

# Chapter 1

# Contest

Template.py

```
1 import sys
2 from collections import *
3 from itertools import permutations #No repeated
   elements
4 sys.setrecursionlimit(10**5)
5 itr = (line for line in sys.stdin.read().strip().split(
   '\n'))
6 INP = lambda: next(itr)
7 def ni(): return int(INP())
8 def nl(): return [int(_) for _ in INP().split()]
9
10
11
12 def solve(n,a):
13     pass
14
15
16 t = ni()
17 for case in range(t):
18     n = ni()
19     a = nl()
20     solve(n,a)
```

Troubleshooting: Pre-submit: Write a few simple test cases if sample is not enough. Are time limits close? If so, generate max cases. Is the memory usage fine? Could anything overflow? Make sure to submit the right file.

Wrong answer: Print your solution! Print debug output, as well. Are you clearing all data structures between test cases? Can your algorithm handle the whole range of input? Read the full problem statement again. Do you handle all corner cases correctly? Have you understood the problem correctly? Any uninitialized variables? Any overflows? Confusing N and

M, i and j, etc.? Are you sure your algorithm works? What special cases have you not thought of? Are you sure the STL functions you use work as you think? Add some assertions, maybe resubmit. Create some testcases to run your algorithm on. Go through the algorithm for a simple case. Go through this list again. Explain your algorithm to a teammate. Ask the teammate to look at your code. Go for a small walk, e.g. to the toilet. Is your output format correct? (including whitespace) Rewrite your solution from the start or let a teammate do it.

Runtime error: Have you tested all corner cases locally? Any uninitialized variables? Are you reading or writing outside the range of any vector? Any assertions that might fail? Any possible division by 0? (mod 0 for example) Any possible infinite recursion? Invalidated pointers or iterators? Are you using too much memory? Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded: Do you have any possible infinite loops? What is the complexity of your algorithm? Are you copying a lot of unnecessary data? (References) How big is the input and output? (consider buffering output) What do your teammates think about your algorithm?

Memory limit exceeded: What is the max amount of memory your algorithm should need? Are you clearing all data structures between test cases?

# Chapter 2

# Mathematics

## 2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by  $x = -b/2a$ .

$$\begin{matrix} ax + by = e \\ cx + dy = f \end{matrix} \Rightarrow \begin{matrix} x = \frac{ed - bf}{ad - bc} \\ y = \frac{af - ec}{ad - bc} \end{matrix}$$

In general, given an equation  $Ax = b$ , the solution to a variable  $x_i$  is given by

$$x_i = \frac{\det A'_i}{\det A}$$

where  $A'_i$  is  $A$  with the  $i$ 'th column replaced by  $b$ .

## 2.2 Recurrences

If  $a_n = c_1a_{n-1} + \dots + c_ka_{n-k}$ , and  $r_1, \dots, r_k$  are distinct roots of  $x^k + c_1x^{k-1} + \dots + c_k$ , there are  $d_1, \dots, d_k$  s.t.

$$a_n = d_1r_1^n + \dots + d_kr_k^n.$$

Non-distinct roots  $r$  become polynomial factors, e.g.  $a_n = (d_1n + d_2)r^n$ .

## 2.3 Trigonometry

$$\sin(v+w) = \sin v \cos w + \cos v \sin w$$

$$\cos(v+w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$

$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$(V+W) \tan(v-w)/2 = (V-W) \tan(v+w)/2$$

where  $V, W$  are lengths of sides opposite angles  $v, w$ .

$$a \cos x + b \sin x = r \cos(x - \phi)$$

$$a \sin x + b \cos x = r \sin(x + \phi)$$

where  $r = \sqrt{a^2 + b^2}$ ,  $\phi = \text{atan2}(b, a)$ .

## 2.4 Geometry

### 2.4.1 Triangles

Side lengths:  $a, b, c$

Semiperimeter:  $p = \frac{a+b+c}{2}$

Area:  $A = \sqrt{p(p-a)(p-b)(p-c)}$

Circumradius:  $R = \frac{abc}{4A}$

Inradius:  $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles):  $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):  $s_a =$

$$\sqrt{bc \left[ 1 - \left( \frac{a}{b+c} \right)^2 \right]}$$

Law of sines:  $\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$

Law of cosines:  $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents:  $\frac{a+b}{a-b} = \frac{\tan \frac{\alpha+\beta}{2}}{\tan \frac{\alpha-\beta}{2}}$

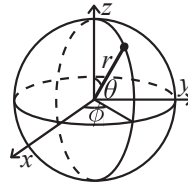
### 2.4.2 Quadrilaterals

With side lengths  $a, b, c, d$ , diagonals  $e, f$ , diagonals angle  $\theta$ , area  $A$  and magic flux  $F = b^2 + d^2 - a^2 - c^2$ :

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is  $180^\circ$ ,  $ef = ac + bd$ , and  $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$ .

### 2.4.3 Spherical coordinates



$$x = r \sin \theta \cos \phi \quad r = \sqrt{x^2 + y^2 + z^2}$$

$$y = r \sin \theta \sin \phi \quad \theta = \arccos(z / \sqrt{x^2 + y^2 + z^2})$$

$$z = r \cos \theta \quad \phi = \text{atan2}(y, x)$$

## 2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}} \quad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x \quad \frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \quad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \text{erf}(x) \quad \int x e^{ax} dx = \frac{e^{ax}}{a^2} (ax - 1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

## 2.6 Sums

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

## 2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)$$

## 2.8 Probability theory

Let  $X$  be a discrete random variable with probability  $p_X(x)$  of assuming the value  $x$ . It will then have an expected value (mean)  $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$  and variance  $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$  where  $\sigma$  is the standard deviation. If  $X$  is instead continuous it will have a probability density function  $f_X(x)$  and the sums above will instead be integrals with  $p_X(x)$  replaced by  $f_X(x)$ .

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent  $X$  and  $Y$ ,

$$V(aX + bY) = a^2 V(X) + b^2 V(Y).$$

2.8.1 Discrete distributions

Binomial distribution

The number of successes in  $n$  independent yes/no experiments, each which yields success with probability  $p$  is  $\text{Bin}(n, p)$ ,  $n = 1, 2, \dots$ ,  $0 \leq p \leq 1$ .

$$p(k) = \binom{n}{k} p^k (1 - p)^{n - k}$$

$$\mu = np, \sigma^2 = np(1 - p)$$

$\text{Bin}(n, p)$  is approximately  $\text{Po}(np)$  for small  $p$ .

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability  $p$  is  $\text{Fs}(p)$ ,  $0 \leq p \leq 1$ .

$$p(k) = p(1 - p)^{k - 1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1 - p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time  $t$  if these events occur with a known average rate  $\kappa$  and independently of the time since the last event is  $\text{Po}(\lambda)$ ,  $\lambda = t\kappa$ .

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between  $a$  and  $b$  and 0 elsewhere it is  $\text{U}(a, b)$ ,  $a < b$ .

$$f(x) = \begin{cases} \frac{1}{b - a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a + b}{2}, \sigma^2 = \frac{(b - a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is  $\text{Exp}(\lambda)$ ,  $\lambda > 0$ .

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean  $\mu$  and variance  $\sigma^2$  are well described by  $\mathcal{N}(\mu, \sigma^2)$ ,  $\sigma > 0$ .

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x - \mu)^2}{2\sigma^2}}$$

If  $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$  and  $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$  then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let  $X_1, X_2, \dots$  be a sequence of random variables generated by the Markov process. Then there is a transition matrix  $\mathbf{P} = (p_{ij})$ , with  $p_{ij} = \text{Pr}(X_n = i | X_{n - 1} = j)$ , and  $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$  is the probability distribution for  $X_n$  (i.e.,  $p_i^{(n)} = \text{Pr}(X_n = i)$ ), where  $\mathbf{p}^{(0)}$  is the initial distribution.

$\pi$  is a stationary distribution if  $\pi = \pi \mathbf{P}$ . If the Markov chain is *irreducible* (it is possible to get to any state from any state), then  $\pi_i = \frac{1}{\mathbb{E}(T_i)}$  where  $\mathbb{E}(T_i)$  is the expected time between two visits in state  $i$ .  $\pi_j / \pi_i$  is the expected number of visits in state  $j$  between two visits in state  $i$ .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors,  $\pi_i$  is proportional to node  $i$ 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1).  $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$ .

A Markov chain is an A-chain if the states can be

partitioned into two sets **A** and **G**, such that all states in **A**

are absorbing ( $p_{ii} = 1$ ), and all states in **G** leads to an absorbing state in **A**. The probability for absorption in state

$i \in \mathbf{A}$ , when the initial state is  $j$ , is  $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$ .

The expected time until absorption, when the initial state is

$i$ , is  $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$ .

Chapter 3

Graph Algorithms

bfs:

```
1 # S is index, G is adjacancy list
2 # finds distance from S to all verticies in G
3 def bfs(S, G):
4     q = [S]
5     INF = 10**18
6     dist = [INF]*len(G)
7     dist[S] = 0
8     while q:
9         q2 = []
10        for u in q:
11            for v in G[u]:
12                # early break here if only interested in
13                length of S -> T path.
14                if dist[u] + 1 < dist[v]:
15                    dist[v] = dist[u] + 1
16                    q2.append(v)
17        q = q2
18    return dist
```

dijkstra:

```

1 from heapq import heappop as pop, heappush as push
2 # adj-list where edges are tuples (node_id, weight)
3 # (1) --2-- (0) --3-- (2) has the adj-list:
4 # adj = [[(1, 2), (2, 3)], [(0, 2)], [0, 3]]
5 def dijk(adj, S, T):
6     N = len(adj)
7     INF = 10**18
8     dist = [INF]*N
9     pq = []
10    def add(i, dst):
11        if dst < dist[i]:
12            dist[i] = dst
13            push(pq, (dst, i))
14    add(S, 0)
15
16    while pq:
17        D, i = pop(pq)
18        if i == T: return D
19        if D != dist[i]: continue
20        for j, w in adj[i]:
21            add(j, D + w)
22
23    return dist[T]

```

## twoSat:

```

1 # used in sevenkingdoms, illumination
2 import sys
3 sys.setrecursionlimit(10**5)
4 class Sat:
5     def __init__(self, no_vars):
6         self.size = no_vars*2
7         self.no_vars = no_vars
8         self.adj = [[] for _ in range(self.size)]
9         self.back = [[] for _ in range(self.size)]
10    def add_imply(self, i, j):
11        self.adj[i].append(j)
12        self.back[j].append(i)
13    def add_or(self, i, j):
14        self.add_imply(i^1, j)
15        self.add_imply(j^1, i)
16    def add_xor(self, i, j):
17        self.add_or(i, j)
18        self.add_or(i^1, j^1)
19    def add_eq(self, i, j):
20        self.add_xor(i, j^1)
21
22    def dfs1(self, i):
23        if i in self.marked: return
24        self.marked.add(i)
25        for j in self.adj[i]:
26            self.dfs1(j)
27        self.stack.append(i)
28
29    def dfs2(self, i):
30        if i in self.marked: return
31        self.marked.add(i)

```

```

32        for j in self.back[i]:
33            self.dfs2(j)
34        self.comp[i] = self.no_c
35
36    def is_sat(self):
37        self.marked = set()
38        self.stack = []
39        for i in range(self.size):
40            self.dfs1(i)
41        self.marked = set()
42        self.no_c = 0
43        self.comp = [0]*self.size
44        while self.stack:
45            i = self.stack.pop()
46            if i not in self.marked:
47                self.no_c += 1
48                self.dfs2(i)
49        for i in range(self.no_vars):
50            if self.comp[i*2] == self.comp[i*2+1]:
51                return False
52        return True
53
54    # assumes is_sat.
55    # If xi is after xi in topological sort,
56    # xi should be FALSE. It should be TRUE otherwise.
57    # https://codeforces.com/blog/entry/16205
58    def solution(self):
59        V = []
60        for i in range(self.no_vars):
61            V.append(self.comp[i*2] > self.comp[i*2+1])
62        return V
63
64 if __name__ == '__main__':
65     S = Sat(1)
66     S.add_or(0, 0)
67     print(S.is_sat())
68     print(S.solution())

```

## networkFlow, minflow:

```

1 # used in mincut @ Kattis
2 from collections import defaultdict
3 class Flow:
4     def __init__(self, sz):
5         self.G = [
6             defaultdict(int) for _ in range(sz)
7         ] # neighbourhood dict, N[u] = {v_1: cap_1, v_2
8         : cap_2, ...}
9         self.Seen = set() # redundant
10
11    def increase_capacity(self, u, v, cap):
12        """ Increases capacity on edge (u, v) with cap.
13        No need to add the edge """
14        self.G[u][v] += cap
15
16    def max_flow(self, source, sink):
17        def dfs(u, hi):
18            G = self.G

```

```

18            Seen = self.Seen
19            if u in Seen: return 0
20            if u == sink: return hi
21
22            Seen.add(u)
23            for v, cap in G[u].items():
24                if cap >= self.min_edge:
25                    f = dfs(v, min(hi, cap))
26                    if f:
27                        G[u][v] -= f
28                        G[v][u] += f
29                    return f
30
31            return 0
32
33        flow = 0
34        self.min_edge = 2**30 # minimal edge allowed
35        while self.min_edge > 0:
36            self.Seen = set()
37            pushed = dfs(source, float('inf'))
38            if not pushed:
39                self.min_edge //= 2
40            flow += pushed
41        return flow

```

## maxflow:

```

1 from collections import defaultdict
2 class Dinitz:
3     def __init__(self, sz, INF=10**10):
4         self.G = [defaultdict(int) for _ in range(sz)]
5         self.sz = sz
6         self.INF = INF
7
8     def add_edge(self, i, j, w):
9         self.G[i][j] += w
10
11    def bfs(self, s, t):
12        level = [0]*self.sz
13        q = [s]
14        level[s] = 1
15        while q:
16            q2 = []
17            for u in q:
18                for v, w in self.G[u].items():
19                    if w and level[v] == 0:
20                        level[v] = level[u] + 1
21                        q2.append(v)
22            q = q2
23        self.level = level
24        return level[t] != 0
25
26    def dfs(self, s, t, FLOW):
27        if s in self.V: return 0
28        if s == t: return FLOW
29        self.V.add(s)
30        L = self.level[s]
31        for u, w in self.G[s].items():
32            if u in self.dead: continue

```

```

33         if w and L+1==self.level[u]:
34             F = self.dfs(u, t, min(FLOW, w))
35             if F:
36                 self.G[s][u] -= F
37                 self.G[u][s] += F
38                 return F
39     self.dead.add(s)
40     return 0
41
42
43 def max_flow(self, s, t):
44     flow = 0
45     while self.bfs(s, t):
46         self.dead = set()
47         while True:
48             self.V = set()
49             pushed = self.dfs(s, t, self.INF)
50             if not pushed: break
51             flow += pushed
52     return flow

```

```

17         setup(2*i + 1, mid+1, hi)
18         self._fix(i)
19         setup(1, 0, self.sz-1)
20     def _fix(self, i):
21         self.value[i] = self.func(self.value[2*i], self
22         .value[2*i+1])
23
24     def _combine(self, a, b):
25         if a is None: return b
26         if b is None: return a
27         return self.func(a, b)
28
29     def query(self, lo, hi):
30         assert 0 <= lo <= hi < self.sz
31         return self.__query(1, lo, hi)
32
33     def __query(self, i, lo, hi):
34         l, r = self.L[i], self.R[i]
35         if r < lo or hi < l:
36             return None
37         if lo <= l <= r <= hi:
38             return self.value[i]
39         return self._combine(
40             self.__query(i*2, lo, hi),
41             self.__query(i*2 + 1, lo, hi)
42         )
43
44     def assign(self, pos, value):
45         assert 0 <= pos < self.sz
46         return self.__assign(1, pos, value)
47
48     def __assign(self, i, pos, value):
49         l, r = self.L[i], self.R[i]
50         if pos < l or r < pos: return
51         if pos == l == r:
52             self.value[i] = value
53             return
54         self.__assign(i*2, pos, value)
55         self.__assign(i*2 + 1, pos, value)
56         self._fix(i)

```

```

57
58     def inc(self, pos, delta):
59         assert 0 <= pos < self.sz
60         self.__inc(1, pos, delta)
61
62     def __inc(self, i, pos, delta):
63         l, r = self.L[i], self.R[i]
64         if pos < l or r < pos: return
65         if pos == l == r:
66             self.value[i] += delta
67             return
68         self.__inc(i*2, pos, delta)
69         self.__inc(i*2 + 1, pos, delta)
70         self._fix(i)
71
72     # for indexing - nice to have but not required
73     def __setitem__(self, i, v):
74         self.assign(i, v)

```

```

74     def __fixslice__(self, k):
75         return slice(k.start or 0, self.sz if k.stop ==
76         None else k.stop)
77     def __getitem__(self, k):
78         if type(k) == slice:
79             k = self.__fixslice__(k)
80             return self.query(k.start, k.stop - 1)
81         elif type(k) == int:
82             return self.query(k, k)

```

## Fenwick Tree:

```

1 # Tested on: https://open.kattis.com/problems/froshweek
2 class FenwickTree: # zero indexed calls!
3     # Give array or size!
4     def __init__(self, blob):
5         if type(blob) == int:
6             self.sz = blob
7             self.data = [0]*(blob+1)
8         elif type(blob) == list:
9             A = blob
10            self.sz = len(A)
11            self.data = [0]*(self.sz + 1)
12            for i, a in enumerate(A):
13                self.inc(i, a)
14
15        # A[i] = v
16        def assign(self, i, v):
17            currV = self.query(i, i)
18            self.inc(i, v - currV)
19
20        # A[i] += delta
21        # this method is ~3x faster than doing A[i] +=
22        # delta
23        def inc(self, i, delta):
24            i += 1 # (to 1 indexing)
25            while i <= self.sz:
26                self.data[i] += delta
27                i += i&-i # lowest oneBit
28
29        # sum(A[:i+1])
30        def sum(self, i):
31            i += 1 # (to 1 indexing)
32            S = 0
33            while i > 0:
34                S += self.data[i]
35                i -= i&-i
36            return S
37
38        # return sum(A[lo:hi+1])
39        def query(self, lo, hi):
40            return self.sum(hi) - self.sum(lo-1)
41
42        # for indexing - nice to have but not required
43        def __fixslice__(self, k):
44            return slice(k.start or 0, self.sz if k.stop ==
45            None else k.stop)
46        def __setitem__(self, i, v):
47            self.assign(i, v)
48        def __getitem__(self, k):
49            if type(k) == slice:
50                k = self.__fixslice__(k)

```

## Chapter 4

# Data Structures

### Segment tree:

```

1 # Tested on: https://open.kattis.com/problems/
2   supercomputer
3 class SegmentTree:
4     def __init__(self, arr, func=min):
5         self.sz = len(arr)
6         assert self.sz > 0
7         self.func = func
8         sz4 = self.sz*4
9         self.L, self.R = [None]*sz4, [None]*sz4
10        self.value = [None]*sz4
11        def setup(i, lo, hi):
12            self.L[i], self.R[i] = lo, hi
13            if lo == hi:
14                self.value[i] = arr[lo]
15            return
16        mid = (lo + hi)//2
17        setup(2*i, lo, mid)

```

```

45         return self.query(k.start, k.stop - 1)
46     elif type(k) == int:
47         return self.query(k, k)

```

## RMQ:

```

1 import math
2 class RMQ:
3     def __init__(self, arr, func=min):
4         self.sz = len(arr)
5         self.func = func
6         MAXN = self.sz
7         LOGMAXN = int(math.ceil(math.log(MAXN + 1, 2)))
8         self.data = [[0]*LOGMAXN for _ in range(MAXN)]
9         for i in range(MAXN):
10             self.data[i][0] = arr[i]
11         for j in range(1, LOGMAXN):
12             for i in range(MAXN - (1<<j)+1):
13                 self.data[i][j] = func(
14                     self.data[i][j-1],
15                     self.data[i + (1<<(j-1))][j-1])
16
17     def query(self, a, b):
18         if a > b:
19             # some default value when query is empty
20             return 1
21         d = b - a + 1
22         k = int(math.log(d, 2))
23         return self.func(self.data[a][k], self.data[b
- (1<<k)+1][k])

```

## Union Find:

```

1 class UnionFind:
2     def __init__(self, N):
3         self.parent = [i for i in range(N)]
4         self.sz = [1]*N
5     def find(self, i):
6         path = []
7         while i != self.parent[i]:
8             path.append(i)
9             i = self.parent[i]
10        for u in path: self.parent[u] = i
11        return i
12    def union(self, u, v):
13        uR, vR = map(self.find, (u, v))
14        if uR == vR: return False
15        if self.sz[uR] < self.sz[vR]:
16            self.parent[uR] = vR
17            self.sz[vR] += self.sz[uR]
18        else:
19            self.parent[vR] = uR
20            self.sz[uR] += self.sz[vR]
21        return True

```