Geeks for Geeks problem discussion

Sofiullah Iqbal Kiron
Mail: sofiul.k.1023@gmail.com [1] and
Muhammad Ali Nirob[2]

[1]BSMRSTU, Department of CSE
[2]Markazul Suffah, Madani

10 April, 2020

Fulfilment

1 Is Binary Number Multiple of 3

Problem Link: LINK

1.1 My solution

A number will be divisible by 3 when the sum of all digits of this number is divisible by 3. Means, if sum of digits of a number is divisible by 3, the number is divisible by 3. School concept, COOL.
It's a long processing method, not efficient.

1.1.1 Algorithm

1. Take a binary number string ”s”
2. Create a Add() function for add two string
3. Create another function for returning power of 2 by string namely Pow2()
4. convert it to decimal, digit by digit, store in another string named ”deci”;
5. All apart of deci add to ”decimal” string.
6. Create a function for give ’long long int’ sum of ”decimal”
7. If sum of all digits are divisible by 3, then the given number is divisible by 3, return true value, 1.
   Else return false, 0.

1.1.2 Code

```

#include<iostream>
#include<algorithm>
#define lli long long int
using namespace std;

void Swap(string &s1, string &s2)
{
    string temp;
    temp = s1;
    s1 = s2;
    s2 = temp;
}

string Add(string s1, string s2)
{
    string result;
    int cs, carry=0;
    if(s1.size()<s2.size())
    {
        Swap(s1, s2);
    }
    int c1, c2;
    c1 = s1.size();
    c2 = s2.size();
    reverse(s1.begin(), s1.end());
    reverse(s2.begin(), s2.end());
    for(int i=c2; i<c1; i++)
    {
        s2.push_back('0');
    }
    for(int i=0; i<c1; i++)
    {
        cs = (s2[i]-'0')+(s1[i]-'0')+carry;
        result.push_back(cs%10+'0');
        carry=cs/10;
```

```cpp
    }
    if(carry)
    {
        result.push_back(carry+'0');
    }
    reverse(result.begin(), result.end());

    return result;
}

string Pow2(int n)
{
    if(n==0)
    {
        return "1";
    }
    else if(n==1)
    {
        return "2";
    }
    string a = "2";
    int i;
    for(i=1; i<n; i++)
    {
        a = Add(a, a);
    }

    return a;
}

string bin_To_deci(string s)
{
    string decimal="0", deci;
    lli i, j;
    for(i=s.size()-1, j=0; i>=0; i--, j++)
    {
        if(s[i]=='1')
        {
            deci=Pow2(j);
            decimal=Add(decimal, deci);
        }
    }

    return decimal;
}

lli digit_sum(string s)
{
    lli sum = 0;
    string ss = bin_To_deci(s);
    for(int i=0; i<ss.size(); i++)
    {
        sum+=(ss[i]-'0');
    }
    return sum;
}

bool isM3(string s)
{
    if(digit_sum(s)%3==0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

int main()
{
    int t;
    cin >> t;
    while(t--)
    {
        string s;
        cin >> s;
        if(isM3(s)==true)
        {
            cout << true << endl;
        }
        else
        {
            cout << false << endl;
        }
    }
}
```

1.2 Geeks official solution

Link: Write an Efficient Method to Check if a Number is Multiple of 3.
There is a pattern in the binary representation of the number. If A: sum of odd places bits, B: sum of even places bits; (A-B) mod 3 is equal to 0; then the number is divisible by 3;

1.2.1 Proof of this method by number theory

Let us consider a 4 digit binary number: $abcd$
Separate digits by place value. (Binary is 2 based number)
$abcd = 2^3a + 2^2b + 2c + d$
$= 8a + 4b + 2c + d$
$= (9a - a) + (3b + b) + (3c - c) + d$
$= (9a + 3b + 3c) - a + b - c + d$
$= (9a + 3b + 3c) + (b + d) - (a + c)$
Here $(9a + 3b + 3c)$ is divisible by 3.
So we can say that, the number will be divisible by 3 if and only if (iff), difference of sum of odd place bits and sum of even place bits is divisible by 3.
Considering one thing, there is no effect that how many 0 bits we add for processing. So, if we count only ”true” bit, that is enough.

1.2.2 Algorithm

1. Take a binary string namely ”bs”
2. Calculate size of string;
3. If size is 1:
   (a) If bs is equal to ”0”, return ”true”
   (b) Else, return ”false”
4. Else:
   (a) Count or sum odd place non-zero bits, store in a variable namely A
      - Write a for loop, initialize to bs.size()-1, decrement two by two, if current character is ’1’, increment A
      - When loop variable becomes 0, over.

   (b) Count or sum even place non-zero bits, store in a variable namely B
   (c) When loop variable becomes 0, over.
      - Write a for loop, initialize to bs.size()-2, decrement two by two, if current character is ’1’, increment B

   (d) If $(A - B) mod\ 3$ is equal to 0, return ”true”
   (e) Else return ”false”

1.2.3 Code

```
#include<iostream>
#define mod %
using namespace std;

bool isM3(string s)
{
    if(s.size()==1)
    {
        if(s=="0")
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    int i, A, B;
    for(i=s.size()-1, A=0; i>=0; i-=2)
    {
        if(s[i]=='1')
        {
            A++;
        }
    }
    for(i=s.size()-2, B=0; i>=0; i-=2)
    {
        if(s[i]=='1')
        {
            B++;
        }
    }
    if((A-B) mod 3 == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

int main()
{
    int T;
    cin >> T;
    while(T--)
    {
        string bs;
        cin >> bs;
        if(isM3(bs)) /*No equal statement means,
                       if (true or non-zero)*/
        {
            cout << true << endl;
        }
        else
        {
            cout << false << endl;
        }
    }
}
```

2 Contiguous subarray with largest sum

Given an array, we need to find a subarray of this such as, sum of all elements of this array must be largest from another subarrays.For implement this, here are many algorithm with $O(n^6), O(n^3), O(n^2)$ but Kadane's algorithm is more efficient thats complexity is linear, $O(n)$. Ow, it's so simple. So let's get started...
Problem source.

2.1 Kadane's algorithm

- Declaring two variable,

3 Majority Element

Problem source is here.

3.1 My Approach

This is the matter of sorrow that, my approach is not accepted, showing time limit exit, because its time complexity gonna be $O(n^2)$.

3.1.1 Algorithm

1. Count frequency of each elements and store in another array.
2. Search in count array for major purpose.
3. If: found, print the array and break.
4. Else: loops over, print -1.

3.1.2 Code

```cpp
using namespace std;

int main()
{
    int t;
    cin >> t;
    while(t--)
    {
        int n, i, major;
        bool major_exist=false;
        cin >> n;
        major=n/2;
        int arr[n];
        for(i=0; i<n; i++)
        {
            cin >> arr[i];
        }
        for(i=0; i<n; i++)
        {
            if(count(arr, arr+n, arr[i])>major)
            {
                major_exist=true;
                cout << arr[i] << endl;
                break;
            }
        }
        if(major_exist==false)
        {
            cout << "-1" << endl;
        }
    }
}
```

Listing 1: Majority Element 1'st code

So, it can be said simply, my approach is a f*****g approach.

3.2 Solution by sorting

Now we take approach from geeks.
Time complexity for this approach $O(n\log(n))$.

n	$O(n\log(n))$
3	1.43136376416
5	3.49485002168

So, we can easily say that, this approach will take least time.
Auxiliary space $O(n)$.

3.2.1 Algorithm

1. Declare

   $major = \frac{sizeofarray}{2}$.

2. Declare

   bool major_exist = false;

3. Sort the array and create two variable ”count” and ”previous”.

   count = 1;

4. Traverse array from start to end.
5. If current element is equal to the previous element, increase count by one.
6. When

   count >= major+1;

   change value of major_exist to true and print the current value, then break.
7. If major_exist till false then print -1.
8. Over.

3.2.2 Code

```cpp
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int t;
    cin >> t;
    while(t--)
    {
        int n, i, c=1, major;
        bool major_exist = false;
        cin >> n;
        major=n/2;
        int arr[n];
        for(i=0; i<n; i++)
        {
            cin >> arr[i];
        }
        sort(arr, arr+n);
        for(i=0; i<n; i++)
        {
            if(arr[i]==arr[i+1])
            {
                c++;
            }
            else
            {
                c=1;
            }
            if(c>=major+1)
            {
                major_exist=true;
                cout << arr[i] << endl;
                break;
            }
        }
        if(major_exist==false)
        {
            cout << "-1" << endl;
        }
    }
}
```

Listing 2: Majority Element $2'nd$ code

4 Leaders in an array

Link
First, me approach uses two nested loop. That's why I got Time Limit Exceed. So, when got TIME LIMIT EXCEED, try to overcome nested loop. It causes increase order of time complexity. Try to use another STL Or simple loops but not nested.

4.1 Geeks Hint

4.1.1 Process

Geeks hint was:

Scan all the elements from right to left in array and keep track of maximum till now. When maximum changes it's value, print it.

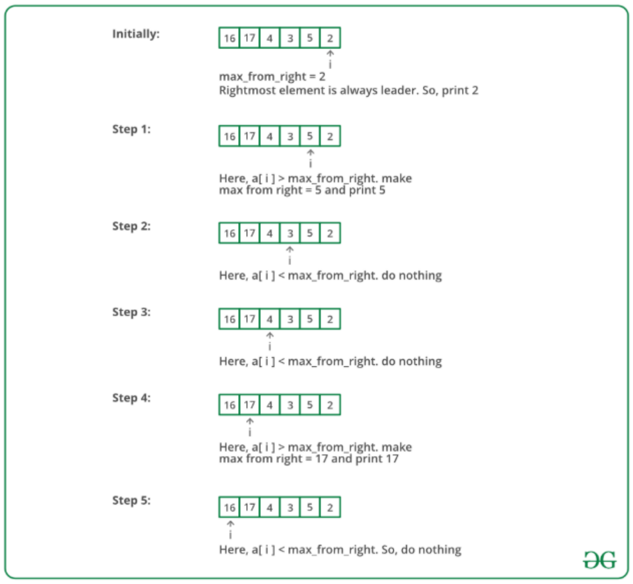
4.1.2 My code by hints

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int t;
    cin >> t;
    while(t--)
    {
        int n, i, j;
        list<int> L;
        cin >> n;
        int arr[n];
        for(i=0; i<n; i++)
        {
            cin >> arr[i];
        }
        int maxx=-1;
        for(i=n-1; i>=0; i--)
        {
            if(arr[i]>=maxx)
            {
                L.push_front(arr[i]);
                maxx=arr[i];
            }
        }
        list<int> :: iterator it;
        for(it=L.begin(); it!=L.end(); it++)
        {
            cout << *it << " ";
        }
        cout << endl;
    }
}
```

Listing 3: Leaders in an array

4.1.3 Picture Run From Geeks


Initially:
16 17 4 3 5 2
i
max_from_right = 2
Rightmost element is always leader. So, print 2
Step 1:
Here, a[i] > max_from_right. make
max from right = 5 and print 5
Step 2:
Here, a[i] < max_from_right. do nothing
Step 3:
Here, a[i] < max_from_right. do nothing
Step 4:
Here, a[i] > max_from_right. make
max from right = 17 and print 17
Step 5:
Here, a[i] < max_from_right. So, do nothing

4.1.4 Geeks official code

```cpp
#include <bits/stdc++.h>
using namespace std;

long long a[10000000];
vector<long long>v;

int main()
{
    long long t;
    cin >> t;
    while (t--)
    {
        long long n;
        cin >> n;
        for(long long i =0;i<n;i++)
        {
            cin >> a[i];
        }
        long long max = a[n-1];
        for(long long i =n-1; i>=0; i--)
        {
            if(a[i] >= max)
            {
                max = a[i];
                v.push_back(max);
            }
        }
        reverse(v.begin(), v.end());
        for(auto it = v.begin(); it!=v.end(); it++)
        {
            cout << *it << " ";
        }
        v.clear();
        cout << endl;
    }
}
```

OVER