MONEY THIEF

****
Author: Parvat Jakhar
****

Explaination:
We can solve this problem by considering the fact that both node and its children can't be in sum at same time, so when we take a node into our sum we will call recursively for its grandchildren or when we don't take this node we will call for all its children nodes and finally we will choose maximum from both of these results.

```
    1
   / \
  2*   3
 /   / \
1   4*   5*
```

*the starred nodes are selected.

It can be seen easily that above approach can lead to solving same subproblem many times, for example in above diagram node 1 calls node 4 and 5 when its value is chosen and node 3 also calls them when its value is not chosen so these nodes are processed more than once. We can stop solving these nodes more than once by memoizing the result at all nodes.
In below code a map is used for memoizing the result which stores result of complete subtree rooted at a node in the map, so that if it is called again, the value is not calculated again instead stored value from map is returned directly.
Please see below code for better understanding.

Code:

```
#include<bits/stdc++.h>
using namespace std;
struct Node
{
   int val;
   Node* left, *right;
};

Node* newNode(int val)
{
   Node* node = (Node*)malloc(sizeof(Node));
   node->val = val;
   node->left = node->right = NULL;
   return (node);
}

Node* insertNode(int arr[], Node* root,int i, int n)
{
   if (i < n)
   {
```

```
        if(arr[i]==-1)
        {
          return NULL;
        }
        Node* temp = newNode(arr[i]);
        root = temp;
        root->left = insertNode(arr,root->left, 2 * i + 1, n);
        root->right = insertNode(arr,root->right, 2 * i + 2, n);
    }
    return root;
}
unordered_map<Node*,long long int> um;
long long int rob(Node* root)
{
        if(!root)
            return 0;

        if(!root->left and !root->right)
        {
            return root->val;
        }

        if(um.count(root)!=0) return um[root];

        long long int curr = root->val;
        if(root->left)
            curr =(curr+ rob(root->left->left) + rob(root->left->right))%1000000007;

        if(root->right)
            curr = (curr+rob(root->right->right) + rob(root->right->left))%1000000007;

        return um[root]=(max(curr,rob(root->left) + rob(root->right)))%1000000007;
}
int main()
{
  int t;
  cin>>t;
  while(t--)
  {
            int n;
            cin>>n;
            long long int a[n];
            for(int i=0;i<n;i++)
                    cin>>a[i];
            Node* root = insertNode(a, root, 0, n);

            cout<<rob(root)<<endl;
            }
}
```

Time Complexity: O(n)
As each node is being processed one time .