МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ПОЛІТЕХНІЧНИЙ УНІВЕРСИТЕТ

Інститут комп’ютерних систем

Кафедра інформаційних систем

Звіт

Лабораторна робота №3

З дисципліни **«**Теорія алгоритмів**»**

**Тема:** **«Алгоритм пірамідального сортування»­**

Виконав:

Студент групи AI-191У

Ігнатюк А.Г

Перевірили:

Смик С. Ю

Одеса ­­­­– 2020

**Тема:**  «Алгоритм пірамідального сортування»

**Мета роботи:** Розглянути алгоритм пірамідального сортування(Heapsort, «Сортування купою»).

**План:**

1. Псевдокод алгоритму;
2. Графічне зображення процесу створення максимальної купи;
3. Графічне зображення фази сортування купою, а також фази просіювання;
4. Результати моделювання;
5. Висновок.
6. **Сортування купою. Псевдокод**

Нижче подано псевдокод даного алгоритму:

# HEAP SORT

**def heapify(arr, n, i):**

**largest = i** # Initialize largest as root

**l = 2 \* i + 1** # left = 2\*i + 1

**r = 2 \* i + 2** # right = 2\*i + 2

# Checking if there's a left element that is bigger than root

**if l < n and arr[i] < arr[l]:**

**largest = l** # then change largest

# Checking if there's a right element that is bigger than root

**if r < n and arr[largest] < arr[r]:**

**largest = r** # then change largest

# If largest element isn't root than swap them

**if largest != i:**

**arr[i],arr[largest] = arr[largest],arr[i]** # swap

# Calling heapify for the root element

**heapify(arr, n, largest)**

# Main function to sort array of n elements

**def heapSort(arr):**

**n = len(arr)** # defining the length of array

# Start making max-heap from the end of array

**for i in range(n, -1, -1):**

**heapify(arr, n, i)**

# Pop the root element and rebuild max-heap

**for i in range(n-1, 0, -1):**

**arr[i], arr[0] = arr[0], arr[i]** # swap

**heapify(arr, i, 0)** # rebuild a max-heap

# T E S T

**array = [28, 76, 27, 10, 5, 35, 95, 16, 33]** # array to be sorted

**heapSort(array)** # calling main function

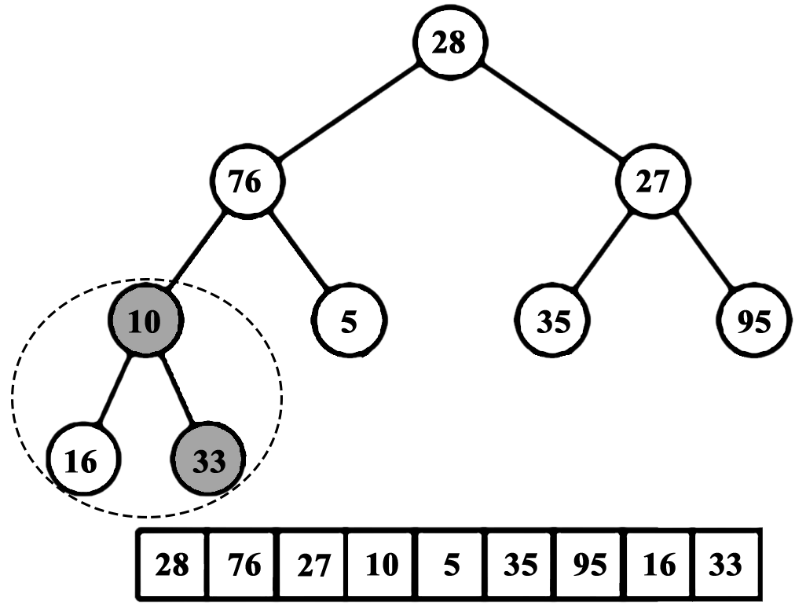
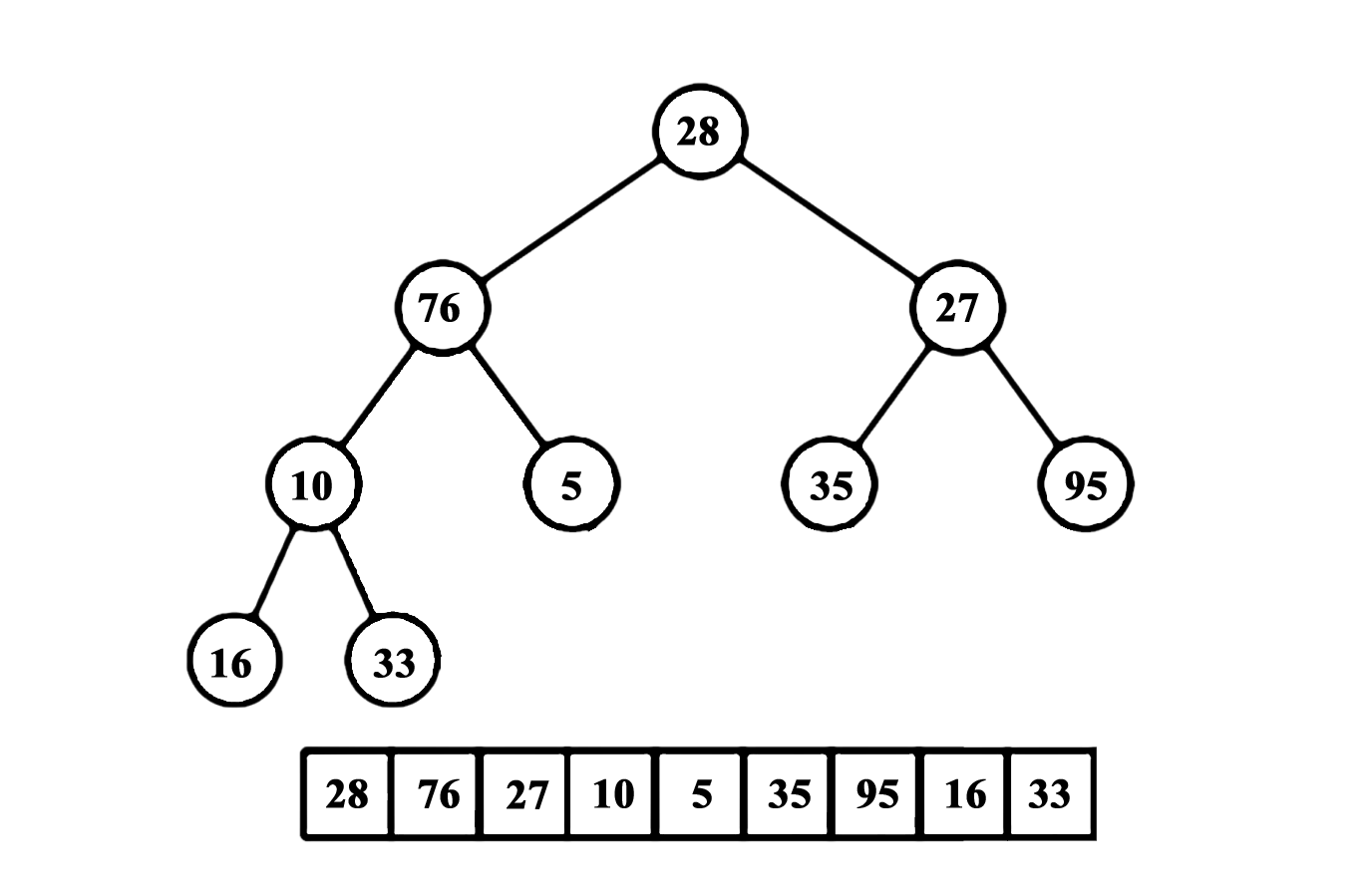
**n = len(array)**

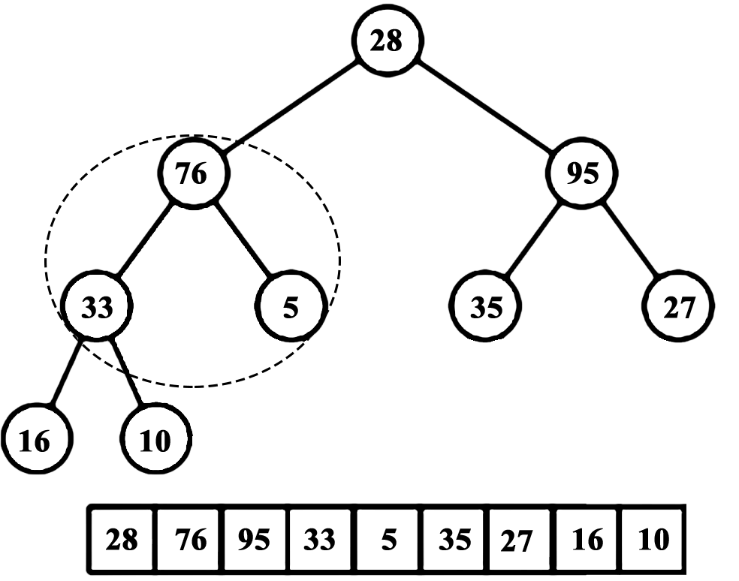
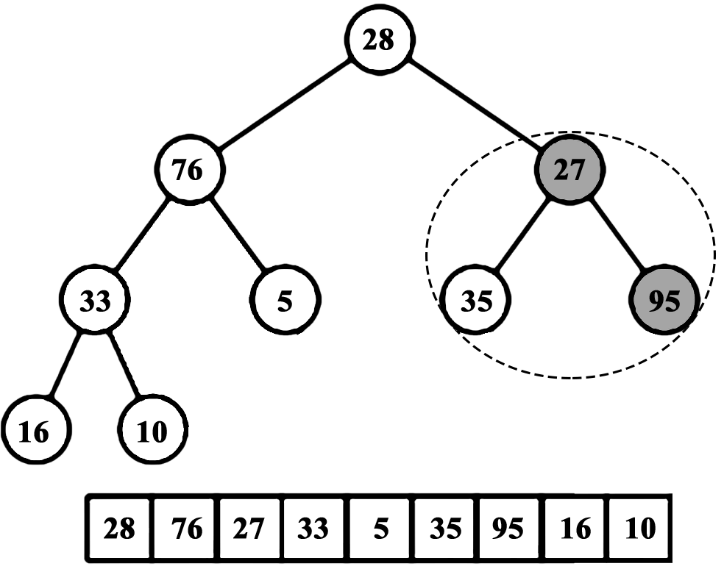
**print ("Sorted array is")**

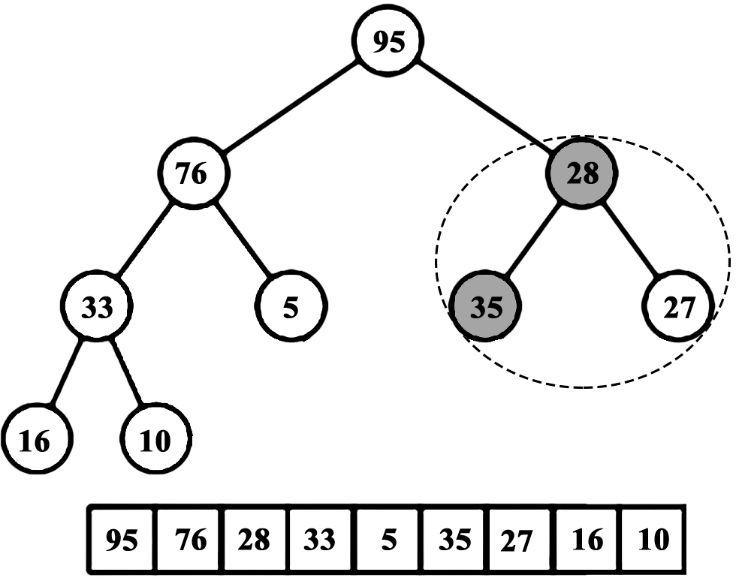
**for i in array :**

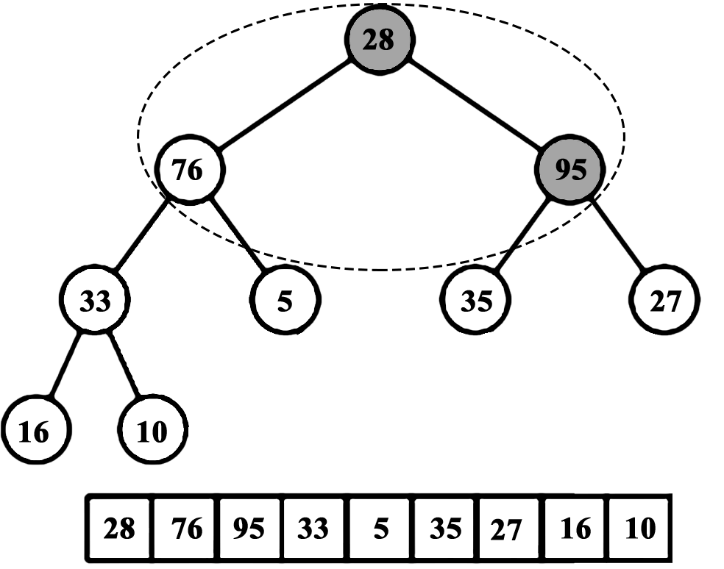
**print(i, end=" ")**

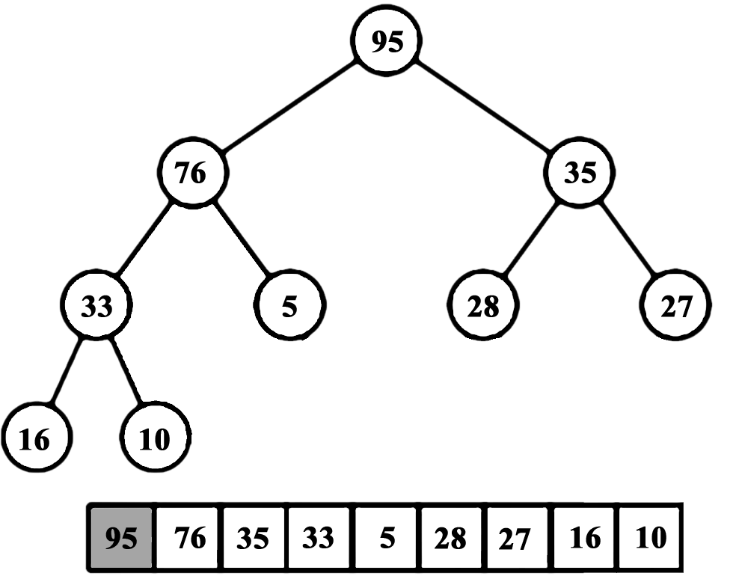
1. **Графічне зображення процесу створення максимальної купи**

****

****

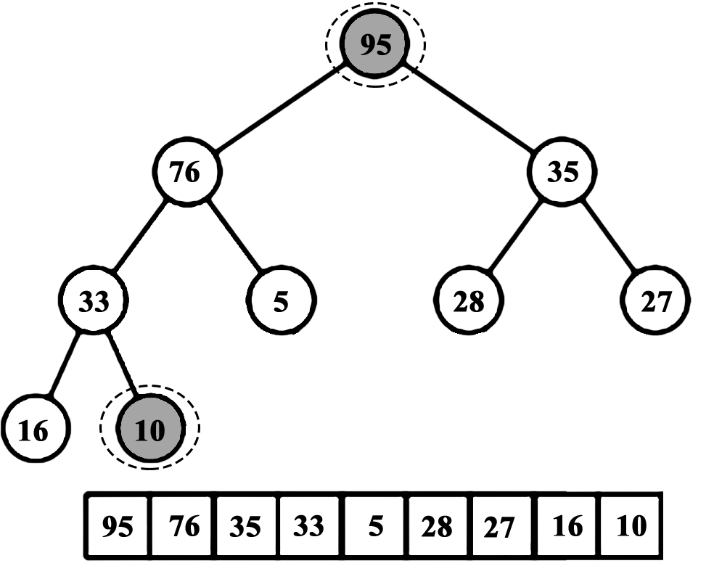
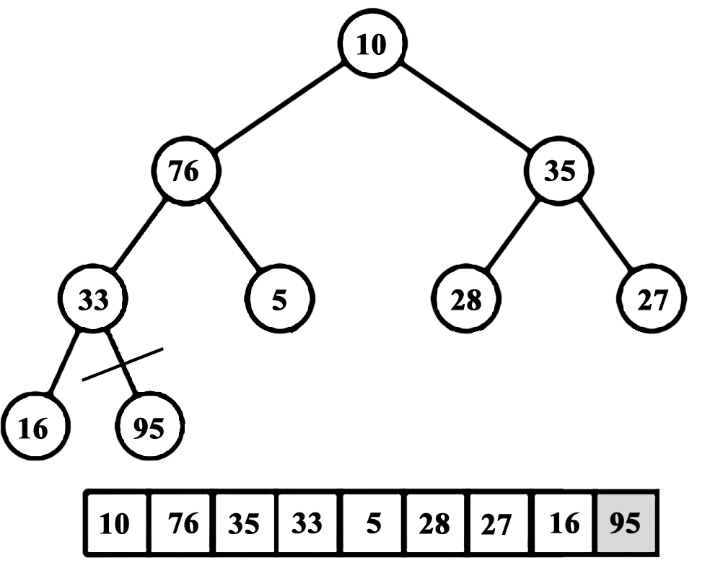
****

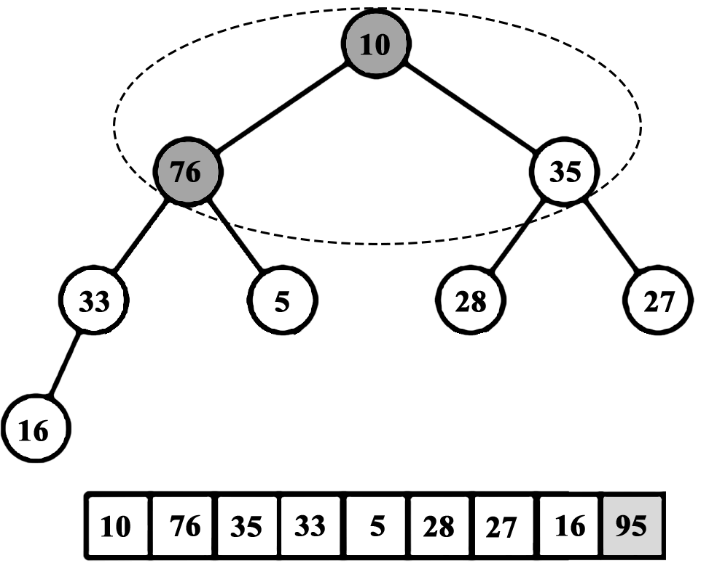
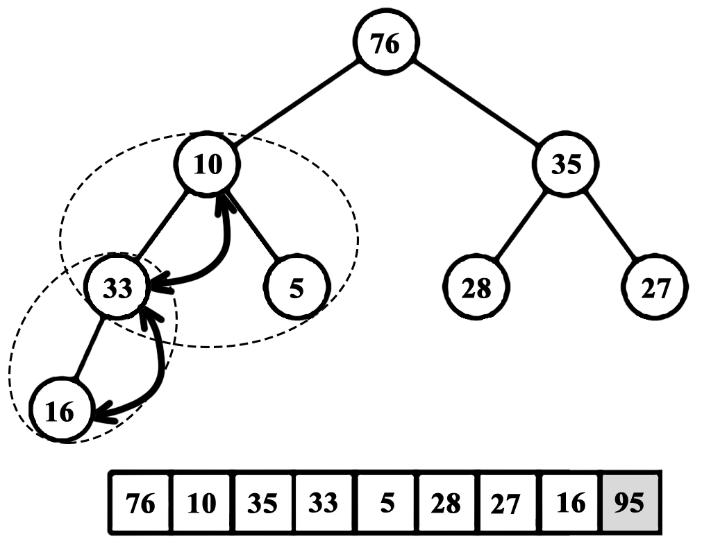
****

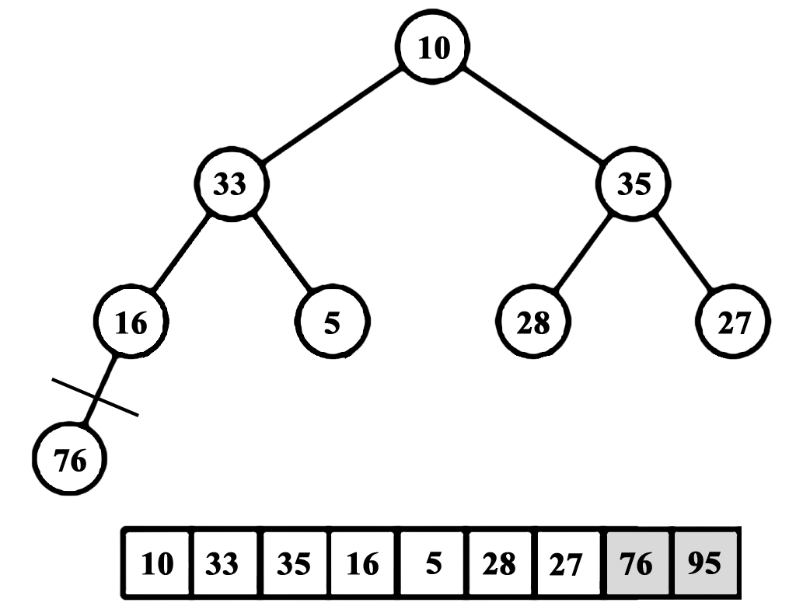
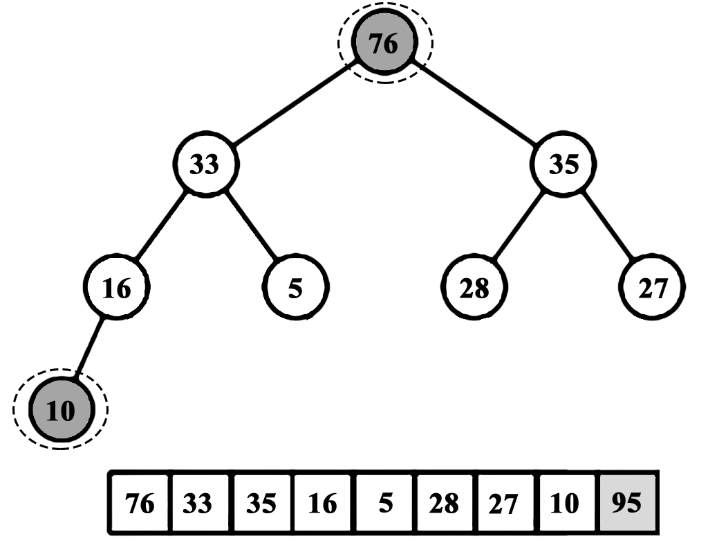
****

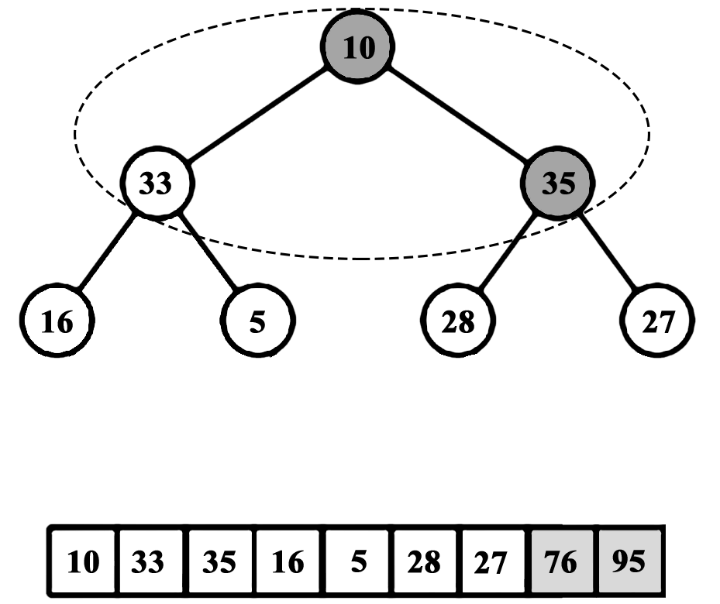
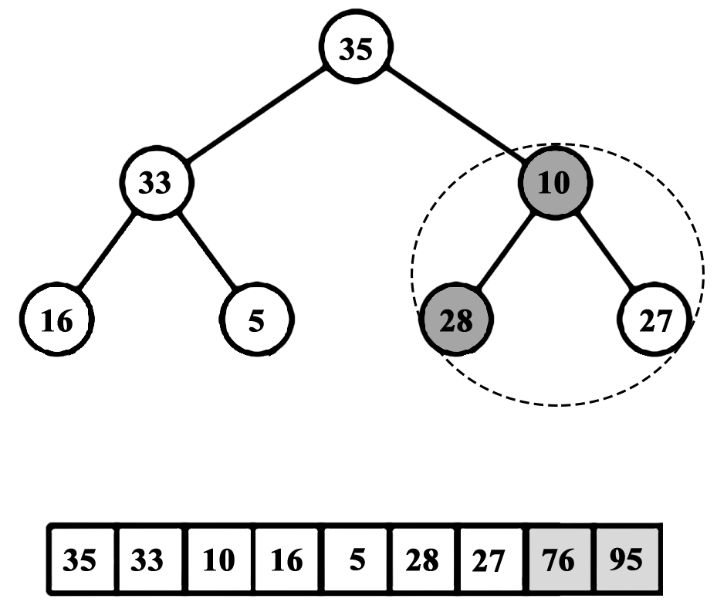
В результаті ми одержали так звану максимальну кучу. У ній кожен батьківський вузол більший за своїх дітей.

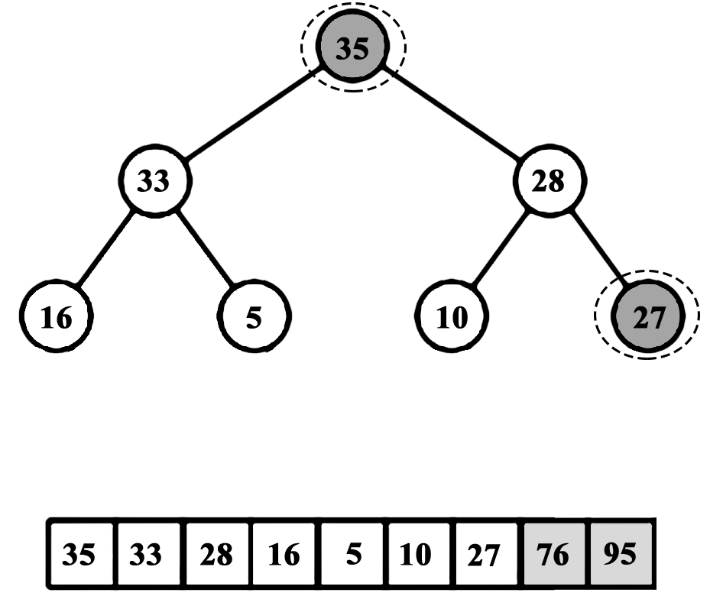
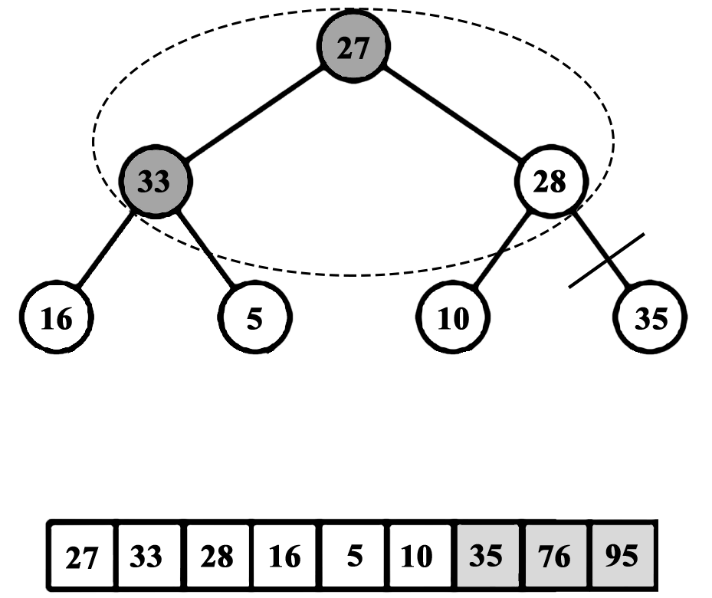
1. **Графічне зображення фази сортування купою, а також фази просіювання**

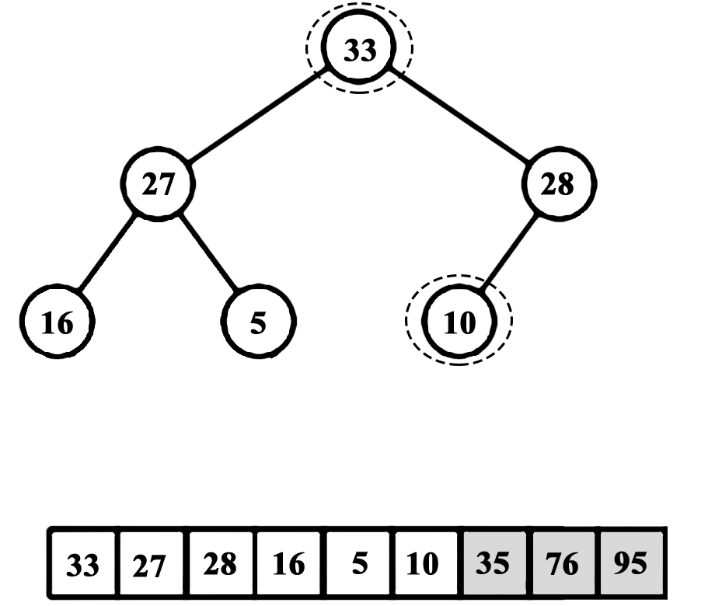
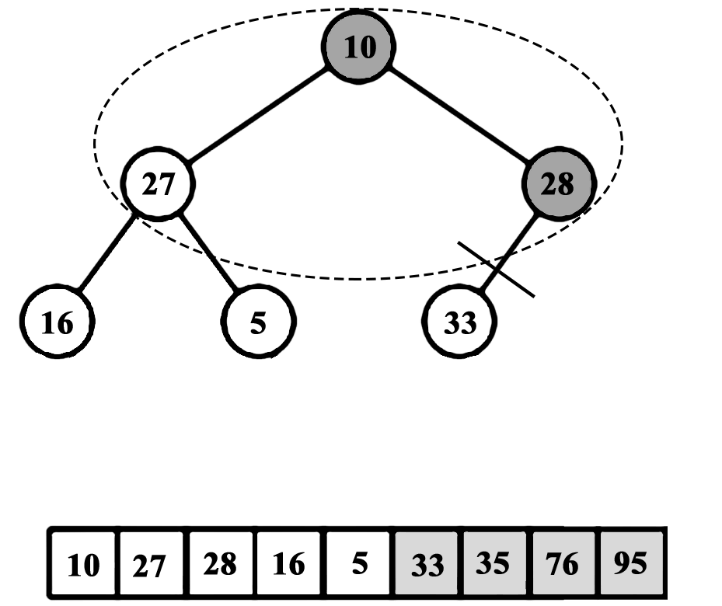
****

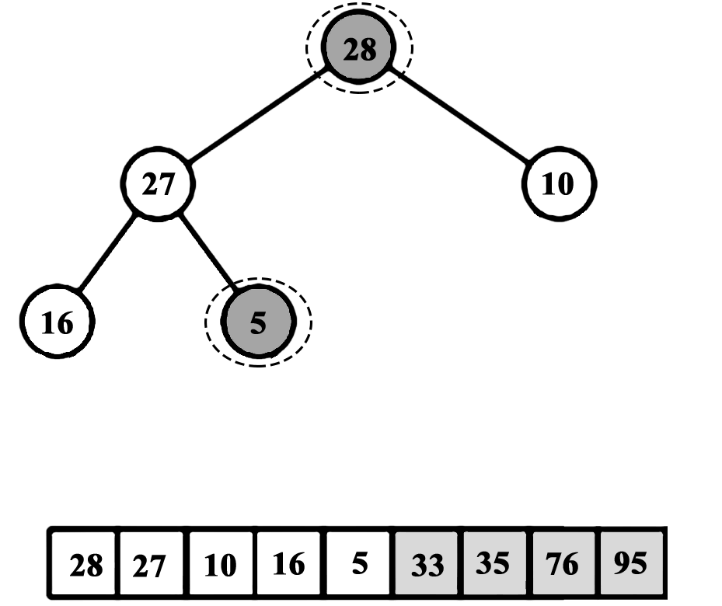
****

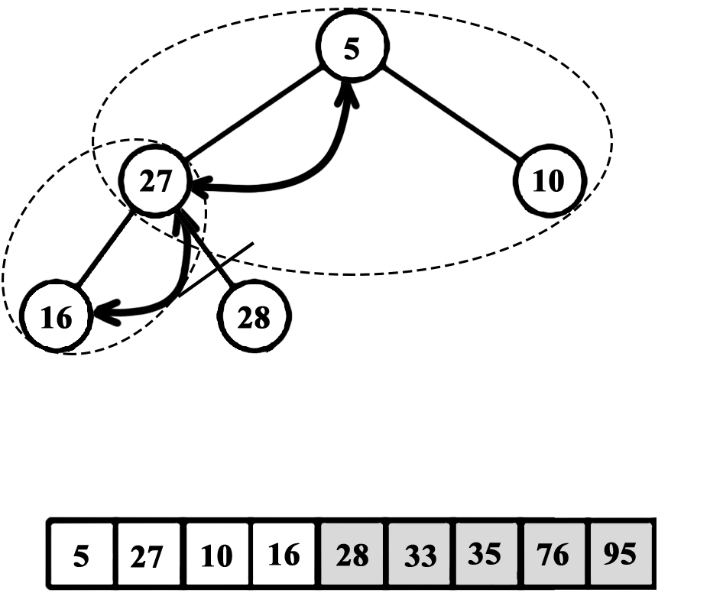
****

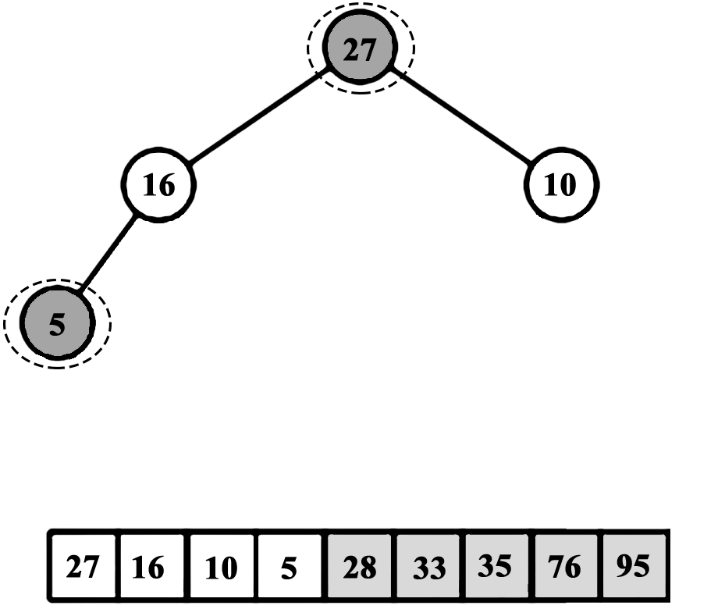
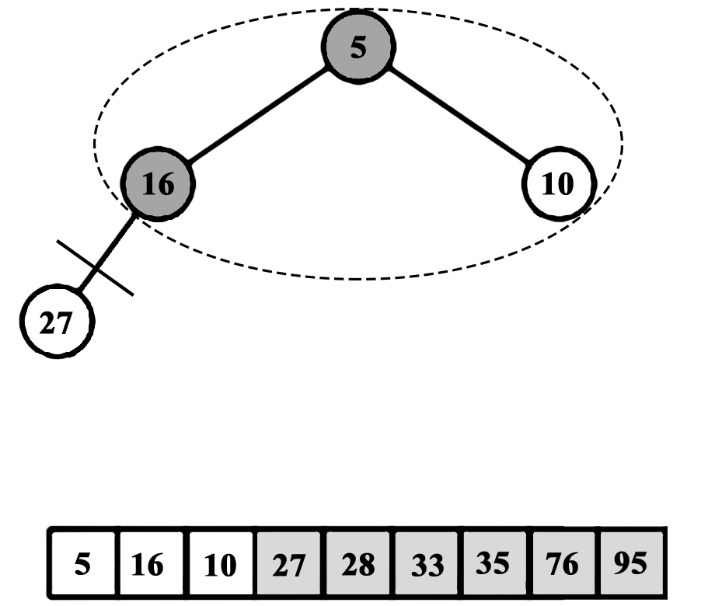
****

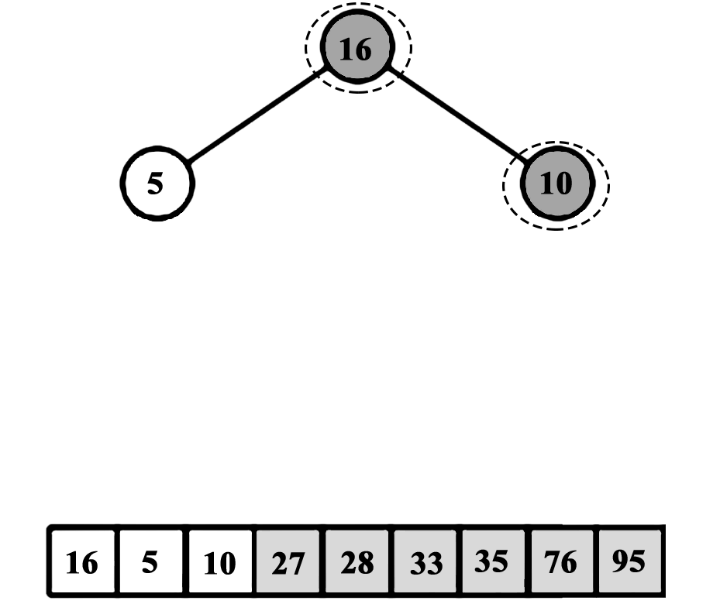
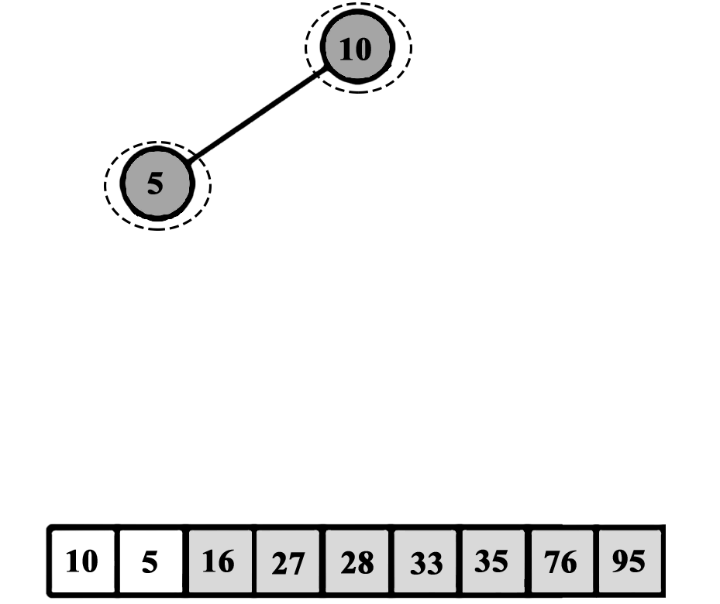
****

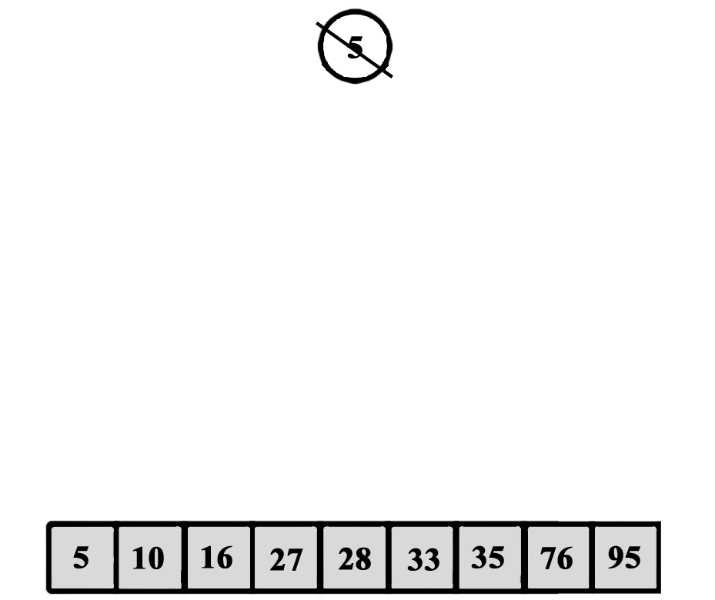
****

****

****

****

****

****

Таким чином масив повністю відсортований.

1. **Результати моделювання**

Вхідні дані:

Масив чисел: array = [28, 76, 27, 10, 5, 35, 95, 16, 33]

heapsort(array)

n = len(arr) [n=9]

for i in range(n, -1, -1) [n=9]

heapify(arr, n, i) [arr, 9, 9]

largest = i [largest = 9]

l = 2 \* i + 1 [l=19]

r = 2 \* i + 2 [r=20]

If l < n and arr[9] < arr[19] – FALSE

if r < n and arr[largest] < arr[r] - FALSE

if largest != i– FALSE

…

…

…

for i in range(n, -1, -1) [i=4]

heapify(arr, n, i) [arr, 9, 4]

largest = i [largest = 4]

l = 2 \* i + 1 [l=9]

r = 2 \* i + 2 [r=10]

If l < n and arr[i] < arr[l] [9<9 and …] – FALSE

if r < n and arr[largest] < arr[r] [10<9 and …] – FALSE

if largest != I – FALSE

for i in range(n, -1, -1) [i=3]

heapify(arr, n, i) [arr, 9, 3]

largest = i [largest = 3]

l = 2 \* i + 1 [l=7]

r = 2 \* i + 2 [r=8]

If l < n and arr[i] < arr[l] [7<9 and 10<16] – TRUE

largest = l [largest=7]

if r < n and arr[largest] < arr[r] [8<9 and 16<33] – TRUE

largest = r [largest=8]

if largest != i – TRUE

arr[i],arr[largest] = arr[largest],arr[i]

[28, 76, 27, 33, 5, 35, 95, 16, 10]

heapify(arr, n, largest) [arr, 9, 8]

largest = i [largest = 8]

l = 2 \* i + 1 [l=17]

r = 2 \* i + 2 [r=18]

If l < n and arr[i] < arr[l] [17<9 and …] – FALSE

if r < n and arr[largest] < arr[r] [18<9 and …] – FALSE

if largest != i – FALSE

for i in range(n, -1, -1) [i=2]

heapify(arr, n, i) [arr, 9, 2]

largest = i [largest = 2]

l = 2 \* i + 1 [l=5]

r = 2 \* i + 2 [r=6]

If l < n and arr[i] < arr[l] [5<9 and 27<35] – TRUE

largest = l [largest=5]

if r < n and arr[largest] < arr[r] [6<9 and 35<95] – TRUE

largest = r [largest=6]

if largest != i – TRUE

arr[i],arr[largest] = arr[largest],arr[i]

[28, 76, 95, 33, 5, 35, 27, 16, 10]

heapify(arr, n, largest) [arr, 9, 6]

largest = i [largest = 6]

l = 2 \* i + 1 [l=13]

r = 2 \* i + 2 [r=14]

If l < n and arr[i] < arr[l] [13<9 and …] – FALSE

if r < n and arr[largest] < arr[r] [14<9 and …] – FALSE

if largest != i – FALSE

for i in range(n, -1, -1) [i=1]

heapify(arr, n, i) [arr, 9, 1]

largest = i [largest = 1]

l = 2 \* i + 1 [l=3]

r = 2 \* i + 2 [r=4]

If l < n and arr[i] < arr[l] [3<9 and 76<33] – FALSE

if r < n and arr[largest] < arr[r] [4<9 and 76<5] – FALSE

if largest != i – FALSE

for i in range(n, -1, -1) [i=0]

heapify(arr, n, i) [arr, 9, 0]

largest = i [largest = 0]

l = 2 \* i + 1 [l=1]

r = 2 \* i + 2 [r=2]

If l < n and arr[i] < arr[l] [1<9 and 28<76] – TRUE

largest = l [largest=1]

if r < n and arr[largest] < arr[r] [2<9 and 76<95] – TRUE

largest = r [largest=2]

if largest != i – TRUE

arr[i],arr[largest] = arr[largest],arr[i]

[95, 76, 28, 33, 5, 35, 27, 16, 10]

heapify(arr, n, largest) [arr, 9, 2]

largest = i [largest = 2]

l = 2 \* i + 1 [l=5]

r = 2 \* i + 2 [r=6]

If l < n and arr[i] < arr[l] [5<9 and 28<35] – TRUE

largest = l [largest=5]

if r < n and arr[largest] < arr[r] [6<9 and 35<27] – FALSE

if largest != i – TRUE

arr[i],arr[largest] = arr[largest],arr[i]

[95, 76, 35, 33, 5, 28, 27, 16, 10]

heapify(arr, n, largest) [arr, 9, 5]

largest = i [largest = 5]

l = 2 \* i + 1 [l=11]

r = 2 \* i + 2 [r=12]

If l < n and arr[i] < arr[l] [11<9 and ... ] – FALSE

if r < n and arr[largest] < arr[r] [12<9 and …] – FALSE

if largest != i – FALSE

for i in range(n, -1, -1) [i=-1] – BREAK

for i in range(n-1, 0, -1) [i=8]

arr[i], arr[0] = arr[0], arr[i]

[10, 76, 35, 33, 5, 28, 27, 16, 95]

heapify(arr, i, 0) [arr, 8, 0]

largest = i [largest = 0]

l = 2 \* i + 1 [l=1]

r = 2 \* i + 2 [r=2]

If l < n and arr[i] < arr[l] [1<8 and 10<76] – TRUE

largest = l [largest=1]

if r < n and arr[largest] < arr[r] [2<8 and 76<35] – FALSE

if largest != i – TRUE

arr[i],arr[largest] = arr[largest],arr[i]

[76, 10, 35, 33, 5, 28, 27, 16, 95]

heapify(arr, n, i) [arr, 8, 1]

largest = i [largest = 1]

l = 2 \* i + 1 [l=3]

r = 2 \* i + 2 [r=4]

If l < n and arr[i] < arr[l] [3<8 and 10<33] – TRUE

largest = l [largest=3]

if r < n and arr[largest] < arr[r] [4<8 and 33<5] – FALSE

if largest != i – TRUE :

arr[i],arr[largest] = arr[largest],arr[i]

[76, 33, 35, 10, 5, 28, 27, 16, 95]

heapify(arr, n, i) [arr, 8, 3]

largest = i [largest = 3]

l = 2 \* i + 1 [l=7]

r = 2 \* i + 2 [r=8]

If l < n and arr[i] < arr[l] [7<8 and 10<16] – TRUE

largest = l [largest=7]

If r < n and arr[largest] < arr[r] [8<8 and …] – FALSE

if largest != i – TRUE

arr[i],arr[largest] = arr[largest],arr[i]

[76, 33, 35, 16, 5, 28, 27, 10, 95]

heapify(arr, n, i) [arr, 8, 7]

largest = i [largest = 7]

l = 2 \* i + 1 [l=15]

r = 2 \* i + 2 [r=16]

If l < n and arr[i] < arr[l] [15<8 and …] – FALSE

If r < n and arr[largest] < arr[r] [16<8 and …] – FALSE

if largest != i – FALSE

for i in range(n-1, 0, -1) [i=7]

arr[i], arr[0] = arr[0], arr[i]

[10, 33, 35, 16, 5, 28, 27, 76, 95]

heapify(arr, i, 0) [arr, 7, 0]

…

…

…

for i in range(n-1, 0, -1) [i=2]

arr[i], arr[0] = arr[0], arr[i]

[10, 5, 16, 27, 28, 33, 35, 76, 95]

heapify(arr, i, 0) [arr, 2, 0]

largest = i [largest = 0]

l = 2 \* i + 1 [l=1]

r = 2 \* i + 2 [r=2]

If l < n and arr[i] < arr[l] [1<2 and 10<5] – FALSE

If r < n and arr[largest] < arr[r] [2<2 and …] – FALSE

if largest != i – FALSE

for i in range(n-1, 0, -1) [i=1]

arr[i], arr[0] = arr[0], arr[i]

[5, 10, 16, 27, 28, 33, 35, 76, 95]

heapify(arr, i, 0) [arr, 1, 0]

largest = i [largest = 0]

l = 2 \* i + 1 [l=1]

r = 2 \* i + 2 [r=2]

If l < n and arr[i] < arr[l] [1<1 and 10<5] – FALSE

If r < n and arr[largest] < arr[r] [2<1 and …] – FALSE

if largest != i – FALSE

for i in range(n-1, 0, -1) [i=0] – BREAK

**АЛГОРИТМ ЗАВЕРШИВ РОБОТУ**

**Результат:** [5, 10, 16, 27, 28, 33, 35, 76, 95]

1. **Висновок:**

У роботі було переглянуто алгоритм сотування купою. Під час виконання лабораторної роботи було усвідомлено принципи роботи даного алгоритму. Алгоритм сортування купою реалізується за допомогою масива, але його дуже зручно представляти у вигляді бінарного дерева. Основна задача алгоритму полягає в тому, щоб з поданого дерева елементів утворити максимальне дерево, тобто дерево, в якому значення кожного батьківського вузла більше за значення вузлів його дітей. Після цього алгоритм міняє місцями головний елемент дерева та останній за індексом. Тепер останній елемент – найбільший і він вже на своєму місці, тому довжина масиву зменшується і алгоритм знову починає будувати максимальне дерево. Це повторюється до тих пір поки з дерева не будуть зібрані всі елементи.