КАТЕДРА: КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ
ДИСЦИПЛИНА: АЛГОРИТМИ СТРУКТУРИ ОТ ДАННИ
ДИСЦИПЛИНА: СИНТЕЗ И АНАЛИЗ НА АЛГОРИТМИ

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 6

ТЕМА: Елементарни алгоритми за сортиране

ЦЕЛ:

Целта на упражнението е студентите да се запознаят с елементарните алгоритми за сортиране. След упражнението студентите би следвало да могат да прилагат съответните алгоритми върху различни структури от данни.

I. ТЕОРЕТИЧНА ЧАСТ

1. Въведение.

Алгоритъм за сортиране е алгоритъм, който подрежда списък от елементи в определена последователност. Най-използваните подредби са числовите и лексикографските подредби. Ефективните алгоритми за сортиране са важни за оптимизацията на други алгоритми, който изискват входните данни да са сортирани в определена последователност. Често също така е полезно за сливане на данни и за генериране на разбираеми за човек крайни резултати. Формално казано, изходният резултат от алгоритъма за сортиране трябва да задоволява две условия:

- Изходният резултат е в ненамаляваща последователност (всеки елемент не трябва да е по-малък от предходните на базата на очакваната обща подредба);
- Изходният резултат е пермутация (пренаредба) на входните елементи.

Сортиращите алгоритми често се класифицират по:

- Изчислителна сложност – при сравняване на елементи в списък от (n) елемента;
- Използвана памет (и други компютърни ресурси) – какви ресурси заема алгоритъма от компютърната система;
- Рекурсивни – Някои алгоритми са или рекурсивни(т.е. способността да извикват сами себе си), или не-рекурсивни, докато други могат да съчетават и двата вида като;
- Стабилност – Възможността да се извършва сортиране по няколко признака;
- Адаптиращи се алгоритми – способността на алгоритмите да се адаптират в различните ситуации.

Съществуват много алгоритми за сортиране, които подреждат елементите на даден масив или списък. Някои от тях са бавни, други бързи; някои изискват много памет, други изискват само две допълнителни променливи; някои се пишат на три реда, други на няколко стотин. В това упражнение ще обърнем внимание на някои прости (но бавни) алгоритми за сортиране. Такива са:

- *Метод на мехурчето (Bubble sort)*

Това е прост сортиращ алгоритъм, който започва в началото на сортиращия се списък. Той сравнява първия и втория елемент, и ако първият е по-голям от втория, ги разменя. Продължава да прави това с всички съседни двойки елементи до края на сортиращия се

списък. След това повтаря същото действие още толкова пъти, докато накрая при обхождането на целия списък не е направено нито една размяна на два съседни елемента.

Сложността на алгоритъма спрямо Big O нотацията е $O(n^2)$ и ще е изключително бавен. Методът е подходящ за да се използва за малки множества или за такива чиито елементи се намират близо до очакваното им място.

- *Сортиране чрез пряка селекция (selection sort)*

Алгоритъмът намира най-малкия елемент в списъка и го разменя с първия елемент. След това търси втория най-малък и го поставя на второ място. Действието се повтаря, докато се стигне до края на списъка. При този алгоритъм не се правят повече от n размени. Този алгоритъм е полезен в случаи, когато размяната на елементи е тежка операция и трябва да се минимизира.

Изчислителната му сложност спрямо Big O нотацията е отново $O(n^2)$ и е изключително бавен. Сортирането с пряка селекция впечатлява с простотата си, а също така в дадени ситуации има предимства пред някои сложни алгоритми. Подобен алгоритъм който има по добро производителност е следващия алгоритъм „сортиране чрез вмъкване."

- *Сортиране чрез вмъкване (Insertion sort)*

Това е най ефективният на практика от „бавните" алгоритми. Често се използва в съчетания с някои по усъвършенствани такива. Алгоритъмът работи като взема всеки елемент един по-един от списъка и го вмъква на съответното си място в нов сортиран списък. При сортиране на масиви елементите могат да споделят базовата памет (пространство), но се извършват прекалено много премествания на елементи което е скъпа операция. Като при останалите бавни алгоритми неговата изчислителна сложност спрямо Big O нотацията е $O(n^2)$.

II. ПРАКТИЧЕСКА ЧАСТ

В практическата част са показани примери, които показват действието на разглежданите алгоритми за сортиране.

ЗАДАЧА1: *Да се сортира следния масив от числа по методите на разгледаните елементарни алгоритми за сортиране в теоретичната част.*

{43, 36, 58, 41, 35}

РЕШЕНИЕ:

1. Метод на мехурчето.

*1) {**43, 36**, 58, 41, 35} -> {36, **43, 58**, 41, 35} -> {36, 43, **58, 41**, 35}-> {36, 43, 41, **58, 35**}->*
*-> {36, 43, 41, 35, **58**}*
*2) {**36, 43**, 41, 35, **58**} -> {36, **43, 41**, 35, **58**} -> {36, 41, **43, 35**, **58**} -> {36, 41, 35, **43**, **58**}*
*3) {**36, 41**, 35, **43**, **58**} -> {36, **41, 35**, **43**, **58**} -> {36, 35, **41**, **43**, **58**}*
*4) {**36, 35**, **41**, **43**, **58**} -> {**35**, **36**, **41**, **43**, **58**}*
*5) {**35**, **36**, **41**, **43**, **58**}*

2. Сортиране чрез пряка селекция

*1) {43, 36, 58, 41, **35**} -> {**35**, 36, 58, 41, 43}*
*2) {**35**, **36**, 58, 41, 43}*
*3) {**35**, **36**, 58, **41**, 43} -> {**35**, **36**, **41**, 58, 43}*
*4) {**35**, **36**, **41**, 58**, 43**} -> {**35**, **36**, **41**, **43**, 58}*

5) *{**35**, **36**, **41**, **43**, **58**}*

3. Сортиране чрез вмъкване

1) *{**43**, 36, 58, 41, 35} -> {**43**, 36, 58, 41, 35}*
2) *{**43**, **36**, 58, 41, 35} -> {**36**, **43**, 58, 41, 35}*
3) *{**36**, **43**, **58**, 41, 35} -> {**36**, **43**, **58**, 41, 35}*
4) *{**36**, **43**, **58**, **41**, 35} -> {**36**, **41**, **43**, **58**, 35}*
5) *{**36**, **41**, **43**, **58**, **35**} -> {**35**, **36**, **41**, **43**, **58**}*

ПОЯСНЕНИЕ

1. Метод на мехурчето:

При първото преминаване първоначално се сравняват първите два елемента. Те си разменят местата понеже 43>36. Алгоритъмът продължава като се сравняват втория и третия елемент в случая 43 и 58. Тук втория елемент (в случая 43) е по малък от третия (в случая 58) и затова те си запазват местата. Алгоритъмът продължава със сравняването на 3 ти и 4ти елемент, след което с 4ти и 5ти елемент. Разменяне на местата се извършва ако предходния елемент е по голям от съседния следващ. Така след първо преминаване през всички елемент чрез алгоритъма на мехурчето се стига до следната подредба *{36, 43, 41, 35, **58**}*.

Алгоритъмът продължава със второ преминаване. То отново започва да сравнява първите два съседни елемента (в случая това 36 и 43). Следват се аналогичните стъпки от първо преминаване. В края на второ преминаване се стига до следната подредба {36, 41, 35, 43, 58} Алгоритъмът продължава по аналогичен начин докато не се стигне преминаване при което не се разменят нито един елемент.

Удебелените елементи са тези които се сравняват в дадения момент, а подчертаните са тези които вече са сортирани.

2. Сортиране чрез пряка селекция

В несортирания списък се търси най малкия елемент и той разменя своето място със първия елемент. В случая това е елемента 35 и затова разменя своето място със елемента 43. Следва търсене на най – малкия елемент в несортираната част. Това е втория елемент той си е на мястото и затова не следва размяна. Алгоритъма се повтаря докато списъка с несортирани елементи не приключи.

Подчертаните елементи са вече сортираните. Удебелените са най малкия елемент в несортираната част.

3. Сортиране чрез вмъкване

Взема се първия елемент и се добавя на подходящото място във сортираната част на алгоритъма. Понеже това е все още първи елемент и няма сортирана част то този елемент сам по себе си е сортиран и не се извършва никакво разместване. Алгоритъма продължава като се взема втория елемент и се добавя на неговото място в сортираната част. Към този момент сортираната част се състои единствено от елемента 43. Елементът 36 трябва да е преди елемента 43. Затова сортираните елементи по големи от 36 се местят с една стъпка на дясно и 36 се поставя на съответното място предназначено за него. Алгоритъма продължава докато не приключат елементите в несортираната част от списъка.

Подчертаните елементи са сортираната част от списъка, а удебелените са първия елемент от несортираната част.

ЗАДАЧА1: *Напишете програмна функция който сортира горния масив по метода на мехурчето.*

КОД:

```
void Bubblesort(int a[], int n)
{
    int r, i, j;
    for (i = 0; i < n - 1; i++)
    {    for (j = 0; j < n - i - 1; j++)
           if (a[j] > a[j + 1])
           {      r = a[j];       a[j] = a[j + 1];     a[j + 1] = r;}
    }
}
```

ПОЯСНЕНИЕ

Функцията приема два аргумента от целочислен тип. Първият е масив от цели числа, който трябва да се сортира, а втората е броя на елементите на масива. Създават се два вложени for цикъла които обхождат масива. Вътрешния цикъл обхожда масива като сравнява два по два съседните стойности. Ако открие че предходния елемент е по голям от съседния след него то ги разменя. В края на този цикъл най големия елемент ще е най накрая. Външния for цикъл служи за граница на вътрешния понеже след като най големия елемент е отишъл на последната позиция той вече е подреден и няма смисъл да се сравнява. Така че външния for цикъл намаля всеки път обхвата с 1 на вътрешния. Накрая когато външния цикъл приключи своето изпълнение значи че вътрешния цикъл е направил всички възможни обхождания, което ще означава че алгоритъма е сортиран.

III. Задача за самостоятелна работа.

1. Демонстрирайте горните алгоритми за следния масив:

   37, 60, 65, 17, 91, 84, 41
2. Напишете програма сортираща по метода на мехурчето за масив от произволно зададени 1000 елемента.
3. Подобрете алгоритъма от задача 1 така че ако алгоритъма не прави повече размени след едно пълно преминаване то да означа че масива вече е сортиран и програмата да приключва своето изпълнение. Сравнете времето за изпълнение за стандартния вариант от задача 1 и от този с подобрението.
4. Напишете програма за сортиране на масив от 1000 елемента чрез метода за пряка селекция и метода чрез вмъкване.
5. Сравнете алгоритмите по бързодействие. Тествайте и с друг списък от числови елементи.