **n**. Редицата се получава по следната зависимост:

$h_1 = 1$

$h_i = 3h_{i-1} + 1;\ i = 2, 3,...$

Началната стойност на **h** се избира да удоволетворява:

| **2h <= n**
| **2(3h+1) > n**.

Тази стойност може да се определи чрез следния програмен фрагмент:

```
int h = 0;
while(2*(3*h+1)  < = a.length) h = 3*h+1;
```

Реализация на метода:

```
void shellSort(double *a,int length){
        int h = 0, i, j;
        double x;
        while(2*(3*h+1)  < = length) h = 3*h+1;
```

```
        while(h  >  0){
                for(i = h; i  <  length; i++){
                        x = a[i];
                        for(j = i-h; j  > = 0; j = j-h)
                                if(x < a[j])
                                        a[j+h] = a[j];
                                else
                                        break;
                        a[j+h] = x;
                }
                h = h/3;
        }
}
```

Б. СОРТИРАНЕ ПО ДЯЛОВЕ

Подобрен метод на базата на размяната е предложен от К. А. Хоаре, който го е нарекъл БЪРЗО СОРТИРАНЕ. Той се основава на принципа, че на колкото по-големи разстояния се правят разместванията, толкова по-ефективно е сортирането.

Нека са дадени N елемента. Един елемент от списъка се определя за основен (в нашият случай това е средният).

Останалите елементи се сравняват с основния и евентуално сменят позициятя си. Когато са по-малки се разполагат в списъка преди основния, а когато са по-големи, се разполагат в списъка след него. Това подреждане представлява етап от сортировката. В края на този етап основният елемент се намира в списъка на позиция, съответстваща на мястото му в окончателния подреден списък.

Списъкът е подреден на две части. Първата - от началото до основния елемент, а втората - от основния елемент до края. Само една от тях може да се обработва веднага в следващата стъпка, а другата се запомня в списъка за заявки за разделяне. Съществено е, че списъкът от заявки се подрежда по обратен ред на заявките.

Методът е ефективен за редица с голяма дължина. Предложен е от C.A.Hoare през 1962 год. и е сред най-ефективните алгоритми за сортиране.

Основава се на факта, че на колкото по-големи разстояния се правят разместванията, толкова по-ефективно е сортирането.

Редицата се разделя на две подредици спрямо един до голяма степен произволно избран елемент **x**, наречен ос. Елементите, които са по-малки от **x** се записват в първата половина, а всички, които са по-големи - във втората половина. Същото действие се повтаря за двете подредици.

Ще дадем един опростен вариант на реализацията на този подход.

Реализация на метода:

```
void qSort(double *a,int length){
```

```
        qSort(a, 0, length-1);
}

void qSort(double *a, int left, int right){
        int i = left, j = right;
        double x = a[(j+i)/2], y;
        do{
                while(a[i] < x)
                        i++;
                while(a[j]>x)
                        j--;

                if(i  <  j){
                        y = a[i];
                        a[i] = a[j];
                        a[j] = y;
                        i++;
                        j--;
                }else
                        if(i == j){
                                i++;
                                j--;
                        }else
                                break;
        }while(i  <= j);

        if(j  >  left)
                qSort(a, left, j);
        if(i  <  right)
                qSort(a, i, right);
}
```

C. Пирамидално сортиране

Нарича се още сортиране чрез двоично дърво. Автор му е J.Williams.

Редицата

$\mathbf{a_0, a_1, ..., a_{n-1}}$

удоволетворява условието за *пирамида*. ако $\mathbf{a_i >= a_{2i+1}}$ и $\mathbf{a_i >= a_{2i+2}}$ са в сила стига тези елемети да принадлежат на редицата.

Пример: Редицата

индекси: **0 1 2 3 4 5 6 7 8 9**

стойности: **8 5 7 3 4 6 2 1 2 1**

е пирамида.

Забележка: Условието за пирамида може да е и $\mathbf{a_i <= a_{2i+1}}$ и $\mathbf{a_i <= a_{2i+2}}$ стига тези елементи да принадлежат на редицата.

Основна операция за работа с пирамида е *пресяването*. Ще я илюстрираме чрез следния пример. Редицата

индекси: **0 1 2 3 4 5 6 7 8 9**

стойности: **4 5 7 3 4 6 2 1 2 1**

не е пирамида. Единствено първият елемент й пречи. Ако разменим елемент **a[0]** с максимал-ния от **a[1]** и **a[2]**. Получаваме:

индекси: **0 1 2 3 4 5 6 7 8 9**

стойности: **7 5 4 3 4 6 2 1 2 1**

Тази редица не е пирамида, тъй като **a[2] = 4** не си е на мястото. Отново разменяме **a[2]** с максималното от **a[5]** и **a[6]**.Получава се редицата

индекси: **0 1 2 3 4 5 6 7 8 9**

стойности: **7 5 6 3 4 4 2 1 2 1**

която вече е пирамида. Процесът на получаване на пирамидата се реализира чрез неколкократно прилагане на операцията пресяване.

Методът за пирамидално сортиране се състои в следното:

1. Чрез прилагане на операцията пресяване, елементите на редицата се пренареждат в пирамида. Така максималният елемент на редицата е в позиция **0**.

2. Разменят се **a[0]** с **a[n-1]**. Така максималният елемент на редицата заема правилната си позиция.

3. Прилага се операцията пресяване за елемента **a[0]** на редицата $\mathbf{a_0, a_1, ..., a_{n-2}}$. В резултат се получава пирамида. Разменят се **a[0]** и **a[n-2]**. Така максималният елемент на редицата $\mathbf{a_0, a_1, ..., a_{n-2}}$ заема правилната си позиция.

4. Действията от т. 3 се повтарят до изчерпване на редицата, като на всяка стъпка максималният елемент заема правилна позиция.

Реализация на метода:

```
void heapSort(double *a,int length){
        int right = length;
        int i = right/2-1;

        for(int j = i; j  >= 0; j--)
                insertHeap(a, j, right);
        right--;

        while(right  >  0){
                double temp = a[0];
                a[0] = a[right];
                a[right] = temp;

                insertHeap(a, 0, right);
                right--;
        }
}

void insertHeap(double *a, int k, int right){
        int i, j = 2*k+1;
        while(j  <  right){
                i=j;
                if(j  <  right-1 && a[j]< a[j+1])
                        i = j+1;
                if(a[k] < a[i]){
                        double temp = a[k];
```

```
                    a[k] = a[i];
                    a[i] = temp;

                    k = i;
                    j = 2*k+1;
                }else
                    j = right;
            }
        }
```

D. Сортиране чрез клатене

Методът се състои в следното:

1. Нека **left = 0, right = n-1**.

2. Подредицата

$a_{left}, ..., a_{right}$

се обхожда два пъти като се сравняват всеки два съседни елемента. Едното обхождане е отляво надясно, а другото - отдясно наляво.

При първото обхождане, аналогично на метода на мехурчето, се намира нова стойност на **right**, чрез която се намалява дължината на редицата, а елементите от позиции **right+1, right+2, ..., n-1** са правилно подредени.

При второто обхождане се извършват симетрични действия като се намира нова стойност на **left**, а елеметите от позиции **0, 1, ..., left-1** са правилно подредени. Ще забележим, че при това обхождане се използва новата стойност на **right**.

3. Процесът продължава докато **right-left > 0**, т.е. докато съществува подредицата.

Реализация на метода:

```
void shakeSort(double *a,int length){
        int k, left = 0, right = length-1;
        while(right-left  >  0){
                k = 0;
                for(int i = left; i  <  right; i++)
                        if(a[i] > a[i+1]){
                                double temp = a[i];
                                a[i] = a[i+1];
                                a[i+1] = temp;
                                k = i;
                        }

                right = k;
```

```
                    k = length-1;
                    for(int i = right; i  >  left; i--)
                            if(a[i] < a[i-1]){
                                    double temp = a[i];
                                    a[i] = a[i-1];
                                    a[i-1] = temp;
                                    k = i;
                            }

                    left = k;
            }
    }
```

3. СОРТИРАНЕ НА ПОСЛЕДОВАТЕЛНИ ФАЙЛОВЕ (СМЕСВАНЕ)

Горните методи за сортиране са неприложими при условие, че данните не се събират в оперативната памет на компютъра, или се намират на външно запомнящо устройство с последователен достъп (например, магнитна лента). В този случай данните се описват като последователен файл, т.е. във всеки момент е пряко достъпен само един елемент от редицата по местоназначение на обектите.

Това е много силно ограничение, което оказва влияние върху методите за сортиране на този тип данни.

Почти всички по-важни методи за сортиране на последователни файлове работят чрез СЛИВАНЕ. Сливането (или обединяването) на практика означава комбиниране на две или повече редици от данни в една, която се подрежда по определен критерий на качеството (сортира) чрез непрекъснато избиране от достъпните в момента елементи. Сливането е много по-проста операция от сортирането и се използва само като допълнителна стъпка (действие) в по-сложния процес на последователното сортиране.

Сливането като отделна фаза не извършва сортиране на елементи от редицата, но създава условия за ефективност на сортировката.

ВЪПРОСИ И ЗАДАЧИ

за самостоятелна работа

1. Разработете блокови алгоритми за привеждане във векторен вид и сортиране на числовите масиви по всички методи. Сравнете алгоритмите по сложност.

а)

10	24	-13	-2,5	1,9
-2	2,5	-19	0	-22
24	-13	2,2	-10	5,6
0	9,3	-22	12,4	-19

б)

06	-4	-9,3
-9	1,5	14,7
14	-2	-4
1,5	6,0	-14

2. Разработете блокови алгоритми за сортиране на последователните файлове:

47	-13	24	1,5	-1,7	33,3	0	-25	25	14,4	-33,3	19	39,4	-13
26	14	-19	2,3	-9,7	-9,7	5	3,5	44	-36,2	34	-26	-2,3	5	13

3. Направете сравнение на методите за сортиране като ги приложите преди и след сливането на последователните файлове от задача №2.