Sparse Table and Fenwick Tree (Binary Indexed Tree or BIT)

Youtube link :
https://youtu.be/ecJDSQov5eA

Contents:

Introduction to Range Query

General Format of Range Query Problem

Given any array Arr of size n, and a function f .
Given q queries of the form L R.
Find the value of function f over the range [L,R].

Q. Given any array Arr of size n.
Given q queries of the form L R.
Find the value of sum of all elements in the range [L,R].

Eg. In the below array, find sum in range [2,5].

$$\sum_{i=2}^{5} \mathtt{arr}[i] = 6 + 4 + 2 + 5 = 17.$$

Index i	1	2	3	4	5	6
$\mathtt{arr}[i]$	1	6	4	2	5	3

(1-based indexing)

Naive Approach:

```
while(q--)
{
  cin >> L >>R;
  int sum=0;
   for(int i=L; i<=R; i++)
   {
      sum=sum+arr[i];
   }
}
```

Time complexity: O(q*n)

When $q<=10^5$, $n<=10^5$, this approach will give TLE
So, we use prefix arrays.

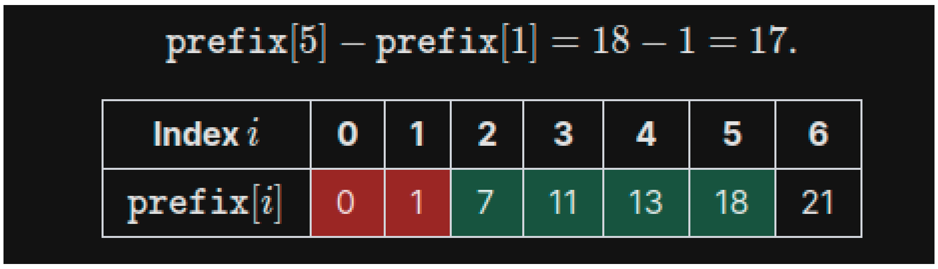
$$\mathtt{prefix}[k] = \sum_{i=1}^{k} \mathtt{arr}[i]$$

$$\mathtt{prefix}[k] = \mathtt{prefix}[k-1] + \mathtt{arr}[k]$$

```cpp
vector<int> prefix(n+1);
prefix[0]=0;
for(int i=1; i<=n; i++)
{
    prefix[i]=prefix[i-1]+arr[i];
}
```

```cpp
while(q--)
{
cin>>L>>R;
cout<<prefix[R] - prefix[L-1];
}
```

Time complexity: O(n + q)
O(n) for pre-computation
O(q) for answering all queries in O(1)



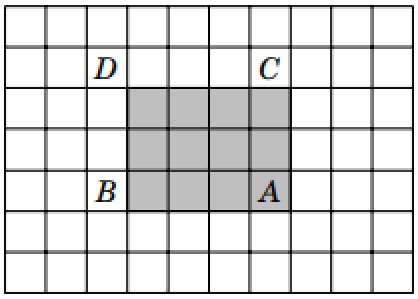
$$\mathtt{prefix}[5] - \mathtt{prefix}[1] = 18 - 1 = 17.$$

Index i	0	1	2	3	4	5	6
$\mathtt{prefix}[i]$	0	1	7	11	13	18	21

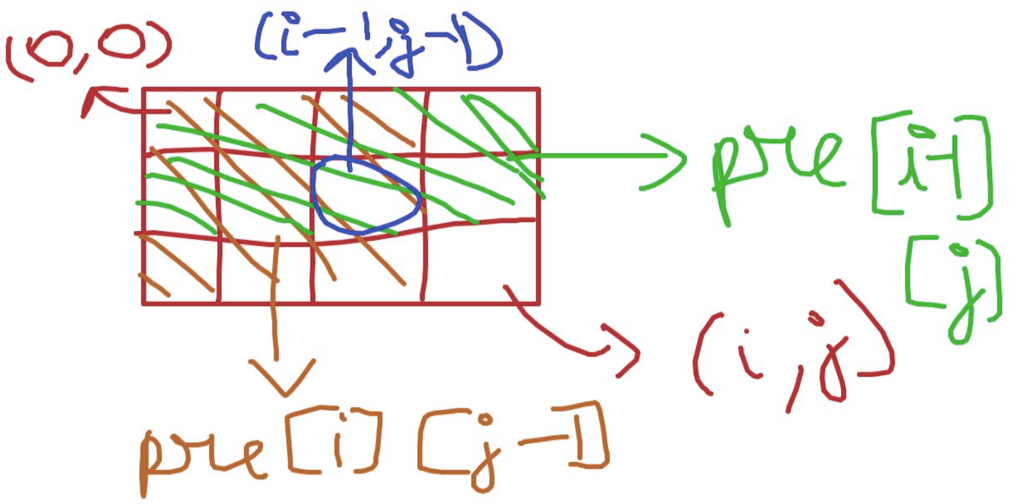
$$\sum_{i=L}^{R} \mathtt{arr}[i] = \mathtt{prefix}[R] - \mathtt{prefix}[L-1]$$

2-D Range Query

Q. Given a 2-D array of size n x m. (n x m <=10^6)
Given q queries (q <= 10^5) of the form: i j u v . For each query, output the sum of all elements lying in the rectangle between (i,j) and (u,v).
For example, in figure D = (i,j) and A = (u,v)



Let ar(S) = area of rectangle with bottom right corner at S
Area of shaded region = ar(A) - ar(B) - ar(C) + ar(D)
Suppose, pre[i][j] = sum of all elements in rectangle between (0,0) and (i,j)



Consider, 1-based indexing.

```
vector<vector<int> > pre(n+1, vector<int>(m+1,0));
// int pre[n+1][m+1]; // initialise all elements with
0
for(int i=1; i<=n; i++)
{
    for(int j=1; j<=m; j++)
    {
pre[i][j] = pre[i-1][j] + pre[i][ j-1] - pre[i-1][j-1]
+ arr[i][j];
    }
}
```

```
while(q--)
{
  cin>>i>>j>>u>>v;
  cout<< pre[u][v] - pre[u][j-1] - pre[i-1][v] +
pre[i-1][j-1];
}
```

Try these problem:
1. https://cses.fi/problemset/task/1646
2. https://cses.fi/problemset/task/1652

Q. Given any array Arr of size n.
Given q queries of the form L R.
Find the value of bitwise xor (^) of all elements in the range [L,R].

(3 properties of XOR:
1. A ^ A = 0
2. A ^ 0 = A
3. A ^ (B ^ C) = (A ^ B) ^ C [XOR is Associative]
)

Suppose, you have a number p = a^b^c^d and you have the value of a, you need to find s = b^c^d ?
-> a^p = a^(a^b^c^d) = (a^a)^(b^c^d) = 0 ^(b^c^d)
        = b^c^d = s
This shows Bitwise XOR (^) is a reversible function.
(You can remove the contribution of any element from ^)

Therefore, we can use prefix array to find bitwise xor (^) of elements in a given range [L, R]

Suppose, you have a number p = min(a,b,c,d) and you have the value of a, you need to find s = min(b,c,d) . Can u find s ?
-> No.
This shows "Minimum" is not a reversible function.
(You can't remove the contribution of any element from minimum)

Therefore, we can't use prefix array to find minimum of elements in a given range [L, R]

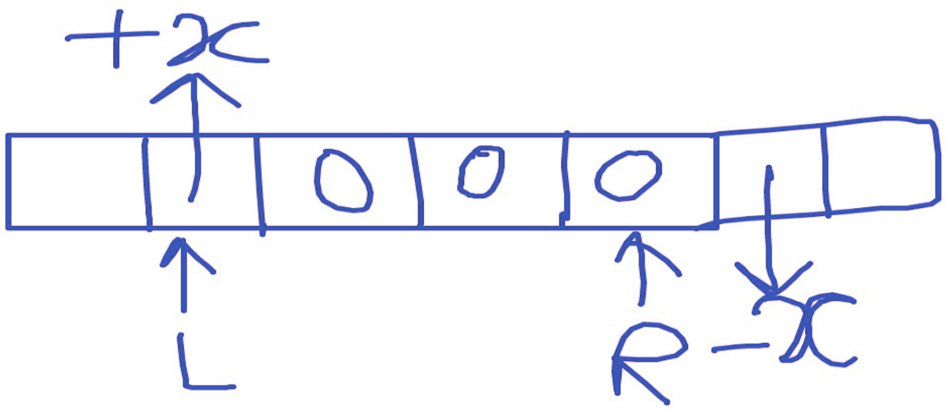
Try to solve this question:
https://cses.fi/problemset/task/1650

Note: For all functions which are **associative** and **reversible**, you can use prefix arrays to answer range queries.

Range Query and Point Update

Q. Given any array Arr of size n .(n<= 10^5) .
Given q (q<= 10^5) queries of the form L R X.
For each query, increase all elements of Arr in range [L,R] by a value X.
Find the final array Arr.

Solution:
We can use a difference array "diff" here.
diff stores the difference of 2 consecutive elements.



```
vector<int> diff(n+1);

while(q--)
{
  cin>>L>>R>>X;
  diff[L] = diff[L]+X;
  diff[R+1] = diff[R+1]-X;
}
```

For final array, just find the prefix sum of this difference array.

```
arr[0]=diff[0];
for(int i=1; i<=N; i++)
{
   arr[i]=arr[i-1]+diff[i];
}
```

Disadvantages of Prefix Array:
1. It is useful only to answer range query of operations which are reversible, like sum, xor, etc.
2. It is useful for static arrays only.

Sparse Table

- Sparse Table can be used when the function **f** is **associative**.
- It can answer most of the queries in $O(\log_2 N)$
- But its power lies in answering range minimum queries / range maximum queries , etc. in O(1).

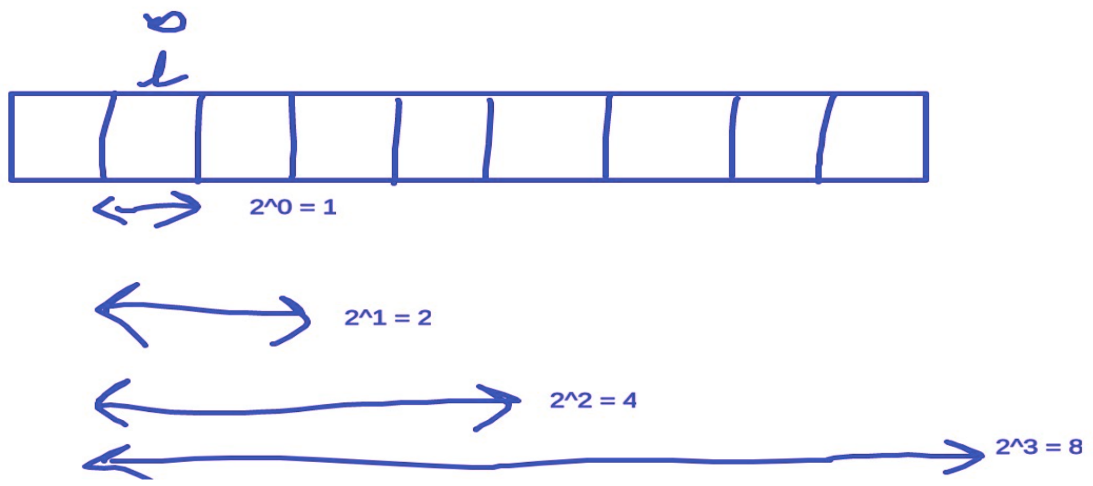
Intuition:
- Any non-negative number can be expressed as the sum of decreasing powers of 2.

(See binary representation of that number)
Eg. 13 = (1101) in binary
$13 = 2^3 + 2^2 + 2^0$

- For any number, there are $O(\log_2 N)$ terms in this expression.

- For any range / interval, the same thing applies
eg. [2, 14] = [2, 9] U [10,13] U [14,14]

Pre-Computation:



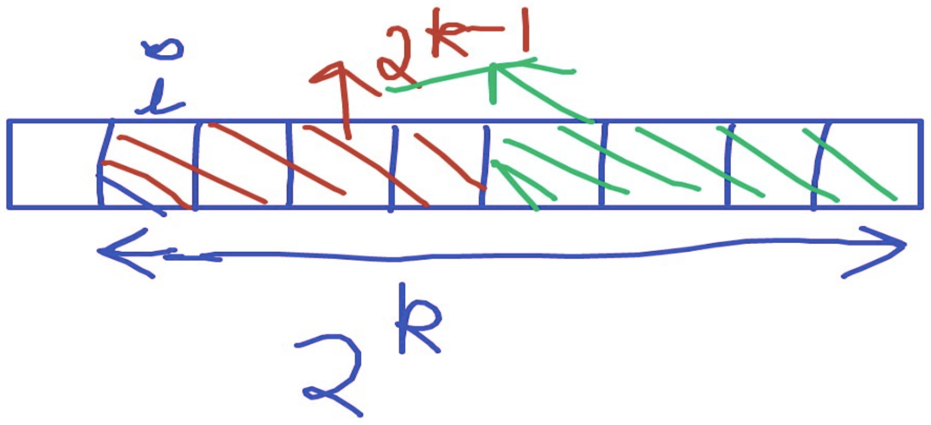
For sum function-

st[i][k] = sum of all elements in range $[i, i+ 2^k)$

```
const int MAXK=25; // For N<= 107
// You can also take MAXK = log2(N) + 1;
int st[ N ] [ MAXK + 1 ];
```

```
for(int i=0; i<N; i++)
{
    st[i][0] = arr[i];
}
for(int k=1; k <= MAXK ; k++)
{
  for(int i=0; i+(1<<k)<=N; i++)
  { // To prevent segmentation fault,
// Loop goes until the range is within the array
       st[i][k] = st[i][k-1]
                   + st[ i + (1<<(k-1)) ][k-1];
  }
}
```

The concept involved in pre-computation is that every range of length 2^k can be divided into 2 equal ranges of length $2^{(k-1)}$



Space Complexity of pre-computation: O(N log N)
Time Complexity of pre-computation: O(N log N)

Query for sum:

Suppose, Range is [L,R]

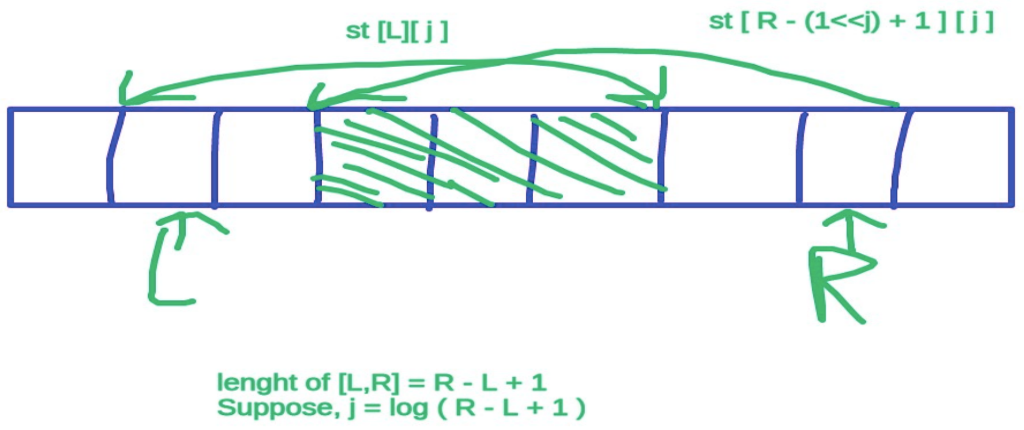
```
int sum=0;
for(int k= MAXK; k>=0; k--)
{
    if ( (1<<k) <= R - L + 1 )
    {
        sum += st[L][k];
        L += 1<<k;
    }
}
```

Time complexity: O($\log_2(N)$)

Query for Range Minimum / Maximum / GCD

When f(x) is an **idempotent** function, we can query in O(1).
f(x,x) = x for all x
Example: min(), max(), __gcd(), lcm(), & (bitwise and), | (bitwise or)
But sum is not an idempotent function. Similarly, xor is also not an idempotent function.

- Overlapping ranges work fine for idempotent functions.


st [L][j]
st [R - (1<<j) + 1] [j]
L
R
lenght of [L,R] = R - L + 1
Suppose, j = log (R - L + 1)

Please note that time complexity for calculating GCD is O(log(N)). So, range query for GCD will also take O(log N)

Algorithm

1. **Pre-compute** st table for minimum / maximum / gcd (as described above)

2. Pre-compute $\log_2$ of all values from 1 to n

$$\log i = \log\left(\frac{i}{2} \times 2\right)$$
$$= \log\left(\frac{i}{2}\right) + 1$$

```
int lg[ N + 1];
```

```
lg[1]=0;
for(int i=2; i<= N ; i++)
{
   lg[i] = lg[i/2] + 1;
}
```

3. **For each query**:

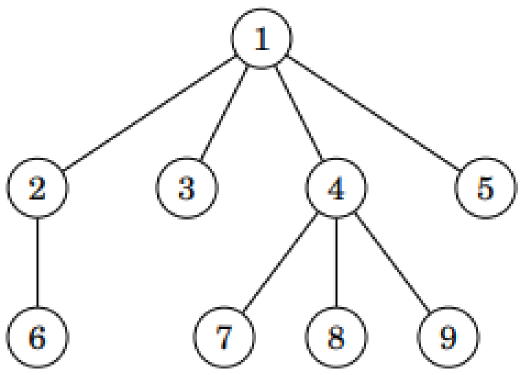
```
int j = log [ R - L +1 ];
int ans = min ( st[L][ j ], st[R - (1<<j) + 1][ j ]);
```

Time complexity for 1 query: O(1) for min,max
[O(log(N)) for GCD]
Try this problem:
http://www.spoj.com/problems/RMQSQ/

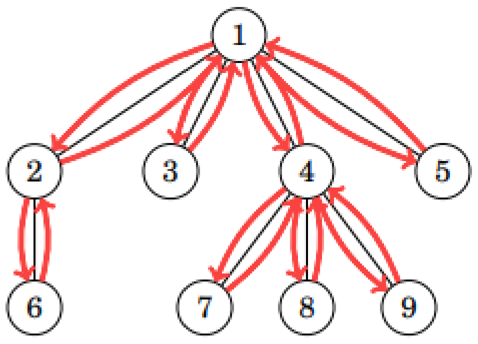
Disadvantage of Spare Table: It is useful for **static** arrays only. If you need to update values, in between queries, then sparse tables aren't efficient.

Converting Tree to Linear Array

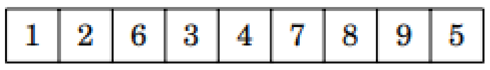
If we need to perform any range queries on subtrees in a tree, or in the path from root to any node, this technique is very helpful.



If you apply DFS on above tree starting from root node 1, it would be something like-



When a node is visited, push it to an array.
(this is also called **DFS Traversal Array**)

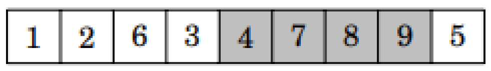


1	2	6	3	4	7	8	9	5

Subtree Queries

Now, all nodes in subtree of a node will become adjacent. Each subtree of the tree corresponds to a continuous subarray of the DFS traversal array.

For eg. shaded subarray here represents subtree of 4:

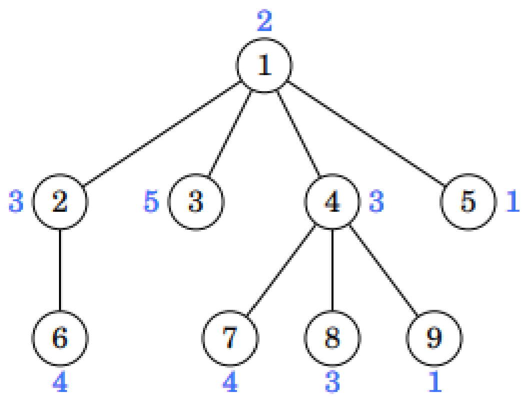


1	2	6	3	4	7	8	9	5

For subtree queries, in DFS, push nodes to an array (in the order of visiting the nodes), when it is visited. And also find the size of subtree of all nodes.

```
vector<int> vec;
dfs(int node,int par)
{
  cnt[node]=1;
  vec.push_back(node);
 for(auto ch: adj[node])
 {
   if(ch!=par)
   {
     dfs(ch,node);
     cnt[i]+=cnt[ch];
   }
 }
}
```

For example, in below figure, blue colour represents value of each node (would be given in question)



node id	1	2	6	3	4	7	8	9	5
subtree size	9	2	1	1	4	1	1	1	1
node value	2	3	4	5	3	4	3	1	1

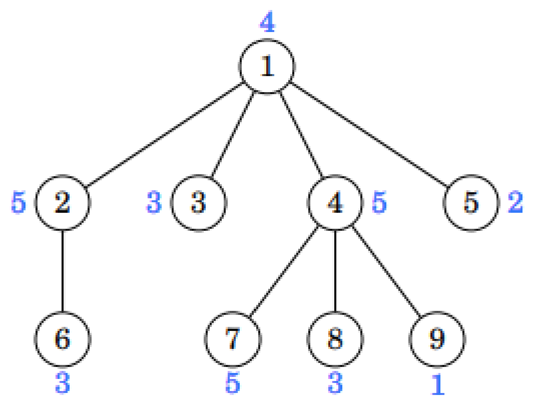
(For each node in the DFS traversal array, we need 3 things: node-id, its subtree size and its node value)

Shaded region below shows subtree of 4:

node id	1	2	6	3	4	7	8	9	5
subtree size	9	2	1	1	4	1	1	1	1
node value	2	3	4	5	3	4	3	1	1

Path Queries from root to node

Similar things can also work for path queries. Path from root to that node.
Consider a problem where our
task is to support the following queries:
• change the value of a node
• calculate the sum of values on a path from the root to a node
For example, in the following tree, the sum of values from the root node to node 7 is 4+5+5 = 14:



For each node in the DFS traversal array, we need 3 things: node-id, subtree size and the path sum from root to that node

node id	1	2	6	3	4	7	8	9	5
subtree size	9	2	1	1	4	1	1	1	1
path sum	4	9	12	7	9	14	12	10	6

When the value of a node increases by x, the sums of all nodes in its subtree increase by x. For example, if the value of node 4 increases by 1, the array changes as follows:

node id	1	2	6	3	4	7	8	9	5
subtree size	9	2	1	1	4	1	1	1	1
path sum	4	9	12	7	10	15	13	11	6

Thus, to support both the operations, we should be able to increase all values in a range and retrieve a single value. To support range updates in $O(\log_2 N)$, we use Fenwick Tree, as explained below.

Fenwick Tree (Binary Indexed Tree or BIT)

Q) CSES - Dynamic Range Sum Queries

Brute Force (using prefix sum array):

Time complexity/query:

query: sum(a,b)

pref[b]-pref[a-1]--> O(1)

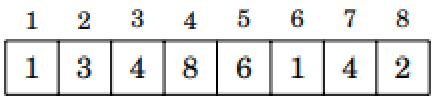
update: O(n)

arr[k]=u

and change the whole prefix array from kth index

Optimized:

Consider this array: **(1- based indexing)**

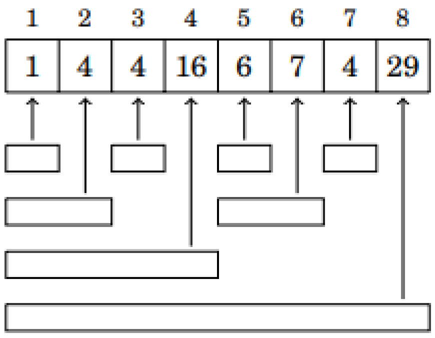


1	2	3	4	5	6	7	8
1	3	4	8	6	1	4	2

arr[3]=1
pref[5]=1+3+1+8+6
pref[2]=1+3
2->parent 4->parent 8
arr[4]+=arr[2]
arr[8]+=arr[2]

Corresponding Fenwick Tree (BIT) of the above array:

1	2	3	4	5	6	7	8
1	4	4	16	6	7	4	29



how to know parents/childs..
→ parent index > child index..
→ Add/subtract LSB
1 -> 2, 4, 8
arr[2]+=1
arr[4]+=1
arr[8]+=1

2 -> 4,8
arr[4]+=3
arr[8]+=3

arr[4]+=4
arr[8]+=4

1000 (8)
0111 (7) // Subtract LSB to get 0110
0110 (6) // Subtract LSB to get 0100

0101 (5)
0100 (4) // +0100
0011 (3) // +0001
0010 (2)
0001 (1)

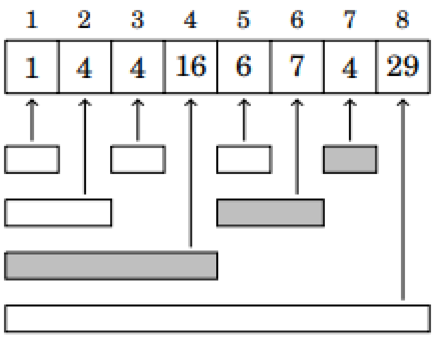
001010→ 001100→ 010000→ 100000
3->parent 4
5->parent 6

In Fenwick Tree, elements are organised by LSB (Least Significant Bit). We say, LSB is the range of responsibility.

Query:

pref(7) ? (Sum(1,7))
bit[7](arr[7])+bit[6]
(arr[5]+arr[6])+bit[4](arr[1]+arr[2]+arr[3]+arr[4])
0011-->0010-->0000
1111
pref(3)=bit[3]+bit[2]
pref(x)-->$\log_2(x)$
O($\log_2(x)$)-->pref(x)
pref(b)-pref(a-1)-->O($\log_2(n)$)
no of operations==no of 1s

For example
Range [1,7] consists of following ranges:



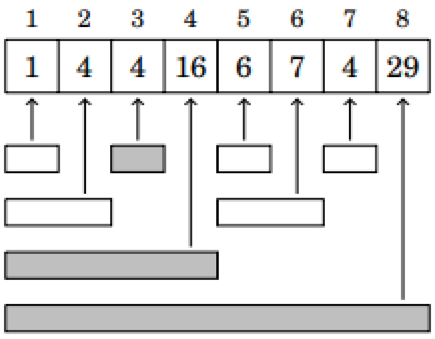
1. Start at current cell (7)
2. Turn the LSB off, until no bits are left.

Sum(1,7) = Sum(1,6) + bit[7]
= Sum(1,4) + bit[6] + bit[7]
= bit[4] + bit[6] + bit[7]

To find Sum (3,7) :
Sum(1,7) - Sum(1,2)
[Similar to prefix sum array]

Update:
arr[3]=k
no of operations==no of 0s(after lsb)==O($\log_2(n)$)
Overall complexity: q * O($\log_2(n)$) = O(q * $\log_2(n)$)

If a value at position 3 changes, the following ranges are updated:



1. Start at current cell (3)

2. Add the LSB, until the end of array is reached.
Update bit[3]
3 = 11 in binary, LSB = 1
3 + 1 = 4 = 100 in binary, LSB = 100 in binary (4)
Update bit[4]
4 + 4 = 8
Update bit[8]

→ **Depends completely on LSB(Least Significant Bit)**

Finding LSB of any number

Q. Given any integer x, find LSB of number x.
(Remember Bit Manipulation)
LSB of x is given by **x & (-x)**
where & is Bitwise AND
x == a1b
b->0s..
-x==2's compliment of x=x'+1==a'0b'+1==a'1b
(a1b)&(a'1b)==000000…(a)1(lsb)0000000
-->how to find the LSB??

Code for BIT

```
vector<int> bit(n+1,0);

// Use 1-based indexing for easy code

void update(int k,int val)
{
  while(k<=n)
  {
    bit[k]+=val;
    k+=(k&(-k));
  }
}

int sum(int k)
{
  int res=0;
  while(k>0)
  {
    res+=bit[k];
    k-=(k&(-k));
  }
return res;
}
```

```
int query(int l,int r)
{
 return sum(r)-sum(l-1);
  // Similar to prefix arrays
}
```

```
vector<int> arr(n+1);
for(int i=1;i<=n;i++)
 {
  cin>>arr[i];
  update(i,arr[i]);
  // initial construction of BIT
 }
```

Time complexity:

Query: $O(\log_2 n)$

Update: $O(\log_2 n)$

Q)SPOJ.com - Problem INVCNT

n->length array

inversion if: i<j ..arr[i]>arr[j]

n==3

3 1 2

ith index..(1….(i-1)) >arr[i]

maxn->max possible element..

vector<int> bit(maxn+1,0)

element ith-->arr[i]

pref(maxn)-pref(arr[i])

n==7

7 6 2 3 1 4 5
maxn==7
7-->0->1
6-->0->1
5-->0->1
4-->0->1
3-->0->1
2-->0->1
1-->0->1

int ans=0+1+2+2+4+2+2=13
n elements..each time pref sum then update

Q)CSES - List Removals

n==5
2 6 1 4 2
3
2 6 4 2
1
6 4 2
3
6 4
1
4
empty..

o/p-->1 2 2 6 4

n positions...

ith position ->present -->bit 1 store
                         else bit 0 store
bit[2]-=1
bit[x]-=1

for(int i=1;i<=n;i++)
update(i,1)

3rd position
1 0 1 0 0 0 0
pref(i)==2

pref(3)==2
pref(4)==2

pref(i)==3/x
3rd position

1 0 0 0 0
2nd position

0 0 0 0 0

xth position -->remove
index in

pref[in]==x

pref function non-decrease with index..

```
lo=1,hi=n,mid;
while(lo<hi)
{
  mid=lo+(hi-lo)/2;
  int tmp=sum(mid);
  // tmp is prefix sum till mid
  // tmp==x for first time
  if(tmp<x)
   lo=mid+1;
 else//tmp>=x
  hi=mid;
}

// lo-->pref(lo)==x for the first time..
cout<<arr[lo];
update(lo,-1);
}
```

Time complexity per query: O($(\log_2 n)^2$)

→ Alternate solution (Better complexity)
https://codeforces.com/blog/entry/61364

Note: For all functions which are **associative** and **reversible**, you can use BIT (Binary Indexed Tree or Fenwick Tree) to answer range queries.
For example, BIT can also be used for Bitwise XOR (^) .
But it can't be used for GCD() or min() or max().

Also, Try these problems:

1) CSES - Range Update Queries
2) CSES - Subtree Queries
3) CSES - Path Queries
4) CSES - Forest Queries II

If you are stuck, you can view my submissions below:
1. Submission to CSES - List Removals
https://cses.fi/paste/5d5c47d94270eba71474da/

2. Submission to CSES - Range Update Queries
https://cses.fi/paste/a638e6fd6c04b4df146f3a/

3. Submission to CSES - Subtree Queries
https://cses.fi/paste/3b013671b6c81329149639/

4. Submission to CSES - Path Queries
https://cses.fi/paste/0c5cf473c9a28e48149791/

5. Submission to CSES - Forest Queries II
https://cses.fi/paste/a0a4cc51311a4ec6148038/