**Experiment No. 10**

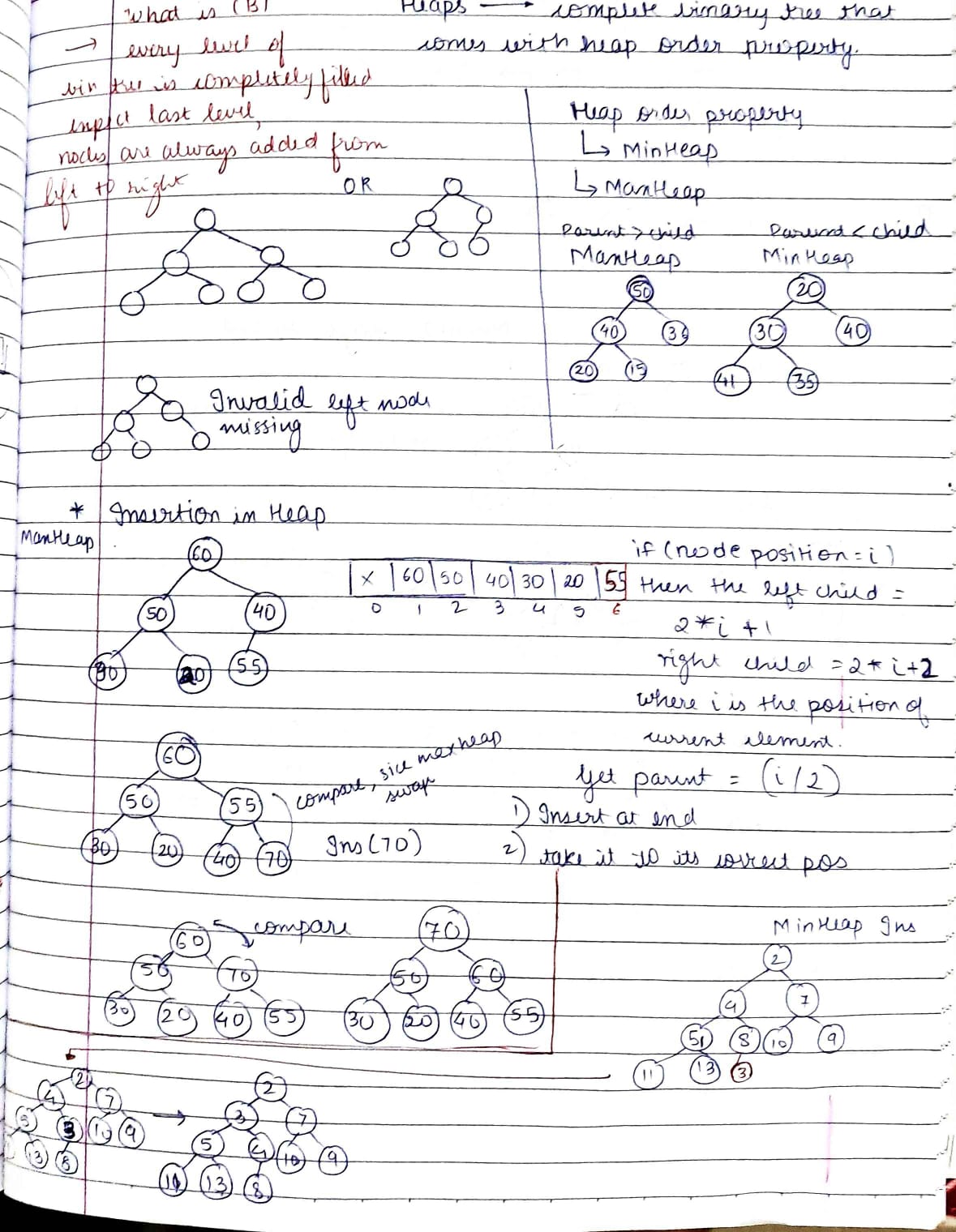
|  |  |
| --- | --- |
| **Name : Manjiri Chavande** | **DSE SY COMPS** |
| **Division: B batch-A** | **UID: 2023301003** |

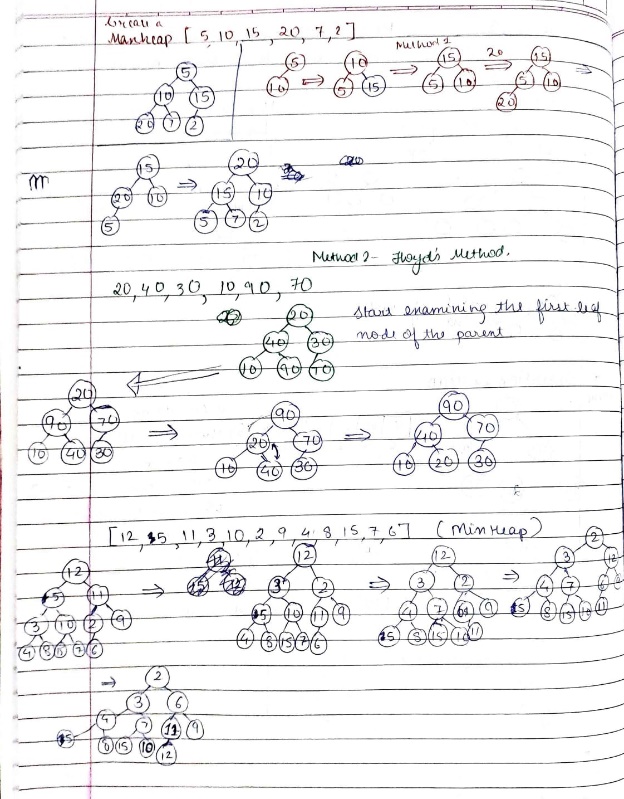
|  |
| --- |
| * **Problem Statement** |

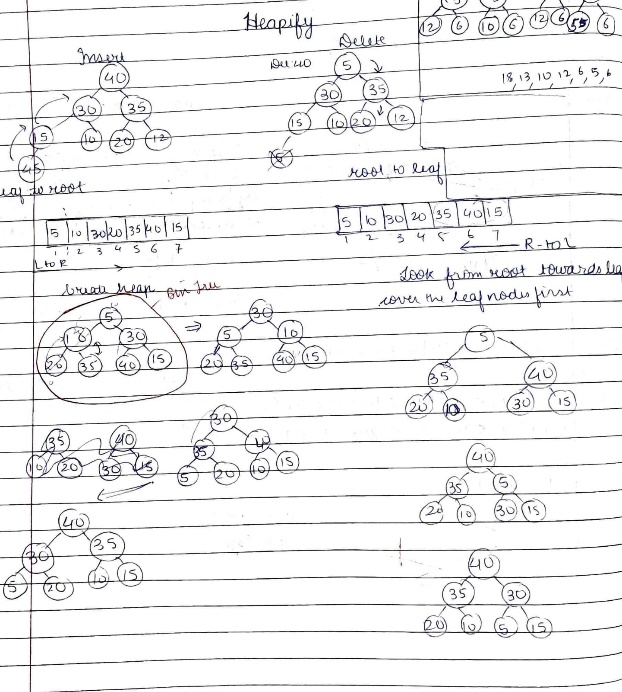
Create a max-heap ADT using array and implement various operations

|  |
| --- |
| * **Theory** |

**Heap Sort:**

Heap Sort is a comparison-based sorting algorithm that uses a binary heap data structure to build a max-heap or min-heap. In the case of Heap Sort, a max-heap is used. The basic idea is to repeatedly extract the maximum element from the heap and rebuild the heap until all the elements are sorted.





|  |
| --- |
| * **Algorithm** |

* **Function:** **initHeap**

**Steps:**

* + 1. Allocate memory for a new MaxHeap structure.
    2. Allocate memory for the array inside the MaxHeap.
    3. Set the initial size to 0.
    4. Return the pointer to the new MaxHeap.

Heap Construction

* **Function:** **constructHeap**

**Steps:**

* + 1. Create a new MaxHeap using **initHeap** with the length of the array.
    2. Copy the elements from the input array to the MaxHeap array.
    3. Set the size of the MaxHeap to the length of the array.
    4. Start from the last non-leaf node and perform heapify for each node.

Heapify Operation

* **Function:** **heapify**

**Steps:**

* + 1. Determine the left and right children indices of the current node.
    2. Find the index of the largest element among the current node and its children.
    3. If the largest element is not the current node, swap them and recursively call **heapify** on the swapped position.

Insertion Operation

* **Function:** **insert**

**Steps:**

* + 1. Check if the heap is full; if so, print an error message and return.
    2. Insert the new value at the end of the heap array.
    3. Swap the value with its parent until the max-heap property is restored.

Display Maximum Element

* **Function:** **peek\_max**

**Steps:**

* + 1. Check if the heap is empty; if so, print an error message and return.
    2. Print the first element in the heap array.

Extract Maximum Element

* **Function:** **extractMax**

**Steps:**

* + 1. Check if the heap is empty; if so, print an error message and return -1.
    2. Save the maximum element.
    3. Replace the first element with the last element.
    4. Reduce the size of the heap.
    5. Call **heapify** to maintain the max-heap property.
    6. Return the saved maximum element.

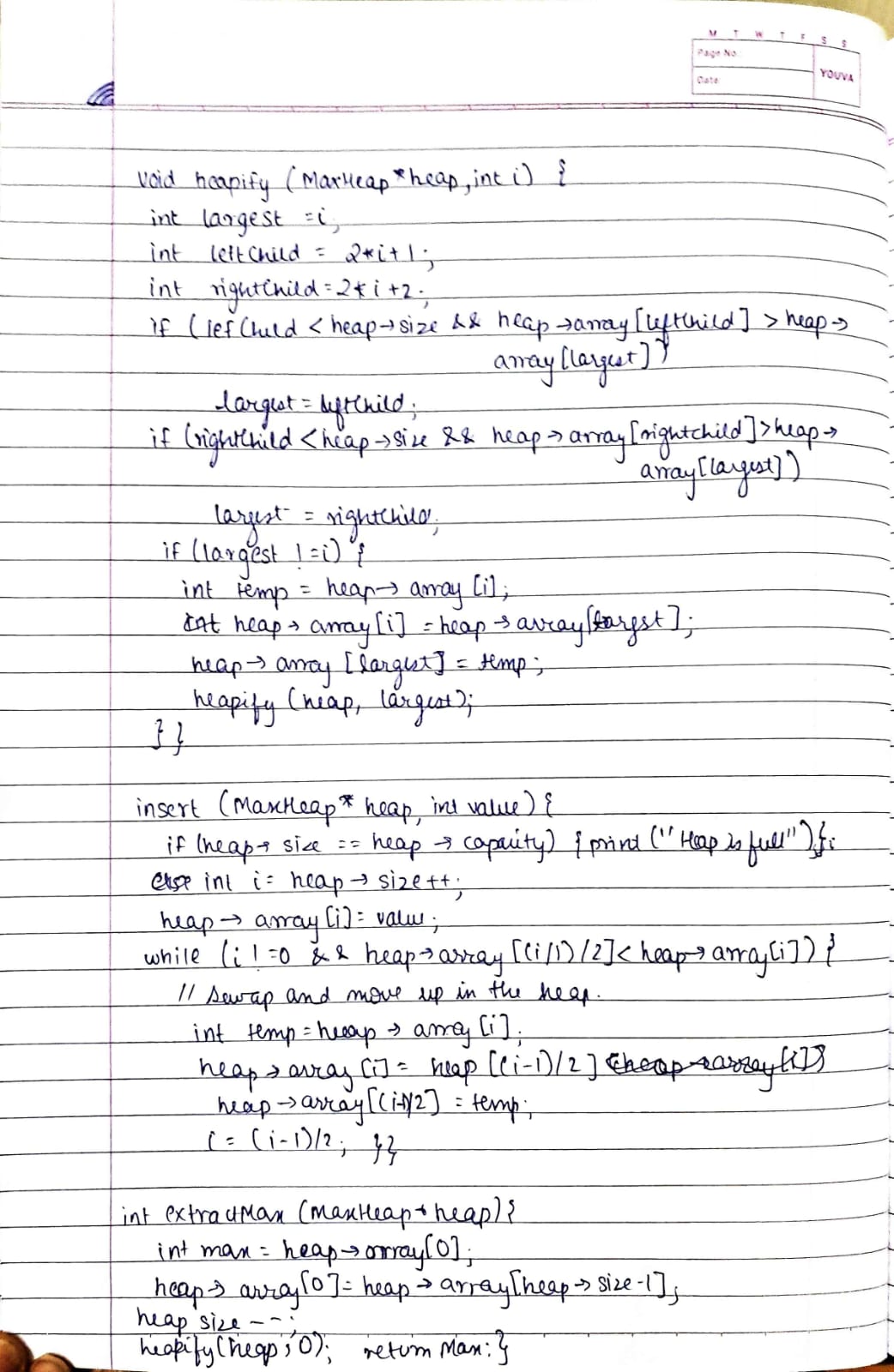
Heap Sort in Ascending Order

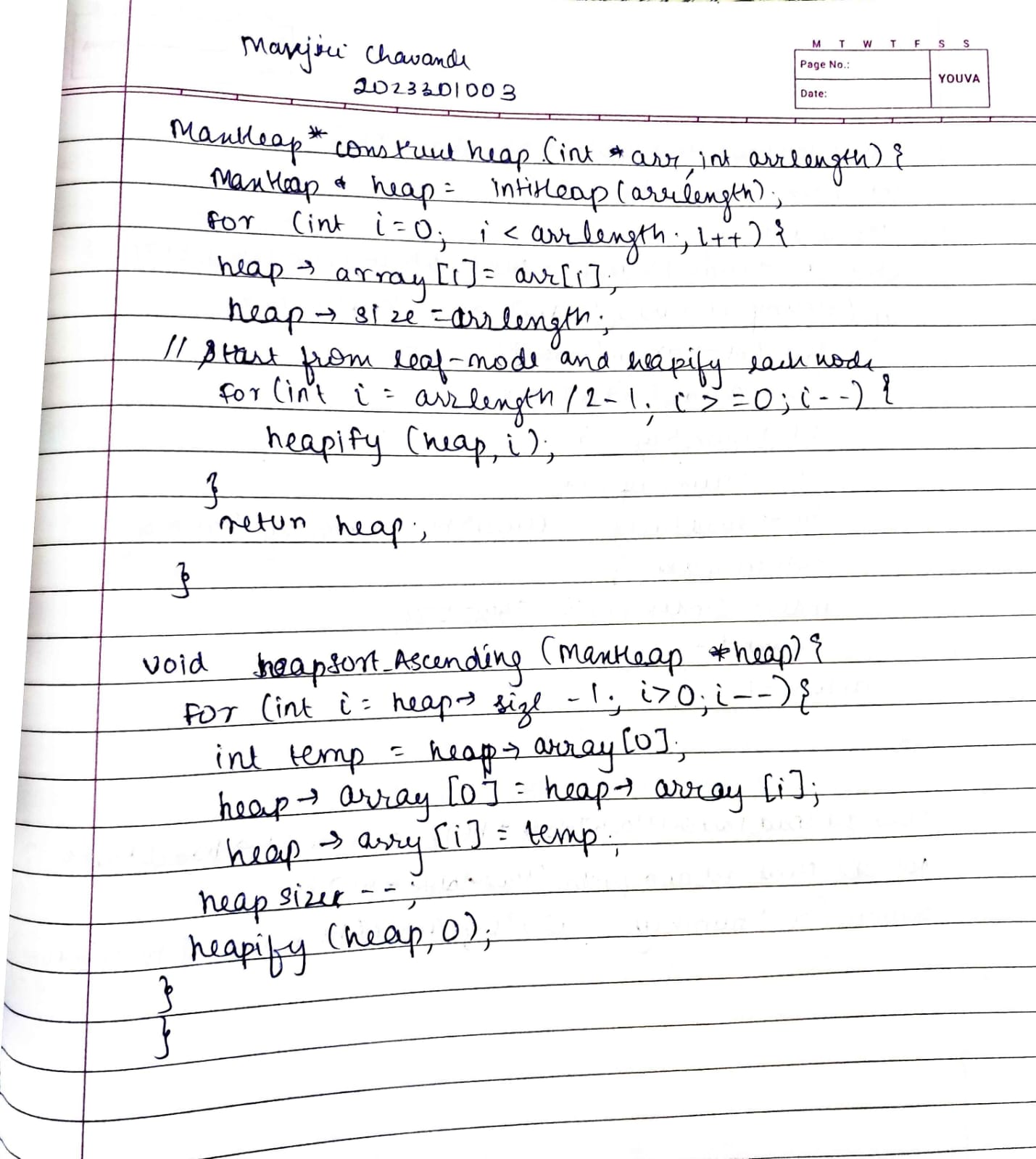
* **Function:** **heapSort\_ascending**

**Steps:**

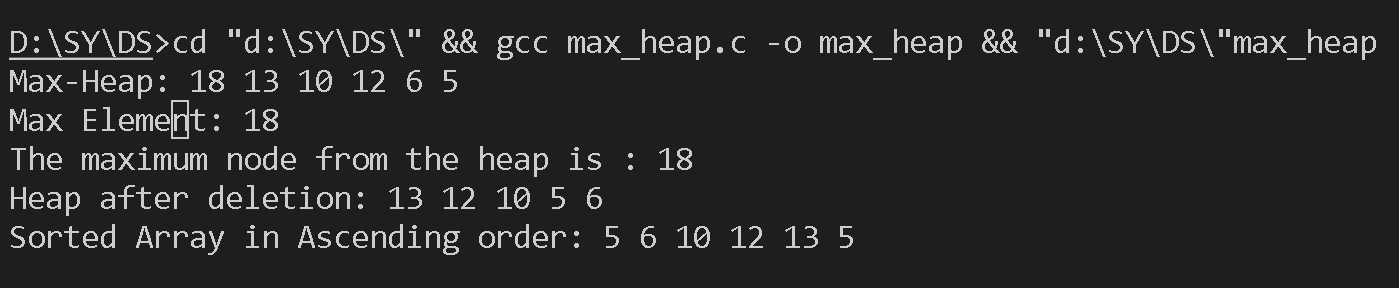
* + 1. Repeat the following steps for each element in the heap (size - 1 times): a. Swap the root (max element) with the last element. b. Reduce the size of the heap. c. Heapify the root to maintain the max-heap property.

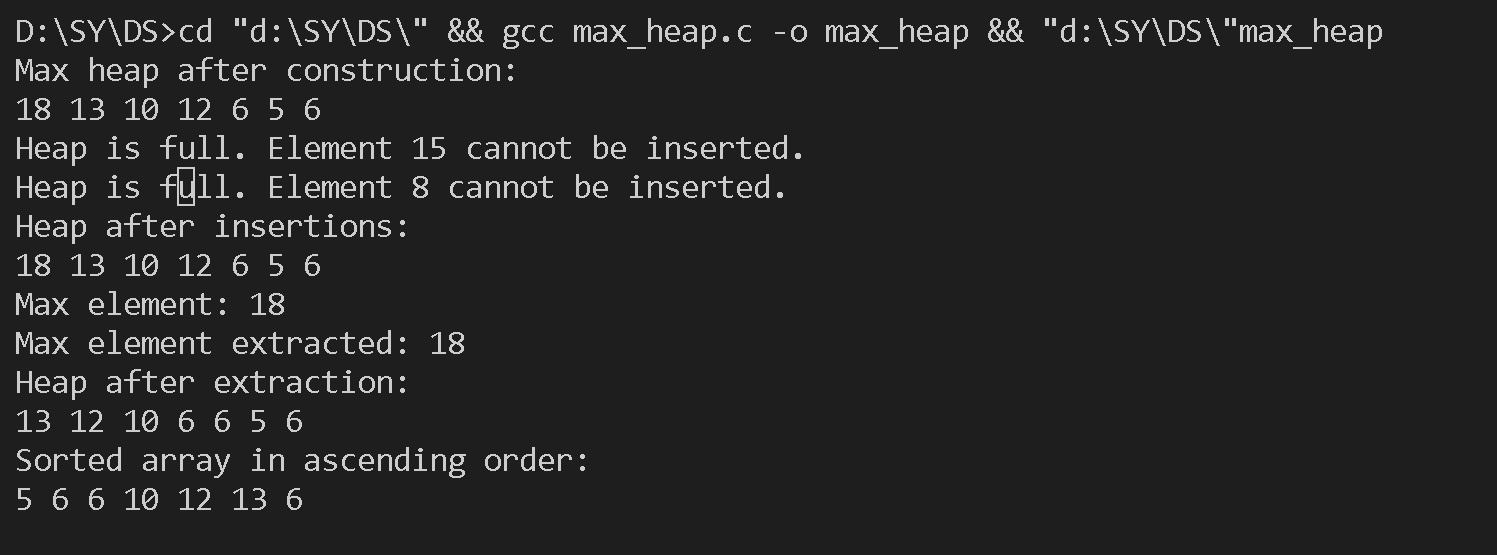
|  |
| --- |
| * **Solution** |



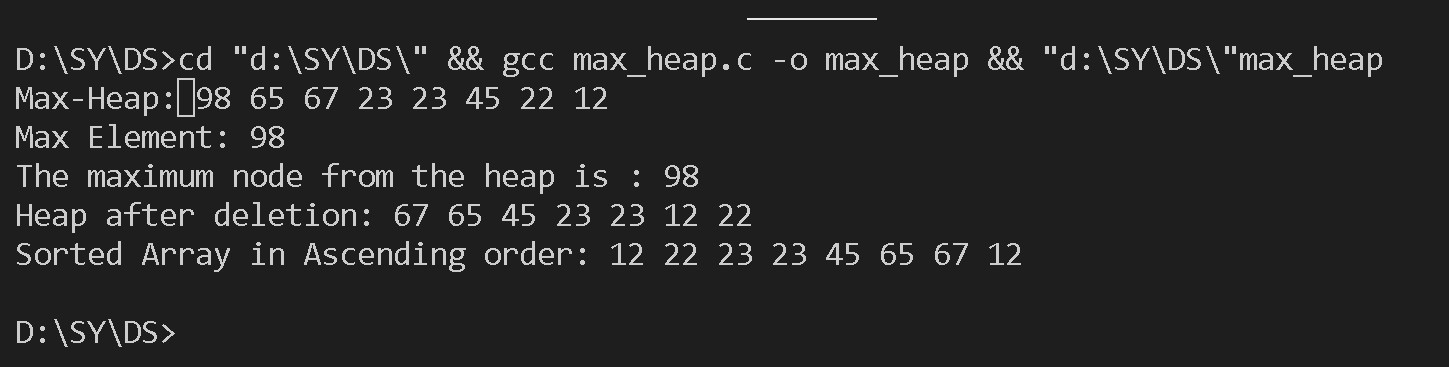


|  |
| --- |
| * **Output** |





|  |
| --- |
| * **Test Cases** |



|  |
| --- |
| * **Conclusion** |

Heap Sort is a reliable choice for sorting large datasets with limited memory constraints, providing a balance between simplicity and efficiency.