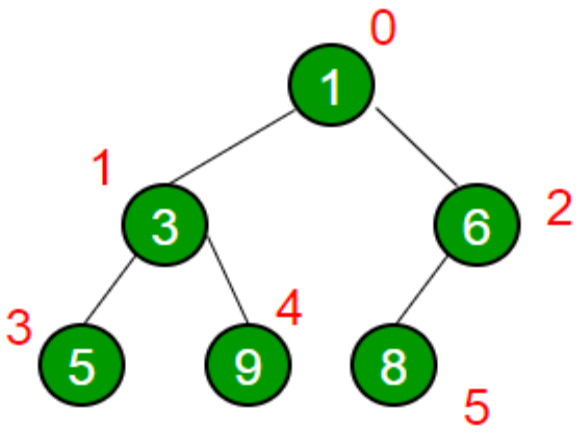
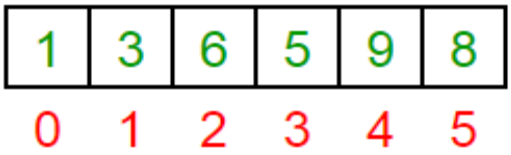
Name	Maureen Miranda
UID no.	2022300060
Experiment No.	10

AIM:	To implement heaps
Program 1	
PROBLEM STATEMENT :	Create a min-heap ADT using array and implement various operations
THEORY:	A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at the root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap. The root element will be at Arr[0]. The below table shows indices of other nodes for the ith node, i.e., Arr[i]: Arr[(i-1)/2] Returns the parent node Arr[(2*i)+1] Returns the left child node Arr[(2*i)+2] Returns the right child node





Operations on Heap:
Below are some standard operations on min heap:

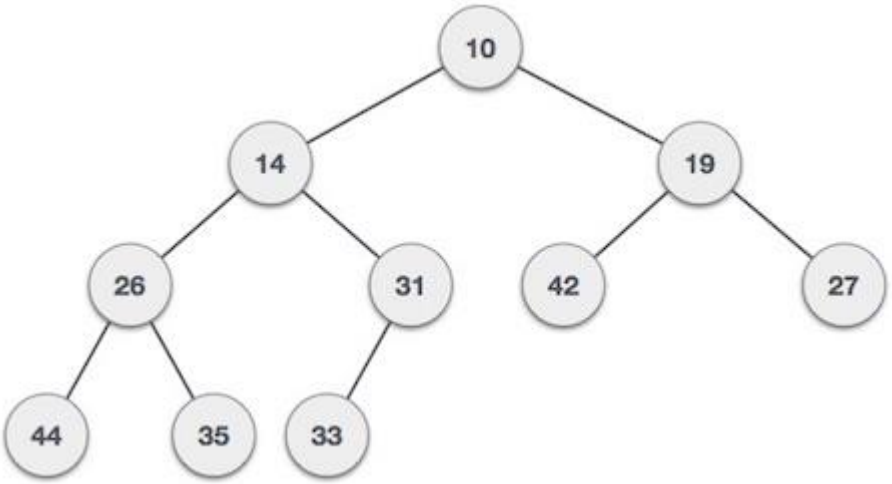
- getMin(): It returns the root element of Min Heap. The time Complexity of this operation is O(1). In case of a maxheap it would be getMax().

- extractMin(): Removes the minimum element from MinHeap. The time Complexity of this Operation is O(log N) as this operation needs to maintain the heap property (by calling heapify()) after removing the root.

- insert(): Inserting a new key takes O(log N) time. We add a new key at the end of the tree. If the new key is greater than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

- delete(): Deleting a key also takes O(log N) time. We replace the key to be deleted with the minimum infinite by calling decreaseKey(). After decreaseKey(), the minus infinite value must reach root, so we call

extractMin() to remove the key.



Heap data structure is a complete binary tree that satisfies the heap property, where any given node is

always greater than its child node/s and the key of the root node is the largest among all other nodes. This property is also called max heap property.
always smaller than the child node/s and the key of the root node is the smallest among all other nodes. This property is also called min heap property.

Algorithm:

MinHeap Structure Definition:

- Define a structure MinHeap with members array (pointer to the heap array), size (current number of elements in the heap), and capacity (maximum capacity of the heap).

- Heap Initialization:

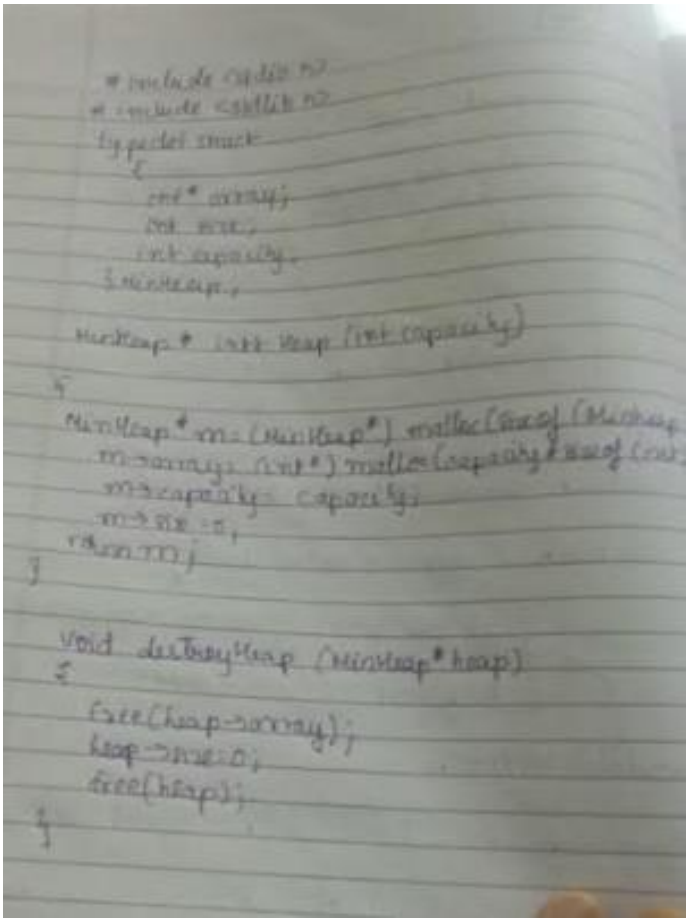
  Create a function initHeap that initializes and returns an empty MinHeap

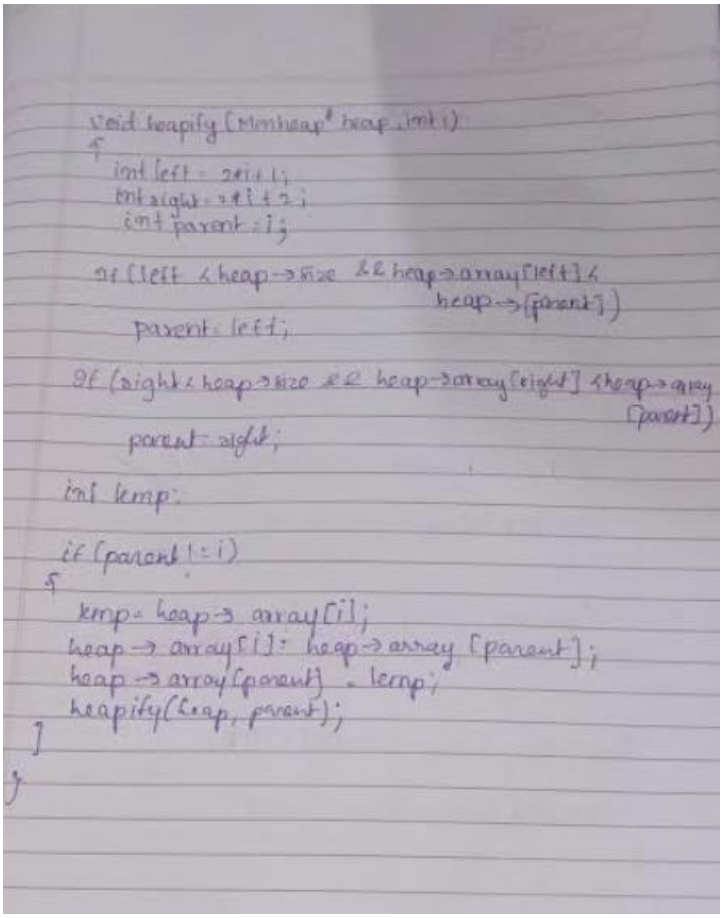
	of a given capacity. Allocate memory for the heap array, set size to 0, and store the capacity. • Heap Destruction: Create a function destroyHeap to free the allocated memory for the heap. • Heapify Operation: • Define a heapify function to restore the heap property at a given index. • Compare the element at the current index with its left and right children. • Swap with the smallest child if needed and recursively heapify the affected subtree. • Insertion Operation: • Create an insert function to insert a new element into the heap. • Place the element at the end of the array and bubble it up by comparing with its parent until the heap property is restored. • Peek Minimum Operation: • Implement a peek_min function to display the minimum element in the heap (root of the heap array). • Extract Minimum Operation: • Implement extractMin to remove and return the minimum element from the heap. • Replace the root with the last element, reduce size, and heapify the root to maintain the heap property. • Display Heap: • Define a display_heap function to visually display the heap in a clear way. • Traverse the heap array level-wise, printing each element. • Heap Construction: • Implement constructHeap using Floyd's method to build a heap from an array.

- Heap Sorting:

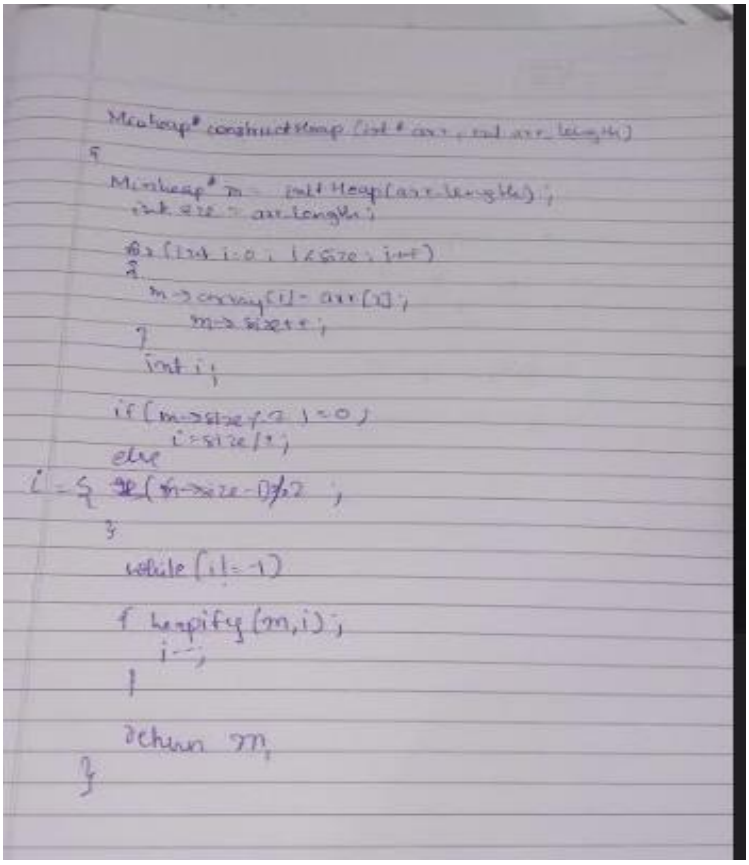
- Create a heapSort_descending function to sort the heap in descending order.
- Swap the root with the last element iteratively and heapify the reduced heap.

Program :

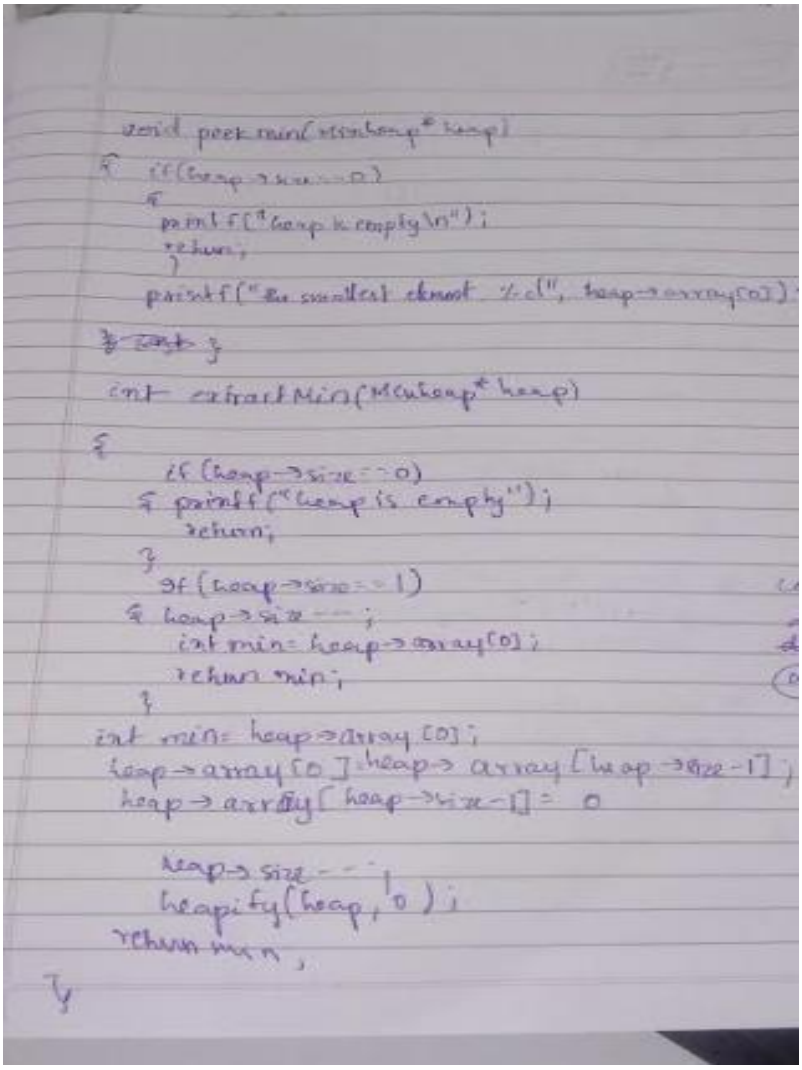




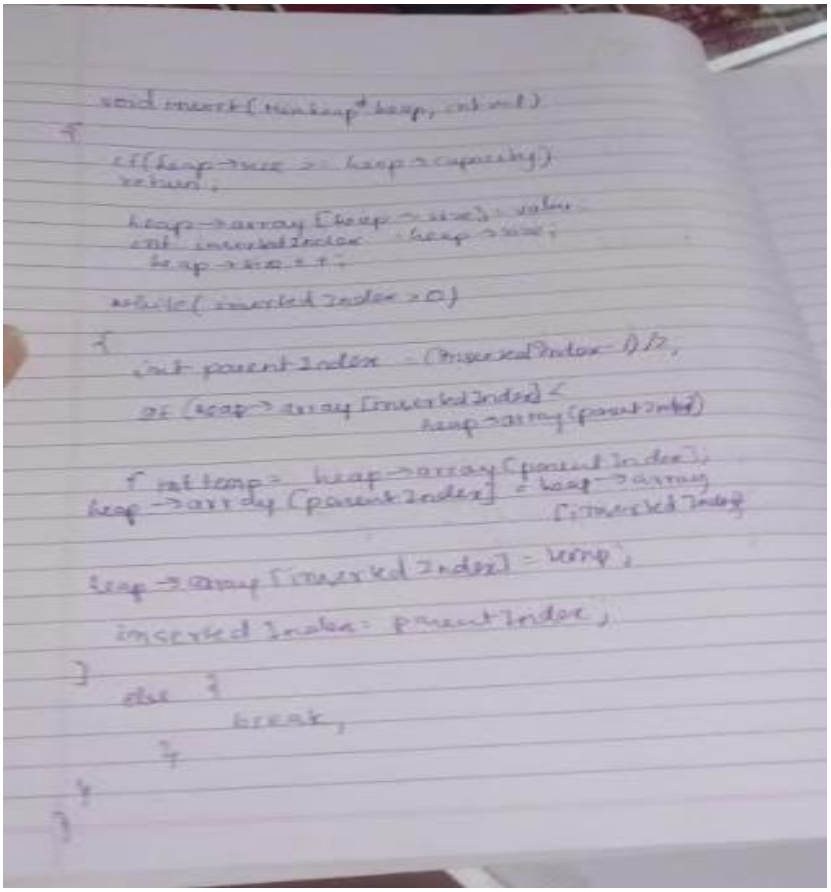
```
void heapify (Minheap* heap, int i)
{
  int left = 2*i+1;
  int right = 2*i+2;
  int parent = i;

  if (left < heap->size && heap->array[left] <
                                heap->[parent])
    parent = left;

  if (right < heap->size && heap->array[right] < heap->array
                                [parent])
    parent = right;

  int temp;

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

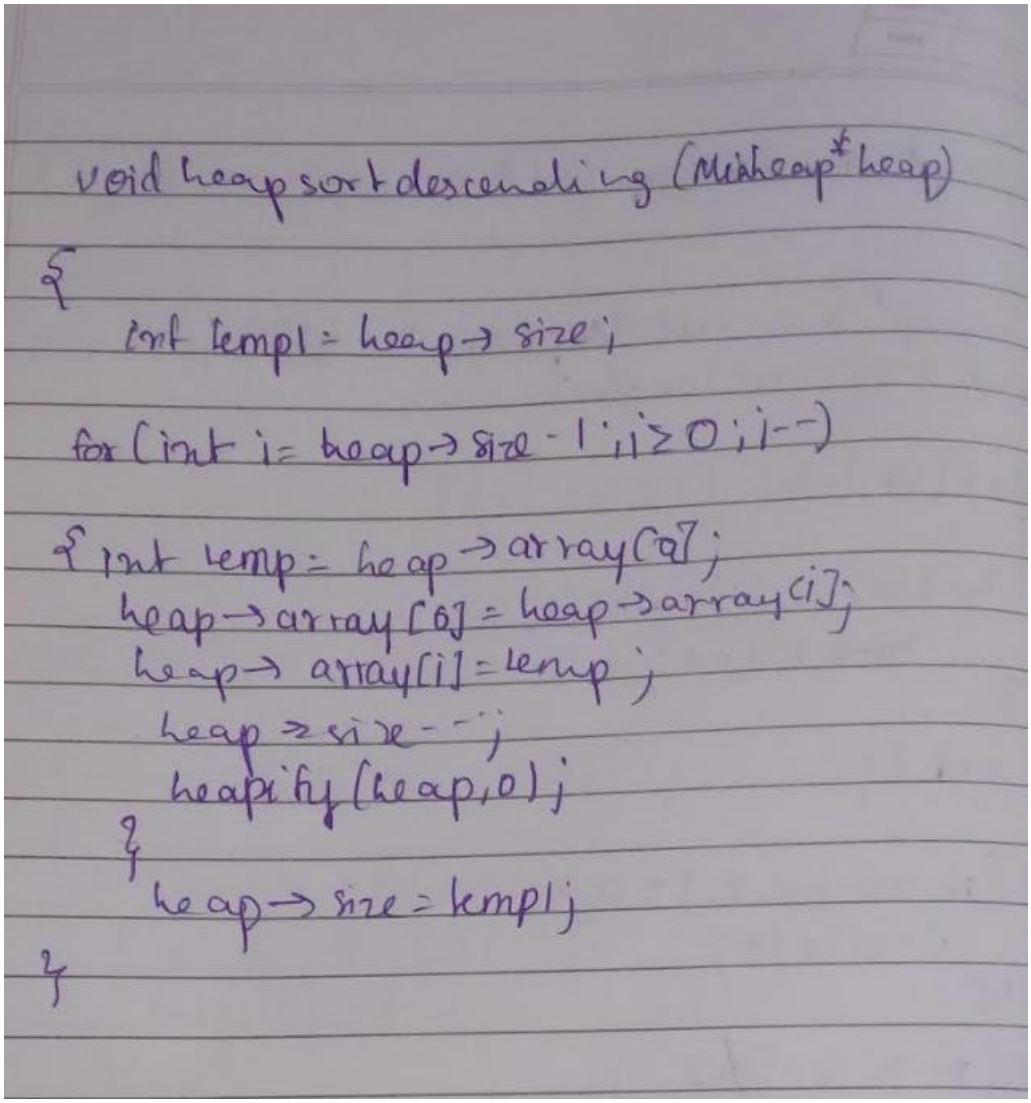


```
Minheap* constructHeap (int* arr, int arr_length)
{
Minheap* m = initHeap(arr_length);
int size = arr_length;

for(int i=0 ; i<size ; i++)
{
  m->array[i] = arr[i];
     m->size++;
}
 int i;

if(m->size%2 != 0)
    i=size/2;
else
i = size(m->size-0)/2 ;
}
 while(i!=-1)

 { heapify(m,i);
    i--;
 }

 return m;
}
```

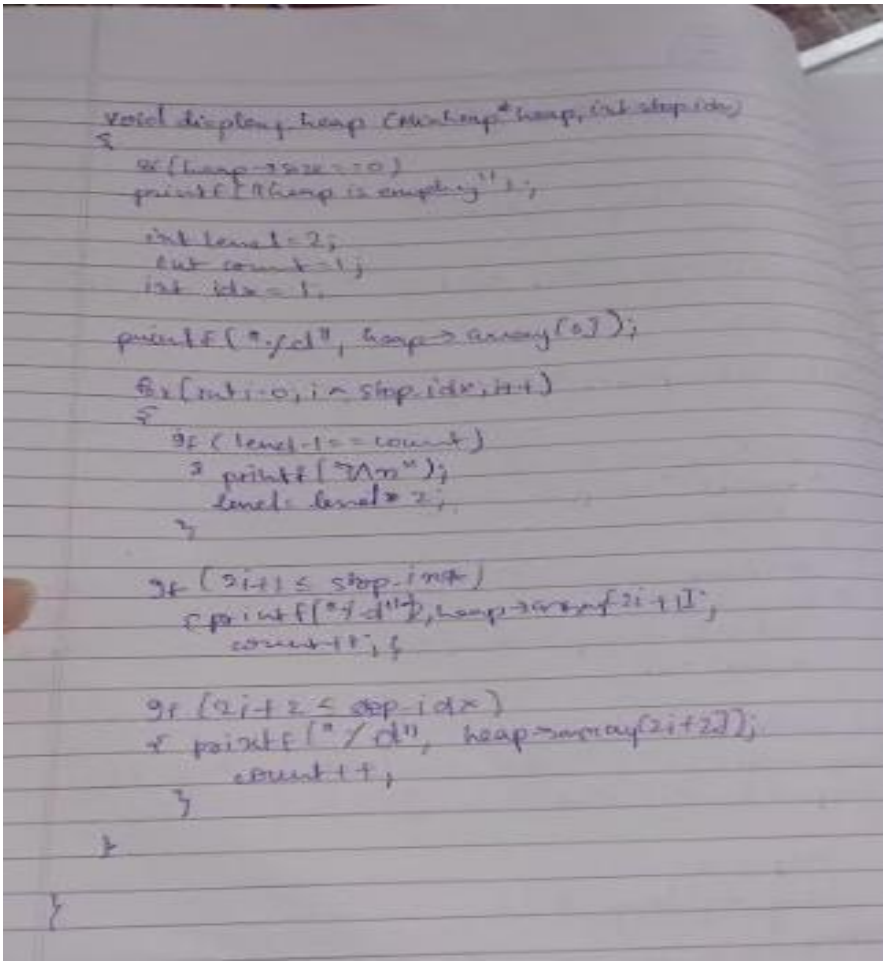


```
void peekmin(struct minheap* heap)
{
  if(heap->size==0)
  {
    printf("heap is empty\n");
    return;
  }
  printf("the smallest element = %d", heap->array[0]);
}

int extractMin(struct minheap* heap)
{
    if (heap->size==0)
  { printf("heap is empty");
    return;
  }
   if(heap->size==1)
  { heap->size--;
    int min= heap->array[0];
    return min;
  }
int min= heap->array[0];
heap->array[0]=heap->array[heap->size-1];
heap->array[heap->size-1]= 0

   heap->size--;
   heapify(heap, 0);
return min;
}
```





```
void heapsortdescending (MinHeap* heap)
{
    int temp1 = heap->size;
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);
}
  heap->size = temp1;
}
```



```
void display_heap (MaxHeap* heap, int stop_idx)
{
  if (heap->size == 0)
  printf("Heap is empty");

  int level = 2;
  int count = 1;
  int idx = 1;

printf("%d", heap->array[0]);

for (int i = 0; i < stop_idx; i++)
{
  if (level-1 == count)
  {
    printf("\n");
    level = level * 2;
  }

  if (2*i+1 <= stop_idx)
    printf(" %d", heap->array[2*i+1]);
    count++;
  }

  if (2*i+2 <= stop_idx)
  {
    printf(" %d", heap->array[2*i+2]);
    count++;
  }
}

}
```

PROGRAM:	#include <stdio.h>
	#include <stdlib.h>
	#include <stdbool.h>
	#include<math.h>
	// Define MinHeap ADT
	typedef struct {
	  int* array;
	  int size;
	  int capacity;
	} MinHeap;
	
	/// @brief Creates an empty min-heap of size 'capacity'
	/// @param capacity - size of heap
	/// @return Pointer to MinHeap
	MinHeap* initHeap(int capacity)
	{
	MinHeap *m = (MinHeap*)malloc(sizeof(MinHeap));
	m->array=(int*)malloc(capacity*sizeof(int));
	m->capacity=capacity;
	m->size=0;
	return m;
	}
	
	// Delete and free the min-heap structure
	void destroyHeap(MinHeap* heap)
	{
	  free(heap->array);
	  heap->size=0;
	  free(heap);
	}
	
	/// @brief This restores the heap-order property for min-heap array at index 'i'
	/// @param heap
	/// @param i - index of the min-heap array, which might potentialy be breaking the heap order
	void heapify(MinHeap* heap, int i)
	{

```
  int left = 2*i+1;
   int right = 2*i+2;
   int parent =i;
   if(left < heap->size&&heap->array[left]<heap->array[parent])
   {
      parent=left;
   }
    if(right<heap->size&&heap->array[right]<heap->array[parent])
   {

      parent=right;
   }
  int temp;

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

/// @brief This inserts a value into a min-heap
/// @param heap
/// @param value
void insert(MinHeap* heap, int value)
{

   if(heap->size>=heap->capacity)
   return;

   heap->array[heap->size]=value;
   int insertedIndex = heap->size;
   heap->size++;
   while (insertedIndex > 0)
   {
      int parentIndex = (insertedIndex - 1) / 2;
      if (heap->array[insertedIndex] < heap->array[parentIndex]) {
         int temp = heap->array[parentIndex];
```

```
            heap->array[parentIndex] = heap->array[insertedIndex];
            heap->array[insertedIndex] = temp;
            insertedIndex = parentIndex;
        } else {
            break;
        }
    }

}




// displays the min element in the heap array
void peek_min(MinHeap* heap)
{
    if(heap->size==0)
    {
        printf("heap is empty\n");
        return;
    }
    printf("%d",heap->array[0]);
}

// extracts the min element from the heap array
int extractMin(MinHeap* heap)
{

    if(heap->size==0)
    {
    printf("heap is empty");
    return;
    }
    if(heap->size==1)
    {
        heap->size--;
        int min= heap->array[0];
        return min;
    }
 int min = heap->array[0];
```

```
 heap->array[0]=heap->array[heap->size-1];
 heap->array[heap->size-1]=0;
 heap->size--;
 heapify(heap,0);

return min;

}

/// @brief - Display the given MinHeap in a visually clear way.
/// @param heap
/// @param stop_idx - This index in the Heap array corresponds to the last item
of the heap

void display_heap(MinHeap* heap, int stop_idx)
{

if(heap->size==0)
{
printf(“heap is empty”);
return;
}
int level=2;
int count=1;
int indx=1;
printf("\n\nDISPLAYING THE HEAP\n");
printf("%d",heap->array[0]);

  for(int i=0;i<stop_idx;i++)
  {
     if(level-1==count)
     {
     printf("\n");
     level=level*2;
     }



      if(2*i+1<=stop_idx)
      {printf("%d ",heap->array[2*i+1]);
```

```
    count++; }
     if (2*i+2<=stop_idx)
     {printf("%d ",heap->array[2*i+2]);
     count++;
     }

  }

}



/// @brief Build min-heap using the Floyd's method. This method should call
initHeap
/// @param heap
MinHeap *constructHeap(int *arr, int arr_length)
{
MinHeap* m = initHeap(arr_length);
int size=arr_length;
for(int i =0;i<size;i++)
{
 m->array[i]=arr[i];
 m->size++;
}
int i;
if(m->size%2!=0)
{
 i = size/2;
}
else{
 i=((m->size-1)/2);
}
while(i!=-1)
{
 heapify(m,i);

 i--;
}

return m;
```

	}
	
	// Sorts the given MinHeap array in descending order
	// Post running this function, heap->array should contain the elements in the sorted order
	// NOTE: This function should not use any additional data structures
	void heapSort_descending(MinHeap* heap)
	{
	  int temp1=heap->size;
	
	  // Extract elements from the heap and place at the end of the array
	  for (int i = heap->size-1 ; i >=0; i--) {
	  // Swap the root (max element) with the last element
	  int temp = heap->array[0];
	  heap->array[0] = heap->array[i];
	  heap->array[i] = temp;
	heap->size--;
	  // Heapify the remaining elements
	  heapify(heap, 0);
	  }
	  heap->size=temp1;
	}

RESULT:

```
22
23  int main()
24  {
25      int a []={2,5,6,3,10,9,4,7};
26      MinHeap*m = constructHeap(a,8);
27
28      insert(m,3);
29       display_heap(m,7);
30     printf("\nextracted element= %d\n",extractMin(m));
31     display_heap(m,6);
32       printf("\n");
33   peek_min(m);
34   heapSort_descending(m);
35     printf("\n");
36
37   display_heap(m,6);
38    destroyHeap(m);
39      return 0;
40  }
41
```

```
DISPLAYING THE HEAP
2
3 4
5 10 9 6
7
extracted element= 2


DISPLAYING THE HEAP
3
5 4
7 10 9 6

the smallest element= 3


DISPLAYING THE HEAP
10
9 7
6 5 4 3
```

Conclusion :	In summary, the C program implements a MinHeap data structure with functions for initialization, insertion, extraction of the minimum element, and sorting elements in descending order. The main function demonstrates the usage of these operations with a sample array. The program provides a practical illustration of how MinHeaps work and their application in sorting algorithms.