17) REPLACE EVERY ELEMENT WITH THE ELEMENT JUST GREATER THAN ITS RIGHT:

METHOD 1:
A naive method is to run two loops. The outer loop will one by one pick array elements from left to right. The inner loop will find the smallest element greater than the picked element on its right side. Finally, the outer loop will replace the picked element with the element found by inner loop.
Time Complexity : O(N^2).
Space Complexity : O(1).

METHOD 2:
In this method, Convert the given array into the BST by traversing linearly through the array from the last element.
When a particular element is inserted into the tree, at the same time we check the INORDER SUCCESSOR of the element inserted and that will be the answer for that element.
The InOrder Successor can be calculated by:

1) Morris traversal(which will take the O(N) time for one element and hence the resultant complexity will be O(N^2)).
2) InOrder Traversal of the tree.(Time Complexity : O(N log N)   Space Complexity : O(N))

CODE OF METHOD 2 USING MORRIS TRAVERSAL:

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

class node{
   public:
   int data;
   node *left=NULL;
   node *right=NULL;
};

void Insert(node *root,int data){
   if(root->data<data){
      if(root->right==NULL){
         node *t=new node;
         t->data=data;
         root->right=t;
      }
      else{
         Insert(root->right,data);
      }
   }
   else{
      if(root->left==NULL){
         node *t=new node;
         t->data=data;
         root->left=t;
```

```
        }
        else{
            Insert(root->left,data);
        }
    }
}

int InOrderSuccessor(node *root,int n){
    node *current=root;
    node *previous=NULL;
    while(current!=NULL){
        if(current->left==NULL){
            if(previous==NULL){
                previous=current;
                current=current->right;
            }
            else if(previous->data==n){
                return current->data;
            }
            else{
                previous=current;
                current=current->right;
            }
        }
        else{
            node *t=current->left;
            while(t->right!=NULL && t->right!=current){
                t=t->right;
            }
            if(t->right==NULL){
                t->right=current;
                current=current->left;
            }
            else{
                t->right=NULL;
                if(previous==NULL){
                    previous=current;
                    current=current->right;
                }
                else if(previous->data==n){
                    return current->data;
                }
                else{
                    previous=current;
                    current=current->right;
                }
            }
        }
    }
    return -1;
```

```
}

vector<int> convertIntoBST(int array[],int n){
    vector<int> answer(n);
    answer[n-1]=-1;
    node *root=new node;
    root->data=array[n-1];
    for(int i=n-2;i>=0;i--){
        Insert(root,array[i]);
        answer[i]=InOrderSuccessor(root,array[i]);
    }
    return answer;
}

//8 58 71 18 31 32 63 92 43 3 91 93 25 80 28

int main(){
    int n;
    cin>>n;
    int array[n];
    for(int i=0;i<n;i++){
        cin>>array[i];
    }
    vector<int> answer=convertIntoBST(array,n);
    cout<<endl;
    for(int i=0;i<answer.size();i++){
        cout<<answer[i]<<" ";
    }
    return 0;
}
```

CODE OF METHOD 2 USING THE INORDER TRAVERSAL:

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

class node{
    public:
    int data;
    node *left=NULL;
    node *right=NULL;
};

void Insert(node *root,int data){
    if(root->data<data){
        if(root->right==NULL){
            node *t=new node;
            t->data=data;
            root->right=t;
        }
        else{
            Insert(root->right,data);
        }
    }
    else{
        if(root->left==NULL){
            node *t=new node;
            t->data=data;
            root->left=t;
        }
        else{
            Insert(root->left,data);
        }
    }
}

void InOrderSuccessor(node *root,int n,node *&previous,node *&current){
    if(root!=NULL){
        InOrderSuccessor(root->left,n,previous,current);
        if(previous==NULL){
            previous=root;
        }
        else if(previous->data==n && current!=NULL){
            return;
        }
        else{
            if(previous->data==n){
                current=root;
                return;
            }
            else{
                previous=root;
            }
        }
        InOrderSuccessor(root->right,n,previous,current);
    }
}

vector<int> convertIntoBST(int array[],int n){
    vector<int> answer(n);
    answer[n-1]=-1;
    node *root=new node;
    root->data=array[n-1];
    for(int i=n-2;i>=0;i--){
        node *previous=NULL;
        node *current=NULL;
        Insert(root,array[i]);
        InOrderSuccessor(root,array[i],previous,current);
        if(current==NULL){
            answer[i]=-1;
        }
        else{
            answer[i]=current->data;
        }
    }
    return answer;
}

//8 58 71 18 31 32 63 92 43 3 91 93 25 80 28

int main(){
    int n;
    cin>>n;
    int array[n];
    for(int i=0;i<n;i++){
        cin>>array[i];
    }
    vector<int> answer=convertIntoBST(array,n);
    cout<<endl;
    for(int i=0;i<answer.size();i++){
        cout<<answer[i]<<" ";
    }
    return 0;
}
```