NAME: ADWAIT S PURAO
UID: 2021300101
EXP NO. : 9
AIM: To perform Heapify(Top Down approach) , Insertion and Heapsort functions on a Heap.

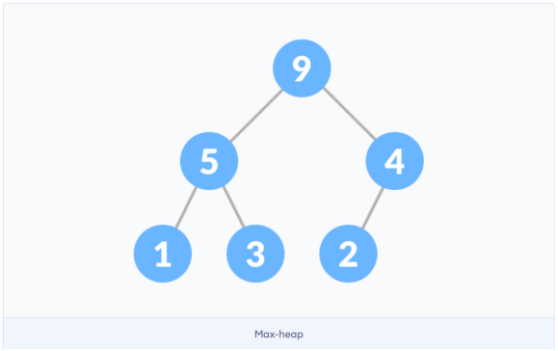
THEORY:

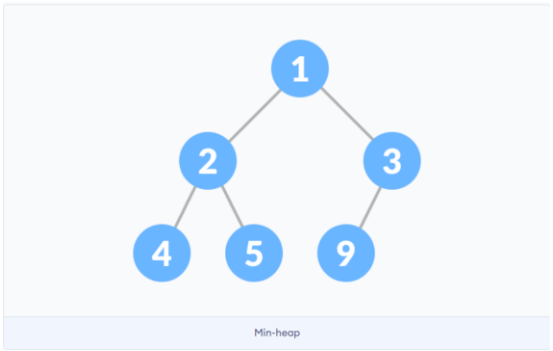
Heap Data Structure

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.

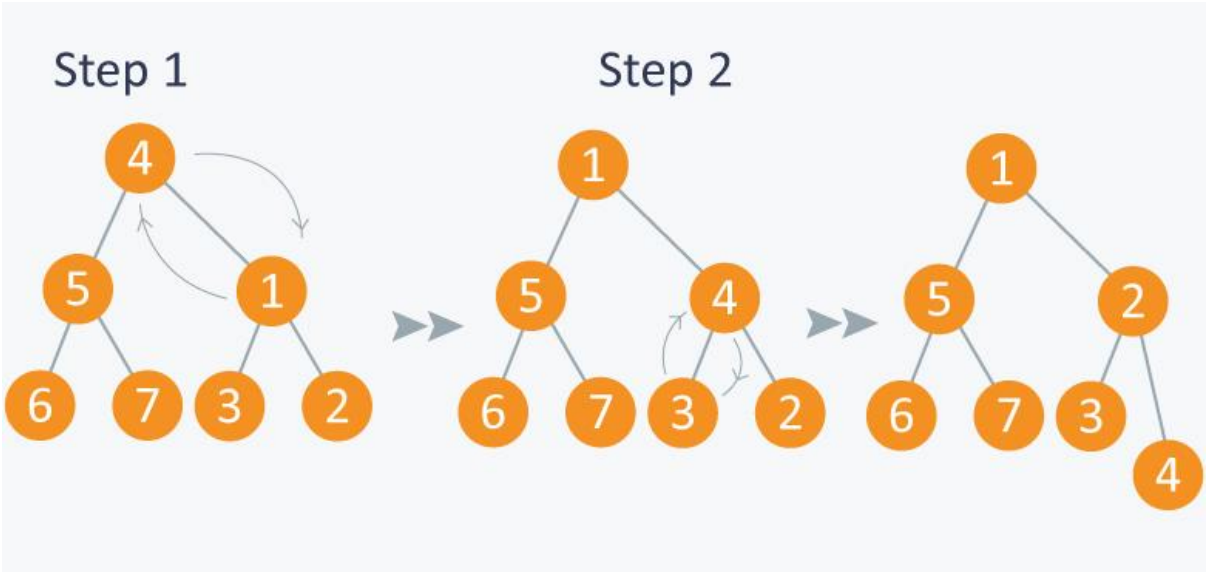




Min Heap:

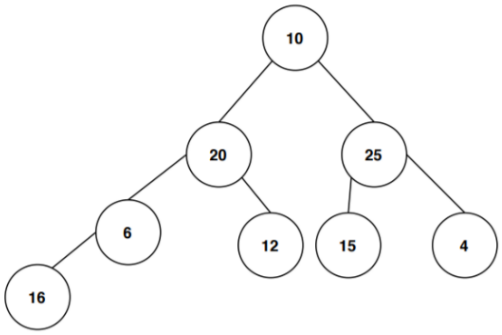
In this type of heap, the value of parent node will always be less than or equal to the value of child node across the tree and the node with lowest value will be the root node of tree.

Suppose you have elements stored in array {4, 5, 1, 6, 7, 3, 2}. As you can see in the diagram below, the element at index 1 is violating the property of min -heap, so performing min_heapify(Arr, 1) will maintain the min-heap.

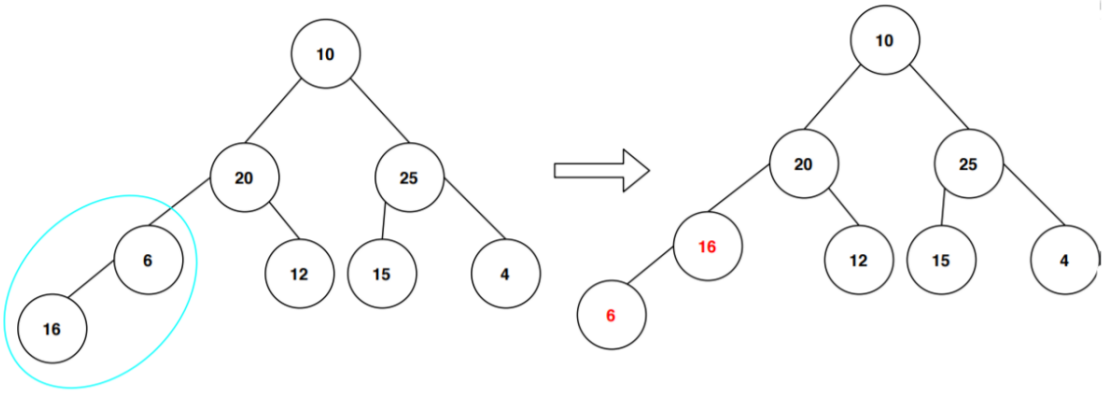


Max-Heapify

Lets take an input array $B = [10, 20, 25, 6, 12, 15, 4, 16]$. The first step is to create a binary tree from the array:

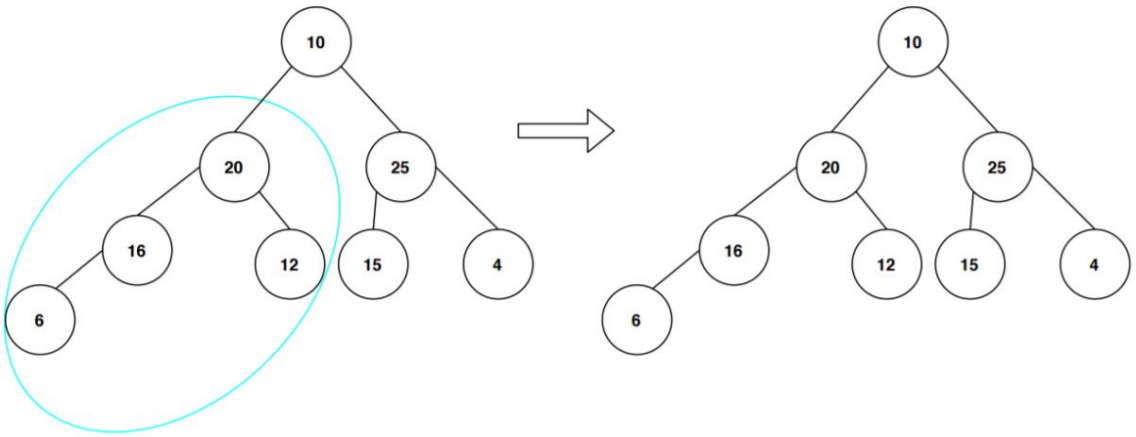


Now we'll take a subtree at the lowest level and start checking whether it follows the max-heap property or not:

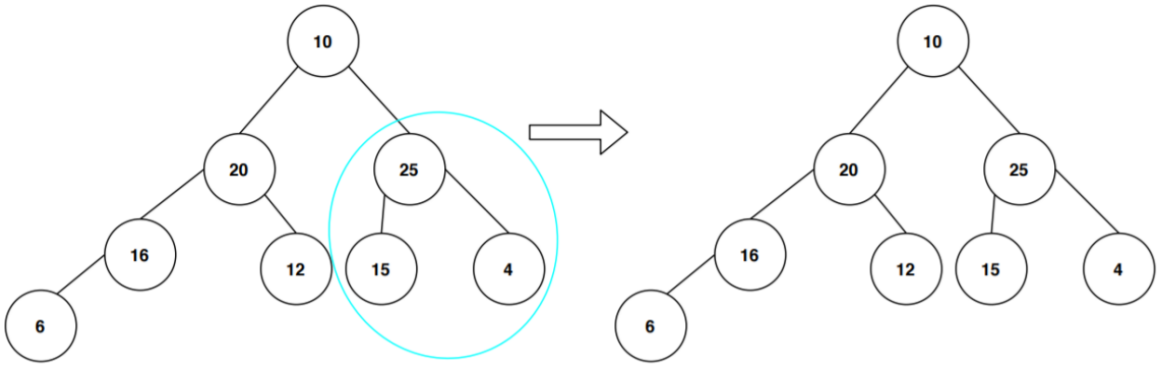


As we can see, the subtree doesn't follow the max-heap property. Here, the parent node should contain a greater value than its children node. So in order to make sure that the tree follows the max-heap property, we swap the key values between the children node and the parent node.

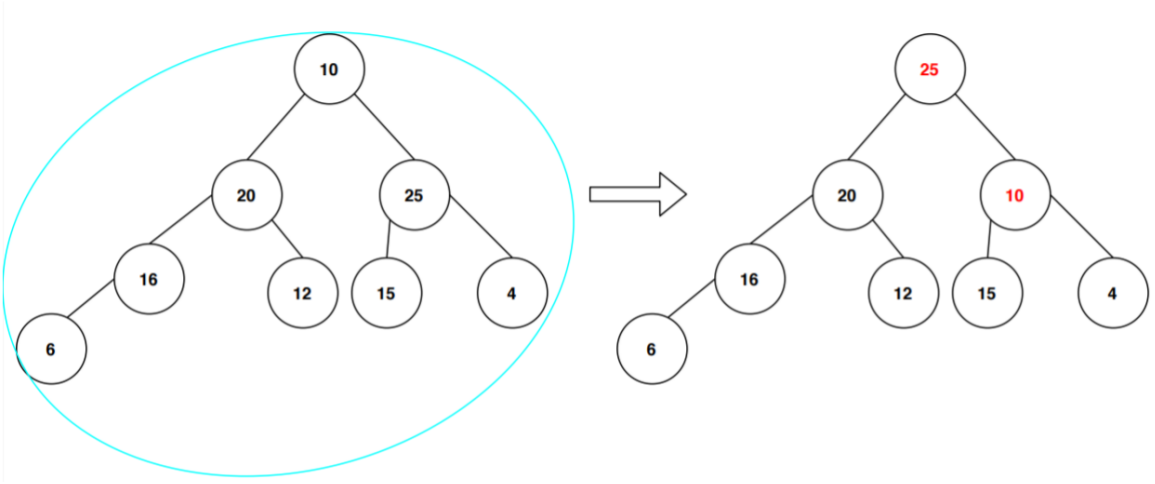
Let's continue and examine all the subtrees from the lowest level to the top level:



This subtree follows the max-heap property, and we don't need to change anything here. Next, we look at the right side branches:

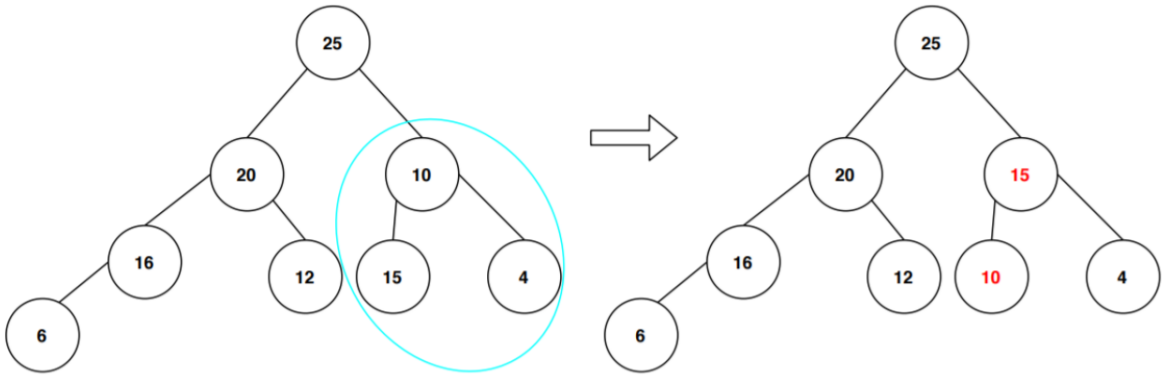


Again the subtree follows the max-heap property. Let's continue this process:

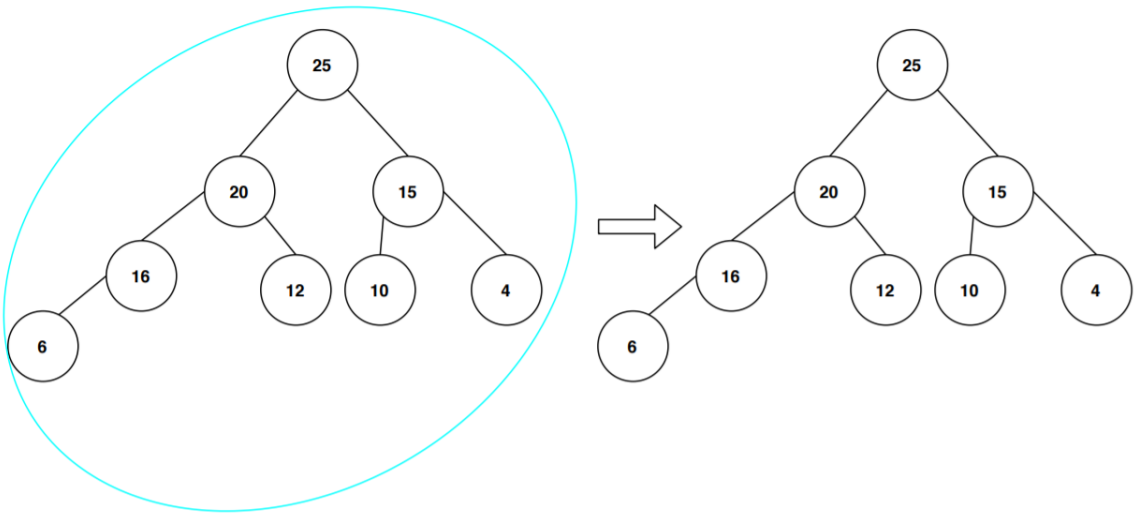


Here again, we can see that the key value of the root node is not the largest among all the nodes in the tree. Hence we swapped the key values of the root node with the key value of its right children node to match with the max-heap property.

Now, after the swap, we need to check the right subtree from the root node in order to see whether it follows the max-heap property or not:



Finally, we've to check the whole tree in order to see if it satisfies the max-heapify property, and then we'll get our final max-heap tree:



Algorithm:

Build a min-heap from the given input array.

After this, the smallest node is stored at the root of the heap. Replace it with the last node of the heap until the size of the heap gets 1.

Heapify the root of the tree.

Repeat the above steps while the size of the heap is greater than 1.

Dry Run:

Step 1

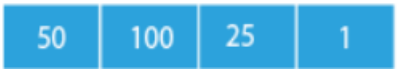
First, construct the heap from the given array & convert it into min heap.

Step 2

Delete the root (1) from the min heap. To delete this node, we have to swap it with the last node i.e. (50). After this, we have to heapify it again.

Step 3

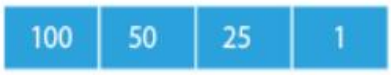
In the next step, we have to delete the root element (25). To delete this, we have to swap it with the last node i.e. (100) & again perform heapify after swapping.



Step 4

In the next step, delete the root node (50). To delete this, swap it with the last node i.e.(100).

Now, heap has left only 1 element, After deleting it heap will be empty and the array is completely sorted:



ALGORITHM:

- Struct Heap to maintain the heap
  - Declare a pointer to an array
  - Size variable to store the total size of array
  - Usize variable to store the Used size of array
- Function Swap pass integer pointers a and b
  - Store a in a temporary variable
  - Store the b in a
  - Store temp in b
- Function int isFull pass Struct Heap
  - If usize == size -1
    - Return 1
  - Return 0
- Function void Display pass struct Heap
  - Iterate a loop from i=1 until i<=usize
    - Print arr[i]
- Function void TDHeapify pass struct Heap
  - If usize==1
    - Return
  - Else
    - Iterate a loop from i=2 until i=usize

Store I in curr_ind
While arr[curr_ind]<arr[curr_ind/2] and curr_ind>1
Swap arr[curr_ind] and arr[curr_ind/2]
Make curr_ind=curr_ind/2

- Function Insertion pass struct Heap and data to be inserted

If heap array isFull

Print Heap is full

Else

Increment usize

Store data in arr[usize]

Heapify the array

Display the array

- Function int HeapSort pass struct Heap and a arrayvsorted to store the sorted elements

Repeat until usize!=1
Set count to 0
Sorted[count]←arr[1]
Swap arr[1] and arr[usize]
Decrement usize
Heapify the tree
Display the heap

Increment count

Sorted[count]←arr[1]

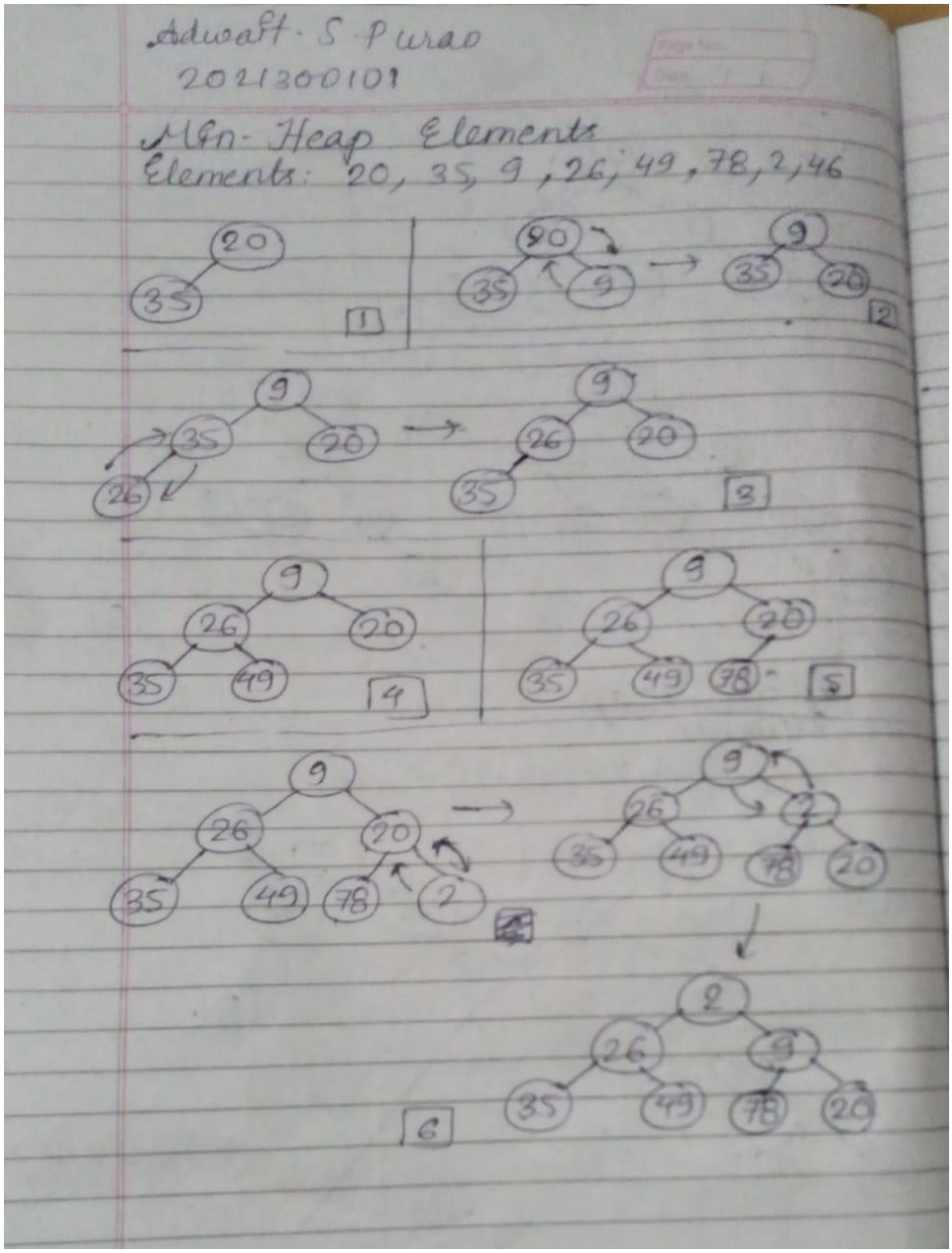
Return count

- Main function

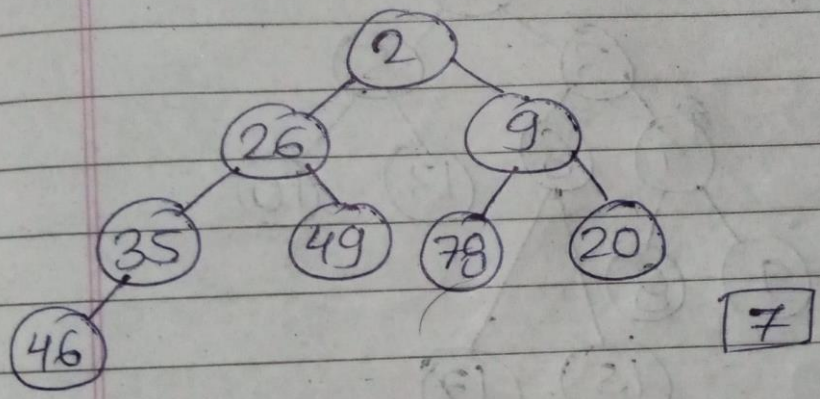
Allocate memory for struct Heap
Take input n size of heap array
Set size to n+1
Set usize to 0
Allocate memory for heap array
Initialize all elements of heap array to 0
Declare an array sorted to store the sorted elements
Repeat until flag ==0
Take the choice of user
If choice =1
Take input the element to be inserted
Insert in heap
Else if choice =2

HeapSort the heap array
Display the sorted array
Else if choice=3
Set flag=1
Else
Print Invalid input

PROBLEM SOLVING ON CONCEPT:



Adwait. S. Purao
2021300101



7

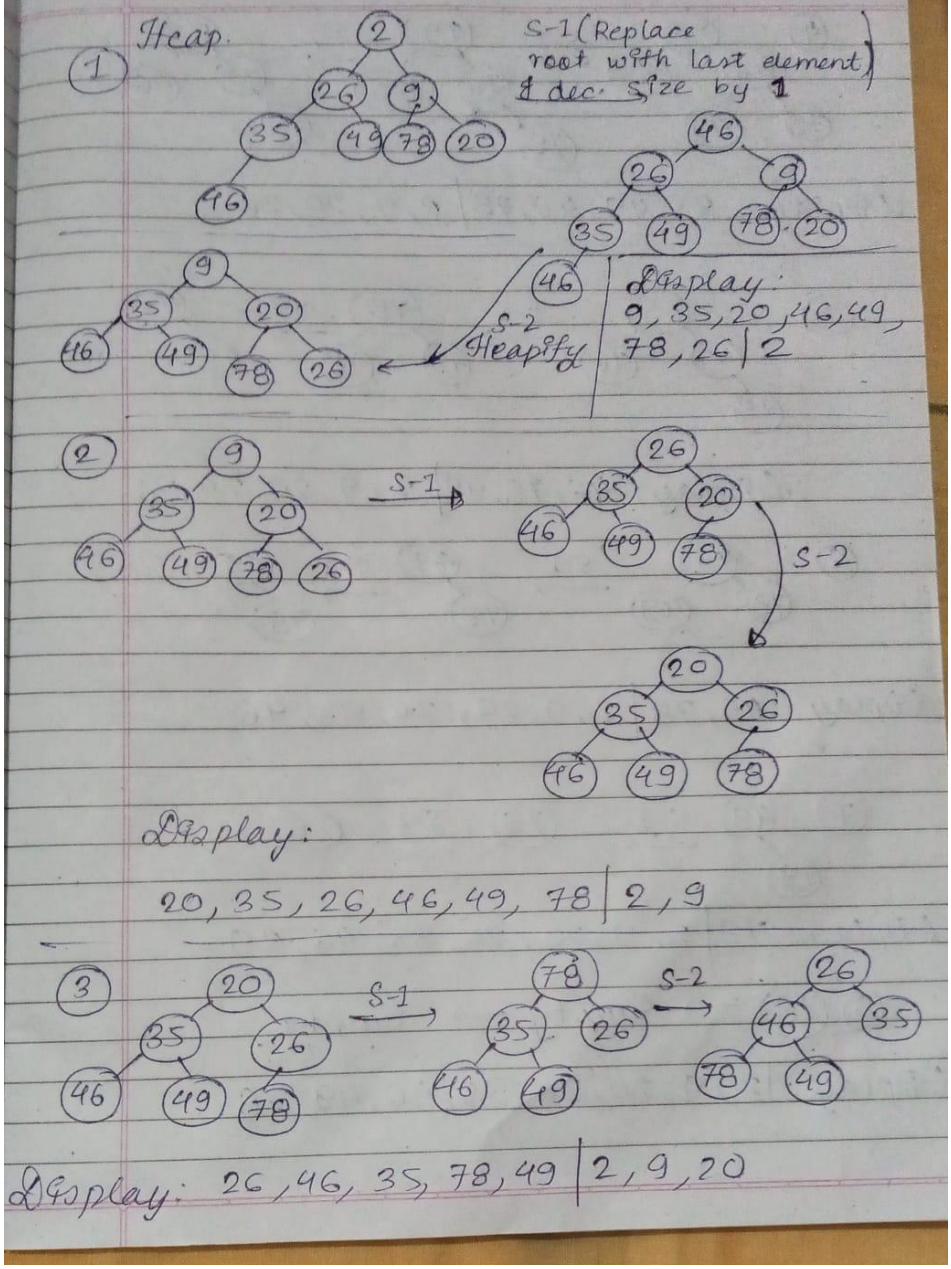
Algorithm

1. Build a min/max heap from data
2. Max./Min. element is stored at the root of heap.
   a. Replace it with last item of heap
   b. Reduce size of array by one
   c. Heapify the root
3. Repeat 2 until size of heap is greater than 1

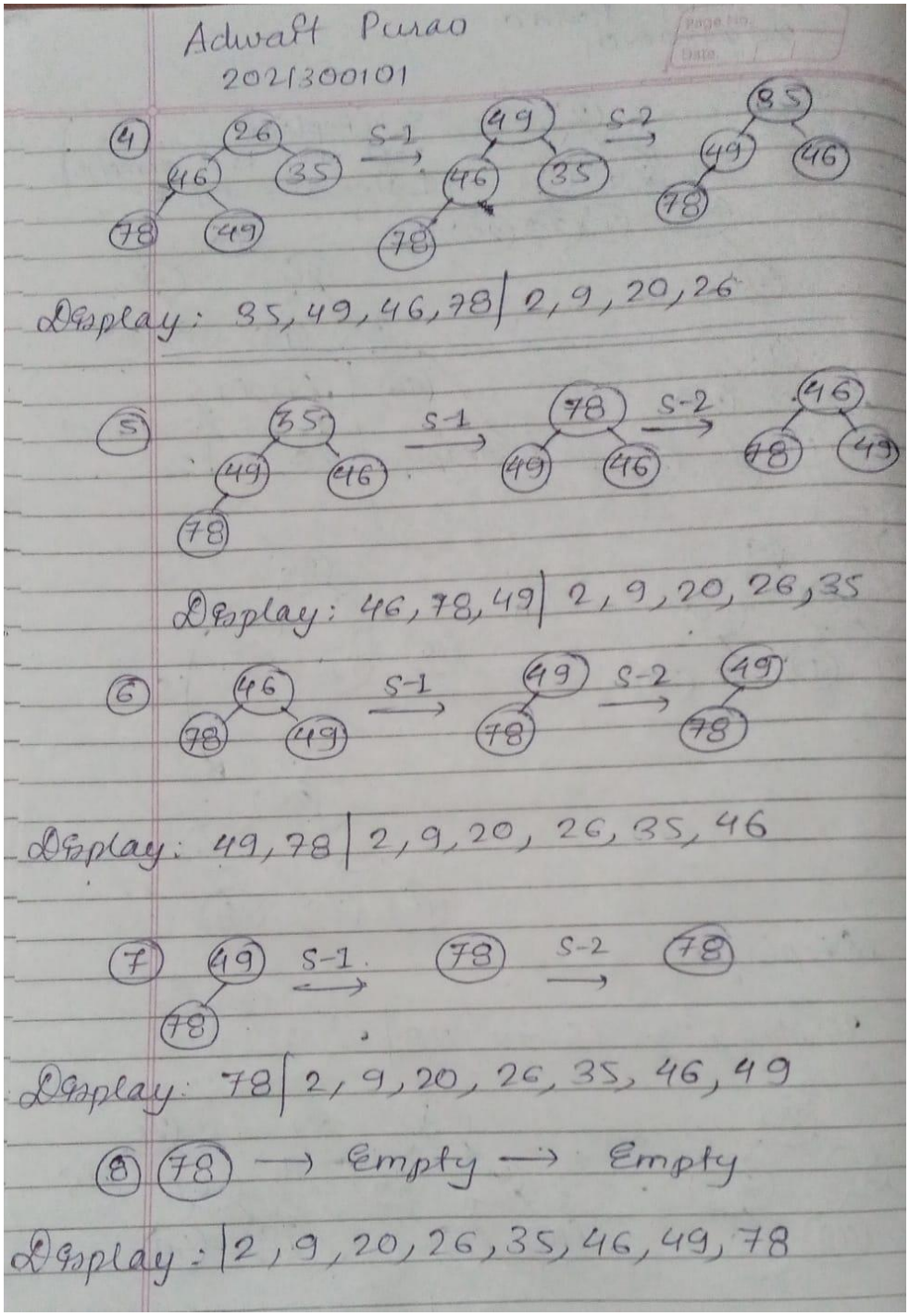
~~Elements~~: Heap: array:

2, 26, 9, 35, 49, 78, 20, 46

Adwait Purao
2021300101



Display: 26, 46, 35, 78, 49 | 2, 9, 20



CODE:

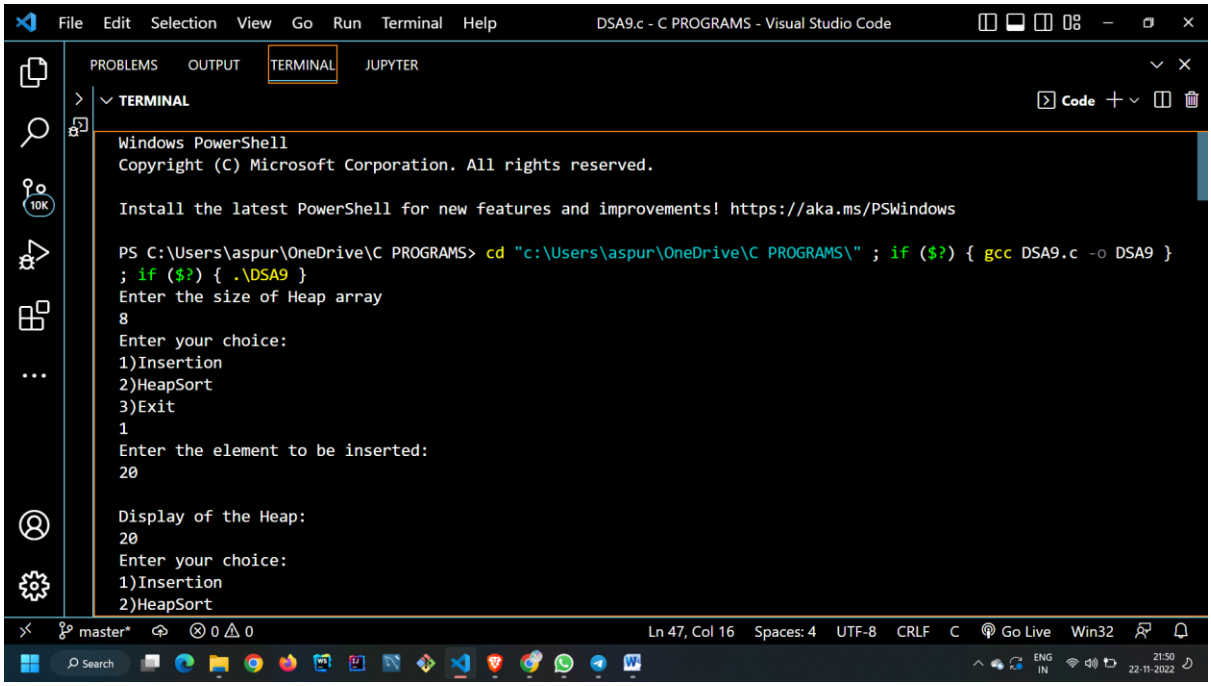
```
#include <stdio.h>
#include <stdlib.h>

struct Heap
{
    int *arr;//Array to maintain the heap
    int size;//Total size of heap array
    int usize;//Used(Current) size of heap array
};
```

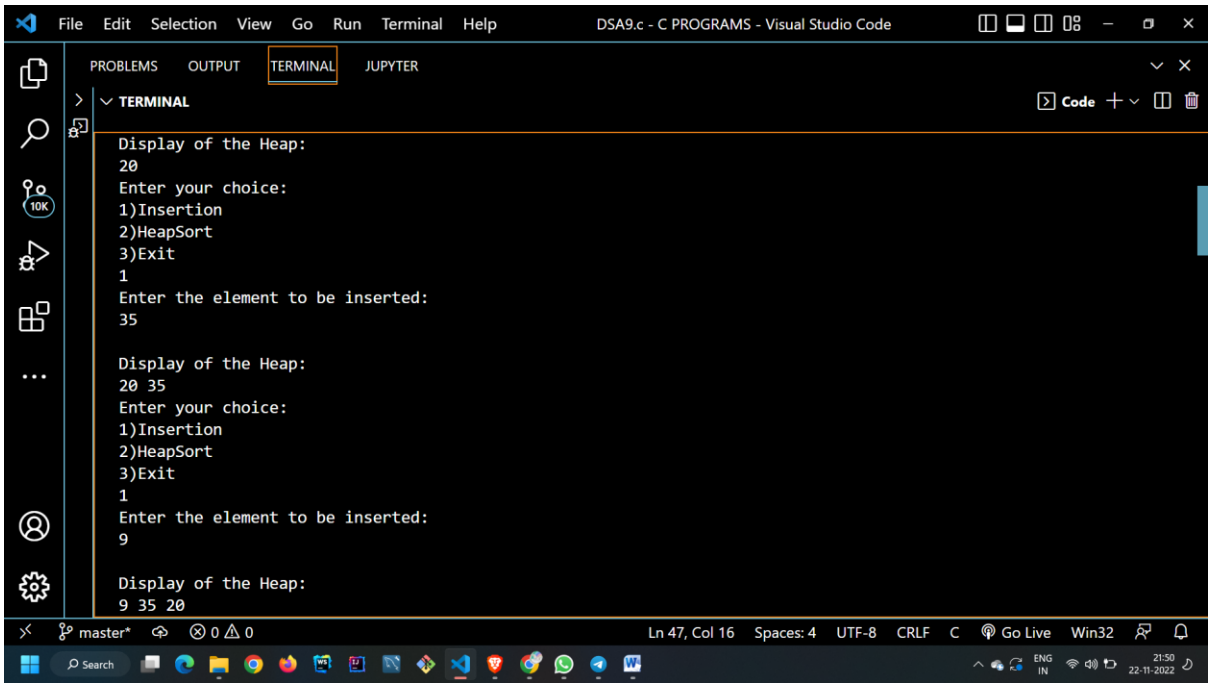
```c
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int isFull(struct Heap *h)
{
    if (h->usize == h->size - 1)
    {
        return 1;
    }
    return 0;
}

void Display(struct Heap *h)
{
    printf("\nDisplay of the Heap:\n");
    for (int i = 1; i <= h->usize; i++)
    {
        printf("%d ", (h->arr[i]));
    }
}

//Heapify Top-Down approach
void TDHeapify(struct Heap *h)
{
    //If size of heap is already 1 ,There is no need to heapify the tree as it
is already heapified
    if (h->usize == 1)
    {
        return;
    }
    else
    {
        for (int i = 2; i <= h->usize; i++)
        {
            int curr_ind = i;
            // If parent is greater than current index max heap property gets
violated , so we need to swap them
            while ((curr_ind > 1) && (h->arr[curr_ind] < h->arr[curr_ind /
2]))
            {
                swap(&(h->arr[curr_ind]), &(h->arr[curr_ind / 2]));
                curr_ind = curr_ind / 2;
            }
```

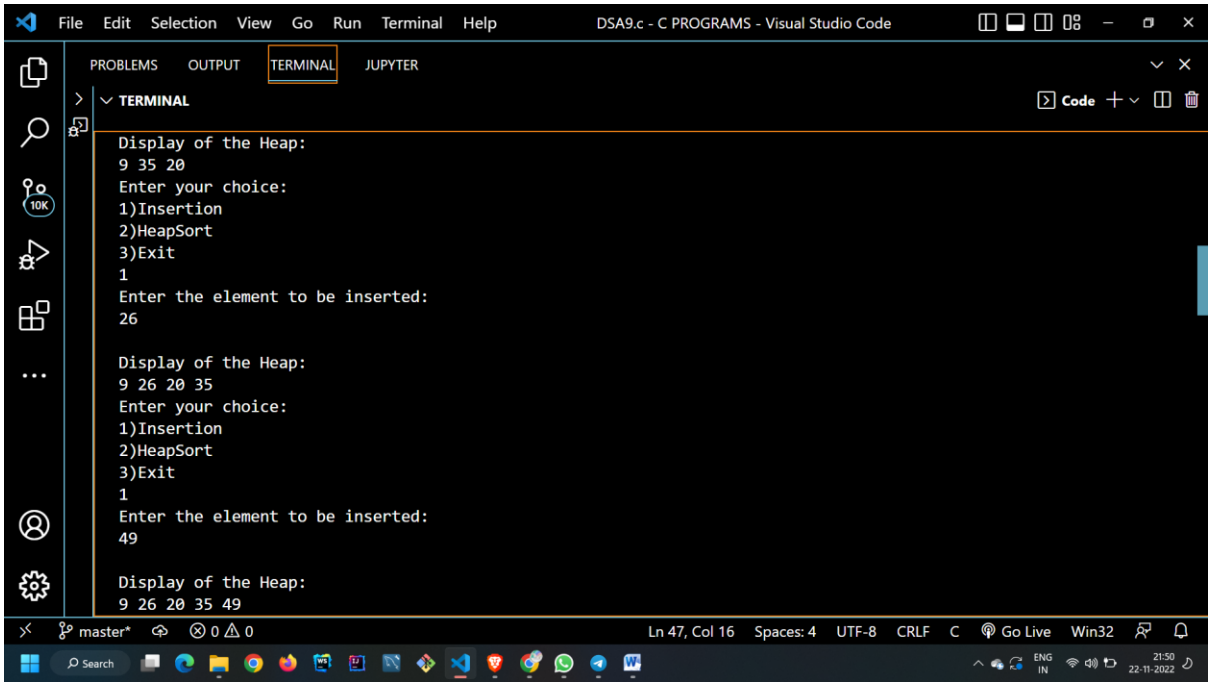
```
        }
    }
}

void Insertion(struct Heap *h, int data)
{
    if (isFull(h) == 1)
    {
        printf("Heap is full!,Can't insert anymore elements\n");
    }
    else
    {
        h->usize++;
        h->arr[h->usize] = data;
        TDHeapify(h);
    }
    Display(h);
}

int HeapSort(struct Heap *h, int sorted[])
{
    int count = 0;
    while (h->usize != 1)
    {
        count++;
        sorted[count] = h->arr[1];
        swap(&(h->arr[h->usize]), &(h->arr[1]));//Replace last element of heap
with the first
        h->usize--;
        TDHeapify(h);//Heapify the tree
        Display(h);
    }
    count++;
    sorted[count] = h->arr[1];
    return count;
}
int main()
{
    struct Heap *h = (struct Heap *)malloc(sizeof(struct Heap));
    int n;
    int element;

    printf("Enter the size of Heap array\n");
    scanf("%d", &n);
    h->size = n + 1;
    h->usize = 0;
    h->arr = (int *)malloc(h->size * sizeof(int));
    for (int i = 1; i < h->size; i++)
```

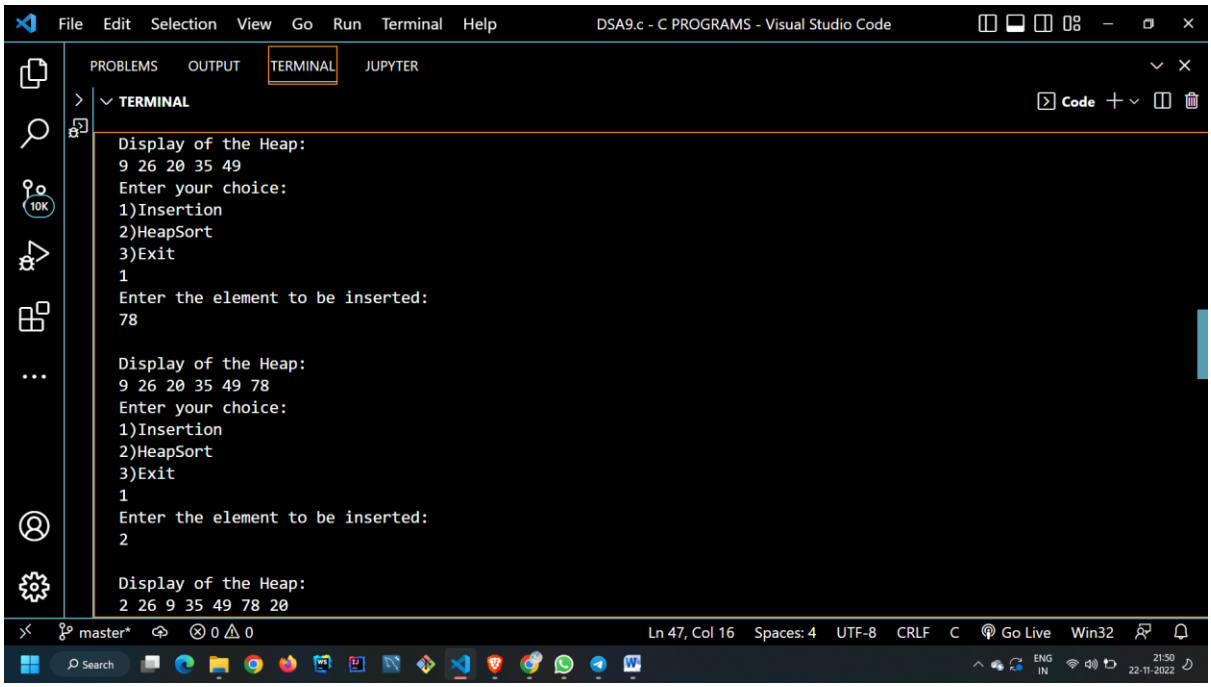
```
    {
        h->arr[i] = 0;
    }

    int sorted[n + 1];//Array to store the sorted elements
    int flag = 0;
    do
    {
        printf("Enter your choice:\n1)Insertion\n2)HeapSort\n3)Exit\n");
        int ch;
        scanf("%d", &ch);
        if (ch == 1)
        {
            printf("Enter the element to be inserted:\n");
            scanf("%d", &element);
            Insertion(h, element);
            printf("\n");
        }
        else if (ch == 2)
        {
            int count = HeapSort(h, sorted);
            printf("\nThe elements in sorted order\n");
            for (int i = 1; i <= count; i++)
            {
                printf("%d ", sorted[i]);
            }
            printf("\n");
        }
        else if (ch == 3)
        {
            printf("Program finished!\n");
            flag = 1;
        }
        else
        {
            printf("Invalid choice!\n");
        }
    } while (flag != 1);
}
```

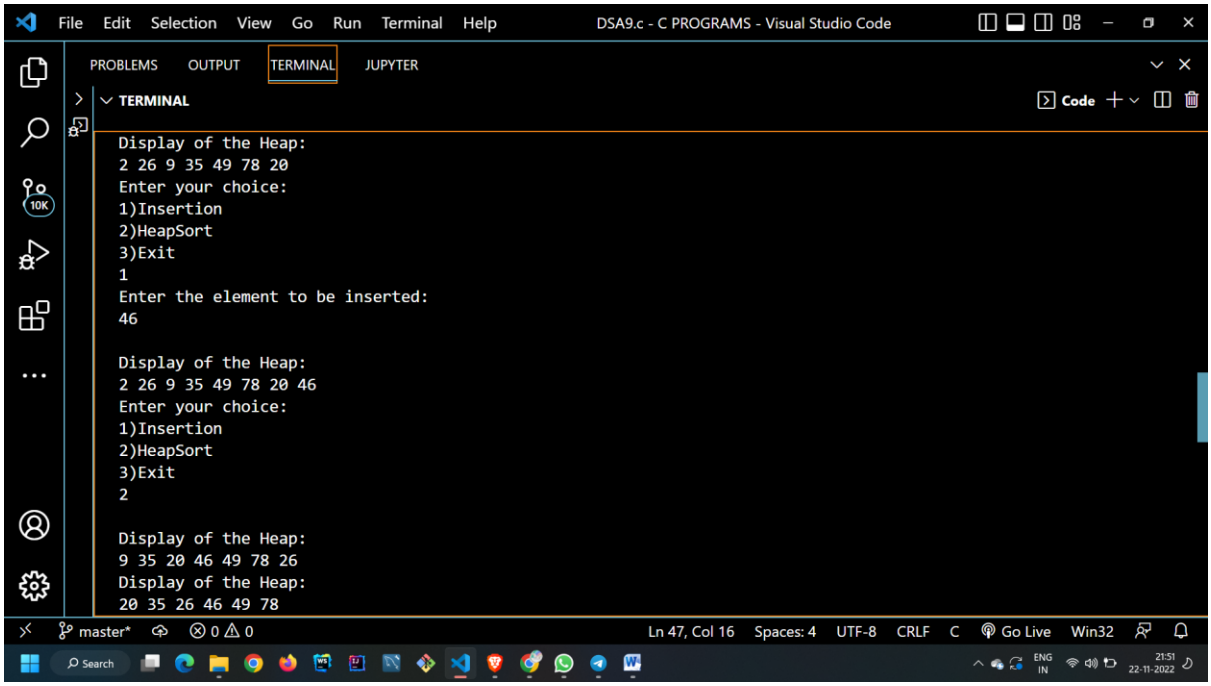
OUTPUT SCREENSHOT:

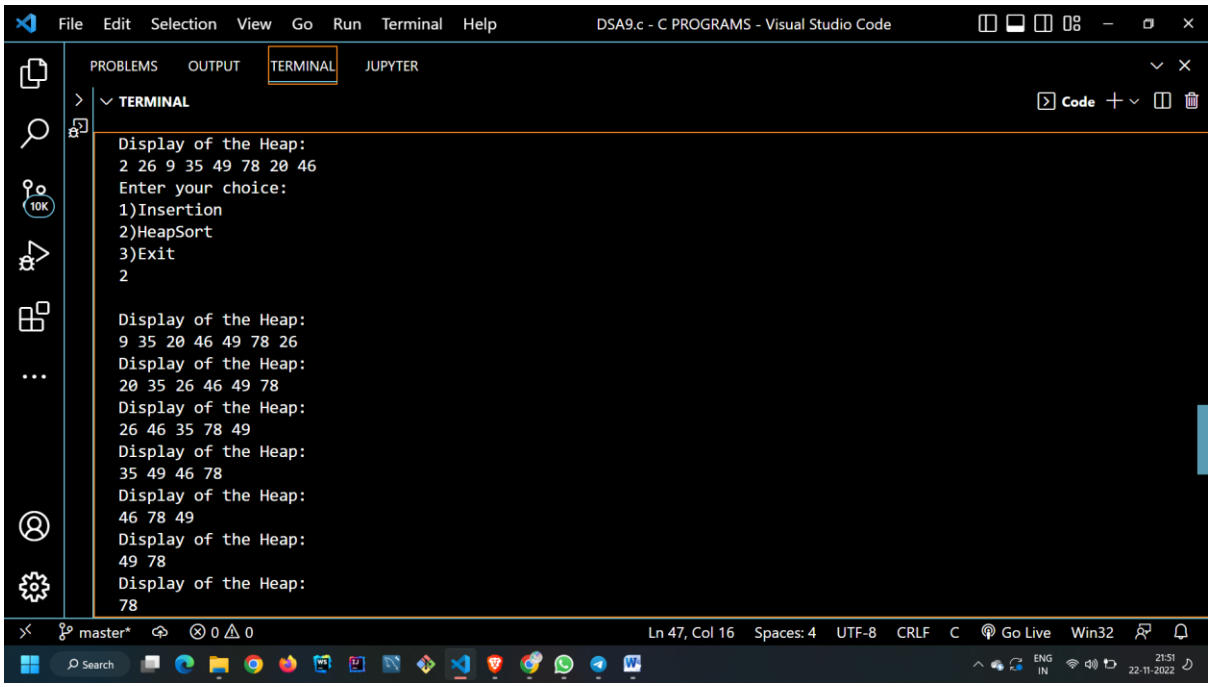

File Edit Selection View Go Run Terminal Help
DSA9.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\aspur\OneDrive\C PROGRAMS> cd "c:\Users\aspur\OneDrive\C PROGRAMS\" ; if ($?) { gcc DSA9.c -o DSA9 }
; if ($?) { .\DSA9 }
Enter the size of Heap array
8
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
1
Enter the element to be inserted:
20
Display of the Heap:
20
Enter your choice:
1)Insertion
2)HeapSort
master*
Ln 47, Col 16 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search
ENG IN
21:50 22-11-2022

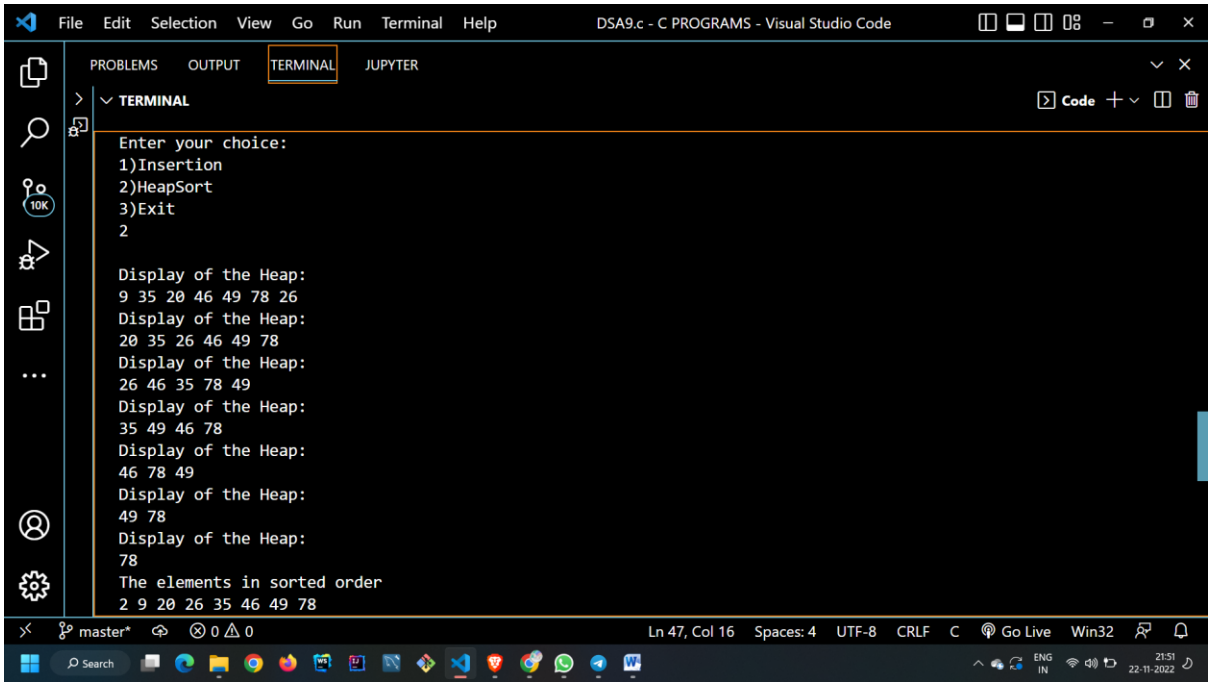

File Edit Selection View Go Run Terminal Help
DSA9.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Display of the Heap:
20
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
1
Enter the element to be inserted:
35
Display of the Heap:
20 35
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
1
Enter the element to be inserted:
9
Display of the Heap:
9 35 20
master*
Ln 47, Col 16 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search
ENG IN
21:50 22-11-2022


File Edit Selection View Go Run Terminal Help
DSA9.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Display of the Heap:
9 35 20
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
1
Enter the element to be inserted:
26
Display of the Heap:
9 26 20 35
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
1
Enter the element to be inserted:
49
Display of the Heap:
9 26 20 35 49
master*
Ln 47, Col 16 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search
ENG IN
21:50 22-11-2022

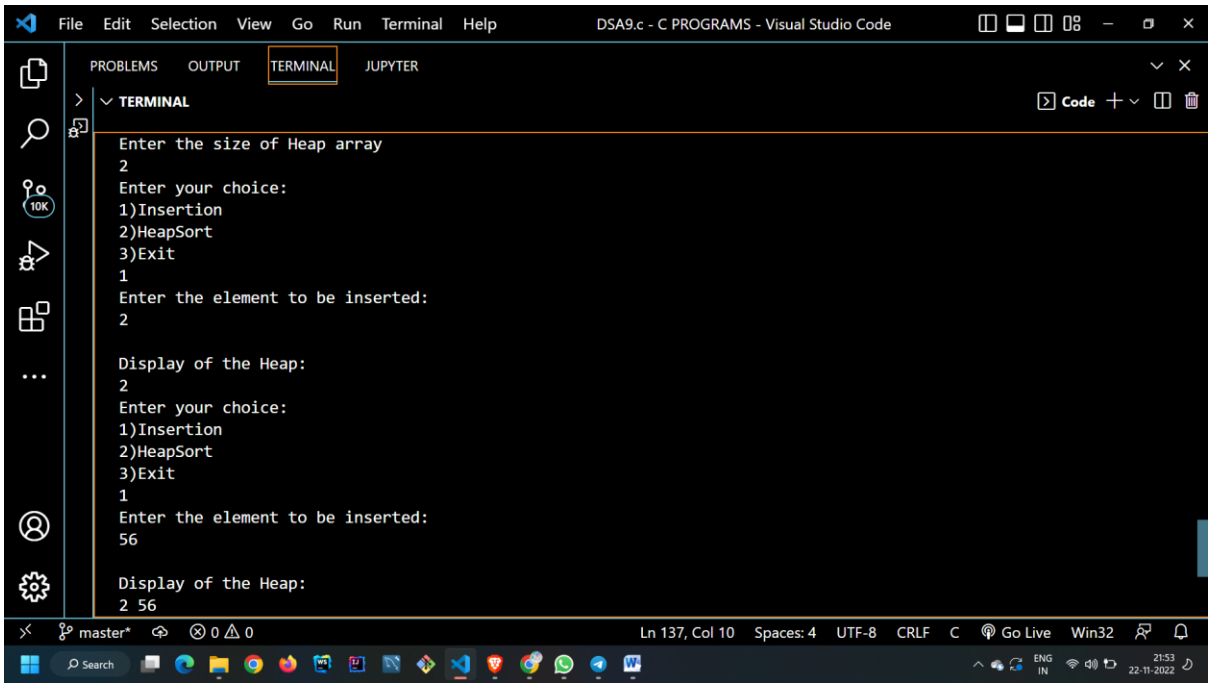

File Edit Selection View Go Run Terminal Help
DSA9.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Display of the Heap:
9 26 20 35 49
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
1
Enter the element to be inserted:
78
Display of the Heap:
9 26 20 35 49 78
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
1
Enter the element to be inserted:
2
Display of the Heap:
2 26 9 35 49 78 20
master*
Ln 47, Col 16 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search
ENG IN
21:50 22-11-2022

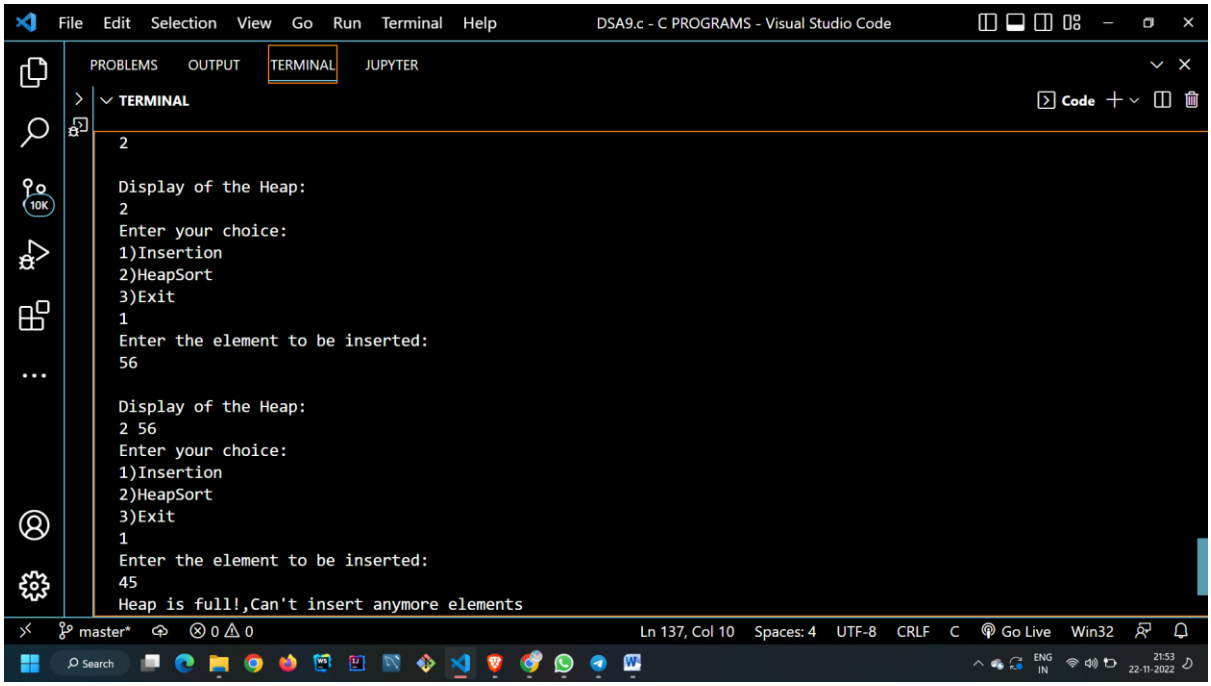

File Edit Selection View Go Run Terminal Help
DSA9.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Display of the Heap:
2 26 9 35 49 78 20
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
1
Enter the element to be inserted:
46
Display of the Heap:
2 26 9 35 49 78 20 46
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
2
Display of the Heap:
9 35 20 46 49 78 26
Display of the Heap:
20 35 26 46 49 78
master*
Ln 47, Col 16 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search
22-11-2022


File Edit Selection View Go Run Terminal Help
DSA9.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Display of the Heap:
2 26 9 35 49 78 20 46
Enter your choice:
1)Insertion
2)HeapSort
3)Exit
2
Display of the Heap:
9 35 20 46 49 78 26
Display of the Heap:
20 35 26 46 49 78
Display of the Heap:
26 46 35 78 49
Display of the Heap:
35 49 46 78
Display of the Heap:
46 78 49
Display of the Heap:
49 78
Display of the Heap:
78
master*
Ln 47, Col 16 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search
22-11-2022

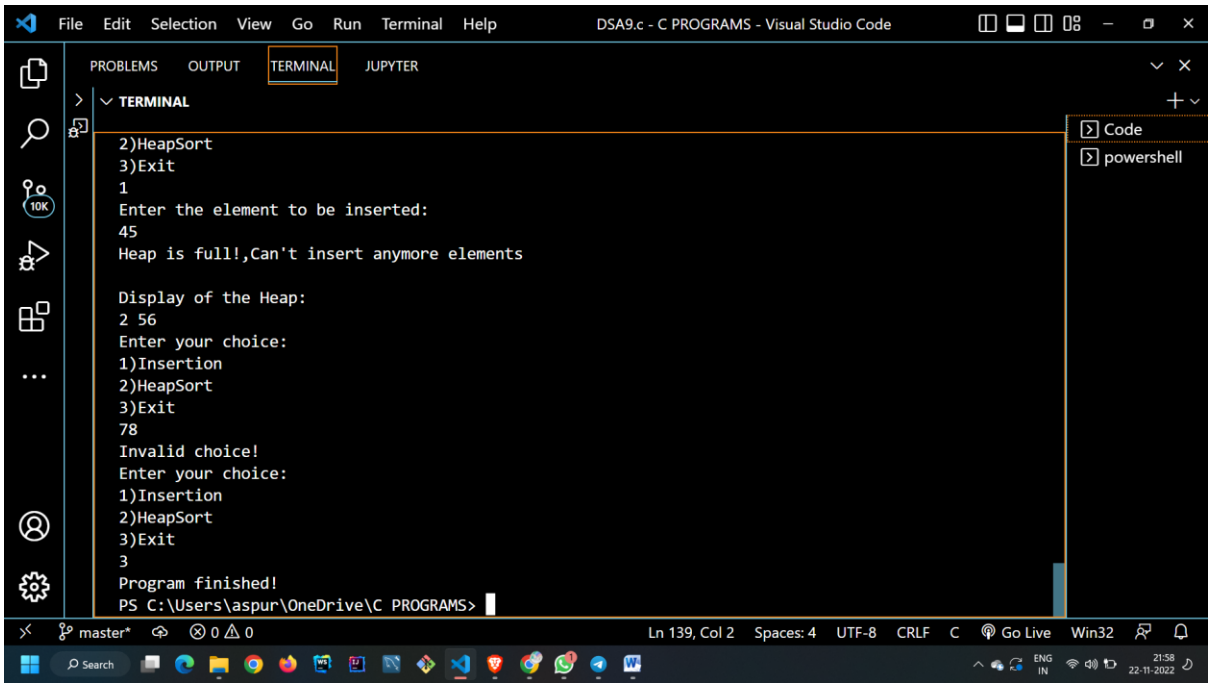


Test Case : When Heap is Full





Test Case: Invalid Choice



CONCLUSION:

In this experiment we learnt about Heap data structure and it's application . We learnt the implementation of Heaps using arrays . We learnt the internal structure of a Heap structure.

In the end we implemented Insertion , Heapify(Top Down Approach) , Heapsort functions with the help of a menu driven program in C.