**Практична робота № 17-18**

**Завдання №1**

Проаналізувати теоретичні відомості та програмний код алгоритму сортування за допомого дерева. Запустити програму та написати коментарі до програми:

#include <iostream>

#include <cstdlib>

using namespace std;

struct tnode // реалізація дерева

{

int field; // поле данних

struct tnode \*left; // лівий нащадок

struct tnode \*right; // правий нащадок

};

void treeprint(tnode \*tree) // обхід дерева у симетричному порядку

{

if (tree != NULL){ // перевірка умови на порожність вузла

treeprint(tree->left); // рекурсія для лівого піддерева

cout << tree->field << " "; // відображення кореня дерева

treeprint(tree->right); // рекурсія для правого піддерева

}

}

struct tnode \* addnode(int x, tnode \*tree) { // додавання вузлів дерева

if (tree == NULL)

{ // перевірка на існування дерева

tree = new tnode;

tree->field = x;

tree->left = NULL;

tree->right = NULL; // задання параметрів для нового дерева

}

else

if (x < tree->field)

tree->left = addnode(x, tree->left);

else

tree->right = addnode(x, tree->right); // умови додавання правих і лівих нащадків

return(tree); // повернення дерева

}

void freemem(tnode \*tree) // видалення піддерев

{

if (tree != NULL)

{ // перевірка умови на порожність вузла

freemem(tree->left);

freemem(tree->right);

delete tree; // видалення піддерева

}

}

int main() // головна функція

{

struct tnode \*root = 0;

system("chcp 1251"); // підтримка кирилиці

system("cls"); // очищення від зайвих символів

int a;

for (int i = 0; i< 8; i++)

{

cout << "Введіть вузол " << i + 1 << ": ";

cin >> a;

root = addnode(a, root);

} // введення значень у вузли

treeprint(root);

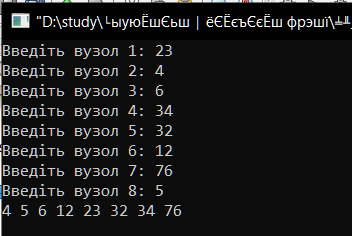
freemem(root); // виклик функцій сортування елементів дерева

cin.get();

cin.get();

return 0;

}



**Завдання №2**

На основі теоретичних відомостей самостійно дослідити алгоритм сортування пірамідою. Записати алгоритм:

1. Заповнюємо піраміду масивом;
2. Порівнюємо нащадків правого піддерева між собою;
3. Визначаємо найбільший з них та порівнюємо з батьківським та міняємо місцями їх для пересування найбільшого елементу вгору (просіювання);
4. Повторюємо дану операцію і з лівою частиною, поки найбільший елемент масиву не буде знаходитися в корені дерева;
5. Елемент, який виявився найбільшим, тобто знаходиться в корені дерева, міняємо місцями з останнім елементом та фіксуємо його там; (heapSort);
6. Викликаємо рекурсію попередніх функцій, поки масив не буде відсортованим.

**Завдання №3**

Реалізувати алгоритм сортування пірамідою:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

const N=10;

void siftDown(int \*numbers, int root, int bottom)

{

int maxChild;

int done = 0;

while ((root \* 2 <= bottom) && (!done))

{

if (root \* 2 == bottom)

{

maxChild = root \* 2;

}

else if (numbers[root \* 2] > numbers[root \* 2 + 1])

{

maxChild = root \* 2;

}

else

maxChild = root \* 2 + 1;

if (numbers[root] < numbers[maxChild])

{

int temp = numbers[root];

numbers[root] = numbers[maxChild];

numbers[maxChild] = temp;

root = maxChild;

}

else

done = 1;

}

}

void heapSort(int \*numbers, int array\_size)

{

for (int i = (array\_size / 2) - 1; i >= 0; i--)

{

siftDown(numbers, i, array\_size - 1);

}

for (int i = array\_size - 1; i >= 1; i--)

{

int temp = numbers[0];

numbers[0] = numbers[i];

numbers[i] = temp;

siftDown(numbers, 0, i - 1);

}

}

int main()

{

int a[N];

srand(time(NULL));

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

{

a[i] = rand() % 774;

}

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

printf("%d ", a[i]);

printf("\n");

heapSort(a, N);

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

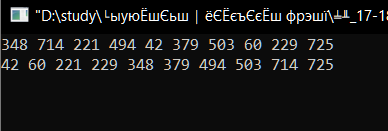
printf("%d ", a[i]);

printf("\n");

getchar();

return 0;

}



**Завдання №4**

Заповнити таблицю 1 для 10, 20 та 50 елементів. Чисельні значення елементів масиву лежать в діапазоні 0 .. 774 і випадково згенеровані.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Об’єм масиву | Критерії порівнянь | Алгоритм сортування | Результат | |
| 10 | Час | Алгоритм вставки | 0.005 | 0.005 |
| Алгоритм обміну | 0.001 | 0.001 |
| Алгоритм Шелла | 0.004 | 0.004 |
| Алгоритм Хоара | 0.005 | 0.004 |
| Сортування пірамідою | 0.003 | 0.004 |
| Кількість порівнянь | Алгоритм вставки | 33 | 35 |
| Алгоритм обміну | 50 | 50 |
| Алгоритм Шелла | 39 | 36 |
| Алгоритм Хоара | 34 | 26 |
| Сортування пірамідою | 45 | 40 |
| 20 | Час | Алгоритм вставки | 0.006 | 0.007 |
| Алгоритм обміну | 0.003 | 0.002 |
| Алгоритм Шелла | 0.007 | 0.005 |
| Алгоритм Хоара | 0.007 | 0.005 |
| Сортування пірамідою | 0.005 | 0.006 |
| Кількість порівнянь | Алгоритм вставки | 125 | 109 |
| Алгоритм обміну | 200 | 200 |
| Алгоритм Шелла | 108 | 107 |
| Алгоритм Хоара | 73 | 78 |
| Сортування пірамідою | 108 | 109 |
| 50 | Час | Алгоритм вставки | 0.013 | 0.016 |
| Алгоритм обміну | 0.006 | 0.007 |
| Алгоритм Шелла | 0.018 | 0.013 |
| Алгоритм Хоара | 0.014 | 0.013 |
| Сортування пірамідою | 0.017 | 0.013 |
| Кількість порівнянь | Алгоритм вставки | 668 | 705 |
| Алгоритм обміну | 1250 | 1250 |
| Алгоритм Шелла | 469 | 434 |
| Алгоритм Хоара | 257 | 288 |
| Сортування пірамідою | 337 | 333 |