ЛЕКЦІЯ 3. МЕТОДИ СОРТУВАННЯ .

3.1. Задача сортування

Загальна задача сортування полягає в наступному:

нехай дано множину елементів, яка є індексованою, тобто довільно пронумерованою від *1* до *n* . Необхідно індексувати цю множину елементів так, щоб з умови $i<j$ витікало $a_i < a_j$ - для всіх$i,j=1..n$.

Отже, процес сортування полягає у послідовних перестановках елементів доти, доки їх індексація не узгодиться з їх впорядкованістю.

Будемо розглядати ефективністьалгоритмів у термінах розмірності множини з точки зору простоти програмування. Задачу сортування зручніше розглядати в застосуванні до одновимірних масивів (векторів).

Нехай маємо вектор з *n* елементів $R_1 ,R_2 ,...,R_n$ .. Задача сортування полягає у тому, щоб знайти таку перестановку елементів вектора, після якої вони розмістилися б у зростаючому порядку їх значень.

Сортування називається стійким, якщо елементи з однаковими значеннями залишаються на попередньому місці. Для простоти аналізу будемо вважати, що всі елементи масиву, який сортується, займають однакові кванти пам'яті і ніякі два елементи не можуть мати рівних значень, але в алгоритмах такий випадок повинен бути передбачений.

Наведемо основні алгоритми у формальному аспекті і розглянемо на прикладах їх роботу.

3.2.Метод простої вибірки.

Задано масив елементів $R_1 ,R_2 ,...,R_n$.Даний алгоритм реорганізує масив у висхідному порядку, тобто для його елементів буде мати місце співвідношення$R_i < R_j$ - для всіх $i,j=1..n$.

Алгоритм S.

S1. Цикл за індексом проходження. Повторювати кроки S2 - *S* 4 при i=1..n-1.

S2. Зафіксувати перший поточний елемент: встановити $R0 = R_i$.

S3. Пошук найменшого значення $min\ R_j$для елементів з індексом j=i+1,i+2,…,n

S4. Перестановка елементів. Якщо $min\ R_j < R0$ та j !=i , то $min\ R_j <-> R0$.

S5. Кінець. Вихід.

З алгоритму S видно, що для сортування потрібно виконати *n-1* проходження послідовності елементів. Одним проходженням називаємо пошук елементу з наступним найменшим значенням.

Проведемо невеликий аналіз алгоритму. При першому проходженні, коли знаходиться елемент з найменшим значенням, порівнюється *n-1* елементів. У загальному випадку при і -му проходженні у процесі сортування порівнюється *n-i*елементів. Тоді загальна кількість порівнянь,

які треба виконати для сортування масиву із *n* елементів буде:

$$\Sigma\ (n-i) = (n-1) + (n-2) + \ldots + (n-n+1) = 1/2n(n-1),\ i=1..n-1$$

Таким чином, ефективність алгоритму пропорційна величині n^2 (говорять, щоалгоритм S має ефективність $O(n^2)$).

Кількість перестановок елементів залежить від того, як на початку був відсортований масив. Але, оскільки при одному проходженні у даному алгоритмі потрібно виконати не більш як одну перестановку, максимальна кількість перестановок при такому сортуванні дорівнює величині *n-1*.

3.3. Метод бульбашки.

Другим добре відомим методом із класу вибірки є метод, що ґрунтується на ідеї спливаючої бульбашки . На відміну від попереднього в даному алгоритмі два елементи обмінюються місцями, як тільки виявлено, що між ними порушений порядок.

Алгоритм В.

Задано масив елементів $R_1, R_2, \ldots, R_n$.

Даний алгоритм реорганізує масив у висхідному порядку, тобто для його елементів буде мати місце співвідношення$R_i < R_j$ - для всіх $i,j=1..n$.

B1. Цикл за індексом проходження. Повторювати кроки B2 і B3 при $i=1..n-1$.

B2. Ініціалізація прапорця перестановки: встановити $Fl=0$.,

B3. Виконання проходження. Повторювати при $j=1,2,\ldots,n-i$: якщо $R_{j+1}<R_j$, то встановити $Flg=1$ та переставити місцями елементи $R_j<->R_{j+1}$, ; якщо $Fl=0$, то завершити виконання алгоритму.

B4. Кінець. Вихід.

Робота алгоритму В очевидна. Перед кожним проходженням змінна *Fl* приймає значення нуль. Її значення аналізується в кінці кожного кроку. Якщо воно не змінилося, сортування виконане повністю. Характеристика сортування бульбашкою в гіршому випадку складає$1/2n(n-1)$ порівнянь і $1/2n(n-1)$ перестановок. Середня кількість проходжень приблизно дорівнює величині $1,25nO\ n$. Наприклад, якщо $n=10$, потрібно виконати шість проходжень послідовності.

Середня кількість порівнянь і перестановок також пропорційна величині n^2. Отже складність алгоритму сортування бульбашкою становить $O(n^2)$.Цей метод неефективний для масивів великого розміру. Існує багато модифікацій даного алгоритму.

3.4.Швидкий метод сортування

В класі алгоритмів вибірки слід відзначити так зване швидке сортування, в якому виконується наступна схема обмінів .

Є два вказівники *i* та *j* , причому на початку *i = 1, j = n.* , де *n* кількість елементів масиву. Довільним чином вибирається за базовий будь-який елемент з масиву (перший, середній або останній). Нехай, наприклад, це буде перший елемент $X= R_1$. Встановлюємо$i=1$ (від першого) , $j =n$ (від останнього), якщо знайдено $R_i >=X$ та $R_j < X$, то потрібно провести обмін $R_i <-> R_j$ при умові що $i<j$. Після першого обміну збільшуємо *i* на одиницю та шукаємо $R_i>=X$. Якщо такий елемент знайдено то *j* зменшуємо на одиницю і шукаємо $R_j < X$. Проводимо наступний обмін. Якщо $R_i>=X$ не знайдено, а $i>=j$, то перша ітерація закінчена. Отже, алгоритм працює за принципом "спалюванню свічки з обох кінців" . Масив буде розділений наступним чином: $R_1 ,R_2 ,...,R_{i-1,} R_i ,R_{i+1} ,...,R_n$ причому $R_l < X$, $l=1,...,i-1;$ $R_m <= X,$ $m=i+1,...,n.$ В лівій частині масиву будуть стояти всі елементи, що є меншими від базового, а в правій – що є більшими та сам базовий елемент. Потім до кожної з цих підмножин рекурсивно застосовується даний метод. Рекурсія закінчується, коли всі підмножини будуть складатися з одного елементу, або весьмасив буде впорядкований.

Розглянемо приклад:

Нехай масив включає наступні елементи:

5 3 2 6 4 1 3 7

Встановлюємо i=1, j=8, X=5.

Для i=1 виконується умова 5>=5, для j=7 виконується умова 3<5. Оскільки $i<j$, то проводимо обмін місцями знайдених елементів 5<->3. Повторюємо кроки для новоутвореного масиву 3 3 2 6 4 1 5 7 :для i=4, 6>5, для j=6, 1<5, $i<j$, проводимо обмін місцями елементів 6<->1. При наступній ітерації умова $i<j$ не виконається

3 3 2 1 4 6 5 7, i=8, j=5 i>j, отже перший прохід розділить масив на дві частини. Рекурсивно відсортовуємо по черзі обидві частини.

Час роботи алгоритму швидкого сортування залежить від збалансованості, що характеризує розбиття. Збалансованість, у свою чергу залежить від того, який елемент обрано як базовий (відносно якого елемента виконується розбиття). Якщо розбиття збалансоване, то асимптотично алгоритм працює так само швидко як і алгоритм сортування злиттям. У найгіршому випадку, асимптотична поведінка алгоритму настільки ж погана, як і в алгоритму сортування включенням.

- Найгірше розбиття. Найгірша поведінка має місце у тому випадку, коли процедура, що виконує розбиття, породжує одну підзадачу з $(n – 1)$ елементом, а другу - з 0 елементами. Нехай таке незбалансоване розбиття виникає при кожному рекурсивному виклику. Для самого розбиття потрібен час $\Theta(n)$. Тоді рекурентне співвідношення для часу роботи, можна записати наступним чином:

$$T(n) = T(n – 1) + T(0) + \Theta(n) = T(n – 1) +\Theta(n).$$

Розв’язком такого співвідношення є: $T(n) =\Theta(n^2)$.

- Найкраще розбиття. В найкращому випадку процедура поділу ділить задачу на дві підзадачі, розмір кожної з яких не перевищує $(n / 2)$. Час роботи описується нерівністю: $T(n) \le 2\cdot T(n / 2) + \Theta(n)$. Тоді: $T(n) =O(n\cdot\log(n))$ — асимптотично найкращий час.
- Середній випадок. Математичне очікування часу роботи алгоритму на всіх можливих вхідних масивах є$O(n\cdot\log(n))$, тобто середній випадок ближчий до найкращого.