 **ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

Катедра „Информационни технологии в индустрията”

**ДОКЛАД**

***Алгоритми за сортиране. Сортиране чрез сравнение.***

Синтез и анализ на алгоритми

Изготвил: Ръководител:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Християн Зарков ас. маг. инж. Константин Костадинов

Фак. №: 501215038

10.05.2017г

**СЪДЪРЖАНИЕ**

1. Алгоритъм за сортиране
2. Определение за алгоритъм за сортиране
3. Стабилност
4. Класификация
5. Сравнение на алгоритмите
6. Популярни алгоритми за сортиране

а) Метод на мехурчето

б) Сортиране чрез сливане

в) Шел сортиране

г) Сортиране чрес вмъкване

д) Сортиране чрез пряка селекция

е) Пирамидално сортиране

ж) Бърза сортировкa

1. Сортиране чрез сравнение
2. Същност
3. Четно-нечетно сортиране

**I.Алгоритъм за сортиране**

**1.Определение за алгоритъм за сортиране** 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.Стабилност**

Когато се сортират някакви данни, само част от техните признаци се анализират при тяхното подреждане. Алгоритмите за стабилно сортиране използват следното правило: ако два елемента при сравнение са еднакви, то тяхната първоначална подредба ще бъде съхранена, така че ако едната от тях е преди другата в началната подредба, то тя ще бъде първа и в крайният резултат.   
 Когато еднаквите елементи са неразличими, като целочислените числа или други по-общи данни където всеки елемент е уникален по всичките си признаци, стабилността не е проблем. Стабилността не е проблем и когато всички елементи са различни.   
 Нестабилното сортиране може да бъде превърнато в стабилно чрез допълнително разширяване логиката за сравняване, така че сравняването на два елемента, които по един признак са еднакви, да бъдат различени по един или повече други признаци. Това допълнително разширение обаче води до нуждата от допълнително време и памет. Един подход за стабилно сортиращи алгоритми е сортиране на списъци по първичен и вторичен ключ.

**3.Класификация**

Сортиращите алгоритми често се класифицират по:

* Изчислителна сложност при сравняване на елементите (най-лош, среден и най-добър случай) при сравняване на елементи в списък от (n) елемента. Добър пример за серийно сортиране O(n log n), за паралелно сортиране O(log2 n), и за лошо сортиране O(n2).
* Рекурсивни. Някои алгоритми са или рекурсивни, или не-рекурсивни, докато други могат да съчетават и двата вида като например (алгоритъмът за сортиране чрез сливане)
* Стабилност
* Дали са сравняващи алгоритми – сравняващите алгоритми сравняват два елемента с оператор за сравнение
* Адаптиращи се алгоритми
* Използвана памет (и други компютърни ресурси)

**5.Най-използвани алгоритми за сортиране**

*а) Метод на мехурчето*

Алгоритъмът работи по следния начин: взимаме първият елемент на масива и го сравняваме със следващият(втория в нашия случай) и разменяме стойностите им, ако първият е по – голям от втория. След това сравняваме вторият елемент с третия и пак разменяме, ако има нужда. Ако нашият масив е от 10 елемента, след 9 такива сравнения най – отгоре ще изплува най – голямата стойност. След това започваме отново да сравняваме като пак взимаме първият елемент и сравняваме с вторият и така нататък.  
 В най-добрия случай методът на мехурчето има сложност Ω(n). Ако масивът е вече сортиран, методът на мехурчето ще премине през масива веднъж и ще установи, че не трябва да разменя никакви елементи. В най – лошия и среднен случай методът на мехурчето има сложност О(n2), където n е броят на елементите, които биват сортирани. Дори и методът на пряката селекция, който има сложност като на метода на мехурчето, е с по – добра производителност. Следователно методът на мехурчето, не е добър избор, ако искаме да сортираме много на брой елементи.   
 Единственото значимо предимство, което има методът на мехурчето спрямо повечето други алгоритми за сортиране – дори и бързият алгоритъм(quicksort), но не и методът на пряката селекция(insertion sort), е че способността му да разбира кога масивът е сортиран, е ефикасно построена в алгоритъма. Производителността на метода на мехурчето спрямо вече сортиран масив е(най – добър случай) е O(n). За справка, повечето други алгоритми за сортиране, дори тези, които имат по – добра производителност в средния случай им отнема повече итерации. Обаче методът на пряката селекция извършва дори по – малко итерации от метода на мехурчето при сортиран масив.

#include <stdio.h>

int main(void)

{

int item[100];

int a, b, t;

int count;

/\*Прочитане на числата\*/

printf("how many numbers?");

scanf("%d", &count);

for(a=0; a<count; a++) scanf("%d", &item[a]);

/\*Сортиране чрез метода на мехурчето\*/

for(a=0; a<count; ++a)

for(b=count-1; b>a; --b){

/\*Сравняване на съседни елементи\*/

if(item[b-1] > item[b]){

t = item[b-1];

item[b-1] = item[b];

item[b] = t;

}

}

/\*Изписване на числата\*/

for(t=0; t<count; t++) printf("chisloto e %d\n", item[t]);

return 0;

}

*б) Сортиране чрез сливане*

В информатиката сортирането чрез сливане е алгоритъм за сортиране, базиран на сравняване, който винаги има сложност (nlog(n)). Алгоритъмът се гради на принципа „разделяй и владей“. Създаден е от Джон фон Нойман през 1945. Стабилен е и паметта, която му трябва, в най-лошия случай е n. По-подробно описание и анализ на сортирането чрез сливане, се е появило в доклад на Goldstine и Neumann още през 1948 година.  
 Принципът на действие е следният:

1.Несортирания списък по произволен начин се разделя на два подсписъка с приблизително обща дължина (за линейно време)

2.Рекурсивно се разделят подсписъците, докато не се достигне до списъци с единична дължина

3.Сливат се два подсписъка в нов сортиран списък (за линейно време)

static int[] Splits(int[] arr)

{

if (arr.Length == 1)

{

return arr;

}

int middle = arr.Length / 2;

int[] leftArr = new int[middle];

int[] rightArr = new int[arr.Length - middle];

for (int i = 0; i < middle; i++)

{

leftArr[i] = arr[i];

}

for (int i = middle, j = 0; i < arr.Length; i++, j++)

{

rightArr[j] = arr[i];

}

leftArr = Splits(leftArr);

rightArr = Splits(rightArr);

return Merge(leftArr, rightArr);

}

*в) Шел сортиране*

Алгоритъмът на Шел е алгоритъм за сортиране, за който се взема предвид положението на елементите. Той обобщава други сортиращи методи като метода на мехурчето и метода за сортиране чрез вмъкване, като започва сравнението и разменянето на елементите, с онези, които са отдалечени, преди да се извърши сравнението със съседните. Именно тази характеристика на метода го отличава и го прави по-бърз от предшествениците му. Доналд Шел публикува първата реализация на този метод през 1959 г. Времето за изпълнение на алгоритъма на Шел строго зависи от различната последователност, с която работи.  
 Идеята е да се подреди листът от елементи, като се започне от някъде (обикновено около средата) и се смята, че всеки h-ти елемент дава сортиран лист. Подобен лист се казва, че е h-сортиран. Накрая се получават листове от интервали, всеки от които индивидуално сортиран. Започването с голяма стойност на h, позволява на елементите да се предвижват на големи разстояния и намалява възможността за доста произволни разбърквания. Също така се оставят по-малко на брой операции за малките h-сортирания. Методът е нестабилен. Той може да смени наредбата на елементи с еднаква стойност.

void shellSortPhase(int arr[], int length, int gap)

{

int i;

for (i = gap; i < length; ++i)

{

int value = arr[i];

int j;

for (j = i - gap; j >= 0 && arr[j] > value; j -= gap)

{

arr[j + gap] = arr[j];

}

arr[j + gap] = value;

}

}

void shellSort(int arr[], size\_t length)

{

static const int gaps[] = {1, 4, 10, 23, 57, 132, 301, 701};

int sizeIndex;

for (sizeIndex = sizeof(gaps)/sizeof(gaps[0]) - 1; sizeIndex >= 0; --sizeIndex)

shellSortPhase(arr, length, gaps[sizeIndex]);}

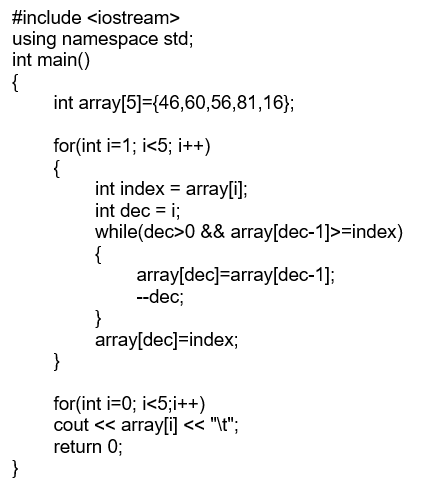
*г) Сортиране чрез вмъкване*

Сортиране чрез вмъкване (на английски: Insertion sort) е прост сортиращ алгоритъм. Чрез сравняващо сортиране сортираният списък се допълва с по един елемент всеки път. Алгоритъмът е доста неефикасен в сравнение с quicksort, heapsort или mergesort, ако се прилага върху големи списъци, но от друга страна има и доста предимства. В най-лошия случай алгоритъмът има сложност O(n2).  
 Принципът на действие е следният:  
 1.Списъкът с елементи, които ще бъдат сортирани се разделя на две части: частта със сортираните елементи и частта с несортираните

2.При всяка стъпка се взема първият елемент от несортирания списък и се вмъква на правилната позиция в сортираната част от списъка

3.Сортирането продължава докато елементите от несортираната част на списъка се изчерпят.

В най-добрия случай масивът е почти сортиран. Тогава за сортирането чрез вмъкване е нужно линейно време (т.е. O(n))). При всяка стъпка елементът, който е на ход, се сравнява с най-десния елемент от сортирания масив.   
 Пример за най-лош случай е когато масивът е напълно обърнат на обратно. Тоест подредбата на елементите ще бъде такава, че всеки следващ елемент ще бъде по-малък от предходния. При тази подредба всяка стъпка във вътрешният цикъл ще се изпълни максимален брой пъти. Това дава на сортирането чрез вмъкване квадратично време за изпълнение (т.е. O(n2)).



*д) Сортиране чрез пряка селекция*

В компютърните науки методът на пряката селекция (на английски: Selection sort) е алгоритъм за сортиране. Той е един от фундаменталните методи за сортиране и е прост и лесен на имплементиране.   
 Алгоритъмът има сложност от Θ(n2), т.е. времето за изпълнението му е пропорционално на квадрата на броя на елементите в масива. Това го прави неефикасен при големи списъци и като цяло работи по-зле от подобния му алгоритъм за сортиране чрез вмъкване (insertion sort). Сортирането чрез пряка селекция впечатлява с простотата си, а също така в дадени ситуации има предимства пред някои сложни алгоритми.  
 Алгоритъмът работи по следния начин:

1.Намира най-малкия елемент в списъка като сравнява първият елемент с всички останали

2.Разменя го с елемента на първа позиция

3.Повтаря горните две стъпки за всеки следващ елемент

Методът на пряката селекция не е труден да се анализира в сравнение с други сортиращи алгоритми. За да намерим най-малкият елемент се изисква да сканираме всички n на брой елементи и след това да го разменим с първата позиция. За да намерим следващият най-малък елемент се изисква да сканираме оставащите n – 1 елементи и така нататък, за (n − 1) + (n − 2) + ... + 2 + 1 = n(n − 1) / 2 ∈ Θ(n2) сравнения.

#include <iostream>

using namespace std;

int main()

{

int array[5]={31,34,12,22,11};

for(int i=0; i<5; ++i)

{

int min = i;

for(int j=i; j<5; ++j)

if(array[min]>array[j])

min = j;

swap(array[min],array[i]);

}

for(int i=0; i<5; i++)

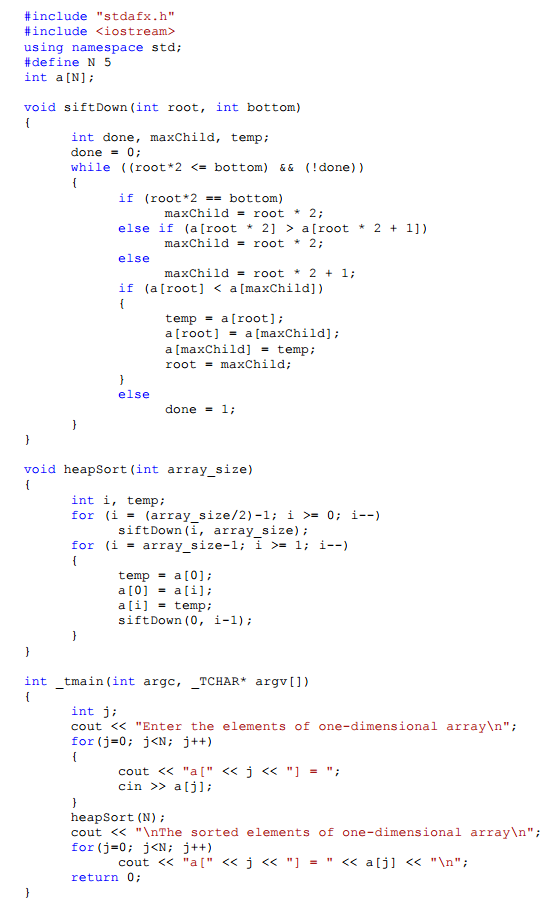
cout << array[i] << "\t";

return 0;

}

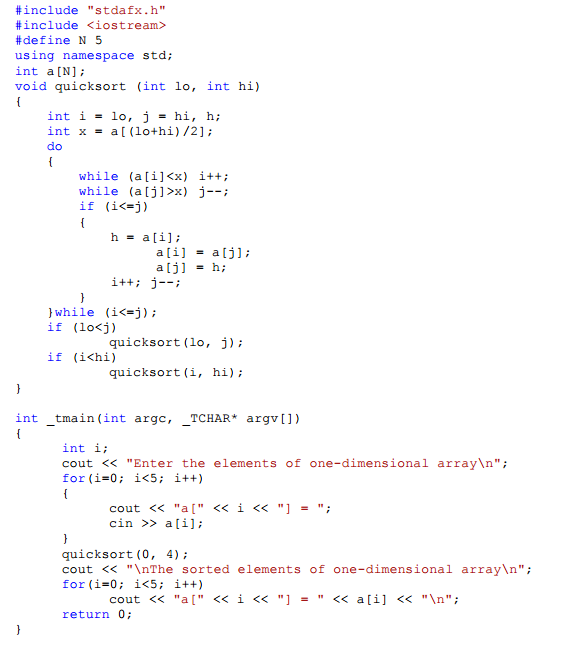
*е) Пирамидално сортиране*

Пирамидално сортиране е детермистичен алгоритъм за сортиране, който създава сортиран масив (или списък). Heapsort е вид алгоритъм за сортиране чрез пряка селекция. Въпреки че е по-бавен, със сложност O (n log n), отколкото едно добро приложение на quicksort алгоритъм, той има предимството за по-благоприятен най-лош случай, защото не използва много масиви или рекурсия. Heapsort е измислен от J. W. J. Williams през 1964 г. Принципът на дейстие е следният. Пирамидалното сортиране може да бъде разделено на две стъпки.   
 Първа стъпка, създава се "купчина" от елементи на множеството.  
 Втора стъпка, създава се сортиран масив, чрез взимане и премахване на най-големия елемент от пирамидална структура от данни (heap) и поставянето му в друг масив. "Купчината" се реконструира при всяко премахване и след като всички обекти са премахнати от heap паметта, се създава пълният сортиран масив. Посоката на сортираните елементи може да бъде променена, като се избере min-heap или max-heap в първата стъпка. За да се изпълни този алгоритъм се изискват два масива - един за елементите в купчината, втори за сортирания масив. Пирамидалното сортиране може да бъде извършено и с един масив - когато някой елемент се премахва от "купчината", той освобождава място, и елементът може да бъде добавен в края на същия масив. Heapsort е конкуриран главно от quicksort, друг високо ефективен сортиращ алгоритъм с общо приложение.  
 Quicksort е по-бърз от heapsort, но най-лошият случай на quicksort е O(n2), което е неприложимо при големи множества от данни. Също така може да бъде достъпена информация за имплементацията на алгоритъма, което само по себе си създава риск за сигурността. Ето защо вградени системи с ограничения в реално време или системи, за които е ключова сигурността, често използват пирамидалното сортиране.



*ж) Бързо сортиране*

Бързо сортиране е добре известен сортиращ алгоритъм, разработен от Ч. А. Р. Хор през 1960 годинаи публикуван през 1961. Основната част на алгоритъма се състои в сравняващо сортиране. В най-лошия случай алгоритъмът има сложност O(n2). В средния случай алгоритъмът има сложност O(log n), а в амортизирания се правят 1.386 nlogn сравнения. В практиката бързото сортиране е по-бързо от други сортиращи алгоритми със сложност O(log n). Освен това последователността на бързото сортиране и локализираните препратки към паметта работят добре с кеша. Бързото сортиране е сортиране чрез сравняване и не е особено стабилно за ефективни приложения. То може да бъде осъществено с алгоритъм без заделяне на допълнителна памет, така че цялото сортиране да бъде реализирано само с O(log n) допълнително място по време на рекурсията.



**II.Сортиране чрез сравнение**

**1.Същност** Сравнителен сорт е тип алгоритъм за сортиране, който чете елементите от списъка само чрез една абстрактна операция за сравняване (често оператор "по-маллко от или равно на" или тристранно сравнение), който определя кой от двата елемента да се получи първо във финалния сортиран списък. Единственото изискване е операторът да изпълнява две условията:

* Ако a ≤ b и b ≤ c тогава a ≤ c (преходност)
* За всички а и b, или ≤ b или b ≤ а (тоталност или трихотомия).

Възможно е и а ≤ b и b ≤ a; този случай всяко едно може да се качи първо в сортирания списък. При стабилно сортирабе редът за въвеждане определя сортирания ред. Една метафора за мислене за сравнителните сортове е, че някой има набор от небелязани тежести и везни. Целта им е да се подредят теглата по реда на теглото им без никаква информация освен тази, получена чрез поставяне на две тежести върху скалата и да се види коя е по-тежка.   
 Съществуват фундаментални ограничения за ефективността на сравнителните алгоритми. Сравнителният сорт трябва да има средна стойност на долната граница на операциите за сравняване Ω (n log n), [1] което е известно като линейно-ватно време. Това е следствие от ограничената информация, налична само чрез сравнения - или, да се каже по различен начин, от неясната алгебрична структура на напълно подредените комплекти. В този смисъл mergesort, heapsort и introsort са асимптотично оптимални по отношение на броя сравнения, които трябва да изпълнят, въпреки че този показател пренебрегва други операции. Неконтролиращите сортове (като примерите, обсъдени по-долу) могат да постигнат O (n) ефективност, като използват други операции, различни от сравнения, като им позволяват да заобиколят тази долна граница (при условие, че елементите са с постоянен размер). Независимо от тези ограничения, сравнителните сортове предлагат забележителното практическо предимство, контролът върху функцията за сравняване позволява сортиране на много различни типове данни и фин контрол върху начина на сортиране на списъка. Например обръщането на резултата от функцията за сравнение позволява списъкът да бъде сортиран обратно; И може да се сортира списък от типове в лексикографски ред само чрез създаване на сравняваща функция, която сравнява всяка част в последователност.   
 Балансираното тризначно означаване позволява да се правят сравнения на една стъпка, чийто резултат ще бъде един от "по-малък", "по-голям от" или "равен на". Сравнителните сортове обикновено се приспособяват по-лесно към сложни поръчки, като например реда на числата с плаваща запетая. Освен това, след като е написана функция за сравнение, всеки сортиращ алгоритъм може да се използва без промяна; Несравнимите сортове обикновено изискват специализирани версии за всеки тип данни.

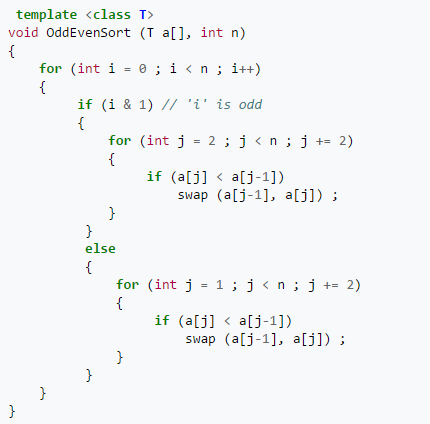
Тази гъвкавост, заедно с ефективността на горните алгоритми за сортиране на сравнението на съвременните компютри, доведе до широко разпространено предпочитание за сравнение в най-практичната работа.

Някои от най-известните видове сравнителни алгоритми включват: бързото сортиране, сортиране чрез сливане, метода на мехурчето, пирамидалната сортировка и други алгоритми, които разгледахме по горе. Един от примерите е четно-нечетното сортиране.

2.Четно-нечетно сортиране

В изчисляването, четно-нечетно сортиране или четно-нечетно транспониране е относително прост алгоритъм за сортиране, разработен първоначално за използване на паралелни процесори с локални връзки. Това е сравнителен алгоритъм, свързан със, с който той споделя много общи характеристики. Той функционира чрез сравняване на всички четни- нечетни равномерно индексирани двойки на съседни елементи в списъка и ако двойката е в грешен ред (първата е по-голяма от втората) елементите се превключват. Следващата стъпка повтаря това за нечетни-четни двойки (на съседни елементи). След това той се редува между четни-нечетни и нечетни-четни стъпки, докато списъкът не бъде сортиран.

На паралелни процесори, с една стойност на процесор и само локално ляво-дясно съседни връзки, процесорите всички едновременно правят операция за сравняване на обмен с техните съседи, като се редуват между нечетни-четни и четни-нечетни двойки. Този алгоритъм първоначално е представен и демонстриран като ефективен за такива процесори от Хаберман през 1972 г.



Използвани източници:

http://stackoverflow.com

[https://en.wikipedia.org](https://en.wikipedia.org/)

<https://mail.bg/Сортиране.pdf>

http://www.math.bas.bg