Міністерство освіти і науки України

Національний технічний університет

«Харківський політехнічний інститут»

Навчально-науковий інститут комп’ютерних наук та інформаційних технологій

ЗВІТ

про виконання лабораторної роботи №4

Варіант 7

за дисципліною

«Алгоритми і структури даних»

Виконав:

студент групи КН-320Б

Миргород В.І.

Перевірила:

старший викладач

Мошко Є.О.

Харків 2022

**Лабораторна робота №4**  
**Тема лабораторної роботи:** Алгоритми сортування та асимптотичний аналіз  
алгоритмів.  
**Мета:** вивчити основні алгоритми сортування масивів і освоїти їх на практиці.  
Перевірити роботу алгоритмів на різних наборах даних, провести асимптотичний  
аналіз алгоритмів сортування.

**Завдання за варіантом:**

* Алгоритм сортування вставками
* Алгоритм порозрядного сортування
* Алгоритм пірамідального сортування

**Порядок виконання роботи:**  
1. Створити функцію сортування масиву алгоритмами сортування, які треба  
обрати у таблиці 3 відповідно до свого варіанту. Варіант завдання вибирається за номером студента в списку групи.  
2. Будь-яким способом заповнити елементи масиву значеннями.  
3. Виконати сортування масиву першим алгоритмом і проаналізувати отримані результати.  
4. Перевірити всі варіанти вихідного заповнення масиву: випадковим чином,  
відсортованого у порядку зростання, відсортованого у порядку спадання.  
Переконатися в правильності сортування у всіх випадках. Зробити висновки.  
5. Повторити пункти 2-4 для другого та третього алгоритму сортування.  
6. Дослідити складність алгоритмів. Провести асимптотичний аналіз алгоритмів сортування та зробити висновки.  
7. Здійснити порівняння алгоритмів сортування для n = 10, 100, 1000, 10000 і в наступному порядку вхідних елементів:  
− елементи вже впорядковані;  
− елементи в зворотному порядку;  
− розстановка елементів випадкова.  
8. Виконання порівняння алгоритмів сортування на основі наступних кроків:  
− створити таблицю асимптотичних оцінок трудомісткості алгоритмів в кращому, середньому, гіршому випадках;  
− розставити лічильники операцій у функціях угруповань;

− провести експеримент, визначити середню кількість операцій для різних  
угруповань, побудувати графіки;  
− для кожного виклику сортування генерувати новий масив;  
− створити таблиці і представити графіки експериментальних оцінок  
алгоритмів.  
Необхідно враховувати, що порівняння алгоритмів проводиться на однакових  
вхідних даних.

Аналіз завдання:

Розробити реалізацію алгоритмів сортування згідно з варіантом. Провести випробування алгоритмів, впевнитись у правильності їх роботи. Відсортувати масиви різного розміру та з різним розташуванням елементів. Порахувати кількість операцій, зробити оцінку складності алгоритмів. Згідно з результатами, скласти таблиці та провести їх аналіз. Визначити найкращі сторони кожного з алгоритмів.

Структура вхідних та вихідних даних:

Масив даних генерується випадково. Далі масив передається у функцію, що виконує сортування. Вихідними даними є відсортований масив.

**Сортування вставками**

Алгоритм сортування вставками базується на знаходженні правильного місця для кожного елемента. Кожний елемент порівняється з попередніми, які вже відсортовані. Таким чином знаходиться місце в масиві, на якому лівий елемент буде менший за поточний, а правий – більший.

void InsertionSort(int\* arr, int n) {

for (int i = 1; i < n; i++) {

int temp = arr[i];

for (int j = i - 1; j >= 0; j--) {

if (arr[j] > temp) {

arr[j + 1] = arr[j];

}

else {

arr[j + 1] = temp;

break;

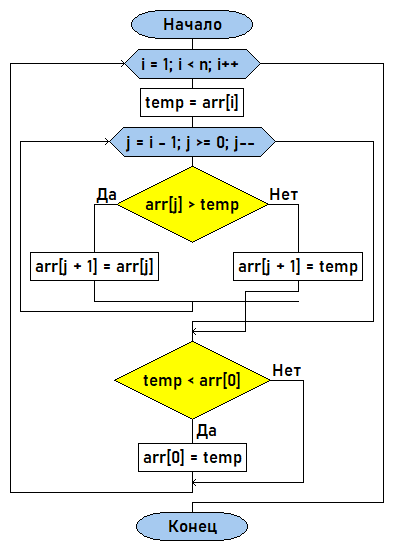
}

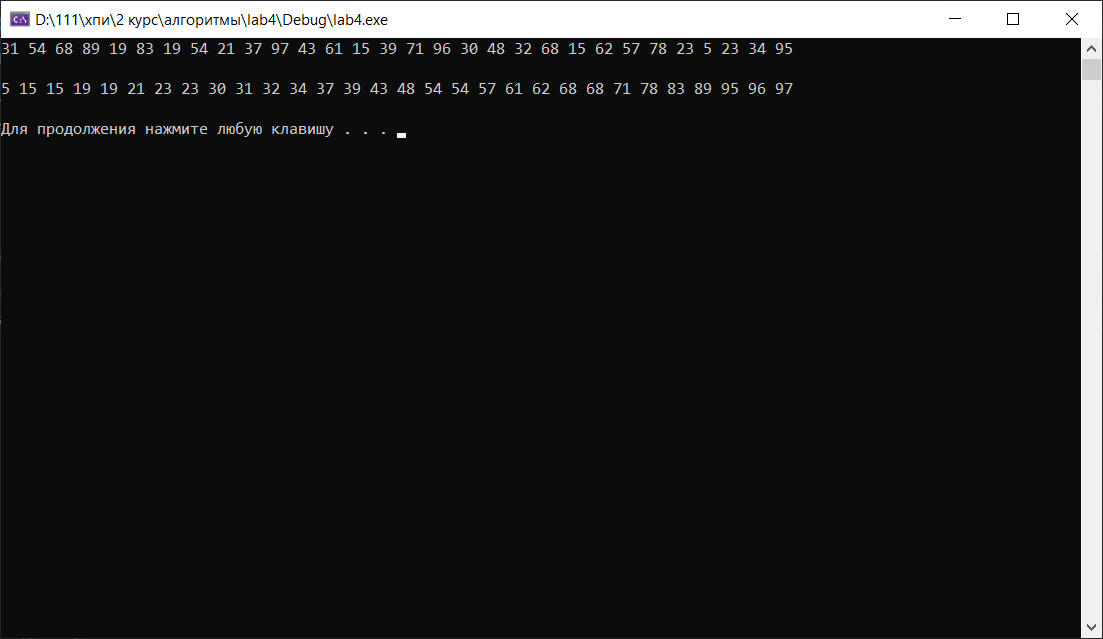
}

if (temp < arr[0])

arr[0] = temp;

}}





Такий алгоритм потребує O(n2) часу у середньому та гіршому випадках, та O(n) часу, якщо масив відсортований. Не потребує додаткової пам’яті для роботи.

**Порозрядне сортування**

Алгоритм порозрядного сортування складається з розбиття масиву на підмасиви, в залежності від значень одного з розрядів числа. Розподіляючи числа по кожному з їх розрядів, дійдемо до того, що всі числа масиву будуть відсортовані.

void RadixSort(int\* arr, int n, int nums) {

int\*\* temp\_arr = new int\* [10];

for (int i = 0; i < 10; i++)

temp\_arr[i] = new int[(int)(0.3 \* n)];

int\* count = new int[10]{ 0 };

int x;

int exp = 1;

for (int y = 0; y < nums; y++, exp \*= 10) {

for (int i = 0; i < 10; i++)

count[i] = 0;

for (int i = 0; i < n; i++) {

x = (arr[i] / exp) % 10;

temp\_arr[x][count[x]] = arr[i];

count[x]++;

}

int k = 0;

for (int i = 0; i < 10; i++) {

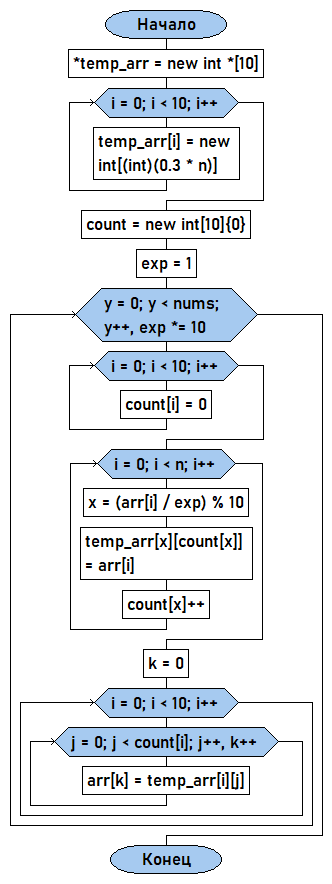
for (int j = 0; j < count[i]; j++, k++) {

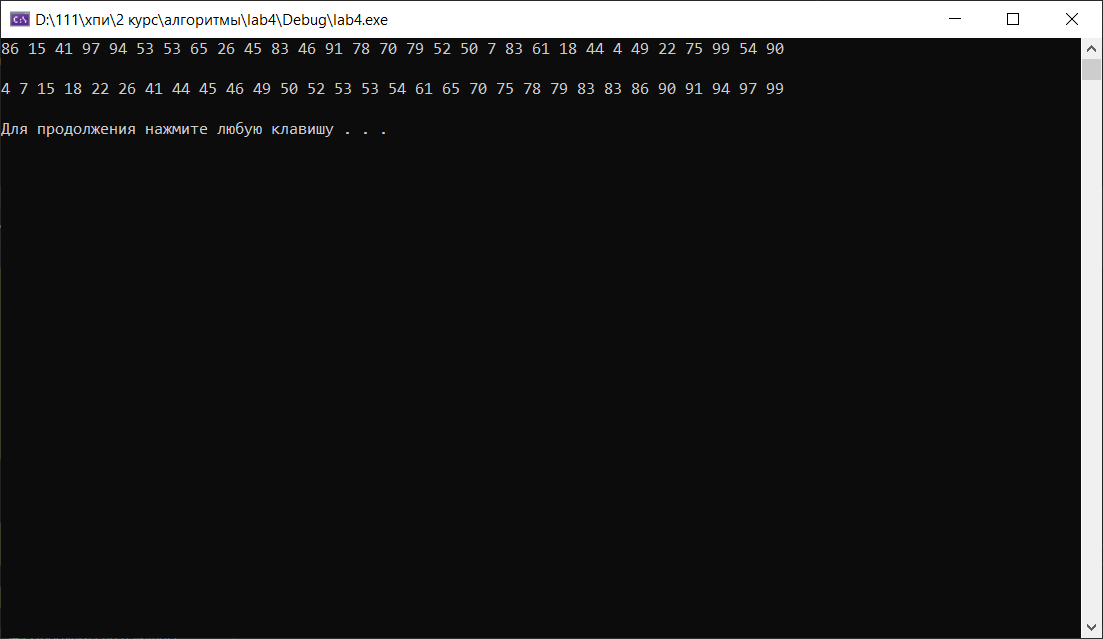
arr[k] = temp\_arr[i][j];

}

}

}

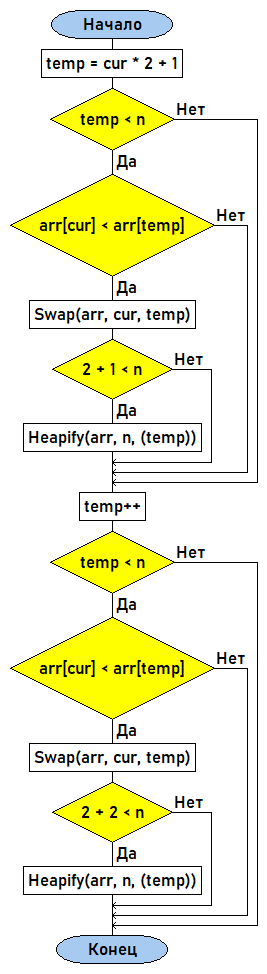




При будь-якому розташуванні елементів, масив сортується за однакову кількість часу. Швидкість сортування – O(w\*n), де w – кількість біт для зберігання елемента масиву. Досить швидко сортує масиви великого розміру, але потребує багато додаткової пам’яті.

**Пірамідальне сортування**

При пірамідальному сортування масив представляється у вигляді піраміди, тобто бінарного дерева, для якого виконуються 2 умови: корінь є максимальним елементом, а також значення листа не перевищує значення його батьківського вузла. Процес сортування полягає в наданні масиву вигляду саме піраміди, так щоб виконувались ці правила. Далі перший елемент копіюється в кінець масиву, а масив знову підводиться під ці 2 правила.



void HeapSort(int\* arr, int n) {

for (int cur = ceil(n / 2); cur >= 0; cur--) {

Heapify(arr, n, cur);

}

//получили max heap

for (int i = n - 1; i > 0; i--) {

Swap(arr, 0, i);

//swap(arr[0], arr[i]);

Heapify(arr, i, 0);

}

}

void Heapify(int\* arr, int n, int cur) {

int temp = cur \* 2 + 1;

if (temp < n) {

if (arr[cur] < arr[temp]) {

//swap(arr[cur], arr[temp]);

Swap(arr, cur, temp);

if (temp \* 2 + 1 < n)

Heapify(arr, n, (temp));

}

}

temp++;

if (temp < n) {

if (arr[cur] < arr[temp]) {

//swap(arr[cur], arr[temp]);

Swap(arr, cur, temp);

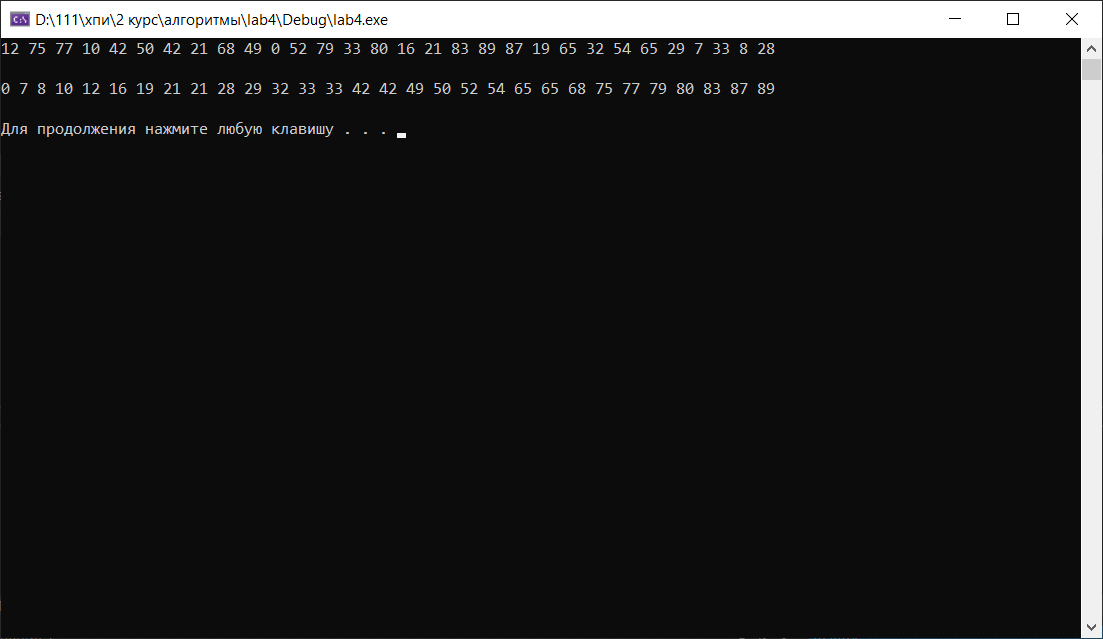
if (temp \* 2 + 2 < n)

Heapify(arr, n, (temp));

}

}

}



Алгоритм пірамідального сортування не потребує додаткової пам’яті. Складність алгоритму – O(nlogn) при будь-якому розташуванні елементів. Не найшвидший спосіб сортування, але працює стабільно, незалежно від порядку елементів.

**Аналіз алгоритмів**

**Час виконання алгоритмів сортування**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Random | | | |
|  | 10 | 100 | 1000 | 10000 |
| Insertion sort | 9.5e-07 | 7.2e-06 | 0.00055 | 0.054 |
| Radix sort | 7.7e-06 | 1.25e-05 | 5.65e-05 | 0.00045 |
| Heap sort | 1.9e-06 | 6.77e-05 | 0.0054 | 0.48 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Sorted | | | |
|  | 10 | 100 | 1000 | 10000 |
| Insertion sort | 4e-07 | 8.3e-07 | 4.2e-06 | 5.39e-05 |
| Radix sort | 8.3e-06 | 1.4e-05 | 4.64e-05 | 0.00042 |
| Heap sort | 1.6e-06 | 6.66e-05 | 0.0061 | 0.46 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Reverse | | | | |
|  | 10 | 100 | 1000 | 10000 |
| Insertion sort | 7e-07 | 6.8e-06 | 0.00055 | 0.063 |
| Radix sort | 6.4e-06 | 1.18e-05 | 4.46e-05 | 0.00047 |
| Heap sort | 1.7e-06 | 7.8e-05 | 0.0048 | 0.46 |

**Висновки:**

Сортування вставками має досить просту програмну реалізацію. Не надто швидкий алгоритм сортування, але й не найгірший. Швидкість роботи також залежить від розміщення елементів у масиві. Тобто масив, що вже відсортовано, сортується швидше. Не вимагає додатковою пам’яті.

Порозрядне сортування має найкращій результат у порівнянні з іншими реалізованими алгоритмами. Сортує великі масиви значно швидше, ніж інші алгоритми. Не залежить від порядку елементів у масиві. Із недоліків – потреба у великій кількості додаткової пам’яті.

Пірамідальне сортування працює повільніше за інші алгоритми. Сортування не залежить від розміщення елементів, не вимагає додаткової пам’яті. Але час виконання на середньому рівні.

Складність алгоритмів сортування:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Insertion sort | O(n^2) | O(n) | O(n^2) |
| Radix sort | O(w\*n) | O(w\*n) | O(w\*n) |
| Heap sort | O(nlogn) | O(nlogn) | O(nlogn) |