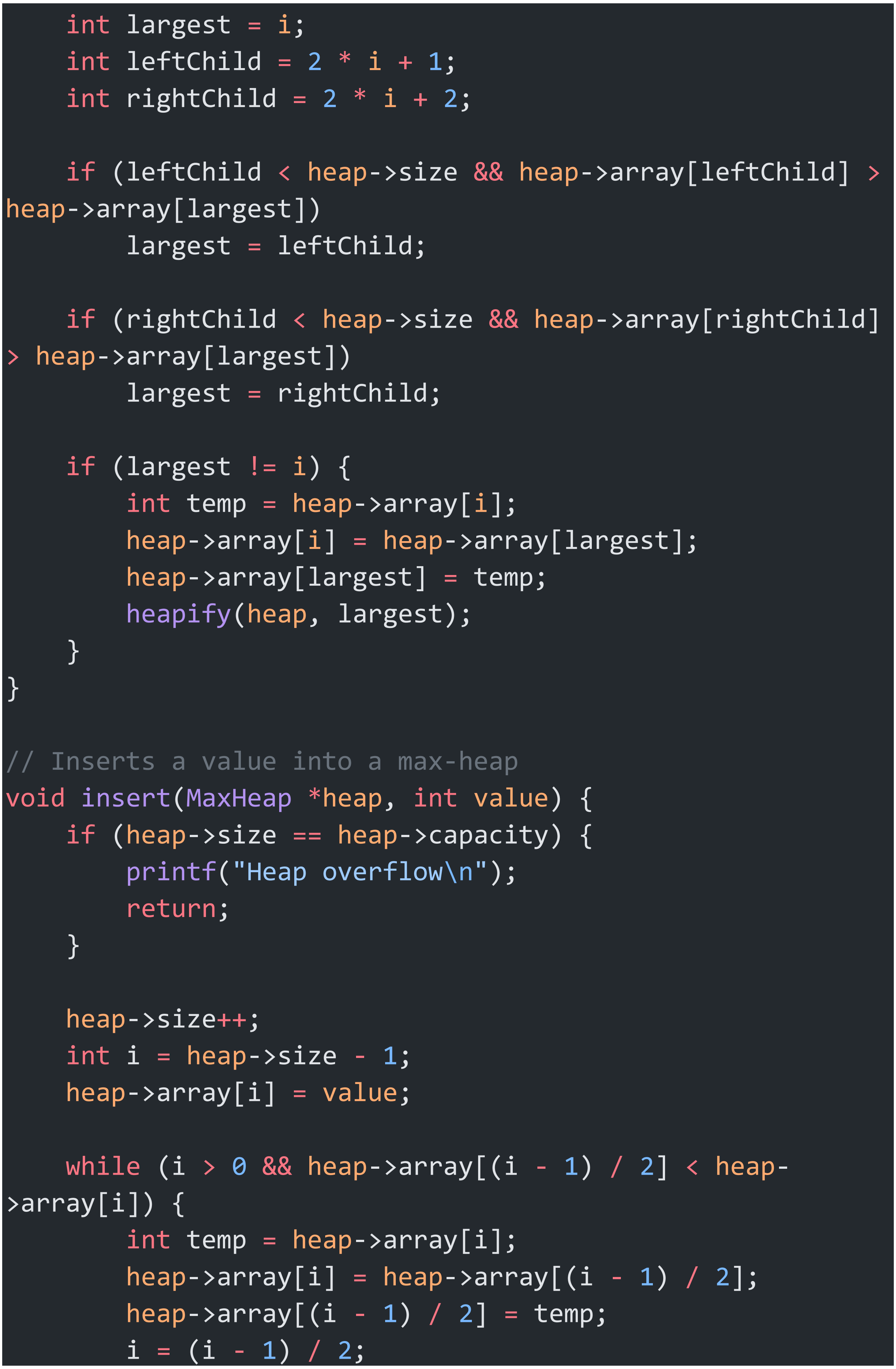
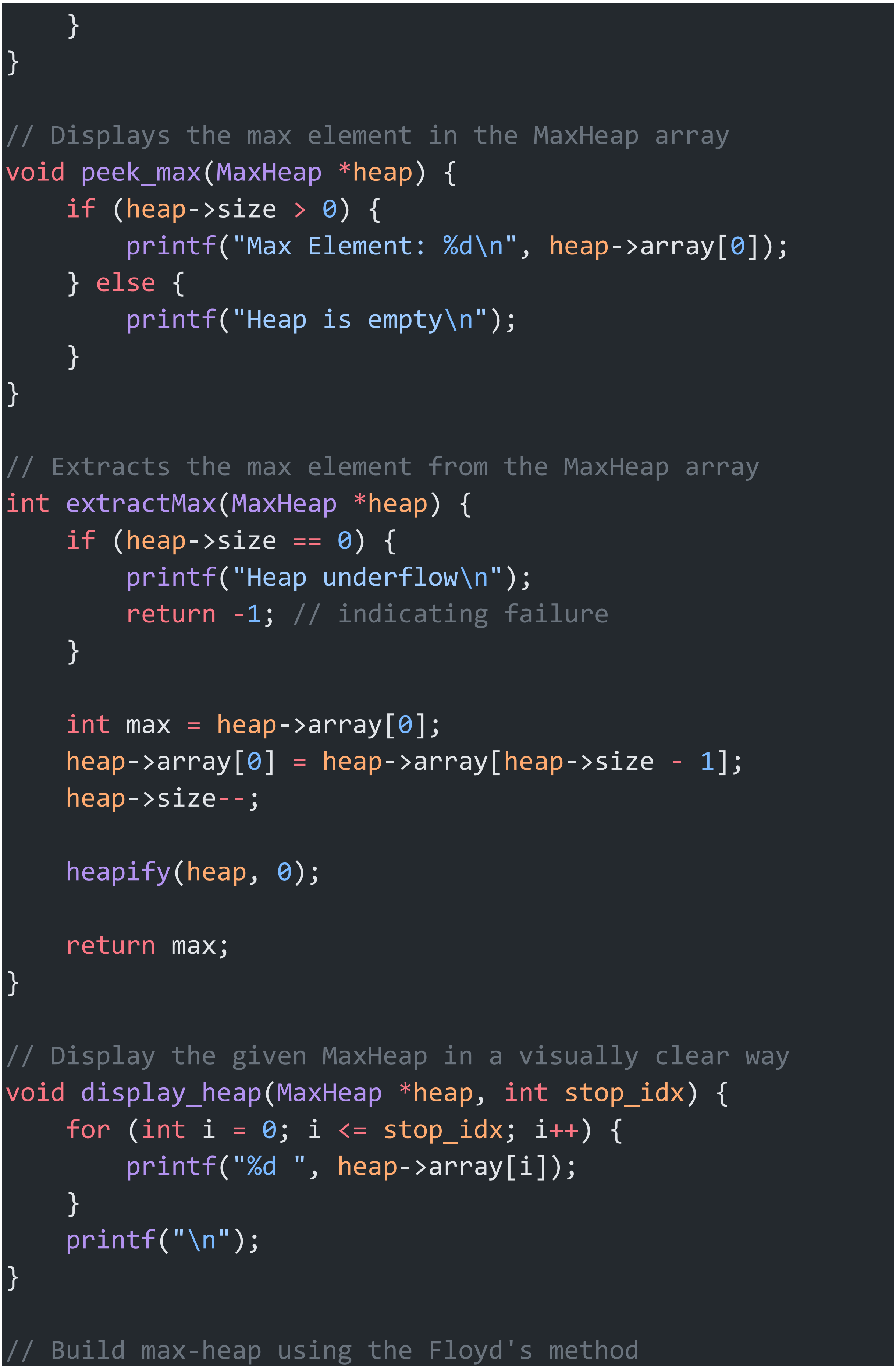
1NAME :- Manish Shashikant Jadhav
UID :- 2023301005.
BRANCH :- Comps -B.   **BRANCH:** B.
EXPERIMENT 10: Create a max-heap ADT using array and implement various operations.
SUBJECT :- DS (DATA STRUCTURES).

CODE :-

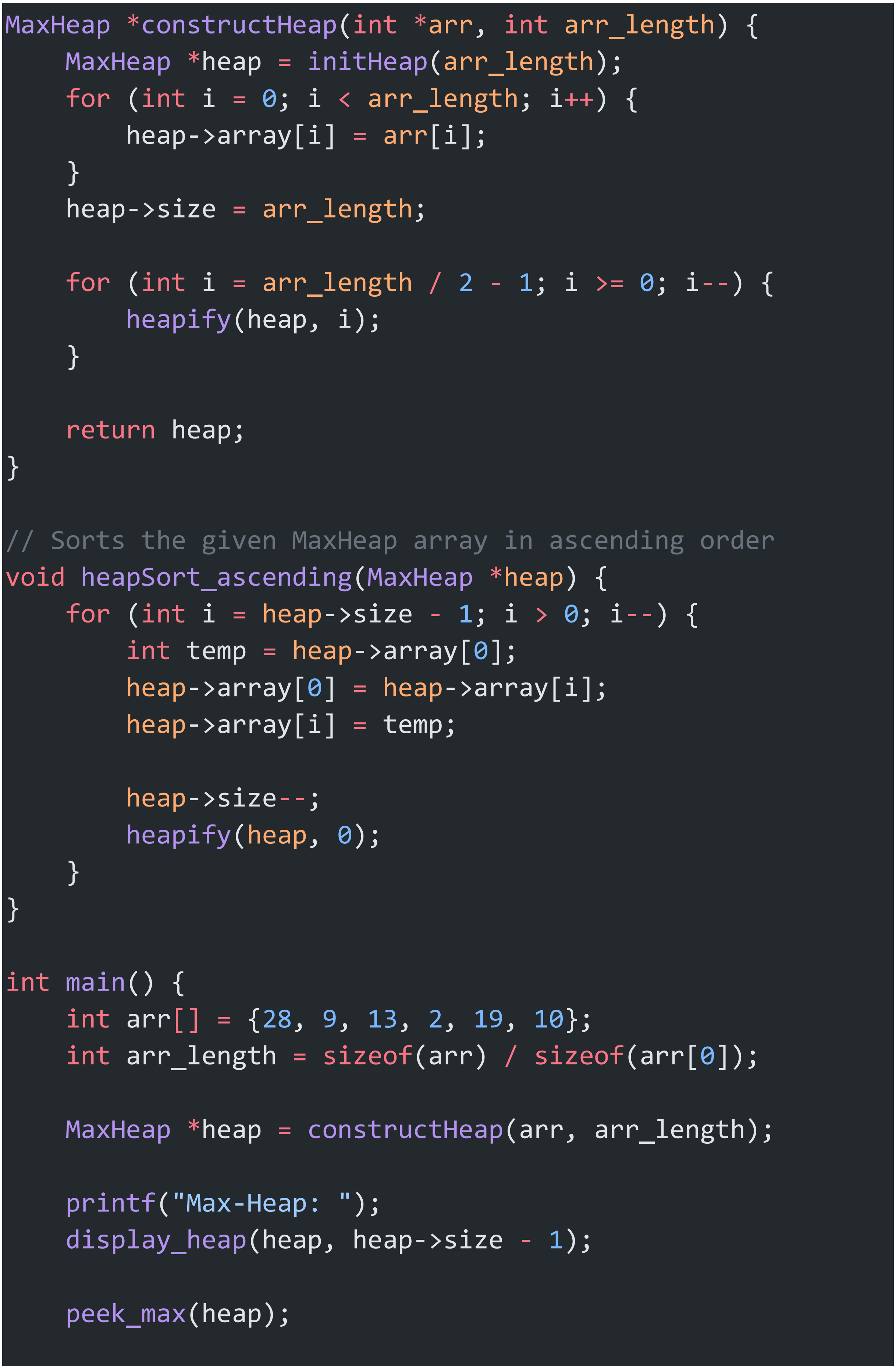
```
// Create a max-heap ADT using array and implement various
operations

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Define MaxHeap ADT
typedef struct {
    int *array;
    int size;
    int capacity;
} MaxHeap;

// Creates an empty max-heap of size 'capacity'
MaxHeap *initHeap(int capacity) {
    MaxHeap *heap = (MaxHeap *)malloc(sizeof(MaxHeap));
    heap->array = (int *)malloc(capacity * sizeof(int));
    heap->size = 0;
    heap->capacity = capacity;
    return heap;
}

// Delete and free the max-heap structure
void destroyHeap(MaxHeap *heap) {
    free(heap->array);
    free(heap);
}

// Restores the heap-order property for max-heap array at
index 'i'
void heapify(MaxHeap *heap, int i) {
```



```
    int largest = i;
    int leftChild = 2 * i + 1;
    int rightChild = 2 * i + 2;

    if (leftChild < heap->size && heap->array[leftChild] >
heap->array[largest])
        largest = leftChild;

    if (rightChild < heap->size && heap->array[rightChild]
> heap->array[largest])
        largest = rightChild;

    if (largest != i) {
        int temp = heap->array[i];
        heap->array[i] = heap->array[largest];
        heap->array[largest] = temp;
        heapify(heap, largest);
    }
}

// Inserts a value into a max-heap
void insert(MaxHeap *heap, int value) {
    if (heap->size == heap->capacity) {
        printf("Heap overflow\n");
        return;
    }

    heap->size++;
    int i = heap->size - 1;
    heap->array[i] = value;

    while (i > 0 && heap->array[(i - 1) / 2] < heap-
>array[i]) {
        int temp = heap->array[i];
        heap->array[i] = heap->array[(i - 1) / 2];
        heap->array[(i - 1) / 2] = temp;
        i = (i - 1) / 2;
```

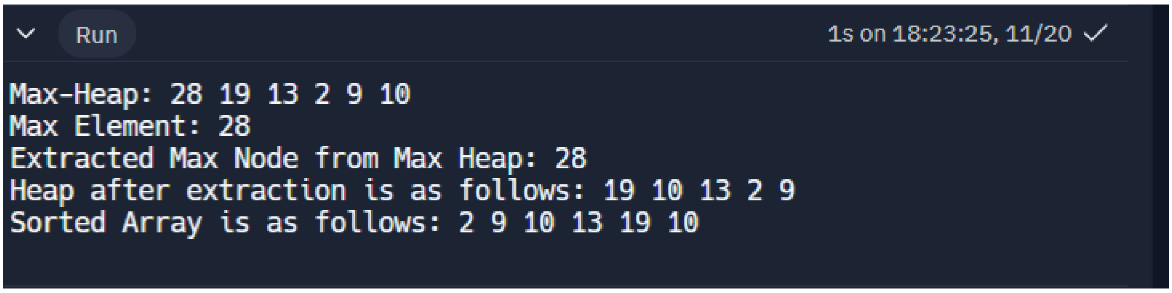


```
    }
}

// Displays the max element in the MaxHeap array
void peek_max(MaxHeap *heap) {
    if (heap->size > 0) {
        printf("Max Element: %d\n", heap->array[0]);
    } else {
        printf("Heap is empty\n");
    }
}

// Extracts the max element from the MaxHeap array
int extractMax(MaxHeap *heap) {
    if (heap->size == 0) {
        printf("Heap underflow\n");
        return -1; // indicating failure
    }

    int max = heap->array[0];
    heap->array[0] = heap->array[heap->size - 1];
    heap->size--;

    heapify(heap, 0);

    return max;
}

// Display the given MaxHeap in a visually clear way
void display_heap(MaxHeap *heap, int stop_idx) {
    for (int i = 0; i <= stop_idx; i++) {
        printf("%d ", heap->array[i]);
    }
    printf("\n");
}

// Build max-heap using the Floyd's method
```



```
MaxHeap *constructHeap(int *arr, int arr_length) {
    MaxHeap *heap = initHeap(arr_length);
    for (int i = 0; i < arr_length; i++) {
        heap->array[i] = arr[i];
    }
    heap->size = arr_length;

    for (int i = arr_length / 2 - 1; i >= 0; i--) {
        heapify(heap, i);
    }

    return heap;
}

// Sorts the given MaxHeap array in ascending order
void heapSort_ascending(MaxHeap *heap) {
    for (int i = heap->size - 1; i > 0; i--) {
        int temp = heap->array[0];
        heap->array[0] = heap->array[i];
        heap->array[i] = temp;

        heap->size--;
        heapify(heap, 0);
    }
}

int main() {
    int arr[] = {28, 9, 13, 2, 19, 10};
    int arr_length = sizeof(arr) / sizeof(arr[0]);

    MaxHeap *heap = constructHeap(arr, arr_length);

    printf("Max-Heap: ");
    display_heap(heap, heap->size - 1);

    peek_max(heap);
    int extractedMax = extractMax(heap);
    printf("Extracted Max Node from Max Heap: %d\n",
extractedMax);

    printf("Heap after extraction is as follows: ");
    display_heap(heap, heap->size - 1);

    heapSort_ascending(heap);

    printf("Sorted Array is as follows: ");
    display_heap(heap, arr_length - 1);

    destroyHeap(heap);

    return 0;
}
```

Output:



```
Max-Heap: 28 19 13 2 9 10
Max Element: 28
Extracted Max Node from Max Heap: 28
Heap after extraction is as follows: 19 10 13 2 9
Sorted Array is as follows: 2 9 10 13 19 10
```

Algorithm:

1. Initialization:

- **Function:** `initHeap(int capacity)`
- **Algorithm:**
  1. Allocate memory for the MaxHeap structure.
  2. Allocate memory for the array based on the specified capacity.
  3. Set `size` to 0.
  4. Set `capacity` to the specified capacity.
  5. Return the initialized MaxHeap structure.

2. Heapify:

- **Function:** `heapify(MaxHeap *heap, int i)`
- **Algorithm:**

  1. Determine the left and right child indices of node `i`.
  2. Find the index of the largest element among `i`, its left child, and its right child.
  3. If the largest element is not `i`, swap the elements at indices `i` and `largest`.
  4. Recursively apply heapify to the affected sub-tree.

3. Insertion:

- **Function:** `insert(MaxHeap *heap, int value)`
- **Algorithm:**

  1. Check for heap overflow.
  2. Increment the `size` of the heap.
  3. Add the new value at the end of the array.
  4. Perform a heap-up operation to restore the heap property.

4. Peek Max:

- **Function:** `peek_max(MaxHeap *heap)`
- **Algorithm:**

  1. Check if the heap is not empty.
  2. Display the maximum element at the root of the heap.

5. Extract Max:

- **Function:** `extractMax(MaxHeap *heap)`
- **Algorithm:**

  1. Check for heap underflow.
  2. Save the maximum element (root) to be returned later.
  3. Replace the root with the last element in the array.
  4. Decrement the `size` of the heap.
  5. Perform a heapify operation on the root to restore the heap property.
  6. Return the extracted maximum element.

6. Display Heap:

- **Function:** `display_heap(MaxHeap *heap, int stop_idx)`

- **Algorithm:**

  1. Iterate through the array up to the specified index (`stop_idx`).

  2. Display each element in the array.

7. Construct Heap:

- **Function:** `constructHeap(int *arr, int arr_length)`

- **Algorithm:**

  1. Create an empty heap with a capacity equal to the length of the input array.

  2. Copy the elements of the input array to the heap array.

  3. Set the size of the heap to the length of the array.

  4. Perform a bottom-up heapify for each non-leaf node.

8. Heap Sort (Ascending):

- Function: `heapSort_ascending(MaxHeap *heap)`

- Algorithm:

  1. Iterate from the last element to the first in the array.

  2. Swap the root (maximum element) with the current element.

  3. Decrement the size of the heap.

  4. Perform a heapify operation on the root to restore the heap property.

9. Main Function:

- **Function:** `main()`

- **Algorithm:**

  1. Create an array.

  2. Construct a max-heap using the array.

  3. Display the max-heap.

  4. Peek at the maximum element.

  5. Extract the maximum element.

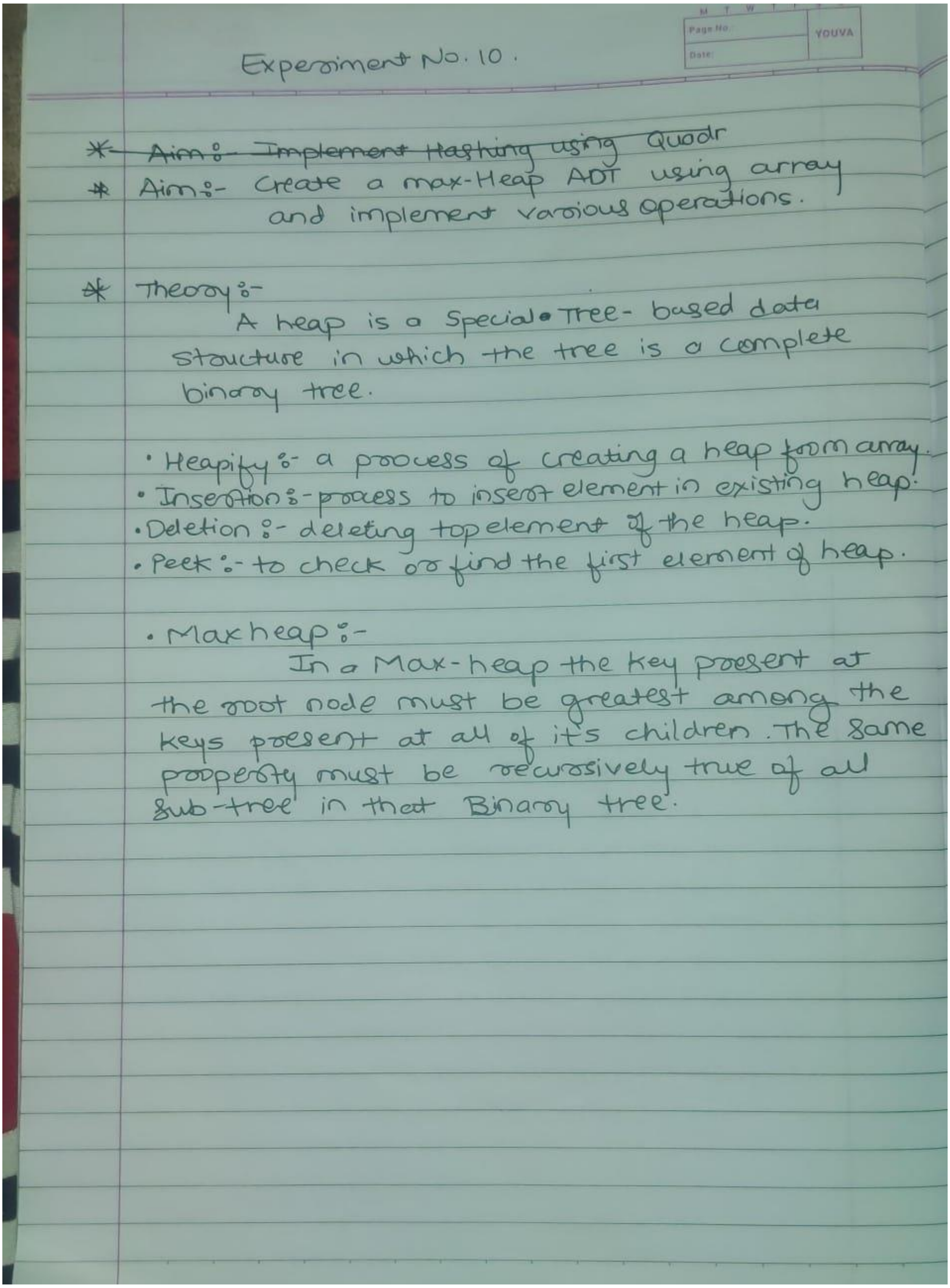
  6. Display the heap after extraction.

  7. Sort the heap in ascending order using heap sort.

  8. Display the sorted array.

  9. Free allocated memory.

free the allocated memory using `destroyHeap` to avoid memory leaks.



Experiment No. 10.

* ~~Aim:- Implement Hashing using~~ Quodr

* Aim:- Create a max-Heap ADT using array and implement various operations.

* Theory:-

A heap is a Special Tree- based data structure in which the tree is a complete binary tree.

- Heapify:- a process of creating a heap from array.
- Insertion:- process to insert element in existing heap.
- Deletion:- deleting top element of the heap.
- Peek:- to check or find the first element of heap.

- Max heap:-

In a Max-heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true of all sub-tree in that Binary tree.

Conclusion:

Hence, by completing this experiment I came to know about creating a max-heap ADT using array and implement various operations.