Team: Zero^Infinity(DUET)

*****Segment Tree**

```
#define mx 100001
using namespace std;
int arr[mx];
int tree[mx * 3];

void init(int node, int b, int e){
   if (b == e) {
      tree[node] = arr[b];
      return;
   }
   int Left = node * 2;
   int Right = node * 2 + 1;
   int mid = (b + e) / 2;
   init(Left, b, mid);
   init(Right, mid + 1, e);
   tree[node] = tree[Left] + tree[Right];
}
int query(int node, int b, int e, int i, int j){
   if (i > e || j < b)
      return 0; //out of range
   if (b >= i && e <= j)
      return tree[node]; //Full range interset
   int Left = node * 2; // more divided
   int Right = node * 2 + 1;
   int mid = (b + e) / 2;
   int p1 = query(Left, b, mid, i, j);
   int p2 = query(Right, mid + 1, e, i, j);
   return p1 + p2; //left and right side summation
}
void update(int node, int b, int e, int i, int newvalue){
   if (i > e || i < b)
      return; //out of range
   if (b >= i && e <= i) { //Full range interset
      tree[node] = newvalue;
      return;
   }
   int Left = node * 2; //more divided
   int Right = node * 2 + 1;
   int mid = (b + e) / 2;
   update(Left, b, mid, i, newvalue);
   update(Right, mid + 1, e, i, newvalue);
   tree[node] = tree[Left] + tree[Right];
}
int main()
{
   int n;
   cin >> n;
   for(int i = 1; i <= n; i++){
      cin >> arr[i];
   }
   init(1, 1, n);
   update(1, 1, n, 2, 0);
   cout << query(1, 1, n, 1, 3) << endl;
   update(1, 1, n, 2, 2);
   cout << query(1, 1, n, 2, 2) << endl;
   return 0;
}
```

***** Segment tree with lazy propagation**

```
#define mx 100001
#define ll long long int
int arr[mx];
struct info {
   ll prop, sum;
} tree[mx * 4]; //sum and extra update value added prop
void update(int node, int b, int e, int i, int j, ll x)
{
   if (i > e || j < b)
      return;
   if (b >= i && e <= j) //Full range interset
   {
      tree[node].sum += ((e - b + 1) * x); //all down node e-
b+1, so,  (e-b+1) * x added.
      tree[node].prop += x; //all down node sum with x
      return;
   }
   int Left = node * 2;
   int Right = (node * 2) + 1;
   int mid = (b + e) / 2;
   update(Left, b, mid, i, j, x);
   update(Right, mid + 1, e, i, j, x);
   tree[node].sum = tree[Left].sum + tree[Right].sum + (e - b
+ 1) * tree[node].prop;
   //up all node added with prop value
   // left and right node sum + extra summation
}
int query(int node, int b, int e, int i, int j, int carry = 0){
   if (i > e || j < b)
      return 0;

   if (b >= i and e <= j)
      return tree[node].sum + carry * (e - b + 1); //sum +
extra value
   int Left = node * 2;
   int Right = (node * 2) + 1;
   int mid = (b + e) / 2;

   int p1 = query(Left, b, mid, i, j, carry +
tree[node].prop); //prop value + carray
   int p2 = query(Right, mid + 1, e, i, j, carry +
tree[node].prop);

   return p1 + p2;
}

void init(int node, int b, int e){
   if (b == e) {
      tree[node].sum = arr[b];
      return;
   }
   int Left = node * 2;
   int Right = node * 2 + 1;
   int mid = (b + e) / 2;
   init(Left, b, mid);
   init(Right, mid + 1, e);
   tree[node].sum = tree[Left].sum + tree[Right].sum;
}
```

Team: Zero^Infinity(DUET)

```
int main()
{
   int n;
   cin >> n;
   for(int i = 1; i <= n; i++){
      cin >> arr[i];
   }
   init(1, 1, n);
   update(1, 1, n, 1, 7, 2);
   update(1, 1, n, 1, 4, 3);
   cout << query(1, 1, n, 1, 7) << endl;
   cout << query(1, 1, n, 1, 4) << endl;
   return 0;
}
```

***last_ans_first_Power_digits**

```
ll lastDigits(ll base, ll pw, ll Mod){
        // Last 3 digit just MOD = 1000
        ll ans = 1;
        while(pw > 0){
                if(pw & 1){
                        ans = (ans * base) % Mod;
                }
                pw >>=1L;
                base = (base*base) % Mod;
        }
        return ans;
}
```

***longest_non_decreasing_subsequnce**

```
ll firstDigits(ll n, ll k){
        long double power;
        power = (double)k * log10(n);
        //cout << power - floor(power) << endl;
        ll ans = pow(10, power - floor(power)) * 100.0;
        return ans;
}
int lengthOfLIS(vector<int>& nums) {
   vector<int> v;
   vector<int>::iterator it;
   for(int i = 0; i < nums.size(); i++){
      it = upper_bound(v.begin(), v.end(), nums[i]);
      if(v.end() == it) v.push_back(nums[i]);
      else v[it-v.begin()] = nums[i];
   }
   return v.size();
}
```

***Policy-based data structures**
order_of_key(n); return n data position
find_by_order(n); return n index data

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

typedef tree<int,null_type,less<int>,rb_tree_tag,
tree_order_statistics_node_update> indexed_set;
indexed_set s;
int main(){
        s.insert(2);
        s.insert(3);
        s.insert(7);
        s.insert(9);
        auto x = s.find_by_order(2);
        cout << *x << "\n"; // 7
        cout << s.order_of_key(7) << "\n"; // 2
        cout << s.order_of_key(6) << "\n"; // 2
        cout << s.order_of_key(8) << "\n"; // 3
return 0;
}
```

***Finding_bridge**

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
   visited[v] = true;
   tin[v] = low[v] = timer++;
   for (int to : adj[v]) {
      if (to == p) continue;
      if (visited[to]) {
         low[v] = min(low[v], tin[to]);
      } else {
         dfs(to, v);
         low[v] = min(low[v], low[to]);
         if (low[to] > tin[v])
            IS_BRIDGE(v, to);
      }
   }
}
void find_bridges() {
   timer = 0;
   visited.assign(n, false);
   tin.assign(n, -1);
   low.assign(n, -1);
   for (int i = 0; i < n; ++i) {
      if (!visited[i])
         dfs(i);
   }
}
```

*** **My Templete**

```
#include <bits/stdc++.h>
#define int int64_t
#define endl '\n'
#define Faster ios::sync_with_stdio(false);cin.tie(nullptr);
#define CHECK(x) cout << (#x) << " is " << (x) << endl;
using namespace std;
const int N = (int)1e5+5;
void solution(int tc){
        int n; cin >> n;
        std::vector<int> v(n);
        for(int i = 0; i < n; i++){
                cin >> v[i];
        }
}
```

```
int32_t main(){
	Faster;
	int tc = 1;
	cin >> tc;
	for(int i = 1; i <= tc; i++){
		solution(i);
	}
  return 0;
}
```

***** Important**

```
8 cell visit..
int fx[] = {-1,-1,-1,0,1,1,1,0};
int fy[] = {-1,0,1,1,1,0,-1,-1};
// 4 cell visit..
int fx[] = {-1,1,0,0};
int fy[] = {0,0,-1,1};
```

**** Setting Sublime Text**
c++.sublime-build

```
{
	"cmd" : ["g++ -std=c++17 $file_name -o
$file_base_name && timeout 4s
./$file_base_name<input.txt>output.txt"],
	"selector" : "source.c",
	"shell":true,
	"working_dir": "$file_path"
}
```

*****STL**

```
 LINKED LISTS
 ==============
 */
 list<int> mylist;
  list<int>::iterator it1,it2,itx;

  // set some values:
  for (int i=1; i<10; ++i) mylist.push_back(10*i);

  // 10 20 30 40 50 60 70 80 90
  it1 = it2 = mylist.begin(); // ^^
  advance (it2,6);          // ^             ^
  ++it1;                    //  ^            ^

  it1 = mylist.erase (it1);   // 10 30 40 50 60 70 80 90
  //  ^         ^

  it2 = mylist.erase (it2);   // 10 30 40 50 60 80 90
  //  ^         ^

  ++it1;                    //      ^       ^
  --it2;                    //      ^    ^

  mylist.erase (it1,it2);     // 10 30 60 80 90

  cout << "\nmylist contains:";
  for (itx=mylist.begin(); itx!=mylist.end(); ++itx)
    cout << ' ' << *itx;
  cout << '\n';

  // NOTE: it1 still points to 40, and 60 is not deleted
  cout << endl << *it1 << "\t" << *it2 <<endl;

  // This will print an unexpected value
  it1++;
  cout << *it1;

  cout << "\nmylist now contains:";
  for (it1=mylist.begin(); it1!=mylist.end(); ++it1)
    cout << ' ' << *it1;
  cout << '\n';
```

***** priority_queue**

```
priority_queue<int, vector<int>, greater<int> > gquiz;
```

***** Trie Algorithm**

```
// C++ implementation of search and insert
// operations on Trie
#include <bits/stdc++.h>
using namespace std;

const int ALPHABET_SIZE = 26;

// trie node
struct TrieNode
{
	struct TrieNode *children[ALPHABET_SIZE];

	// isEndOfWord is true if the node represents
	// end of a word
	bool isEndOfWord;
};
// Returns new trie node (initialized to NULLs)
struct TrieNode *getNode(void)
{
	struct TrieNode *pNode = new TrieNode;

	pNode->isEndOfWord = false;

	for (int i = 0; i < ALPHABET_SIZE; i++)
		pNode->children[i] = NULL;

	return pNode;
}
// If not present, inserts key into trie
// If the key is prefix of trie node, just
// marks leaf node
void insert(struct TrieNode *root, string key){
	struct TrieNode *pCrawl = root;

	for (int i = 0; i < key.length(); i++){
		int index = key[i] - 'a';
		if (!pCrawl->children[index])
			pCrawl->children[index] = getNode();
		pCrawl = pCrawl->children[index];
	}
	// mark last node as leaf
	pCrawl->isEndOfWord = true;
}
```

```
// Returns true if key presents in trie, else
// false
bool search(struct TrieNode *root, string key)
{
	struct TrieNode *pCrawl = root;

	for (int i = 0; i < key.length(); i++)
	{
		int index = key[i] - 'a';
		if (!pCrawl->children[index])
			return false;

		pCrawl = pCrawl->children[index];
	}

	return (pCrawl->isEndOfWord);
}

// Driver
int main()
{
	// Input keys (use only 'a' through 'z'
	// and lower case)
	string keys[] = {"the", "a", "there",
					"answer", "any", "by",
					"bye", "their" };
	int n = sizeof(keys)/sizeof(keys[0]);

	struct TrieNode *root = getNode();

	// Construct trie
	for (int i = 0; i < n; i++)
		insert(root, keys[i]);

	// Search for different keys
	char output[][32] = {"Not present in trie", "Present in
trie"};

	// Search for different keys
	cout<<"the"<<" --- "<<output[search(root,
"the")]<<endl;
	cout<<"these"<<" --- "<<output[search(root,
"these")]<<endl;
	cout<<"their"<<" --- "<<output[search(root,
"their")]<<endl;
	cout<<"thaw"<<" --- "<<output[search(root,
"thaw")]<<endl;
	return 0;
}
```

Insertion O(n), Searching O(n)