# G. H. RAISONI COLLEGE OF ENGG., NAGPUR

(An Autonomous Institute under UGC Act 1956)

## Department of Artificial Intelligence

**Date: 19/08/2020**

### Practical Subject: Data Structures and Algorithms
### Session: 2020-21

**Student Details:**

| | |
|---|---|
| **Roll Number** | 63 |
| **Name** | Vishal Narnaware |
| **Semester** | 3 |
| **Section** | A |
| **Branch** | Artificial Intelligence |

**Practical Details: Practical Number- 6**

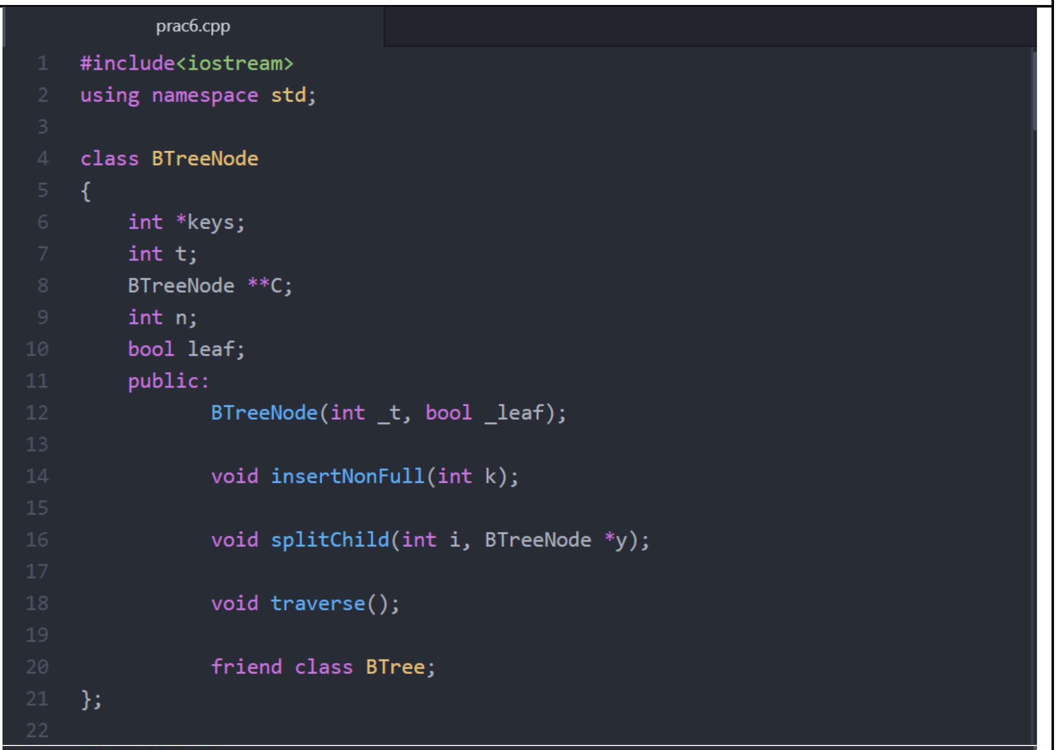
| | |
|---|---|
| **Practical Aim** | Design, develop and implement a program in C to perform the following operation:<br>a) Insertion into a B-tree<br>b) Heap sort algorithm for sorting a given list of integers in ascending order |
| **Theory** | **B Tree:**<br>B Tree is a self-balancing data structure based on a specific set of rules for searching, inserting, and deleting the data in a faster and memory efficient way. In order to achieve this, the following rules are followed to create a B Tree.<br>• All leaves will be created at the same level.<br>• B-Tree is determined by a number of degrees, which is also called "order" (specified by an external actor, like a programmer) depends upon the block size on the disk on which data is primarily located.<br>• The left subtree of the node will have lesser values than the right side of the subtree. This means that the nodes are also sorted in ascending order from left to right.<br>• The maximum number of child nodes, a root node as well as its child nodes can contain are calculated by the formula m – 1 |

| | |
|---|---|
| | **Heap Sort:**<br>Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for the remaining elements. |
| **Procedure** | **B-Tree:**<br>1. Using the SEARCH procedure for M-way trees (described above) find the leaf node to which X should be added.<br>2. add X to this node in the appropriate place among the values already there. Being a leaf node there are no subtrees to worry about.<br>3. if there are M-1 or fewer values in the node after adding X, then we are finished.<br>4. If there are M nodes after adding X, we say the node has *overflowed*. To repair this, we split the node into three parts:<br>Left: the first (M-1)/2 values<br>Middle: the middle value (position 1+((M-1)/2)<br>Right: the last (M-1)/2 values<br><br>**Heap Sort:**<br>1. Build a max heap from the input data.<br>2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.<br>3. Repeat above steps while size of heap is greater than 1 |
| **Algorithm** | **B-Tree:**<br>Step 1: START<br>Step 2: Run the search operation and find the appropriate place of insertion.<br>Step 3: Insert the new key at the proper location, but if the node has a maximum number of keys already:<br>Step 4: The node, along with a newly inserted key, will split from the middle element.<br>Step 5: The middle element will become the parent for the other two child nodes.<br>Step 6: The nodes must re-arrange keys in ascending order.<br>Step 7: STOP |

***Heap Sort:***

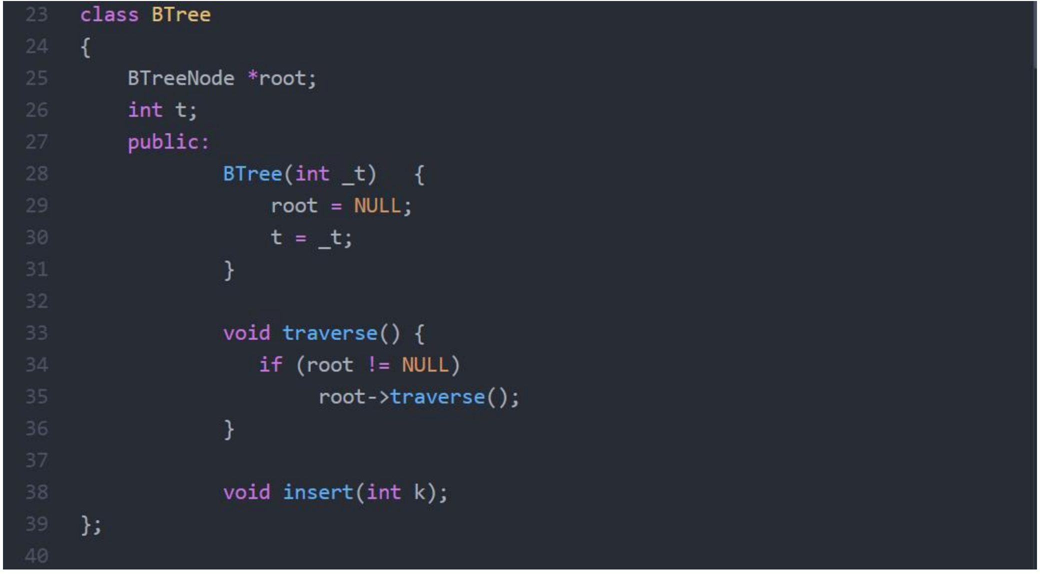
```
Step 1: START
Step 2: Construct a Binary Tree with given list of Elements.
Step 3: Transform the Binary Tree into Min Heap.
Step 4: Delete the root element from Min Heap using Heapify
        method.
Step 5: Put the deleted element into the Sorted list.
Step 6: Repeat the same until Min Heap becomes empty.
Step 7: Display the sorted list.
Step 8: STOP
```

**Program**

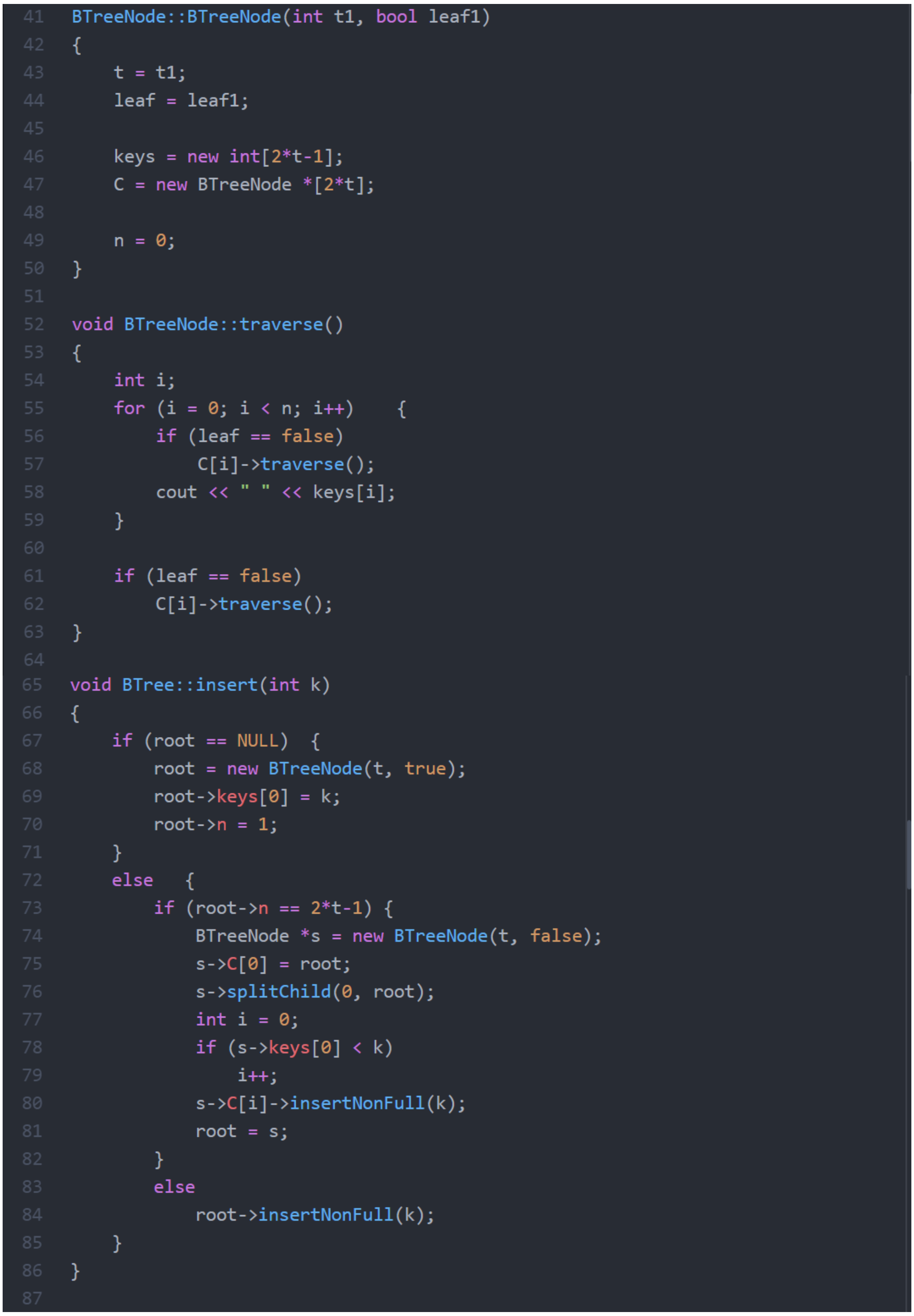


prac6.cpp

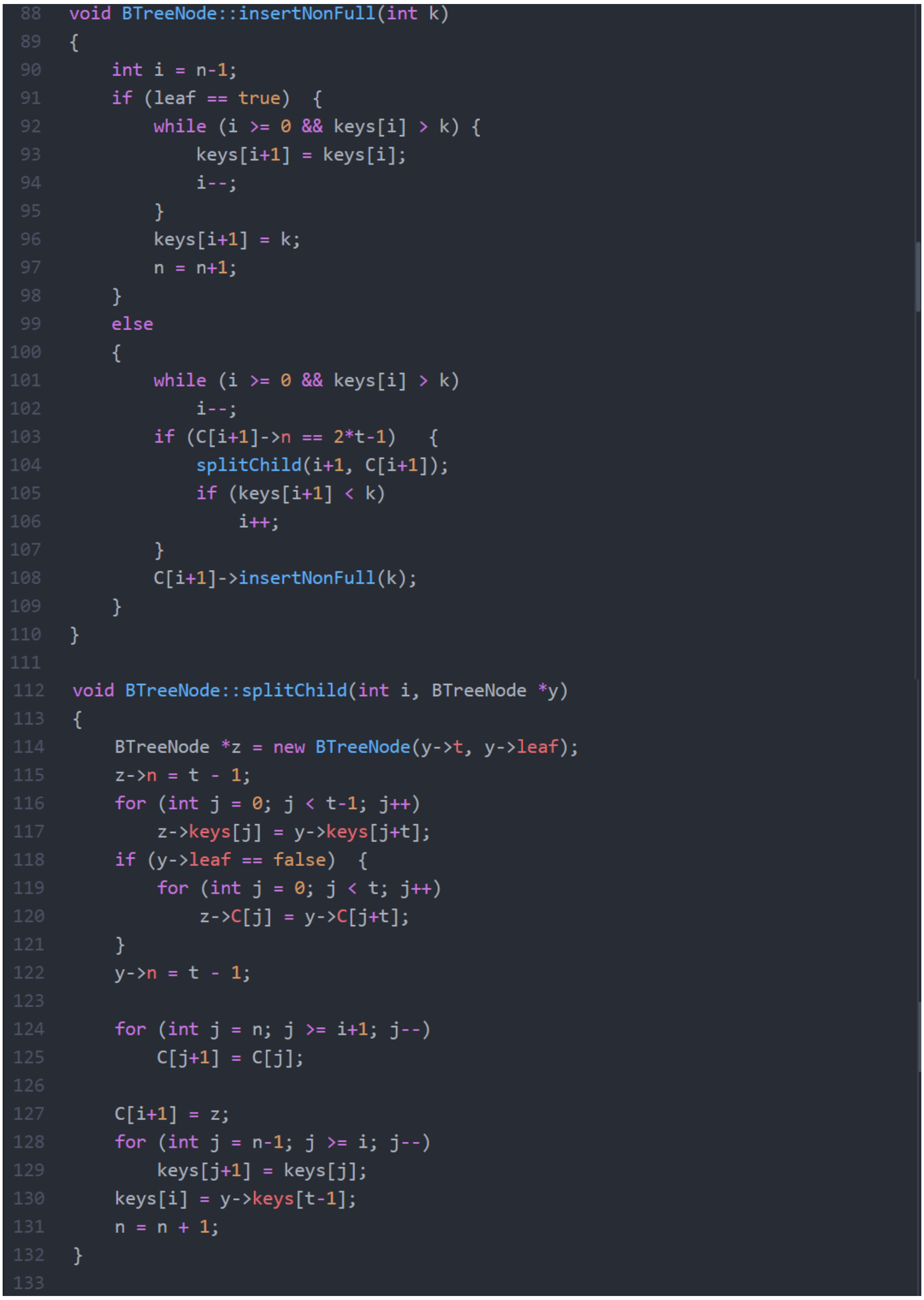
```cpp
#include<iostream>
using namespace std;

class BTreeNode
{
    int *keys;
    int t;
    BTreeNode **C;
    int n;
    bool leaf;
    public:
           BTreeNode(int _t, bool _leaf);

           void insertNonFull(int k);

           void splitChild(int i, BTreeNode *y);

           void traverse();

           friend class BTree;
};

```



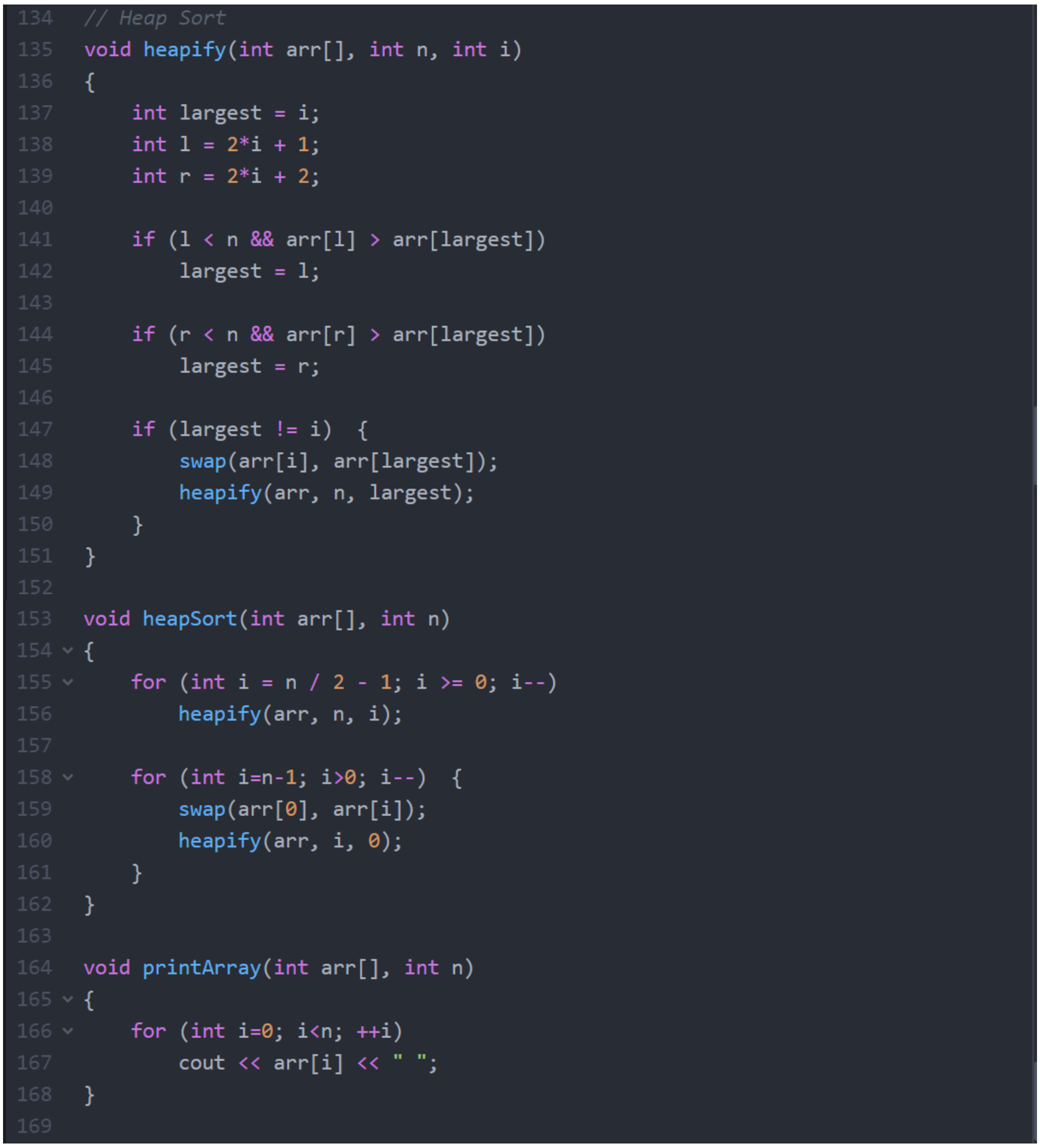
```cpp
class BTree
{
    BTreeNode *root;
    int t;
    public:
            BTree(int _t)   {
                root = NULL;
                t = _t;
            }

            void traverse() {
               if (root != NULL)
                    root->traverse();
            }

            void insert(int k);
};

```



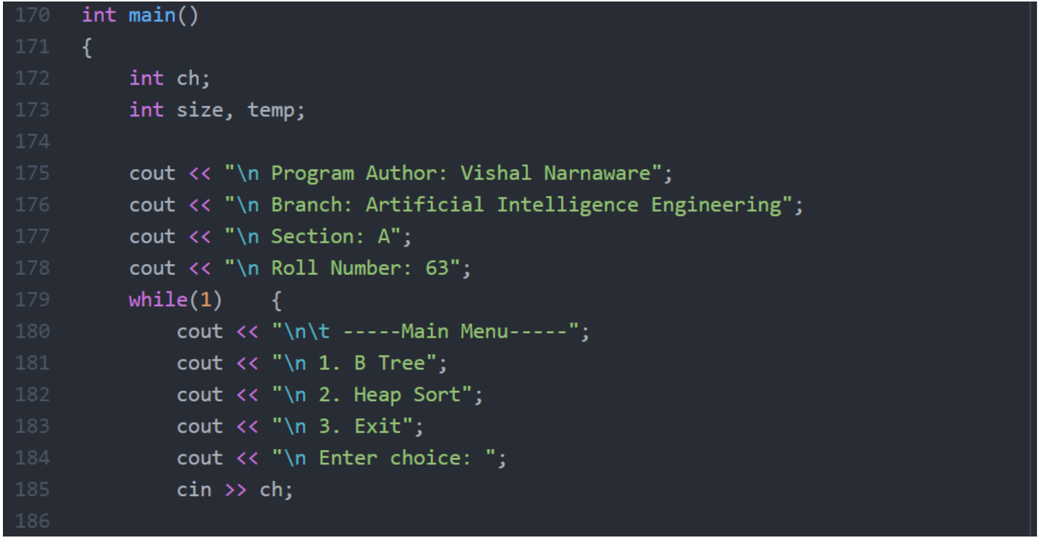
```cpp
BTreeNode::BTreeNode(int t1, bool leaf1)
{
    t = t1;
    leaf = leaf1;

    keys = new int[2*t-1];
    C = new BTreeNode *[2*t];

    n = 0;
}

void BTreeNode::traverse()
{
    int i;
    for (i = 0; i < n; i++)    {
        if (leaf == false)
            C[i]->traverse();
        cout << " " << keys[i];
    }

    if (leaf == false)
        C[i]->traverse();
}

void BTree::insert(int k)
{
    if (root == NULL)  {
        root = new BTreeNode(t, true);
        root->keys[0] = k;
        root->n = 1;
    }
    else   {
        if (root->n == 2*t-1) {
            BTreeNode *s = new BTreeNode(t, false);
            s->C[0] = root;
            s->splitChild(0, root);
            int i = 0;
            if (s->keys[0] < k)
                i++;
            s->C[i]->insertNonFull(k);
            root = s;
        }
        else
            root->insertNonFull(k);
    }
}
```



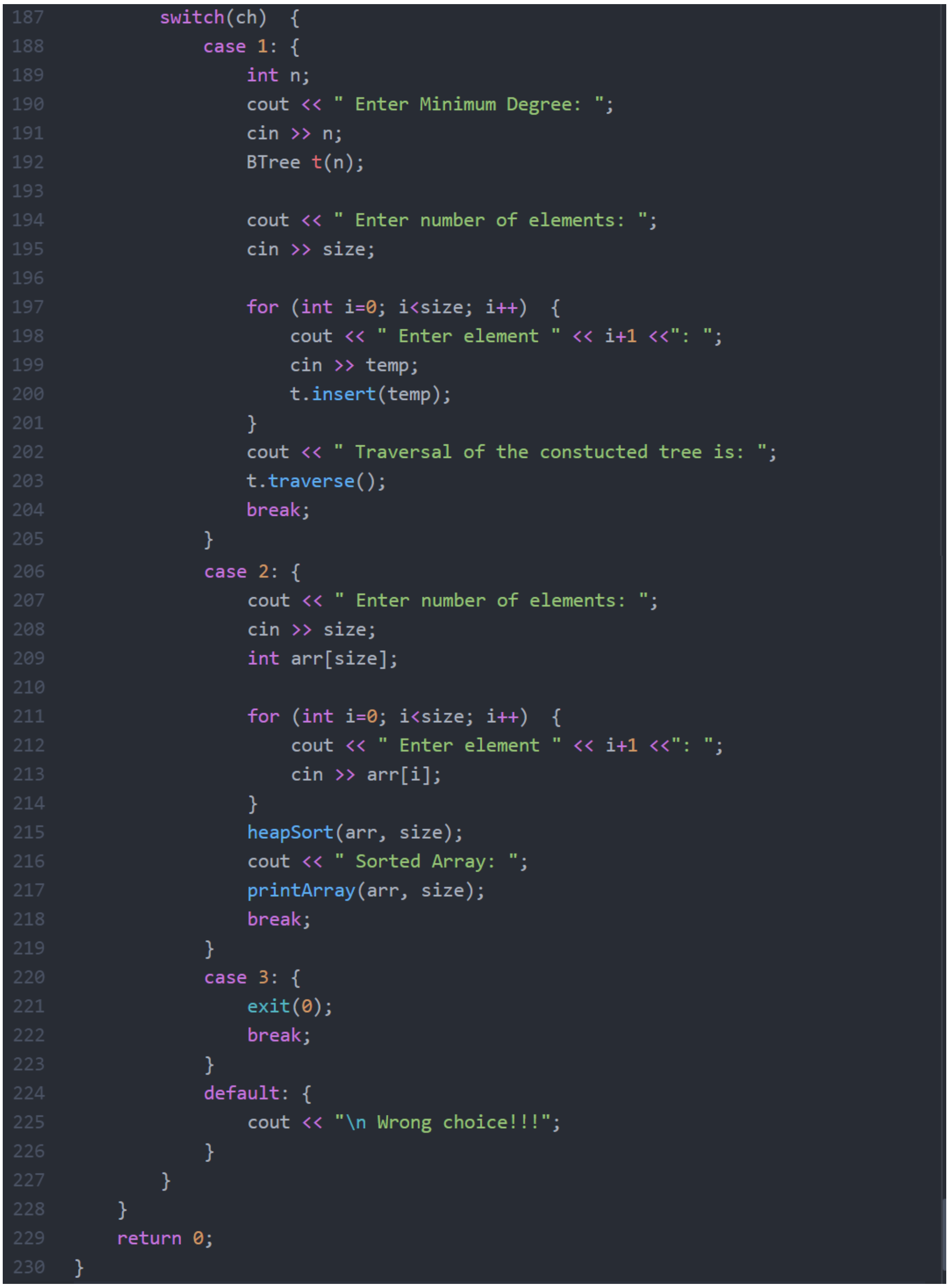
```cpp
void BTreeNode::insertNonFull(int k)
{
    int i = n-1;
    if (leaf == true)  {
        while (i >= 0 && keys[i] > k) {
            keys[i+1] = keys[i];
            i--;
        }
        keys[i+1] = k;
        n = n+1;
    }
    else
    {
        while (i >= 0 && keys[i] > k)
            i--;
        if (C[i+1]->n == 2*t-1)   {
            splitChild(i+1, C[i+1]);
            if (keys[i+1] < k)
                i++;
        }
        C[i+1]->insertNonFull(k);
    }
}

void BTreeNode::splitChild(int i, BTreeNode *y)
{
    BTreeNode *z = new BTreeNode(y->t, y->leaf);
    z->n = t - 1;
    for (int j = 0; j < t-1; j++)
        z->keys[j] = y->keys[j+t];
    if (y->leaf == false)  {
        for (int j = 0; j < t; j++)
            z->C[j] = y->C[j+t];
    }
    y->n = t - 1;

    for (int j = n; j >= i+1; j--)
        C[j+1] = C[j];

    C[i+1] = z;
    for (int j = n-1; j >= i; j--)
        keys[j+1] = keys[j];
    keys[i] = y->keys[t-1];
    n = n + 1;
}
```



```cpp
// Heap Sort
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < n && arr[l] > arr[largest])
        largest = l;

    if (r < n && arr[r] > arr[largest])
        largest = r;

    if (largest != i)  {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i=n-1; i>0; i--)  {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
}

```



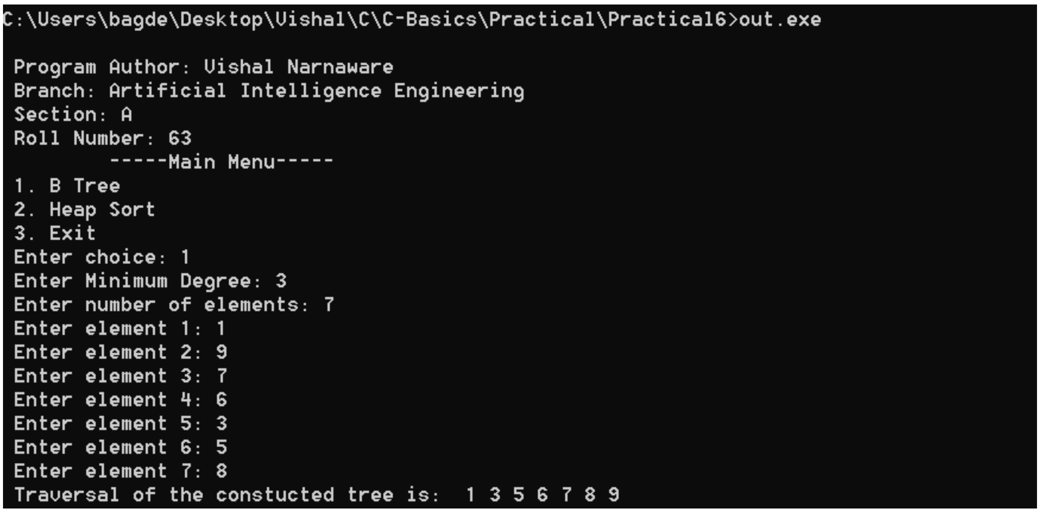
```cpp
int main()
{
    int ch;
    int size, temp;

    cout << "\n Program Author: Vishal Narnaware";
    cout << "\n Branch: Artificial Intelligence Engineering";
    cout << "\n Section: A";
    cout << "\n Roll Number: 63";
    while(1)    {
        cout << "\n\t -----Main Menu-----";
        cout << "\n 1. B Tree";
        cout << "\n 2. Heap Sort";
        cout << "\n 3. Exit";
        cout << "\n Enter choice: ";
        cin >> ch;

```



```
        switch(ch)  {
            case 1: {
                int n;
                cout << " Enter Minimum Degree: ";
                cin >> n;
                BTree t(n);

                cout << " Enter number of elements: ";
                cin >> size;

                for (int i=0; i<size; i++)  {
                    cout << " Enter element " << i+1 <<": ";
                    cin >> temp;
                    t.insert(temp);
                }
                cout << " Traversal of the constucted tree is: ";
                t.traverse();
                break;
            }
            case 2: {
                cout << " Enter number of elements: ";
                cin >> size;
                int arr[size];

                for (int i=0; i<size; i++)  {
                    cout << " Enter element " << i+1 <<": ";
                    cin >> arr[i];
                }
                heapSort(arr, size);
                cout << " Sorted Array: ";
                printArray(arr, size);
                break;
            }
            case 3: {
                exit(0);
                break;
            }
            default: {
                cout << "\n Wrong choice!!!";
            }
        }
    }
    return 0;
}
```

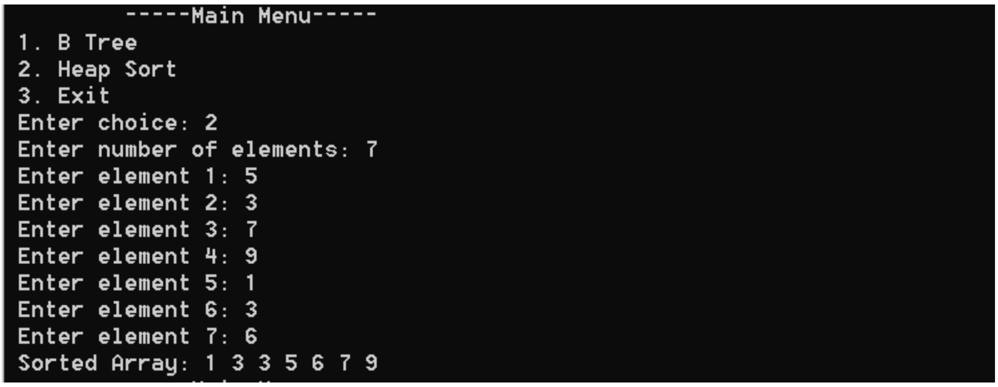
**Output**

***B Tree:***



```
C:\Users\bagde\Desktop\Vishal\C\C-Basics\Practical\Practical6>out.exe

 Program Author: Vishal Narnaware
 Branch: Artificial Intelligence Engineering
 Section: A
 Roll Number: 63
         -----Main Menu-----
 1. B Tree
 2. Heap Sort
 3. Exit
 Enter choice: 1
 Enter Minimum Degree: 3
 Enter number of elements: 7
 Enter element 1: 1
 Enter element 2: 9
 Enter element 3: 7
 Enter element 4: 6
 Enter element 5: 3
 Enter element 6: 5
 Enter element 7: 8
 Traversal of the constucted tree is:  1 3 5 6 7 8 9
```

***Heap Sort:***



```
         -----Main Menu-----
 1. B Tree
 2. Heap Sort
 3. Exit
 Enter choice: 2
 Enter number of elements: 7
 Enter element 1: 5
 Enter element 2: 3
 Enter element 3: 7
 Enter element 4: 9
 Enter element 5: 1
 Enter element 6: 3
 Enter element 7: 6
 Sorted Array: 1 3 3 5 6 7 9
```

**Conclusion**

Hence, successfully implemented a program in C to perform the Insertion into a B-tree and Heap sort algorithm for sorting a given list of integers in ascending order.