Лабораторна робота № 8

Тема: програмування та аналіз алгоритму сортування лінійного рівня складності.
Мета: проаналізувати та дослідити алгоритм сортування за лінійний час

Основні теоретичні відомості

Алгоритми сортування за лінійний час

У ряді попередніх лабораторних робіт ми досліди різні алгоритми, які можуть відсортувати n чисел за час $O(n^2)$ та $O(n \cdot \log n)$. Алгоритми сортування злиттям (merge sort) і сортування за допомогою купи (heap sort) працюють за час $O(n \cdot \log n)$ у найгіршому випадку, а для алгоритму швидкого сортування – це є середній час роботи. Оцінка $O(n \cdot \log n)$ достатньо точна: для кожного із цих алгоритмів можна подати послідовність із n чисел, час обробки якої буде $\Omega(n \cdot \log n)$.

Усі вищезгадані алгоритми здійснюють сортування, ґрунтуючись виняткова на попарних порівняннях елементів, тому їх іноді називають сортуваннями порівнянням (comparison sort). Проте будь-який алгоритм такого типу сортує n елементів за час не менше $\Omega(n \log n)$ у найгіршому випадку. Таким чином, алгоритми сортування злиттям і за допомогою купи асимптотично оптимальні: не існує алгоритму сортування порівнянням, що перевищував би зазначені алгоритми більш ніж у скінченне число раз.

У рамках даної лабораторної роботи розглядаються три алгоритми сортування (сортування підраховуванням, цифрове сортування і сортування вичерпуванням), що працюють за лінійний час. *Ваша задача розібратися зі усіма трьома, але реалізувати лише один з них.* Зрозуміло, що дані алгоритми поліпшують оцінку $\Omega(n \log n)$ за рахунок того, що використовують не лише попарні порівняння, але й внутрішню структуру об'єктів, що відсортовуються.

1 Сортування підраховуванням

Алгоритм сортування підраховуванням (counting sort) застосовують, якщо кожний з n елементів послідовності, що відсортовується, – ціле додатне число у відомому діапазоні (що не переважає заздалегідь відоме k). Якщо $k=O(n)$, то алгоритм сортування підраховуванням працює за час $O(n)$.

Ідея цього алгоритму в тому, щоб для кожного елементу x попередньо підрахувати, скільки елементів вхідної послідовності менше x, після

чого записати *x* прямо у вихідний масив відповідно до цього числа (якщо, скажімо, 17 елементів вхідного масиву менше *x*, то у вихідному масиві *x* повинен бути записаний на місце номер 18). Якщо в послідовності, що відсортовується, присутні рівні числа, цю схему треба дещо модифікувати, щоб не записати кілька чисел на одне місце.

У псевдокоді, що приводиться нижче, використовується допоміжний масив *C[1... k]* із *k* елементів. Вхідна послідовність записана в масиві *A[1...n]*, відсортована послідовність записується у масив *B[1...n]*.

```
Counting-Sort(A, B, k)
1     for i ← l  to k
2           do C[i] ← 0
3     for j ←l  to length [A]
4           do C[A[j]] ← C [A[j]] +1
5     ⇒ C[i] дорівнює кількості елементів рівних i.
6     for i ←2 to k
7           do C[i] ←C[i] ←C[i -1]
8     ⇒ C[i] дорівнює кількості елементів, що не більші i.
9     for j ←length [A] downto 1
10          do B[C[A[j]]] ←A[j]
11                C[A[j]] ←C[A[j]] - 1
```

Роботу алгоритму сортування підраховуванням проілюстровано на рис. 1. Після ініціалізації (рядки 1-2) ми спочатку поміщаємо в *C[i]* кількість елементів масиву *A*, рівних *i* (рядки 3-4), а потім, знаходячи часткові суми послідовності *C[1], C[2],..., C[k]* – кількість елементів, що не перевищують *i* (рядки 6-7). Нарешті, у рядках 9-11 кожний з елементів масиву *A* розміщується на потрібне місце в масиві *B*. Справді, якщо всі *n* елементів різні, то у відсортованому масиві число *A[j]* повинно стояти на місці номер *C[A[j]]*, тому що саме стільки елементів масиву *A* не перевищують *A[j]*; якщо в масиві *A* зустрічаються повторення, то після кожного записування числа *A[j]* у масив *B* число *C[A[j]]* зменшується на одиницю (рядок 11), так що при наступній зустрічі із числом, рівним *A[j]*, воно буде записано на одну позицію лівіше.

Оцінимо час роботи алгоритму сортування підраховуванням. Цикли в рядках 1-2 і 6-7 працюють за час *O(k)*, цикли в рядках 3-4 і 10-11 – за час *O(n)*, а весь алгоритм працює за час *O(k+ n)*. Якщо *k=O(n)*, то час роботи є *O(n)*.

Варто відзначити, що алгоритм сортування підраховуванням не порівнює числа, що відсортовуються, між собою, а використовує їх як індекси масиву.

Алгоритм сортування підраховуванням має важливу властивість, яку називають стійкістю (is stable). А саме, якщо у вхідному масиві присутні декілька рівних чисел, то у вихідному масиві вони розташовуються у тому ж порядку, що й у вхідному. Ця властивість має сенс, тільки якщо в масиві разом із числами записані додаткові дані.

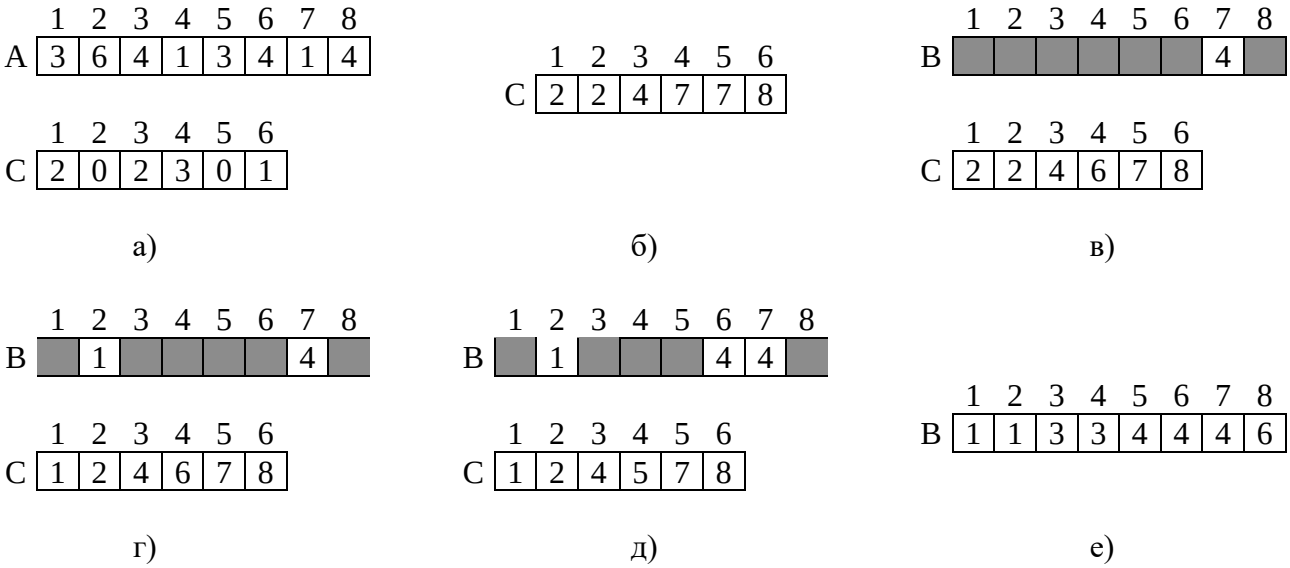

Рисунок 1 – Робота алгоритму Counting-Sort, застосованого до масиву A[1..8], що складається з натуральних чисел, які не перевищують *k=6.*
а – масив *A* та допоміжний масив *C* після виконання циклу в рядках 3-4;
б – масив *C* після виконання циклу в рядках 6-7;
в, г, д – вихідний масив *B* і допоміжний масив *C* після одного, двох і трьох повторень циклу в рядках 9-11. Закреслені клітинки відповідають елементам масиву, значення для яких ще не присвоєні;
е – масив *B* після закінчення роботи алгоритму

Більш детально уявимо собі, що ми сортуємо не лише числа, а пари *(t,x)*, де *t* – число від 1 до *k*, а *x* – довільний об'єкт, і хочемо переставити їх так, щоб перші компоненти пар йшли в неспадному порядку. Описаний алгоритм дозволяє це зробити, причому відносне розташування пар з рівними першими компонентами не змінюється. Ця властивість і називається стійкістю.

2 Цифрове сортування

Алгоритм цифрового сортування (radix sort) використовувся у машинах для сортування перфокарт (зараз такі машини є історією комп’ютерних систем). У картонних перфокартах спеціальний перфоратор пробивав дірки. У кожному з 80 стовпців були місця для 12 прямокутних дірок. Взагалі ж в одній колонці можна було пробити кілька дірок, їхня комбінація відповідала символу (таким чином на карті було місце

для 80 символів), але цифри 0-9 кодувалися одиночними дірками в рядках 0-9 відповідного стовпця.

Сортувальній машині вказували стовпець, за яким потрібно зробити сортування, і вона розкладала колоду перфокарт на 10 стовпців залежно від того, яка з дірок 0-9 була пробита в зазначеному стовпці.

Як відсортувати колоду перфокарт із багатозначними числами (розряд одиниць в одному стовпці, десятків – у попередньому і т. д.)? Перше, що спадає на думку, – почати сортування зі старшого розряду. При цьому вийде 10 стовпців, кожен з яких на наступному кроці необхідно буде розбивати на 10 стовпців і так далі – вийде багато стовпців перфокарт, у яких легко заплутатися.

Як не дивно, виявляється зручніше почати з молодшого розряду, розклавши колоду на 10 стовпців залежно від того, де пробитий отвір в «молодшому» стовпці. Отримані 10 стовпців треба після цього скласти в один в такому порядку: спочатку карти з 0, потім карти з 1 і т.д. Отриману колоду знову розсортуємо на 10 стовпців, але вже відповідно до розряду десятків, складемо отримані стовпців в одну колоду і т.д.; якщо на перфокартах були записані *d*-значні числа, то знадобиться *d* раз скористатися сортувальною машиною. На рис. 3.22 зображено, як діє даний алгоритм, який застосовано до семи тризначних чисел.

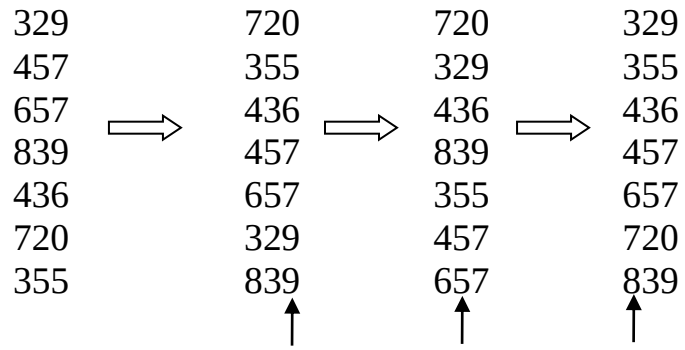

Рисунок 2 – Цифрове сортування послідовності із 7 тризначних чисел. На вхід подаються числа першого стовпця. Далі показано порядок чисел після сортування за третьою, другою і першою цифрами (вертикальні стрілки вказують за якою цифрою здійснювалось сортування)

Важливо, щоб алгоритм, за допомогою якого відбувається сортування за даним розрядом, був стійким: картки, в яких у даному стовпці знаходиться одна і та ж цифра, повинні вийти із сортувальної машини в тій же послідовності, в якій вони туди подавалися (зрозуміло, що при складанні 10 стовпців в один змінювати порядок карт у стовпцях теж не потрібно).

У комп'ютерах цифрове сортування іноді використовується для впорядкування даних, що містять декілька полів. Нехай, наприклад, нам потрібно відсортувати послідовність дат. Це можна зробити за допомогою будь-якого алгоритму сортування, порівнюючи дати в такий спосіб: порівняти роки, якщо роки збігаються – порівняти місяці, якщо збігаються й місяці – порівняти числа. Проте замість цього можна просто тричі відсортувати масив дат за допомогою стійкого алгоритму: спочатку за днями, потім за місяцями, потім за роками.

Програму для цифрового сортування написати нескладно. Ми припускаємо, що кожний елемент *n*-елементного масиву *A* складається з *d* цифр, причому цифра номер 1 – молодший розряд, а цифра номер *d* – старший.

```
RADIX-SORT(A, d)
1     for i ← 1 to d
2           do відсортувати масив A стійким
                  алгоритмом за значенням цифри номер i
```

Правильність алгоритму цифрового сортування доводиться індукцією за номером розряду. Час роботи залежить від часу роботи обраного стійкого алгоритму. Якщо цифри можуть приймати значення від 1 до *k,* де *k* не занадто велике число, то очевидний вибір – сортування підраховуванням. Для *n* чисел з *d* знаками від 0 до *k - 1* кожний прохід забирає час $\theta(n+\kappa)$; оскільки ми робимо *d* проходів, час роботи цифрового сортування дорівнює $\theta(dn + kd)$. Якщо *d* стале та $k=O(n)$, то цифрове сортування здійснюється за лінійний час.

При цифровому сортуванні важливо правильно вибрати основу системи числення, оскільки від неї залежить розмір необхідної додаткової пам'яті та час роботи. Конкретний приклад: нехай треба відсортувати мільйон 64-бітових чисел. Якщо розглядати їх як чотиризначні числа в системі числення з основою 2^{16}, то при цифровому сортуванні ми впораємося з задачею за чотири проходи, використовуючи в процедурі ***Counting-Sort*** масив *B* розміром 2^{16} (це невелике значення в порівнянні з розміром відсортованого масиву). Це вигідно відрізняється від сортування порівнянням, коли на кожне число витрачається по $\log n \approx 20$ операцій. На жаль, цифрове сортування, що базується на сортуванні підраховуванням, потребує ще одного масиву (того ж розміру що й масив сортуваннння) для зберігання проміжних результатів, у той час як багато алгоритмів сортування порівнянням обходяться без цього. Тому, якщо треба заощаджувати пам'ять, алгоритм швидкого сортування може виявитися більш вигідним для застосування.

3 Сортування вичерпуванням

Алгоритм сортування вичерпуванням (bucket sort) працює за лінійний середній час. Як і сортування підраховуванням, сортування вичерпуванням застосовується не для будь-яких вихідних даних: вказуючи на лінійний середній час, ми припускаємо, що на вхід подається послідовність незалежних випадкових чисел, рівномірно розподілених на проміжку [0,1].

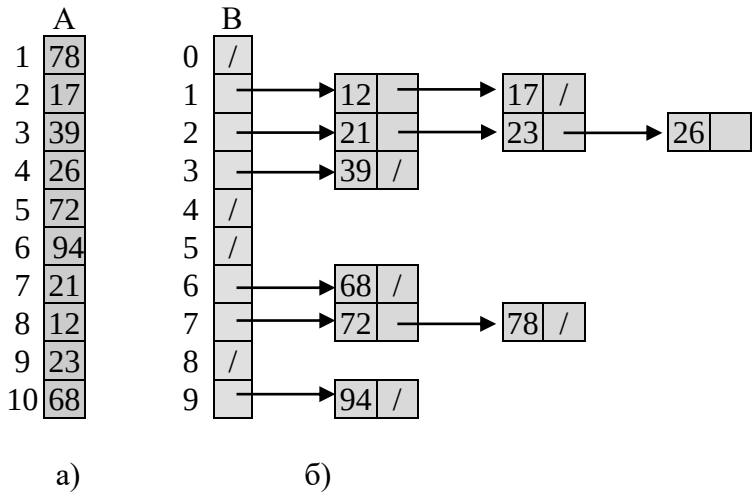

Рисунок 3 – Робота алгоритму Bucket-Sort
а – на вхід подано масив *A[1.. 10]*;
б – масив списків *B[0.. 9]* після виконання рядка 5.
Список з індексом *i* містить числа, у яких перший знак після коми є *i*.
Відсортований масив буде тоді, якщо послідовно виписати списки *B[0],..., B[9]*

Зауважимо, що цей алгоритм – детермінований (не використовує генератора випадкових чисел); поняття випадковості виникає лише під час аналізу часу його роботи.

Ідея алгоритму полягає в тому, що проміжок [0,1] поділяється на *n* рівних частин, після чого для чисел з кожної частини виділяється свій ящик-черпак (bucket), і *n* чисел, що підлягають сортуванню, розкладаються по цих ящиках. Оскільки числа рівномірно розподілені на проміжку [0,1], варто очікувати, що в кожному ящику їх буде небагато. Тепер відсортуємо числа в кожному ящику окремо й пройдемося по ящиках у порядку зростання, виписуючи числа, що потрапили в кожний з них, також у порядку зростання.

Будемо вважати, що на вхід подається *n*-елементний масив *A*, причому *0≤A[i]<1* для всіх *i*. Використовується також допоміжний масив *B[0.. n-1]*, що складається зі списків, що відповідають ящикам. Алгоритм використовує операції зі списками.

```
Bucket-Sort(A)
1    n ← length[A]
2    for i ← l to n
3         do додати A[i] до списку B[nA[i]]
4    for i ← 0 to n - 1
5         do відсортувати список B[i] (сортування вставками)
6    з'єднати списки B[0], B[1],..., B[n - 1] (в зазначеному порядку)
```

На рис. 3 наведено роботу цього алгоритму на прикладі масиву з 10 чисел.

Щоб довести, що алгоритм сортування вичерпуванням правильний, розглянемо два числа *A[i]* та *A[j]*. Якщо вони потрапили в різні ящики, то менше з них потрапило в ящик з меншим номером, і у вихідній послідовності воно виявиться раніше; якщо вони потрапили в один ящик, то після сортування вмісту ящика менше число буде також передувати більшому.

Проаналізуємо час роботи алгоритму. Операції у всіх рядках, крім п'ятого, потребують загального часу *O(n)*. Перегляд всіх ящиків також забирає час *O(n)*. Таким чином, нам залишається тільки оцінити час сортування вставками всередині ящиків.

Нехай у ящик *B[i]* потрапило n_i чисел (n_i – випадкова величина). Оскільки сортування вставками працює за квадратичний час, математичне очікування часу сортування чисел у ящику номер *i* має значення $O(M[n_i^2])$, а математичне очікування сумарного часу сортування у всіх ящиках має значення

$$\sum_{i=0}^{n-1} O(M[n_i^{\,2}]) = O\left(\sum_{i=1}^{n-1} M[n_i^{\,2}]\right). \qquad (1)$$

Знайдемо функцію розподілу випадкових величин n_i. Оскільки числа розподілені рівномірно, а довжини всіх відрізків рівні, ймовірність того, що дане число потрапить у ящик номер *i*, дорівнює *1/n*. Отже, ми перебуваємо в відомій ситуації з кулями й урнами: у нас *n* куль-чисел, *n* урн-ящиків, і ймовірність влучення даної кулі в дану урну дорівнює *p = 1/n*. Тому числа n_i розподілені біноміально: ймовірність того, що $n_i=k$, дорівнює $C_n^k p^k (1-p)^{n-k}$, математичне очікування дорівнює $M[n_i]=np=1$, і дисперсія дорівнює $D[n_i]=np(1-p)=1-1/n$. Отже, можемо записати:

$$M[n_i^2] = M[n_i] + M^2[n_i] = 2 - \frac{1}{n} = \Theta(1).$$

Підставляючи цю оцінку в (1), одержуємо, що математичне очікування сумарного часу сортування вмісту всіх ящиків має значення *O(n)*, так що математичне очікування часу роботи алгоритму сортування вичерпуванням справді лінійно залежить від кількості чисел.

Контрольні питання

1. Поясніть чому задача сортування елементів є однією з найцікавіших та показових задач для курсу теорії алгоритмів.
2. Поясніть, що таке стійкість алгоритму сортування.
3. За якими критеріями на Ваш погляд можна класифікувати алгоритми сортування?
4. Наведіть класифікацію алгоритмів сортування.
5. Перерахуйте та порівняйте відомі Вам алгоритми сортування за лінійний час.
6. Поясніть чому при оцінці складності алгоритму нас частіше за все цікавить робота у найгіршому випадку.
7. Охарактеризуйте особливості алгоритмів сортування, що працюють за лінійний час.
8. Наведіть та поясніть основні властивості алгоритму сортування підраховуванням.
9. Сфери застосування алгоритму цифрового сортування.
10. Наведіть власний приклад алгоритму цифрового сортування.
11. Який час роботи алгоритму сортування вичерпуванням у найгіршому випадку? Наведіть його просту модифікацію, що зберігає лінійний середній час роботи та зменшує час роботи в найгіршому випадку до O(n log n).

Звіт повинен містити

1. Тему, мету та порядок виконання роботи.
2. Ідея відповідного алгоритму сортування.
3. Власний приклад роботи відповідного алгоритму сортування на масиві з 10 чисел.
4. Алгоритм сортування у графічному вигляді.
5. Теоретична оцінка складності алгоритму.
6. Сирець алгоритму відповідного сортування.
7. Практична оцінка складності відповідного алгоритму в трьох випадках для масивів різного розміру. Навести таблиці та відповідні графіки часу виконання програми для трьох випадків. Кількість зна-

чень часу має бути не менше семи.
Увага! Мінімальне та максимальне значення часу ***кожним студентом підбирається індивідуально****, залежно від потужностей Вашого ПК. При цьому врахуйте, що мінімальне значення часу повинно бути порядка кількох секунд (до 10), а максимальне – не більше 5 – 10 хвилин (за Вашим бажанням останній час може бути збільшено)*.

8. Порівняльний аналіз теоретично та практично отриманих графіків залежностей часів виконання програми від розмірів входу.
9. Переваги та недоліки дослідженого Вами алгоритму сортування та Ваші міркування.
10. Розширені висновки з роботи.