Лабораторна робота №5

Прості методи сортування

Мета роботи: реалізація простих алгоритмів сортування та дослідження їх характеристик (швидкодія, необхідний обсяг пам'яті, застосування тощо).

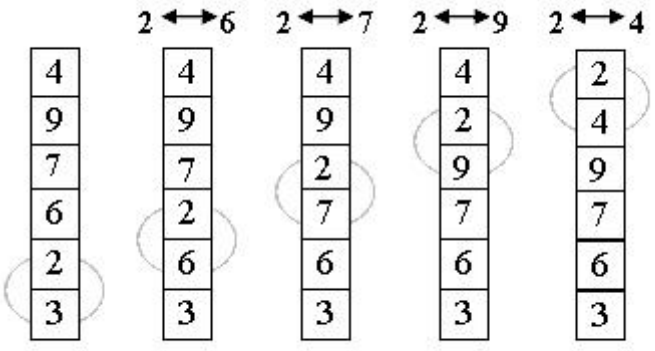
Методичні вказівки

Під **сортуванням** звичайно розуміють процес перестановки об'єктів заданої множини у певному порядку.

Мета сортування – полегшення подальшого пошуку елементів у відсортованій множині. У цьому сенсі елементи сортування присутні майже у всіх завданнях.

1. Сортування бульбашкою

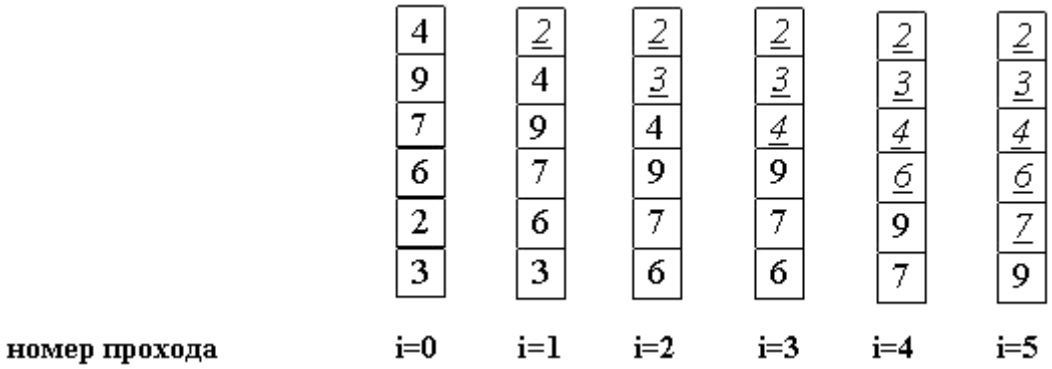
Розташуємо масив зверху вниз, від нульового елемента – до останнього. Ідея методу: крок сортування полягає в проході знизу вгору по масиву. По дорозі проглядаються пари сусідніх елементів. Якщо елементи деякої пари знаходяться в неправильному порядку, то міняємо їх місцями.



Нульовий прохід. Показано пари, які обмінюються.

Після нульового проходу по масиву, зверху розташовується самий "легкий" елемент – звідси аналогія з бульбашкою. Наступний прохід виконується до другого верхнього елементу, таким чином другий за величиною елемент піднімається на правильну позицію ...

Виконуємо проходи по всій нижній частині масиву до тих пір, поки в ній не залишиться тільки один елемент. На цьому сортування закінчується, так як послідовність впорядкована за зростанням.



Середнє число порівнянь і обмінів мають квадратичний порядок зростання: $O(n^2)$, звідси можна зробити висновок, що алгоритм бульбашки дуже повільний і малоефективний. Проте, у нього є величезний плюс: він простий і його можна по-всякому поліпшувати.

По-перше, розглянемо ситуацію, коли на якому-небудь з проходів не відбулося жодного обміну. Що це означає? Це означає, що всі пари розташовані в правильному порядку, так що масив вже відсортований і продовжувати процес не має сенсу (особливо, якщо масив був відсортований з самого початку!). Отже, перше поліпшення алгоритму полягає в запам'ятовуванні, чи проводився на даному проході будь-якої обмін. Якщо немає – алгоритм закінчує роботу.

Процес поліпшення можна продовжити, якщо запам'ятовувати не тільки сам факт обміну, але і індекс **k** останнього обміну. Дійсно, всі пари сусідів елементів з індексами меншими за k, вже розташовані в потрібному порядку. Подальші проходи можна закінчувати на індексі k, замість того щоб рухатися до встановленої заздалегідь верхньої межі.

Якісно інше поліпшення алгоритму можна отримати з наступного спостереження. Хоча “легка” бульбашка знизу підніметься наверх за один прохід, важкі бульбашки опускаються з мінімальною швидкістю: один крок за ітерацію. Тому масив 2 3 4 5 6 1 буде відсортований за 1 прохід, а сортування послідовності 6 1 2 3 4 5 зажадає 5 проходів.

Щоб уникнути подібного ефекту, можна змінювати напрямок наступних проходів. Одержаний алгоритм іноді називають "шейкер-сортуванням".

Наскільки описані зміни впливають на ефективність методу? Середня кількість порівнянь хоч і зменшилася, але залишається $O(n^2)$, в той час як число обмінів залишається незмінною. Середня (гірша) кількість операцій залишається квадратичною.

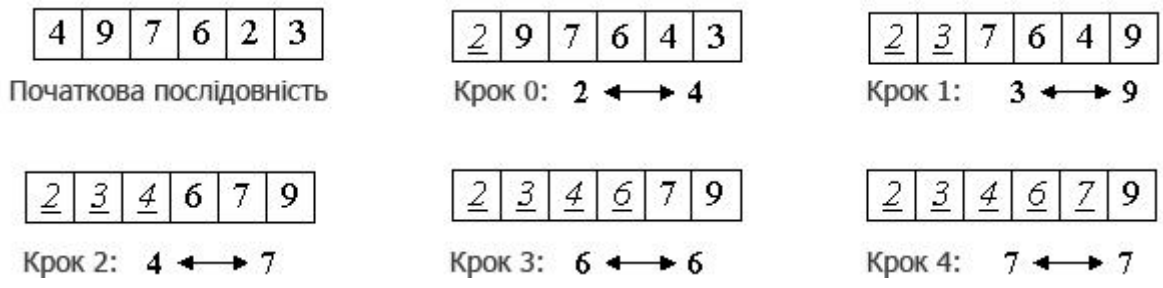
Додаткова пам'ять, очевидно, не потрібна. Поведінка вдосконаленого методу досить природня, майже відсортований масив буде відсортований набагато швидше випадкового. Сортування бульбашкою стійке, однак шейкер-сортування втрачає цю якість. На практиці метод бульбашки, навіть з поліпшеннями, працює занадто повільно. Тому майже не застосовується.

2. Сортування вибором

Ідея методу полягає в тому, щоб створювати відсортовану послідовність шляхом приєднання до неї одного елемента за одним в правильному порядку.

Будемо будувати готову послідовність, починаючи з лівого кінця масиву. Алгоритм складається з **n** послідовних кроків, починаючи від нульового і закінчуючи (**n-1**).

На i-му кроці вибираємо найменший з елементів a[i] ... a[n-1] і міняємо його місцями з a[i]. Послідовність кроків при n = 6 зображена на малюнку нижче:



Незалежно від номера поточного кроку i, послідовність a[0]...a[i] є впорядкованою. Таким чином, на (n-2)-му кроці вся послідовність, крім a[n-1] виявляється відсортованої, а a[n-1] стоїть на останньому місці по праву (всі менші елементи вже розміщені звліва.

Для знаходження найменшого елемента з n елементів, необхідно зробити n-1 порівнянь. З урахуванням того, що кількість аналізованих елементів на кожному кроці зменшується на одиницю, загальна кількість операцій:

$(n-1) + (n-2) + (n-3) + ... + 1 = 1/2 * (n^2+n) = O(n^2)$.

Таким чином, оскільки число обмінів завжди буде меншою за кількість порівнянь, час сортування зростає квадратично щодо кількості елементів.

Алгоритм не використовує додаткову пам'ять (всі операції відбуваються "на місці").

Чи стійкий цей метод? Розглянемо послідовність з трьох елементів, кожен з яких має два поля, а сортування йде за першою з них.

Результат сортування можна побачити вже після кроку 0, так як більше обмінів не буде. Порядок ключів 2a, 2b був змінений на 2b, 2a, так що метод нестійкий.

Якщо вхідна послідовність майже впорядкована, то порівнянь буде стільки ж, значить алгоритм поводиться неприродно.

3. Сортування вставками

Сортування простими вставками в чомусь схожа на вищевикладені методи.

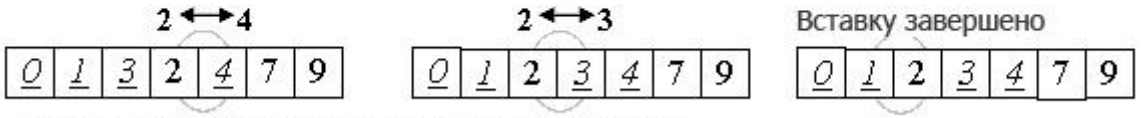
Аналогічним чином виконуються проходи по частині масиву та аналогічним же чином на початку масиву "виростає" відсортована послідовність. Однак в сортуванні бульбашкою або вибором можна було чітко заявити, що на i-му кроці елементи a[0]... a[i] стоять на правильних місцях і нікуди більш не перемістяться. Тут же подібне твердження буде більш слабким: послідовність a[0] ... a[i] впорядкована, але при цьому по ходу алгоритму в неї будуть вставлятися нові елементи.

Будемо розбирати алгоритм, розглядаючи його дії на i-му кроці. Як говорилося вище, послідовність до цього моменту розділена на дві частини: готову a [0] ... a [i] і невпорядковану a [i + 1] ... a [n].

На наступному, (i + 1)-м кожному кроці алгоритму, беремо a[i + 1] і вставляємо на потрібне місце в готову частину масиву.

Пошук відповідного місця для чергового елемента вхідної послідовності здійснюється шляхом послідовних порівнянь з елементом, що стоїть перед ним. Залежно від результату порівняння елемент або залишається на поточному місці (вставка завершена), або вони міняються місцями і процес повторюється.

Послідовність на даний момент. Перші 3 елементи упорядковані.



Вставка числа 2. Показано пари, які обмінюються.

Таким чином, в процесі вставки ми "просіюємо" елемент x до початку масиву, зупиняючись в разі, коли:

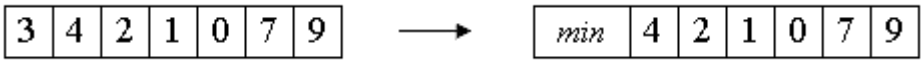
1. Знайдено елемент, менший x або ...

2. Досягнуто початок послідовності.

Аналогічно сортуванні вибором, середнє, а також гірше число порівнянь і пересилань оцінюються як O (n^2), додаткова пам'ять при цьому не використовується.

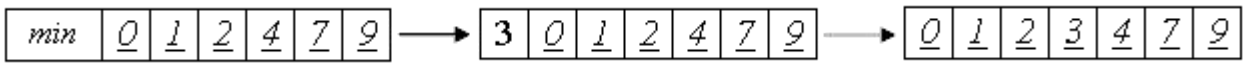
Хорошим показником сортування є дуже природна поведінка: майже відсортований масив буде відсортован дуже швидко. Це, вкупі зі стійкістю алгоритму, робить метод хорошим вибором в відповідних ситуаціях.

Алгоритм можна злегка поліпшити. Зауважимо, що на кожному кроці внутрішнього циклу перевіряються 2 умови. Можна об'єднати їх в одне, поставивши в початок масиву спеціальний сторожовий елемент. Він повинен бути свідомо менше всіх інших елементів масиву.

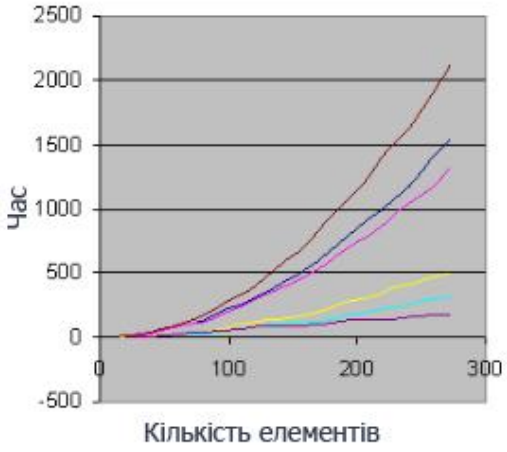


Тоді при j=0 буде наперед вірно a[0] <= x. Цикл зупиниться на нульовому елементі, що і було метою умови j>=0.

Таким чином, сортування відбуватиметься правильним чином, а у внутрішньому циклі стане на одне порівняння менше. З урахуванням того, що воно проводилося O(n^2) раз, то це – реальна перевага. Однак, відсортований масив буде не повний, так як з нього зникло перше число. Для закінчення сортування це число слід повернути назад, а потім вставити в відсортовану послідовність a[1]...a[n].



Зображений нижче графік ілюструє різницю в ефективності декількох алгоритмів сортування.



- коричнева лінія – сортування бульбашкою;
- синя лінія – шейкер-сортування;
- рожева лінія – сортування вибором;
- жовта лінія – сортування вставками;
- блакитна лінія – сортування вставками зі сторожовим елементом;
- фіолетова лінія – сортування Шелла.

Порядок виконання роботи

1. Реалізувати алгоритми сортування:
   1) сортування вибором (структура даних – двусвязний список);
   2) сортування вставками (структура даних – масив);
   3) сортування вставками (структура даних – двусвязний список);
2. Виміряти час сортування даних різної розмірності: 10, 100, 500, 1000, 2000, 5000, 10000. Дані сформувати з використанням генератора випадкових чисел, де в якості початкового значення використати номер Вашого варіанту (для вимірювання інтервалів часу можна використовувати клас Stopwatch з простору імен System.Diagnostics, якщо для C # або використати бібліотеку C++ <chrono>).
3. За отриманими даними побудувати графіки залежностей часу сортування від кількості вхідних даних (з використанням Excel).

Формат вхідних даних:

Єдиний рядок вхідних даних містить 3 цілих числа, розділених пробілами:

N V K

де:

N – номер алгоритму сортування (від 1 до 3);

V – номер варіанту (вхідний параметр для функції srand());

K – приймає кількість значень для сортування

Формат вихідних даних:

Програма повинна вивести розраховані перші 5 значень через пробіл

Приклад:

Вказуємо, наприклад, такі значення:

1 12 100

Результат виведення перших 5 значень у консоль:

12 345 2 123 30149 5376

Зміст звіту

1. Опис алгоритмів (словесна форма або блок-схема алгоритму).
2. Текст функцій сортування з коментарями.
3. Таблиця результатів вимірів часу.
4. Графіки результатів вимірів часу.
5. Висновки по роботі (опис досліджених характеристик кожного алгоритму, порівняння алгоритмів, відзначити переваги та недоліки).