Convert BST to Min Heap

You are given a 'ROOT' of a binary search tree of integers. The given BST is also a complete binary tree.
Your task is to convert the given binary search tree into a Min Heap and print the preorder traversal of the updated binary search tree.
Note:

Binary Search Tree is a node-based binary tree data structure that has the following properties:

1. The left subtree of a node contains only nodes with keys lesser than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. The left and right subtree each must also be a binary search tree.

A Binary Heap is a Binary Tree with the following property:

1. It's a complete tree (all levels are filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

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 Min Heap.

For example:

Given:- BST's 'ROOT' = 4 2 6 -1 -1 -1 -1
Then the min-heap in pre-order fashion would be 2 4 6.

→ Approach :

↳ Traverse BST, convert it to vector

↳ Heapify the vector

↳ Convert the vector to tree.

```
BinaryTreeNode *getPredecessor(BinaryTreeNode *node) {
  BinaryTreeNode *curr = node;

  if (curr->left) {
    curr = curr->left;
    while (curr->right != NULL && curr->right != node)
      curr = curr->right;
  }
  return curr;
}

void morrisTraversal(BinaryTreeNode *root, vector<int> &arr) {
  if (!root)
    return;

  BinaryTreeNode *curr = root;

  while (curr != NULL) {
    if (curr->left) {
      BinaryTreeNode *predecessor = getPredecessor(curr);
      if (predecessor->right) {
        predecessor->right = NULL;
        curr = curr->right;
      } else {
        predecessor->right = curr;
        arr.push_back(curr->data);
        curr = curr->left;
      }
    } else {
      arr.push_back(curr->data);
      curr = curr->right;
    }
  }
}

void heapify(vector<int> &arr, int n, int i) {
  int smallest = i;
  int left = 2 * i + 1;
  int right = 2 * i + 2;

  if (left < n && arr[left] < arr[smallest])
    smallest = left;
  if (right < n && arr[right] < arr[smallest])
    smallest = right;

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

BinaryTreeNode *convertBST(BinaryTreeNode *root) {
  // traverse bst, convert to vector
  vector<int> arr;
  morrisTraversal(root, arr);
  // convert vector to minheap
  int n = arr.size();
  for (int i = n / 2 - 1; i >= 0; i--) {
    heapify(arr, n, i);
  }
  // vector to tree
  map<int, BinaryTreeNode *> mpp;
  for (int i = 0; i < n; i++) {
    BinaryTreeNode *temp = new BinaryTreeNode(arr[i]);

    if (i != 0) {
      int parentIndex = (i - 1) / 2;
      BinaryTreeNode *parentNode = mpp[arr[parentIndex]];
      if (!parentNode->left)
        parentNode->left = temp;
      else
        parentNode->right = temp;
    }

    mpp[arr[i]] = temp;
  }

  return mpp[arr[0]];
}
```

Not working !!!

Some error ⚠

→ Approach 2 :-

```
void getinorder(BinaryTreeNode *root, vector<int> &v) {
  if (!root) {
    return;
  }
  getinorder(root->left, v);
  v.push_back(root->data);
  getinorder(root->right, v);
}
void fillpreorder(BinaryTreeNode *&root, vector<int> inorder, int &i) {
  if (!root) {
    return;
  }
  root->data = inorder[i++];
  fillpreorder(root->left, inorder, i);
  fillpreorder(root->right, inorder, i);
}
BinaryTreeNode *convertBST(BinaryTreeNode *root) {
  // step1: get inorder of bst which is in sorted order
  vector<int> inorder;
  getinorder(root, inorder);
  // step 2: min order && for all  L data < for all  R  data ⇒ N<L<R which is a
  // preorder
  int i = 0;
  fillpreorder(root, inorder, i);
  return root;
}
```

✓