1 Introduction

Data structure is the way to store and organize data in computers. From small variables and arrays to large segment trees and balanced trees, they are all data structures.

Programs cannot run without data structures. Different data structures have their own advantages and disadvantages, and can handle different problems. Choosing the right data structure according to the specific problem can greatly improve the efficiency of the program. Therefore, it is necessary to learn a variety of data structures.

Without specified mentioned, assume `arr` is declared as a signed integer array. The following code is added to the beginning of all C++ programs.

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

2 Sparse Table

Definition

Sparse Table is a data structure based on **Dynamic Programming** and **Binary Lifting** to answer ***Range Minimum/Maximum Query* (RMQ) Problems** in **O(1)** time. It works only for **immutable datum** (no updates).

Binary Lifting

When we are recursing, if the state space is very large and the usual linear recursion cannot meet the requirements of time and space complexity, then we can use the method of binary lifting to recurse only the values at the integer power position of k (often 2) in the state space as representatives.

When values at other positions are needed, we use the property that **"any integer can be expressed as the sum of several powers of k"** to use the previously calculated representative values to piece together the required values. (Recall the binary grouping method in DP(III))

Sparse Table

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
x=3	8							
x=2	4	4	5	8	8			
x=1	4	4	2	2	5	8	8	
x=0	2	4	2	1	2	5	8	0

Let k be the maximum integer such that $R - L + 1 \geq 2^k$

$[L, L + 2^k - 1]$ and $[R - 2^k + 1, R]$ must cover all positions from L to R

$\therefore$ find(L, R) = max(f(L, k), f(R - 2^k + 1, k))

Time Complexity

Preprocessing: **O(N log N)**

Idempotent operation queries: **O(1)**

- max / min / and / or / gcd

Non-idempotent operation queries: **O(log N)**

- sum / product / xor

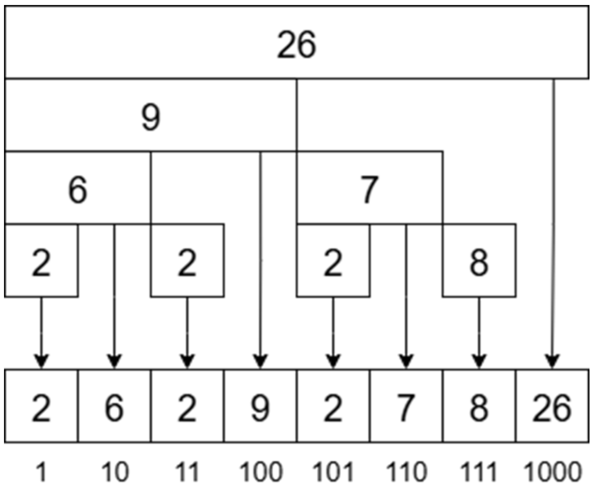
C++ Code Implementation

```
// K >= floor(log2(MAXN))
// for MAXN around 1e7, K = 25 is a good value.
int ST[MAXN][K + 1];
void build() {
    for (int i = 1; i <= N; i++) ST[i][0] = arr[i];
    for (int x = 1; x <= K; i++)
        for (int i = 1; i + (1 << x) <= N; i++) // (1 << x) means 2^x
            ST[i][x] = max(ST[i][x - 1], ST[i + (1 << (x - 1))][x - 1]);
}
int query() {       // max query
    int k = floor(log2(R - L + 1));
    return max(ST[L][k], ST[R - (1 << k) + 1][i]);
}
```

2 Fenwick Tree

Definition

Fenwick Tree is also called Binary Indexed Tree (BIT). It supports point updates and range queries.



A BIT representation of the prefix sum of the data set {2, 4, 2, 1, 2, 5, 8, 0}.

Lowbit Function

Let lowbit(x) be the value of the rightmost "1" in binary representation of x.

- **lowbit(x) = x & -x**

In BIT, node x stores the information of interval [x - lowbit(x) + 1, x].

Point Update

By observation, you will find that the length of next BIT[i] we need to update would be one of current BIT[i] more.

i.e. After updating BIT[i], we have to update BIT[i + (i & -i)].

Time Complexity

Preprocessing: **O(N log N)**

Query: **O(log N)**

C++ Code Implementation

```
int lowbit(int x) {return x & -x;}

void add(int x, int k) {
    while (x <= n) {
        BIT[x] += k;
        x += lowbit(x);
    }
}
int query(int x) {  // sum of BIT[1]..BIT[x]
    int ans = 0;
    while (x > 0) {
        ans += BIT[x];
        x -= lowbit(x);
    }
    return ans;
}
```