Лабораторна робота № 4

Тема: розробка та дослідження алгоритму сортування шляхом вибору (або внутрішнього обмінного сортування).
Мета: детально проаналізувати та дослідити алгоритм сортування шляхом вибору (або внутрішнього обмінного сортування).

Основні теоретичні відомості

1. Сортування методом прямого вибирання

Ідея алгоритму сортування методом прямого вибирання така.

1. Серед n елементів масиву вибирається найменший.
2. Він міняється місцями з першим елементом a_1.
3. Далі цей процес повторюється із рештою $n-1$ елементами, $n-2$ елементами і т. д. доти, поки не залишиться один, найбільший елемент.

Такий алгоритм називають сортуванням за допомогою *прямого вибору.* І він у певному смислі протилежний алгоритму сортування вставками. В останньому випадку на кожному кроці розглядається тільки один черговий елемент вихідної послідовності та всі елементи готової послідовності, серед яких відшуковується точка вставки; при прямому виборі для пошуку *одного* елементу з найменшим ключем проглядаються всі елементи вихідної послідовності і знайдений розташовується як черговий елемент у готову послідовність.

Псевдокод, що дозволяє реалізувати дану ідею, наведений на рисунку 1. Тут же наведено вартість кожної операції а також кількість разів її виконання.

Аналіз складності алгоритму

Як і у випадку сортування вставками, склавши внески всіх рядків отримаємо час виконання в загальному випадку

$$T(n)= c_1 n+(c_2+c_3+c_8+c_9)(n-1)+c_4\sum_{j=2}^{n} t_j +$$

$$+ c_5\sum_{j=2}^{n}(t_j-1)+(c_6+c_7)\sum_{j=2}^{n}(t_j-1).$$

Оцінимо час роботи в найбільш та найменш сприятливих випадках. Отже, найсприятливіший випадок буде у випадку, коли масив вже відсортовано і блоки 6 і 7 виконуватись не будуть. Тоді формула часу вико-

нання з урахуванням формул (3.1 – 3.2) набуває вигляду

$$T(n)= c_1n+(c_2+c_3+c_8+c_9)(n-1)+c_4\sum_{j=2}^{n} j + c_5\sum_{j=2}^{n}(j-1)+(c_6+c_7)\cdot 0 =$$

$$= c_1n+(c_2+c_3+c_8+c_9)(n-1)+c_4\left(\frac{n(n+1)}{2}-1\right)+c_5\left(\frac{n(n-1)}{2}\right)=$$

$$= n^2\left(\frac{c_4}{2}+\frac{c_5}{2}\right)+n\left(c_1+c_2+c_3+\frac{c_4}{2}-\frac{c_5}{2}+c_8+c_9\right)-(c_1+c_3+c_4+c_8+c_9).$$

Номер	Псевдокод	Вартість	Кількість разів
1	**for** *i*← *1* **to** *length*[*A*]-*1*	c_1	n
2	**do** *min*←*A*[*i*]	c_2	$n-1$
3	*N_min*← *i*	c_3	$n-1$
4	**for** *j*← *i+1* **to** *length*[*A*]	c_4	$\sum_{j=i+1}^{n} t_j = \sum_{j=2}^{n} t_j$
5	**do if** *A*[*i*]<*min* **then**	c_5	$\sum_{j=i+1}^{n}(t_j-1)=\sum_{j=2}^{n}(t_j-1)$
6	min←*A*[*N_min*]	c_6	$\sum_{j=i+1}^{n}(t_j-1)=\sum_{j=2}^{n}(t_j-1)$
7	*N_min*←*j*	c_7	$\sum_{j=i+1}^{n}(t_j-1)=\sum_{j=2}^{n}(t_j-1)$
8	*A*[*N_min*]←*A*[*i*]	c_8	$n-1$
9	*A*[*i*]←*min*	c_9	$n-1$

Рисунок 1 – Псевдокод алгоритму сортування прямим вибором

Найнесприятливіший випадок має місце коли масив відсортовано у зворотному порядку. Тоді формула визначення часу набуває вигляду

$$T(n)=c_1n+(c_2+c_3+c_8+c_9)(n-1)+c_4\left(\frac{n(n+1)}{2}-1\right)+$$

$$(c_5+c_6+c_7)\left(\frac{n(n-1)}{2}\right)=n^2\left(\frac{c_4+c_5+c_6+c_7}{2}\right)+$$

$$+n\left(c_1+c_2+c_3-\frac{c_4+c_5+c_6+c_7}{2}+c_8+c_9\right)-(c_2+c_3+c_4+c_8+c_9).$$

Отже, даний алгоритм в обох випадках має значний час сортування $O(n^2)$. Переваги – стійкість та відсутність додаткової пам’яті.

2 Сортування методом бульбашки

Тут обмін місцями двох елементів є характерною особливістю процесу сортування. Викладений нижче алгоритм ґрунтується на порівнянні та зміні місць для пари сусідніх елементів і продовженні цього процесу доти, поки не будуть упорядковані всі елементи (тому даний алгоритм отримав ще назву „сортування прямим обміном”). І такі проходи по масиву слід повторювати, пересуваючи щоразу найменший елемент послідовності, яка залишилась, до лівого кінця масиву. Якщо розглядати масиви як вертикальні, а не горизонтальні, то елементи можна інтерпретувати як бульбашки у чані з водою, причому вага кожного відповідає його ключу [4]. Тут під час кожного проходу одна бульбашка „піднімається” до рівня, що відповідає її вазі (табл. 1).

Таблиця 1 – Сортування методом бульбашки

Елементи масиву / проходу	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
	44	6	6	6	6	6	6	6
	55	44	12	12	12	12	12	12
	32	55	44	18	18	18	18	18
	42	12	55	44	42	42	42	42
	94	42	18	55	44	44	44	44
	18	94	42	42	55	55	55	55
	6	18	94	67	67	67	67	67
	67	67	67	94	94	94	94	94

Псевдокод, що реалізує такий підхід у найпростішому вигляді, наведено на рисунку 2.

```
BUBBLESORT(A)
1.  for i←2 to n
2.     do  for j←n downto i
3.            do  if A[j-1] > A[j] then
4.                                  виконати обмін A[j]↔A[j-1]
```

Рисунок 2 – Псевдокод сортування методом бульбашки

Поліпшення цього алгоритму очевидне. На прикладі у табл. 1 бачимо, що три останніх проходи не впливають на порядок елементів, оскільки вони вже відсортовані. Отже, слід запам’ятовувати, мали місце перестановки під час деякого проходу або ні. І якщо у черговому проході перестановок не було – алгоритм можна завершувати.

Зазначимо, що це поліпшення можна знову ж поліпшити, якщо за-

пам’ятовувати не лише факт обміну, а й індекс *k* останнього обміну. Цілком зрозуміло, що усі пари сусідніх елементів вище цього індексу вже відсортовані. Тому перегляд можна закінчувати на цьому індексі, а не йти до заздалегідь визначеної нижньої межі для *i*.

Крім того, тут можна помітити ще деяку своєрідну асиметрію: „спливає” бульбашка на потрібне місце за один прохід, а „тоне” на потрібне місце дуже повільно – лише на один крок за один прохід. Наприклад, масив

12 18 42 44 55 67 94 06

за допомогою удосконаленого бульбашкового сортування можна упорядкувати за один прохід, а для масиву

94 06 12 18 42 44 55 67

потрібно сім проходів. Така асиметрія наводить на думку про третє поліпшення: послідовно чергувати напрямок проходів. Алгоритм, що дозволяє реалізувати таке поліпшення називають *шейкерним сортуванням* (ShakerSort) [4].

Псевдокод, що дозволяє реалізувати алгоритм шейкерного сортування наведено на рисунку 3 (тут змінні *L, R, k* – позначають ліву та праву границі ділянки сортованого масиву, а також індекс останнього обміну, відповідно). А таблиця 2 ілюструє процес сортування тих же (табл. 1) восьми ключів.

```
SHAKERSORT(A);
1.  L←2
2.  R←n
3.  k←n
4.   repeat
5.     for j←R downto L
6.       do  if A[j-l]>A[j] then
7.                              виконати обмін A[j]↔A[j-1]
8.                              k←j
9.     L←k+l:
10.    for j← L to R
11.      do if A[j-l]>A[j] then
12.                             виконати обмін A[j]↔A[j-1]
13.                             k←j
14.    R←k-1
15.  until L←R
```

Рисунок 3 – Псевдокод реалізації шейкерного сортування

Аналіз алгоритмів сортування методом бульбашки та його модифікацій ми пропонуємо виконати читачу самостійно, за аналогією з попередньо розглянутими алгоритмами. Проте, слід зауважити, усі алгоритми, що ми розгядали з Вами раніше, а також у цій роботі мають складність порядку $O(n^2)$, оскільки усі вищеперераховані вдосконалення не впливають на кількість переміщень, а лише скорочують кількість зайвих подвійних перевірок. Крім того, обмін місцями двох елементів – найчастіше більш трудомістка операція, ніж порівняння ключів, тому наведені поліпшення не дають значного виграшу [3, 4].

Таблиця 2 – Приклад шейкерного сортування

L	2	3	3	4	4
R	8	8	7	7	4
Напрямок проходу	↑	↓	↑	↓	↑
Елементи масиву	44	6	6	6	6
	55	44	44	12	12
	32	55	12	44	18
	42	12	42	18	42
	94	42	55	42	44
	18	94	18	55	55
	6	18	67	67	67
	67	67	94	94	94

Взагалі, шейкерне сортування рекомендується використовувати у випадках, коли відомо, що елементи майже впорядковані, що на практиці зустрічається досить рідко.

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 квадратичний час.
6. Поясніть чому при оцінці складності алгоритму нас частіше за все цікавить робота у найгіршому випадку.
7. Виконайте детальний аналіз алгоритму внутрішнього обмінного сортування (бульбашкового сортування).
8. Виконайте детальний аналіз алгоритму сортування шляхом вибору.

Звіт повинен містити

Увага! Студенти з непарним номером варіанту виконують дослідження алгоритму сортування шляхом вибору, а з парним номером – внутрішнього обмінного сортування (бульбашкового сортування).

1. Тему, мету та порядок виконання роботи.
2. Ідея алгоритму сортування (шляхом вибору або внутрішнього обмінного сортування, відповідного Вашого варіанту).
3. Власний приклад роботи відповідного алгоритму сортування на масиві з 10 чисел.
4. Алгоритм сортування у графічному вигляді.
5. Теоретична оцінка складності алгоритму для трьох випадків.
    5.1 для найкращого випадку.
    5.2 для найгіршого випадку.
    5.3 для середнього випадку.
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 значень 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).*
    7.1 Таблиця та графік для найкращого випадку.
    7.2 Таблиця та графік для найгіршого випадку.
    7.3 Таблиця та графік для середнього випадку.
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.