**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Алгоритми та складність**

**Завдання № 4**

**Звіт**

**Виконав:**

студент групи К-29

Грищенко Юрій Анатолійович

**Київ-2019**

**Умова задачі**

Нехай маємо масив, що містить n записів з даними для сортування, і що ключ кожного запису приймає значення 0 або 1. Алгоритм для сортування такого набору записів повинен мати наступні з трьох характеристик:

1. час роботи O(n)
2. сортування має бути стійким
3. сортування проводиться на місці, тобто крім вхідного масиву використовується додаткова память, що не перевищує константи.

Необхідно створити алгоритми, що задовільняють 2 з 3 умов.

**Опис алгоритмів:**

Оскільки в цій роботі ми працюємо зі стійким та нестійким сортуванням, потрібно якимось чином розрізняти записи «0» один від одного і записи «1» один від одного (щоб знати, чи не порушився їх порядок).

В моїй імплементації програма працює з довільними цілими числами, умовно позначаючи числа, більші або рівні нулю, як «1», а відємні числа як «0».

**bool** **is\_zero**(**int** element)

{

**return** element < 0;

}

**bool** **is\_one**(**int** element)

{

**return** element >= 0;

}

На виході повинні мати масив, що на початку містить всі «0», а в кінці всі «1». За рахунок того, що у нас лише дві групи елементів, можна зробити певні оптимізації в порівнянні зі звичайним сортуванням.

**a. Час роботи O(n), сортування на місці**

Пропонується наступний алгоритм:

1. Задаємо last\_zero=-1
2. Задаємо i=0, тобто розглядаємо перший елемент масиву.
3. Входимо в цикл, виконуємо, поки i менше за довжину масива.
4. Якщо знайшли «1», пропускаємо його і переходимо на наступний елемент. Повертаємось до кроку 3.
5. Інакше, якщо знайшли «0», то міняємо його місцями з елементом, що лежить після останнього попереднього нуля (тобто за індексом last\_zero + 1). Зберігаємо нове значення last\_zero = last\_zero + 1 і переходимо на наступний елемент. Повертаємось до кроку 3.

На кожному кроці циклу ми обробляємо один елемент, отже складність O(n). Виділяються лише 2 додаткові комірки памяті. Цей алгоритм не є стійким, оскільки є операції swap(), що змінюють порядок «одиниць».

**b. Стійкий алгоритм, сортування на місці**

Пропонується наступний алгоритм, який нагадує insertion sort:

1. Задаємо last\_zero=-1
2. Задаємо i=0, тобто розглядаємо перший елемент масиву.
3. Входимо в цикл, виконуємо, поки i менше за довжину масива.
4. Якщо знайшли «1», пропускаємо його і переходимо на наступний елемент. Повертаємось до кроку 3.
5. Інакше, якщо знайшли «0», то зберігаємо його у комірку element.
6. Для всіх елементів від (last\_zero + 1) до (i -1) виконуємо зміщення направо, тобто array[j + 1] = array[j]
7. Записуємо element у комірку, що знаходиться правіше від останнього попереднього нуля (тобто array[last\_zero + 1] = element). Повертаємось до кроку 3.

Складність цього алгоритму, як і для bubble sort, становить O(n^2). Оптимізація полягає у тому, що ми не порівнюємо елементи на кожному кроці — ми відразу знаємо, куди саме вписати наші елементи.

**с. Час роботи O(n), стійкий алгоритм**

В цьому алгоритмі будемо використовувати додаткову память.

Пропонується такий варіант bucket sort:

1. Виділяємо два масиви довжиною n — масив одиниць (ones) і масив нулів (zeroes).
2. Виділяємо дві змінні — ones\_count = 0 та zeroes\_count = 0
3. Для кожного елементу масиву (i = 1, … n — 1):
4. Якщо елемент є «0», записуємо його у zeroes[zeroes\_count]. Збільшуємо zeroes\_count на 1, розглядаємо наступний елемент.
5. В іншому випадку (якщо елемент є «1») записуємо його у ones[ones\_count]. Збільшуємо ones\_count на 1, розглядаємо наступний елемент.
6. Коли розглянемо всі елементи масиву, скопіюємо в нього елементи з zeroes (від 0 до zeroes\_count), а потім елементи з ones (від 0 до ones\_count)
7. Звільняємо память, використану для ones та zeroes.

Цей алгоритм зберігає порядок «нулів» та «одиниць» і працює за час O(n), але при цьому використовує O(n) додаткової памяті.

Оптимізація в порівнянні зі звичайним bucket sort полягає у тому, що нам не потрібно сортувати елементи всередині, вони вважаються рівними між собою, а отже, вони вже відсортовані.

**Інтерфейс користувача**

Користувач вводить розмір вхідного масиву і всі числа. Програма виводить результати трьох алгоритмів сортування у консоль.

Enter array size: 10

Enter 10 numbers: -2

{ -2 }

-3

{ -2, -3 }

-5

{ -2, -3, -5 }

0

{ -2, -3, -5, 0 }

1

{ -2, -3, -5, 0, 1 }

3

{ -2, -3, -5, 0, 1, 3 }

-4

{ -2, -3, -5, 0, 1, 3, -4 }

2

{ -2, -3, -5, 0, 1, 3, -4, 2 }

-5

{ -2, -3, -5, 0, 1, 3, -4, 2, -5 }

6

{ -2, -3, -5, 0, 1, 3, -4, 2, -5, 6 }

In-place fast sort: { -2, -3, -5, -4, -5, 3, 0, 2, 1, 6 }

Stable fast sort: { -2, -3, -5, -4, -5, 0, 1, 3, 2, 6 }

In-place stable sort: { -2, -3, -5, -4, -5, 0, 1, 3, 2, 6 }

**Список використаних джерел.**

https://en.wikipedia.org/wiki/Bubble\_sort

https://en.wikipedia.org/wiki/Bucket\_sort