Завдання до комп'ютерного практикуму № 3

з дисципліни «Алгоритми і структури даних»

Тема: «Методи сортування масивів»

Мета роботи

Порівняти теоретичну оцінку для кількості операцій та експериментально пораховану кількість операцій для сортування масивів різними методами.

Основні теоретичні відомості

Задача сортування

Формальне визначення:

Вхід. Послідовність із n чисел $\langle a_1, a_2, \ldots, a_n \rangle$.

Вихід. Перестановка (перевпорядкування) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ вхідної послідовності, така що $a'_1 \le a'_2 \le \ldots \le a'_n$.

Наприклад, якщо на вхід подається послідовність $\langle 6,7,4,9,2,3,0,1,5,8 \rangle$, то вихід алгоритму сортування має бути таким $\langle 0,1,2,3,4,5,6,7,8,9 \rangle$. Подібна вхідна послідовність $\langle 6,7,4,9,2,3,0,1,5,8 \rangle$ називається екземпляром задачі сортування.

Числа, які необхідно відсортувати, називають ключами. Концептуально ми сортуємо послідовність, хоча вхідні дані ми отримуємо у вигляді масиву з n елементами.

1. Сортування за допомогою вставки

Принцип сортування: масив розподіляється на відсортовану та невідсортовану частини. На першому кроці за відсортовану частину (послідовність) приймається перший елемент масиву. Кожний наступний елемент з невідсортованої частини вставляємо в заздалегідь відсортовану послідовність так, щоб ця послідовність залишалась відсортованою.

Основні етапи:

1. Знайти місце, куди потрібно вставити черговий елемент.

2. Зсунути елементи, що стоять справа у відсортованій частині, на одну позицію вправо.
3. На звільнене місце поставити елемент, який аналізують (вставляють).

Інваріант циклу: на початку кожної ітерації циклу елементи *Array*[0]… *Array*[*j* – 1] є елементами відсортованої частини, зазначимо, що елементи *Array*[0]... *Array*[*j* – 1] від самого початку знаходились в позиціях від 0 до *j* – 1 (але не були впорядкованими), однак зараз вони відсортовані, де *j* – індекс, який розділяє масив на відсортовану та невідсортовану частини.

Час роботи: O(n^2).

2. Сортування за допомогою прямого вибору

Принцип сортування: масив також поділити на відсортовану та невідсортовану частини. На першому кроці весь масив вважати невідсортованим. У невідсортованій частині знайти мінімальний (або максимальний) елемент та поміняти місцями з першим елементом (з нульовим індексом) невідсортованої частини. Тепер цей елемент вважати першим елементом відсортованої частини. Таку саму процедуру повторити з новою не відсортованою частиною (n–1) разів. Після кожного разу межа, що розділяє відсортовану і не відсортовану частини, буде зсунута на одну позицію праворуч.

Інваріант зовнішнього циклу:

На початку кожної ітерації циклу підмасив *Array*[0]…*Array*[*j* – 1] містить *j* найменших елементів масиву у відсортованому порядку, де *j* – індекс, який розділяє масив на відсортовану та невідсортовану частини.

Інваріант внутрішнього циклу(пошук мінімального елементу):

На початку кожної ітерації циклу елемент *Array*[*min*] є найменшим елементом у підмасиві *Array*[*j*]…*Array*[*i* – 1].

Час роботи: O(n^2).

3. Сортування за допомогою прямого обміну (бульбашкове)

Принцип сортування: зліва направо почергово іде порівняння двох сусідніх елементів. Якщо вони не впорядковані між собою, то їх міняють місцями. В базовому алгоритмі проходження масиву та чергове впорядкування повторюють n–1 разів.

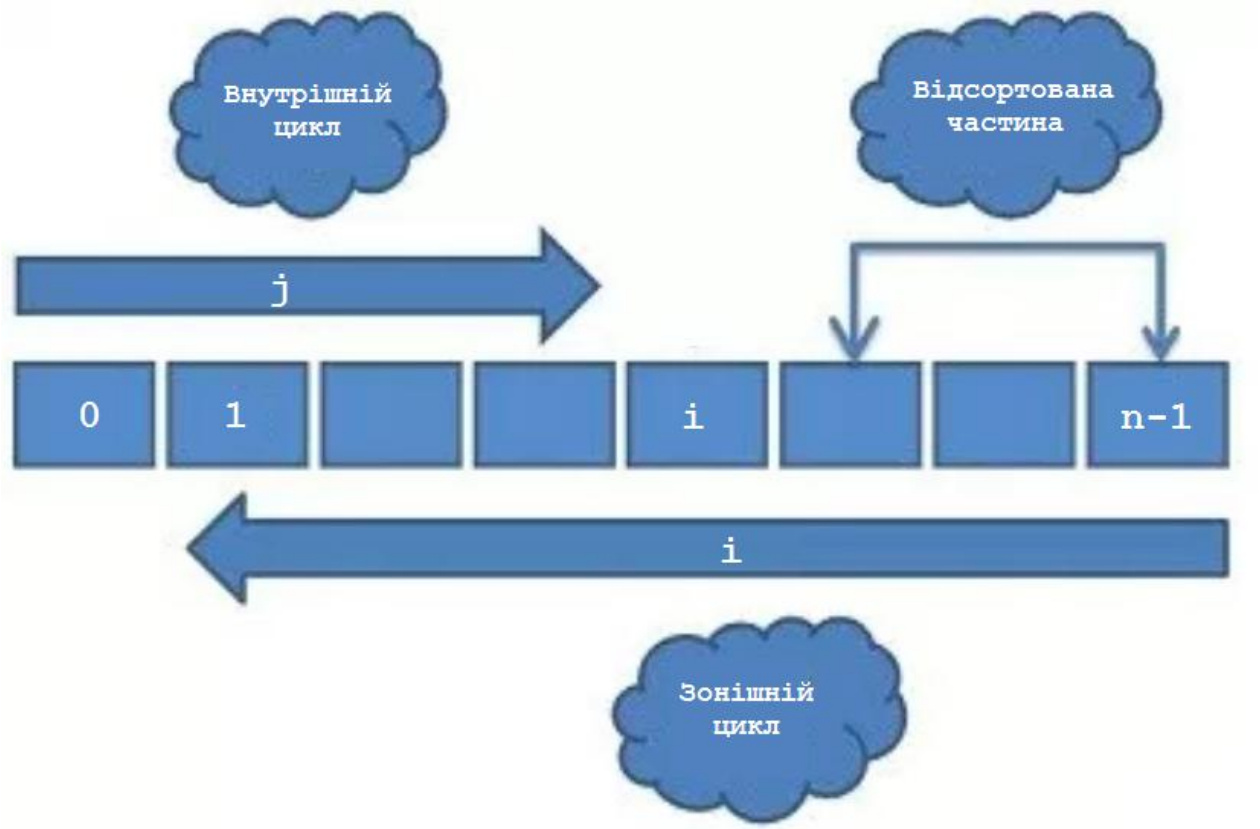

Рис. 3.1. Схематичне зображення інваріантів циклу сортування за допомогою прямого обміну (бульбашкове)

Інваріант зовнішнього циклу:

На початку кожної ітерації циклу підмасив *Array*[i + 1]…*Array*[n – 1] містить (n – i – 1) найбільших елементів масиву у відсортованому порядку, де i – індекс, який розділяє масив на відсортовану та невідсортовану частини.

Інваріант внутрішнього циклу:

На початку кожної ітерації циклу елемент *Array*[j] є найбільшим елементом у підмасиві *Array*[0]… *Array*[j].

Час роботи: O(n^2).

4. Сортування злиттям

Принцип сортування: рекурсія досягає нижньої межі, коли розмірність послідовності, яку необхідно відсортувати дорівнює 1. Ключова операція – об'єднання двох відсортованих послідовностей – злиття.

Три етапи:

1. Розділення послідовності з n елементів на дві послідовності розмірності $n/2$.
2. Володарювання: рекурсивно сортуємо ці дві послідовності з використанням сортування злиттям.
3. Комбінування: об'єднуємо дві відсортовані послідовності в одну відсортовану.

Принцип злиття: у двох відсортованих послідовностей відслідковуємо найменші елементи; у циклі в вихідну послідовність переносимо той елемент, який виявися меншим із найменших елементів відсортованих послідовностей. Цей крок повторюється поки в одній з послідовностей не закінчаться елементи, після чого елементи з послідовності, що залишилась поміщаються в вихідну послідовність.

Інваріант циклу для злиття: На початку кожної ітерації циклу підмасив *Array*[*left*]…*Array*[*k*–1] містить *k*–*left* найменших елементів відсортованих масивів *L* та *R* у відсортованому порядку. *L*[*i*] та *R*[*j*] є найменшими елементами цих масивів, які ще не були скопійованими назад у *Array*, де:

L – копія відсортованого підмасиву *Array*[*left*]…*Array*[*m*];

R – копія відсортованого підмасиву *Array*[*m*+1]…*Array*[*right*];

m = (*left* + *right*)/2;

left – лівий індекс вхідного масиву *Array*;

right – правий індекс вхідного масиву *Array*;

Array – вхідний масив з елементами: *Array*[*left*]…*Array*[*right*].

i, *j* – індекси масивів *L*, *R*;

k – індекс *Array*, куди копіюється слідуючий найменший елемент.

Час роботи: $O(n \log_2(n))$.

5. Швидке сортування

Принцип сортування: рекурсія досягає нижньої межі, коли розмірність послідовності, яку необхідно відсортувати дорівнює 1. Ключова операція – розділення на підмасиви.

Три етапи:

1. Розділення: масив *Array*[*left*]…*Array*[*right*] розділяється на два (можливо порожніх) підмасива *Array*[*left*]…*Array*[*q*–1] та *Array*[*q*+1]…*Array*[*right*], таких, що кожний елемент *Array*[*left*]…*Array*[*q*–1] менше або дорівнює елементу *Array*[*q*], який не більше елементів підмасиву *Array*[*q*+1]…*Array*[*right*].
2. Володарювання: підмасиви *Array*[*left*]…*Array*[*q*–1] та *Array*[*q*+1]…Array[*right*] сортуємо за допомогою рекурсивного виклику функції швидкого сортування.
3. Комбінування: весь підмасив *Array*[*left*]…*Array*[*right*] у результаті роботи алгоритму виявляється відсортованим, тому об’єднання не потребує.

Принцип розділення: функція розділення шукає індекс опорного елементу q та впорядковує елементи підмасивів без залучення додаткової пам’яті, рис. 5.1.

Вхідні дані: *Array* – масив; *left*, *right* – лівий, правий індекси підмасиву *Array*, який необхідно розділити.

Інваріант циклу для розділення:

На початку кожної ітерації для будь-якого індексу k справедливим є:

1) якщо $left \le k \le i$, тоді $Array[k] \le x$;
2) якщо $i + 1 \le k \le j - 1$, тоді $Array[k] > x$;
3) якщо $k = right$, тоді $Array[k] = x$.

Індекси між j та *right* не потрапляють в жоден випадків, тому значення відповідних елементів не мають визначеного зв’язку з опорним елементом x.

Час роботи: $O(n \log_2(n))$.

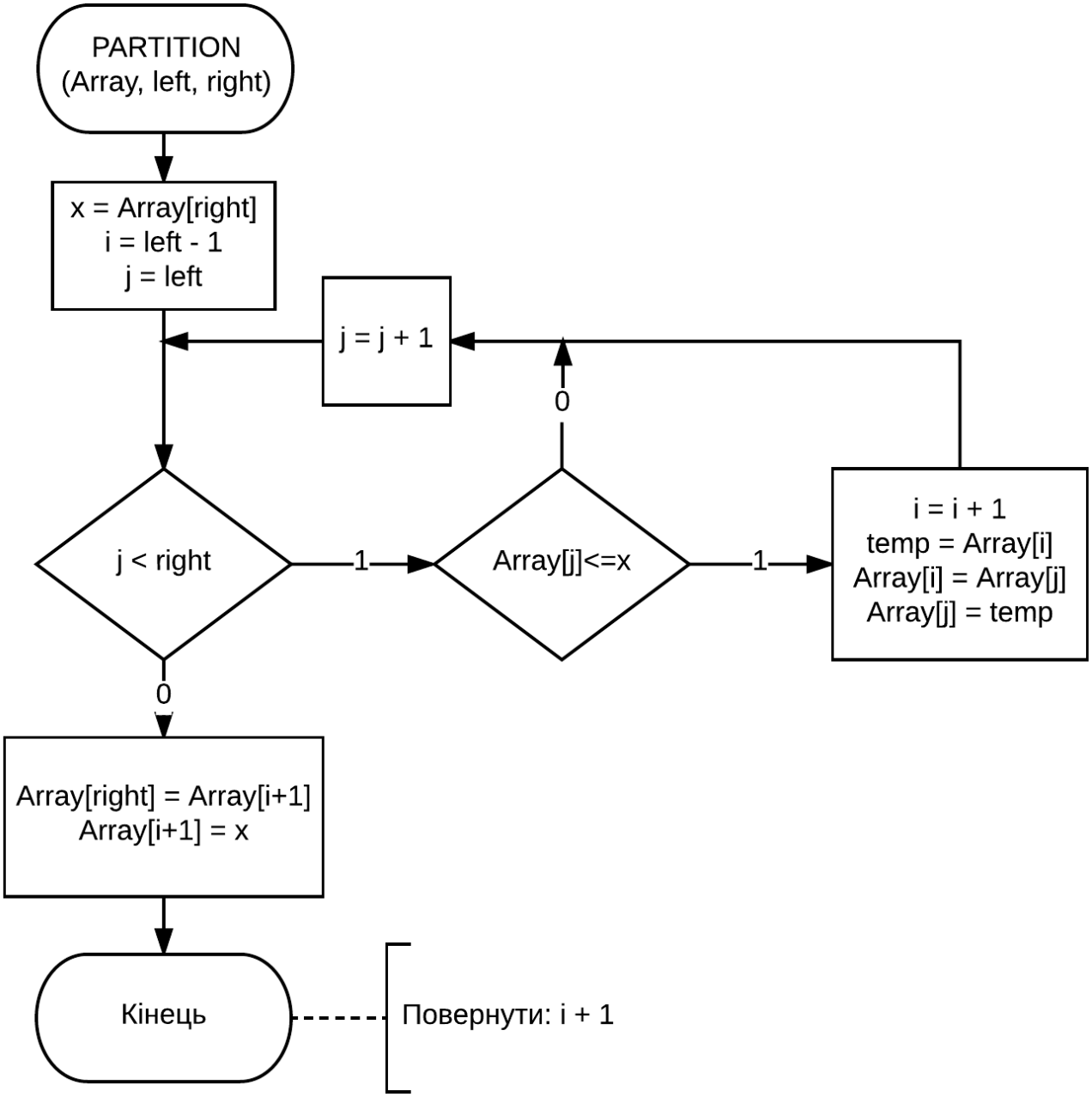

Рис. 5.1. Блок-схема функції розділення швидкого сортування

6. Пірамідальне сортування

Пірамідальне сортування використовує спеціалізовану структури даних – бінарну піраміду.

6.1. Бінарна піраміда

Структура даних бінарна піраміда є масивом, який можна розглядати, як бінарне дерево рис. 6.1, кожний вузол цього дерева відповідає елементу масиву. Дерево повністю заповнене на усіх рівнях, за винятком, можливо, найнижчого, який заповнюється зліва направо.

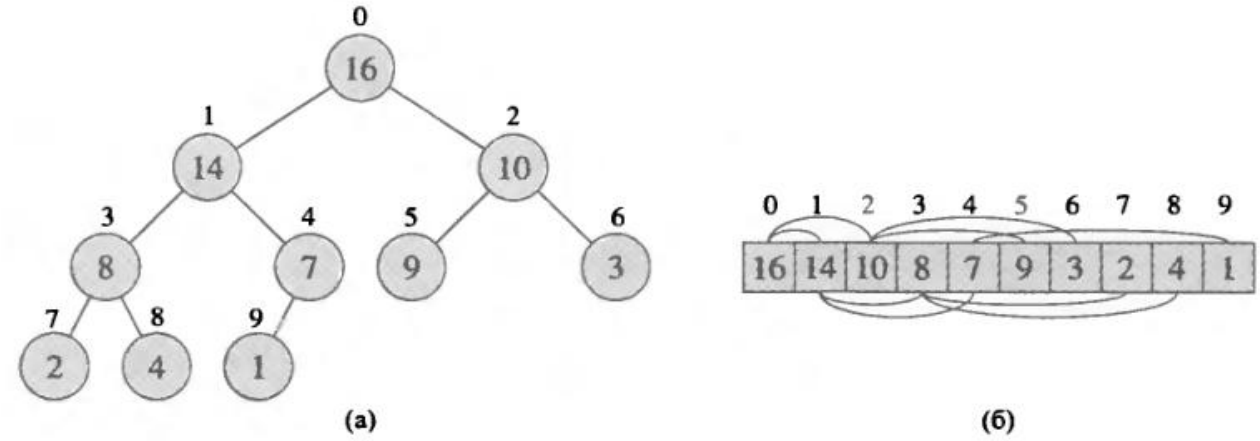

Рис. 6.1. Незростаюча піраміда у вигляді
(а) бінарного дерева та (б) масиву

Коренем дерева є *Array*[0], а для заданого індексу і вузла можна обчислити індекси:

– батьківського: індекс дорівнює (*i* – 1)/2;
– дочірнього лівого вузла: індекс дорівнює 2*(*i* + 1) – 1;
– дочірнього правого вузла: індекс дорівнює 2*(*i* + 1).

Операцію ділення на 2 можна виконати за допомогою оператора бітового зсуву праворуч (зсув на один біт), а операцію множення на 2 – за допомогою оператора бітового зсуву ліворуч (зсув на один біт).

6.2. Властивість незростаючої піраміди.

Для кожного не кореневого вузла і виконується: *Array*[(*i* – *1)*/2] ≥ *Array*[*i*], тобто значення батьківського вузла більше за значення дочірніх.

6.3. Алгоритм сортування:

Принцип сортування: алгоритм починається з побудови незростаючої піраміди з вхідного масиву *Array*[0].. *Array*[*n* – 1]. Оскільки, найбільший елемент масиву знаходиться у корені (*Array*[0]), його можна поміняти місцями з останнім елементом масиву (*Array*[*n* – 1]). Після чого зсунути межу відсортованості масиву на 1 ліворуч. У отриманої піраміди новий корінь порушує властивість незростаючої піраміди – виконаємо відновлення цієї властивості шляхом виклику функції MAX-HEAPIFY.

Принцип роботи функції MAX-HEAPIFY:

Вхідні дані: *Array* – масив, *i* – індекс, *n* – розмірність масиву.

Блок-схема роботи представлена на рис. 6.2:

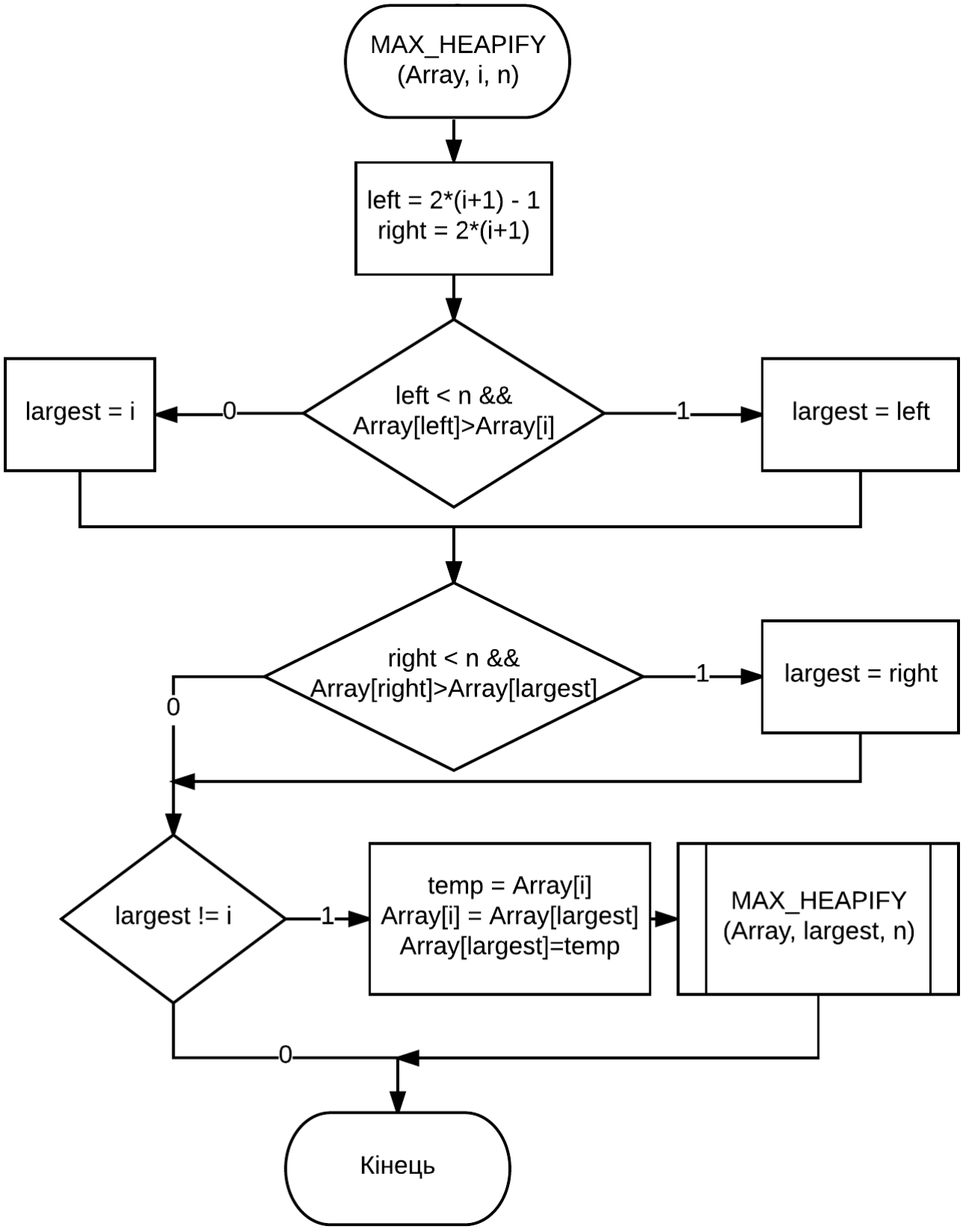

Рис. 6.2. Блок-схема функції MAX-HEAPIFY

При виклику функції MAX-HEAPIFY, вважаємо, що бінарні дерева з коренями в дочірніх елементах, з індексами: 2*(*i*+1) – 1 та 2*(*i*+1)), є незростаючими пірамідами, але елемент *Array*[*i*] може бути меншим, ніж значення дочірніх вузлів (порушуючи властивість незростаючої піраміди). Функція MAX-HEAPIFY «сплавляє» значення *Array*[*i*] вниз по незростаючій піраміді так, що піддерево з кореневим елементом з індексом і задовольняє властивості незростаючої піраміди.

Час роботи функції MAX-HEAPIFY: $O(\log_2(n))$

6.4. Побудова незростаючої піраміди.

Для побудови незростаючої піраміди скористаємось функцією BUILD-MAX-HEAP, рис.6.3.

Вхідні дані: *Array* – масив, *n* – розмірність масиву.

Функцію MAX-HEAPIFY можна використати у зворотному напрямку: піднімаючись від листів дерева до кореня. Елементи *Array*[*n*/2]…*Array*[*n* – 1] є листами дерева, тому кожний з них можна вважати одноелементною пірамідою.

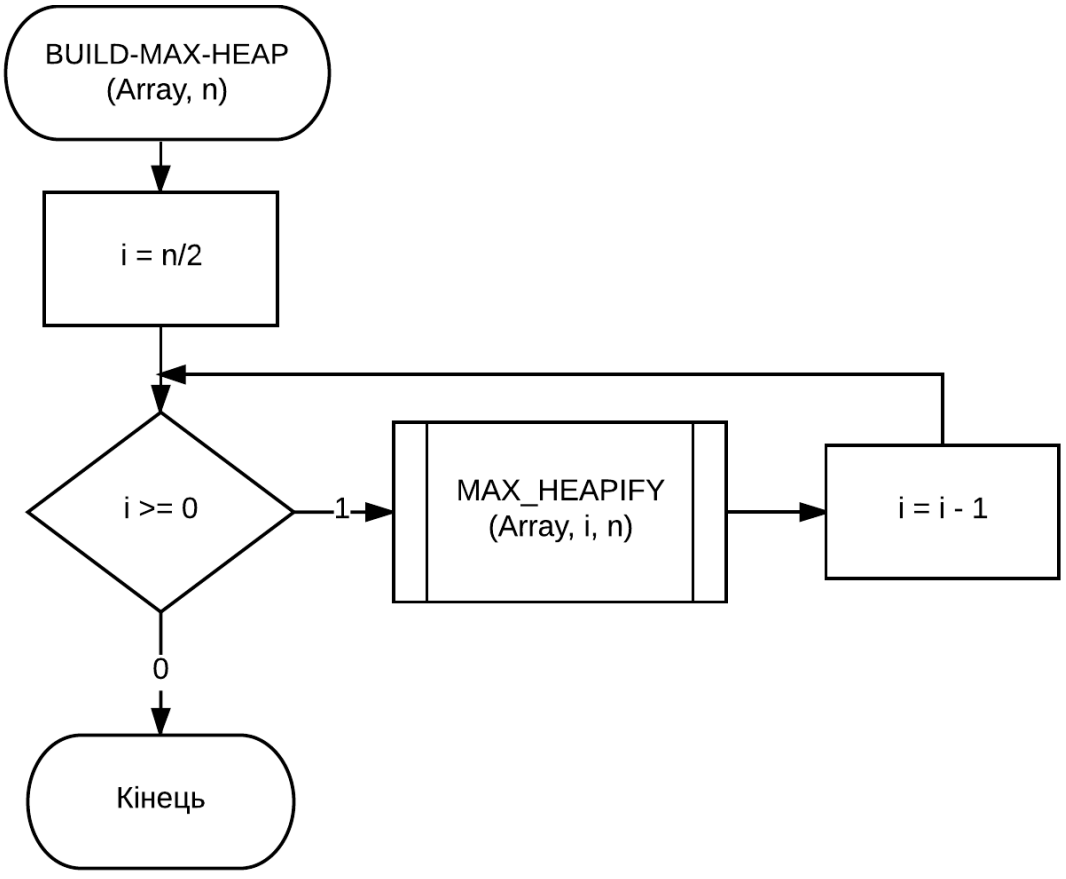

Рис. 6.3. Блок-схема функції BUILD-MAX-HEAP

Інваріант циклу функції BUILD-MAX-HEAP:

На початку кожної ітерації, кожний вузол $i + 1$, $i + 2$, …, $n - 1$ є коренем незростаючої піраміди.

Час роботи функції BUILD-MAX-HEAP: $O(n \log_2(n))$

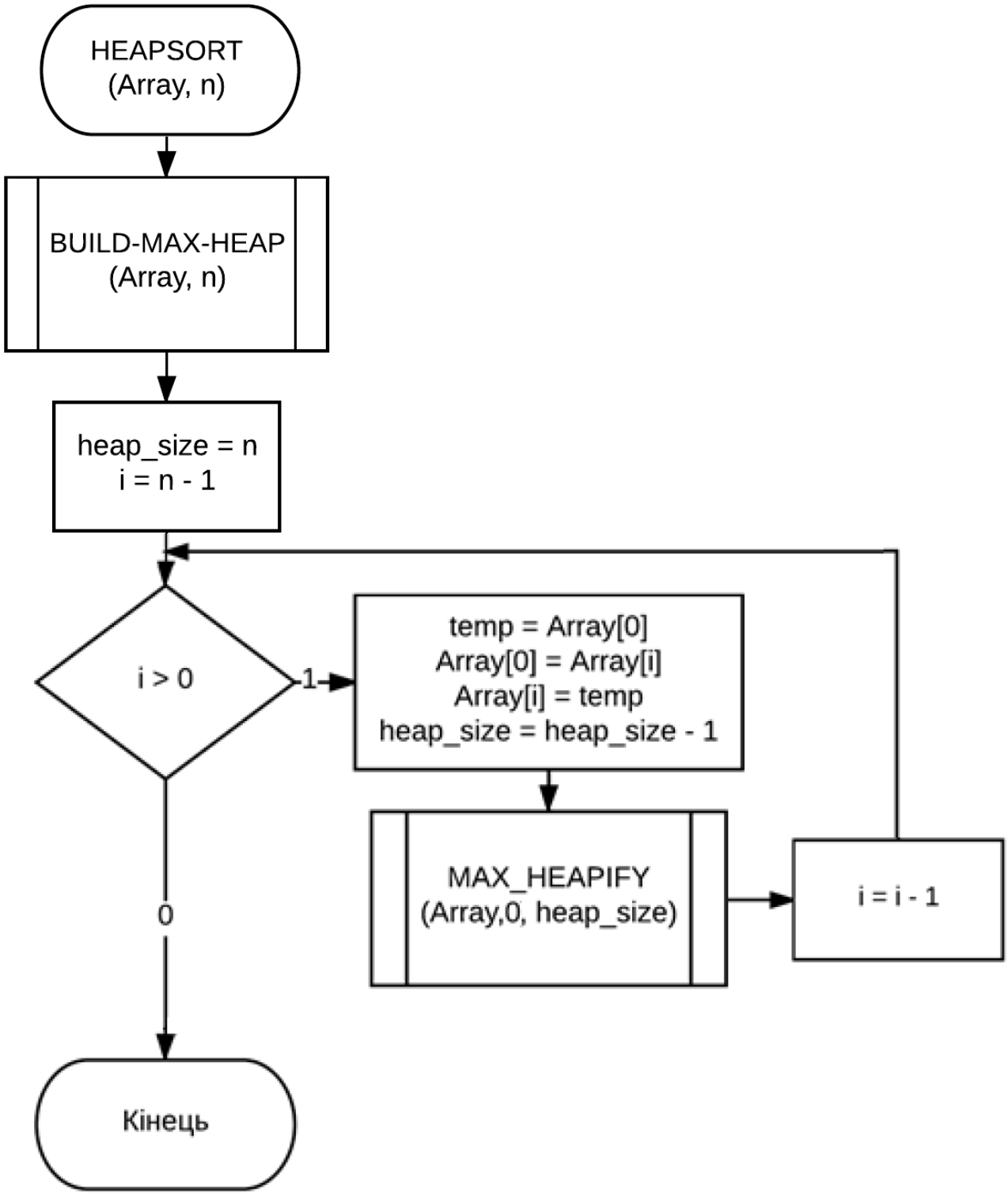

Рис. 6.4. Блок-схема функції HEAPSORT

Час роботи алгоритму пірамідального сортування: $O(n \log_2(n))$.

7. Сортування підрахунком

Вважаємо, що кожний із n вхідних елементів масиву *Array* є цілим числом з інтервалу від 0 до $k - 1$, де k – деяка ціла константа.

Принцип сортування: для кожного елемента масиву *Array*[j] визначаємо кількість елементів, що менші за нього. За допомогою цієї інформації визначаємо індекс в якому він має знаходитись.

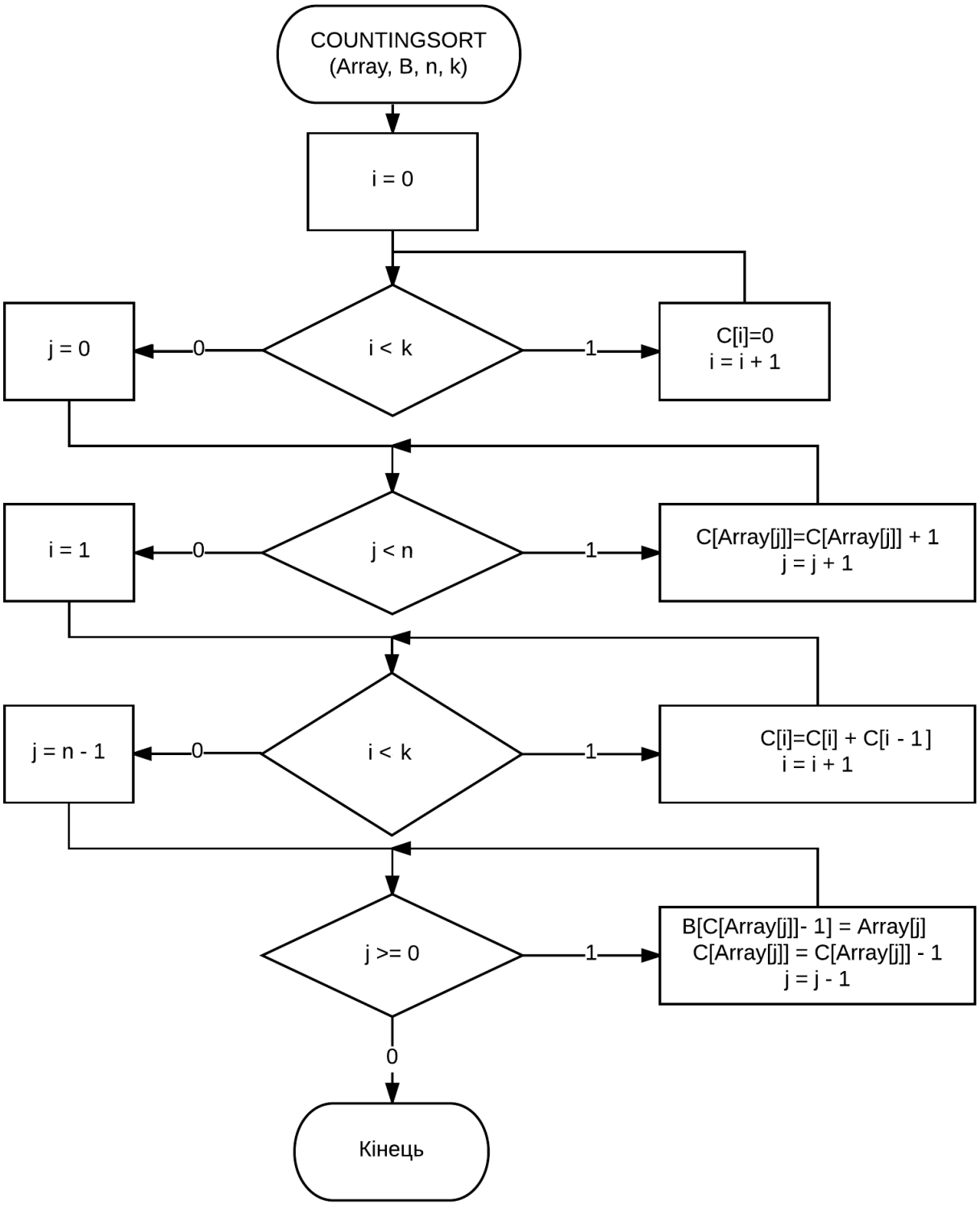

Рис. 7.1. Блок-схема алгоритму сортування підрахунком

Цей алгоритм потребує додаткового використання пам’яті:

$B[0]\ldots B[n-1]$ – містить відсортовані вхідні дані;

$C[0]\ldots C[k-1]$ – тимчасовий робочій простір.

Ініціалізуємо C нульовими значеннями, якщо значення $Array[j] = i$ – збільшуємо $C[i]$ на одиницю. Після проходу по масиву *Array*, у елементах масиву C ($C[i]$) міститься кількість елементів, що дорівнює i.

Для кожного $i = 0, 1, \ldots, k$ шляхом додавання елементів масиву C визначаємо скільки вхідних елементів не перевищують i.

Копіювання елементів реалізовуємо проходом від останнього елементу ($Array[n-1]$) до першого ($Array[0]$). Оскільки різні елементи можуть мати однакові значення, копіюючи значення $Array[j]$ в масив B, зменшуємо значення $C[Array[j]]$ на одиницю. Наступний елемент, значення якого є $Array[j]$ у масиві B розміщуємо безпосередньо перед $Array[j]$.

Стійкість алгоритму підрахунком: елементи з однаковим значенням знаходяться у вихідному масиві в тому ж порядку, що й у вхідному.

Час роботи алгоритму сортування підрахунком: $O(n)$.

8. Сортування за розрядами

Цей алгоритм використовували в машинах, призначених для сортування перфокарт. Перфокарти мали 80 стовпців, в кожному з яких можна було зробити 12 отворів. Сортувальник перевіряв заданий стовпчик кожної перфокарти, що знаходилась у колоді, і розподіляв перфокарти по 12 приймачам залежно від того, в якій позиції розташований отвір.

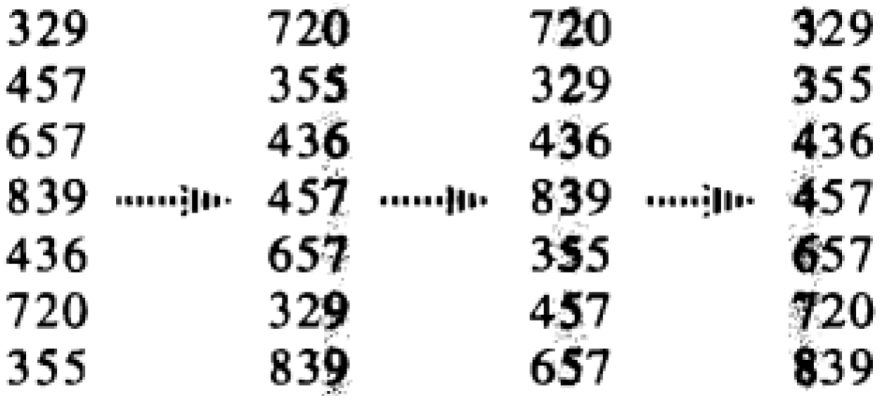

Рис. 8.1. Приклад сортування за розрядами

Принцип сортування: спочатку проводиться стійке сортування за молодшою цифрою, після чого перфокарти об’єднуються в одну колоду, в якій спочатку йдуть перфокарти з нульового приймача, потім – з першого, потім – з другого тощо. Після чого проводиться сортування за передостанньою цифрою, перфокарти об’єднуються в одну колоду за тим же правилом. Процес продовжується доти, поки перфокарти не будуть відсортованими за усіма розрядами.

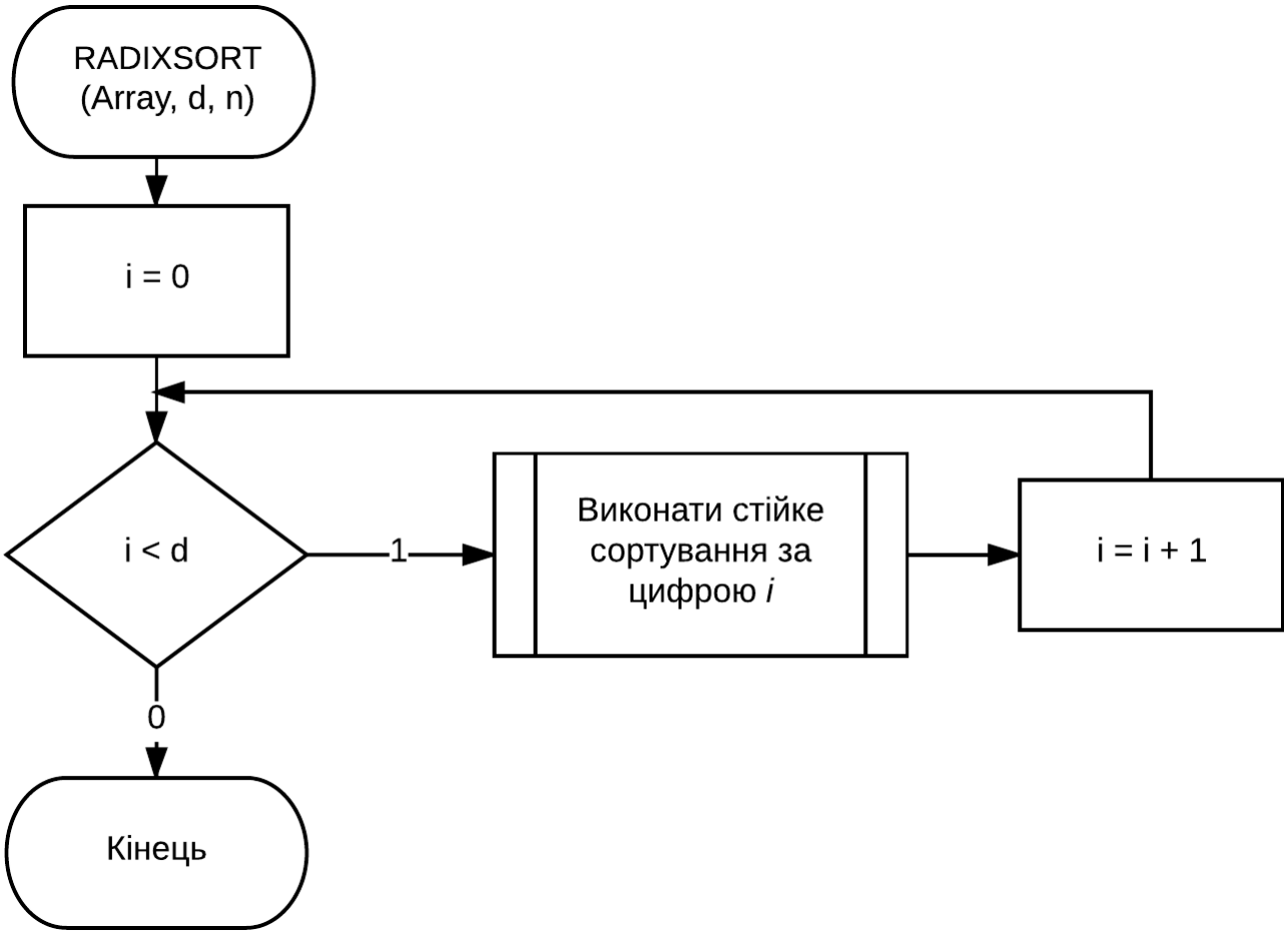

Рис. 8.2. Блок-схема алгоритму сортування за розрядами

Кількість проходів дорівнює кількості розрядів найбільшого числа.

Лема 8.1. Нехай є n d-значних чисел, в яких кожна цифра приймає одне з k можливих значень. Тоді алгоритм сортування за розрядами дозволяє виконати коректне сортування цих чисел за час $O(d*(n+k))$, якщо стійке сортування, що використане в алгоритмі має час роботи $O(n+k)$.

Доведення. Коректність сортування за розрядами доводиться методом математичної індукції за стовпцями, що сортуються. Аналіз часу роботи алгоритму залежить від того, який із методів стійкого сортування

використовувати в якості алгоритму сортування. Якщо кожна цифра належить інтервалу від 0 до *k-1* і *k* не надто велике, то оптимальним вибором є сортування підрахунком. Для обробки кожної із *d* цифр *n* чисел знадобиться час O(*k*+*n*), усі цифри будуть опрацьовані за час O(*d*(n+k)*).

Лема 8.2. Нехай є *n* *b*-бітових чисел і деяке натуральне число $r \le b$. Тоді алгоритм сортування за розрядами дозволяє виконати коректне сортування цих чисел за час O(*(b/r)*(n+*2^r*)*), якщо стійке сортування, що використане в алгоритмі має час роботи O(*n*+*k*), для вхідних даних з діапазону від 0 до $k - 1$.

Доведення. Для значення $r \le b$ кожний ключ можна розглядати, як число, що складається з $d = b/r$ по *r* біт. Усі цифри є цілими числами з інтервалу від 0 до $2^r - 1$, тому можемо скористатись алгоритмом сортування підрахунком, в якому $k = 2^r - 1$ (наприклад 32-бітове слово можна розглядати як число, що складається з чотирьох 8-бітових цифр: $b = 32$, $r = 8$; $k = 2^8 - 1 = 255$; $d = b/r = 4$). Кожний прохід сортування підрахунком займає час $O(n+2^r)$, всього проходів *d*, час роботи алгоритму: $O((b/r)*(n+2^r))$.

9. Сортування комірками

Принцип сортування: інтервал [0, 1) розбивається на *n* однакових інтервалів, або комірок, після чого алгоритм розподіляє по комірках *n* вхідних чисел. Для отримання вихідної послідовності, необхідно виконати сортування чисел вставкою у кожній комірці.

Передбачається, що на вхід подається масив *Array*, який складається з *n* елементів.

Для розподілення елементів по комірках виконаємо масштабування значень масиву:

$0 \le (Array[i] - \text{MIN\_VALUE}) / (\text{MAX\_VALUE} - \text{MIN\_VALUE} + 1) < 1$,

MAX_VALUE – максимальне значення;

MIN_VALUE – мінімальне значення;

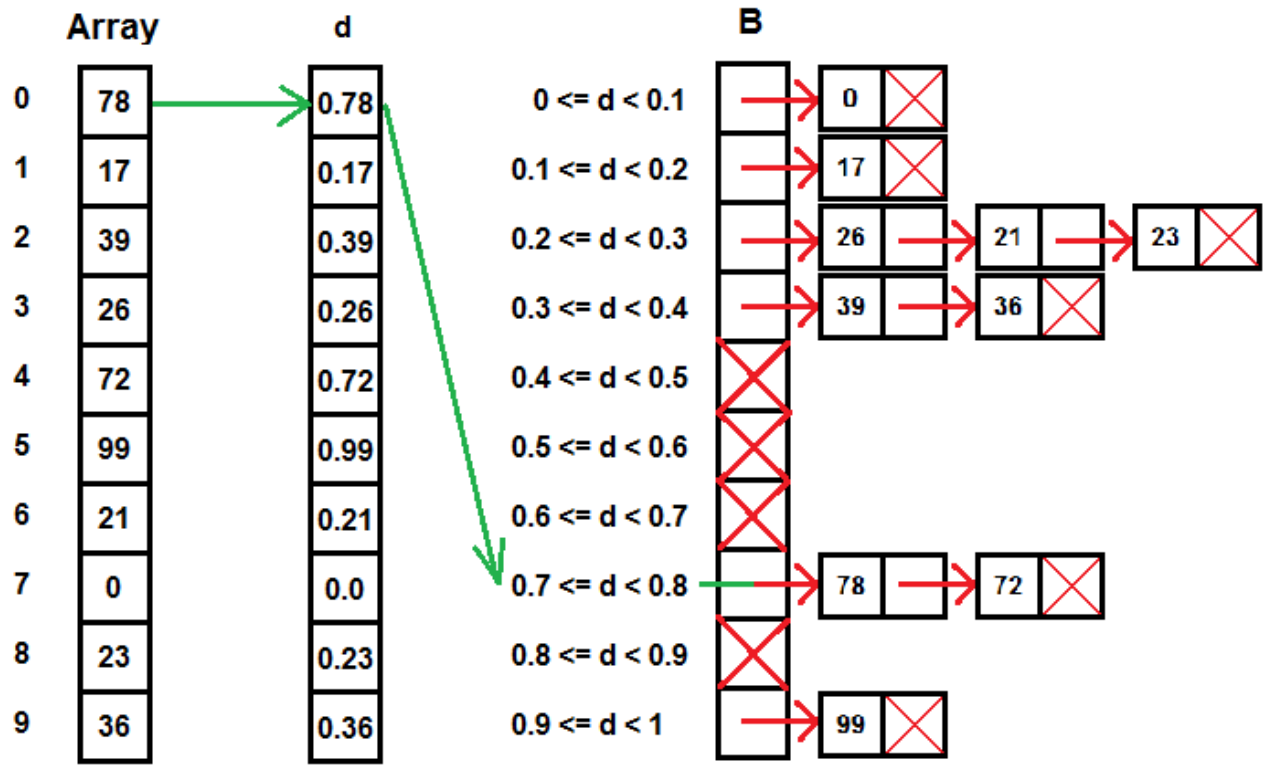

Рис. 9.1. Приклад сортування комірками

Для реалізації алгоритму знадобиться допоміжний масив зв’язних списків.

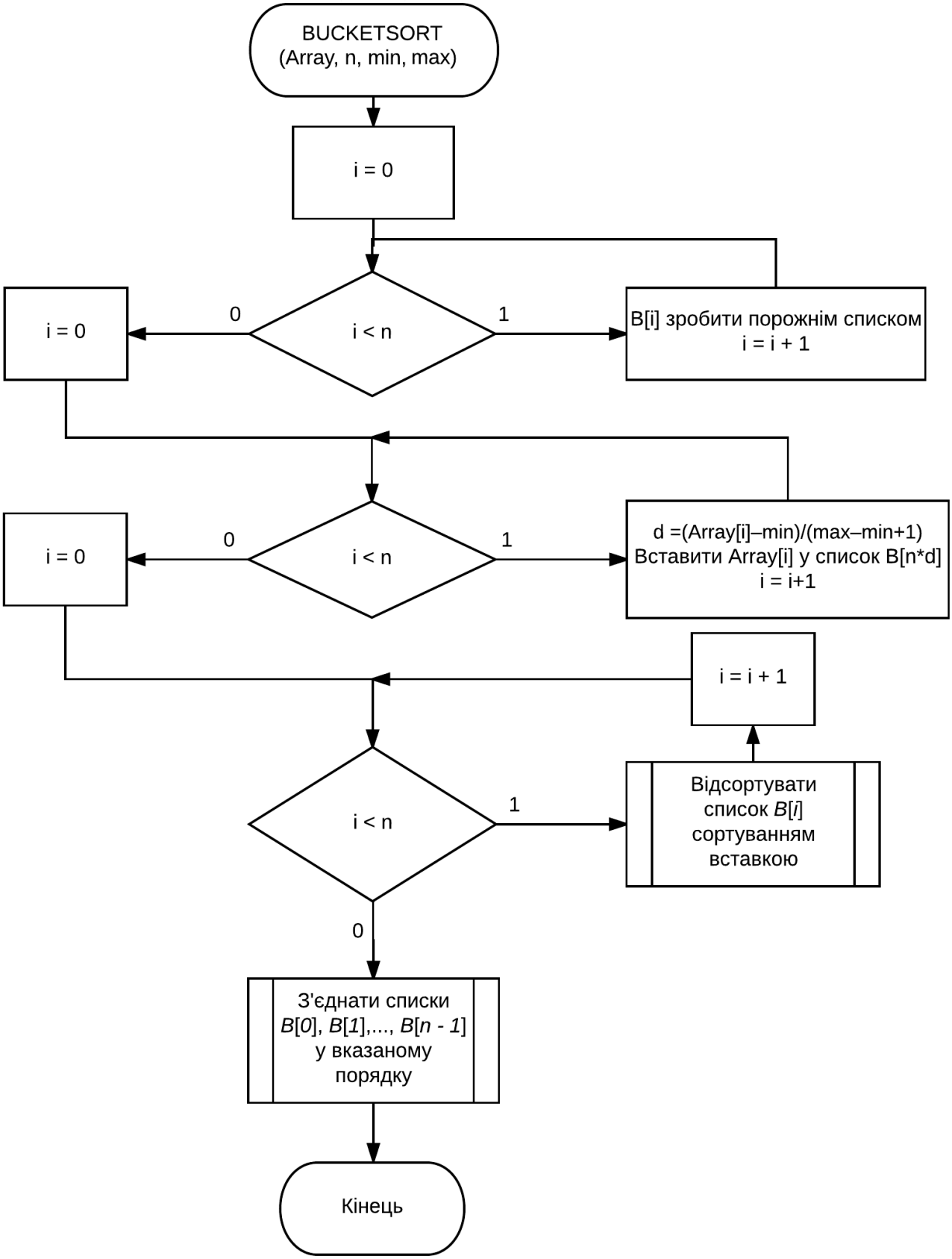

Рис. 9.2. Блок-схема алгоритму сортування комірками

Час роботи алгоритму: O(n).

Порядок виконання роботи

1. Написати програму, що реалізує методи сортування (табл.2).
2. Згенерувати масив розмірності 1000, 10000, 100000 елементів:
    a. відсортований за зростанням;
    b. з випадковими елементами (кількість генерувань = 1000);
    c. відсортований за спаданням.
3. Відсортувати одержані масиви за зростанням, отримавши такі змінні:
    a. кількість порівнянь;
    b. кількість обмінів;
4. Результати оформити в звіт:
    1. Титульний лист.
    2. Варіант завдання.
    3. Завдання.
    4. Лістинг програми.
    5. Результати експерименту у вигляді таблиць (табл.1.).
    6. Висновки.

Табл. 1. Результати порівняння методів сортування

N	Метод					
	Кількість порівнянь			Кількість копіювань		
	Теор.	Експерим.	Відношення	Теор.	Експерим.	Відношення
	Найкращий випадок					
1000						
10000						
100000						
	Середній випадок					
1000						
10000						
100000						
	Найгірший випадок					
1000						
10000						
100000						

Табл. 2. Варіанти завдань

№	Методи		
	I	II	III
1	вставка	злиття	підрахунком
2	вибір	злиття	за розрядами
3	обмін, базовий	злиття	комірками*
4	обмін, із фіксацією наявності пересувань	злиття	підрахунком
5	обмін, із фіксацією останнього місця пересування	злиття	за розрядами
6	обмін, шейкерний	злиття	комірками*
7	вставка	пірамідальне	підрахунком
8	вибір	пірамідальне	за розрядами
9	обмін, базовий	пірамідальне	комірками*
10	обмін, із фіксацією наявності пересувань	пірамідальне	підрахунком
11	обмін, із фіксацією останнього місця пересування	пірамідальне	за розрядами
12	обмін, шейкерний	пірамідальне	комірками*
13	вставка	швидке	підрахунком
14	вибір	швидке	за розрядами
15	обмін, базовий	швидке	комірками*
16	обмін, із фіксацією наявності пересувань	швидке	підрахунком
17	обмін, із фіксацією останнього місця пересування	швидке	за розрядами
18	обмін, шейкерний	швидке	комірками*
19	вставка	злиття	за розрядами
20	вибір	злиття	комірками*
21	обмін, базовий	злиття	підрахунком
22	обмін, із фіксацією наявності пересувань	злиття	за розрядами
23	обмін, із фіксацією останнього місця пересування	злиття	комірками*
24	обмін, шейкерний	злиття	підрахунком
25	вставка	пірамідальне	за розрядами
26	вибір	пірамідальне	комірками*
27	обмін, базовий	пірамідальне	підрахунком
28	обмін, із фіксацією наявності пересувань	пірамідальне	за розрядами

№	Методи		
	I	**II**	**III**
29	обмін, із фіксацією останнього місця пересування	пірамідальне	комірками*
30	обмін, шейкерний	пірамідальне	підрахунком
31	вставка	швидке	за розрядами
32	вибір	швидке	комірками*
33	обмін, базовий	швидке	підрахунком
34	обмін, із фіксацією наявності пересувань	швидке	за розрядами
35	обмін, із фіксацією останнього місця пересування	швидке	комірками*
36	обмін, шейкерний	швидке	підрахунком

* замість сортування комірками можна обрати підрахунком або за розрядами.

Контрольні питання

1. Формальне визначення задачі сортування.
2. Яка ідея реалізована у базовому методі сортування вибором?
3. В чому полягає ідея базового методу сортування обміном?
4. Які ви можете навести варіанти методу сортування обміном та в чому полягають покращення?
5. Поясніть базовий метод сортування вставкою.
6. Який принцип сортування злиттям?
7. Який принцип швидкого сортування?
8. Який принцип пірамідального сортування?
9. Який принцип сортування підрахунком?
10. Який принцип сортування за розрядами?
11. Який принцип сортування комірками?
12. Сформулюйте інваріант циклу для сортування за допомогою вставки.
13. Сформулюйте інваріант циклу для сортування за допомогою вибору.
14. Сформулюйте інваріант циклу для сортування за допомогою обміну.
15. Сформулюйте інваріант циклу для сортування злиттям.
16. *Сформулюйте інваріант циклу для швидкого сортування.
17. Сформулюйте інваріант циклу для пірамідального сортування.
18. Час роботи сортування вибором.
19. Час роботи сортування обміном.
20. Час роботи сортування вставкою.
21. Час роботи сортування злиттям.
22. Час роботи швидкого сортування.
23. Час роботи пірамідального сортування.
24. Час роботи сортування підрахунком.
25. Час роботи сортування за розрядами.
26. Час роботи сортування комірками.