JOI 2025/2026 ファイナルステージ Day1 伝説の団子食通 (Legendary Dango Eater) 解説

解説: 蜂矢 倫久 (Mitsubachi)

Step 0

問題概要

0 問題概要

- 味が甘いか辛いかのどちらかである団子の列があります
- ランレングス圧縮した形で与えられます
- 団子の区間を食べることを考えます
- いくつかの連続部分列に区切ります
- 連続部分列で，(甘い味の数) - (辛い味の数) ≧ K を満たす個数を最大化したいです
- Q 通りの区間について解いてください

0 制約

- N ≦ 500 000 (ランレングス圧縮後の区間数)
- Q ≦ 500 000
- K ≦ 10^9
- A_i ≦ 10^9 (各区間の個数)

0 小課題

小課題番号	N	Q	K	A	配点
1		10			6 点
2			2		5 点
3			10		18 点
4				総和 500 000	27 点
5	200 000	200 000			17 点
6					27 点

Subtask 1

$Q \leqq 10$

1 条件の言い換え

- 甘い味の団子を +1，辛い味の団子を -1 とみなす
- (甘い味の数) - (辛い味の数) は単に総和となる
- 以降この設定で考える
- 実装上は A_2, A_4, ... を -1 倍するなどで対応できる

1 最適な操作

- まず最適な区切り方について考える

- 総和が K になったら切っていい
- 0 になっても切っていい

- これでシュミレーションができる形に
- クエリあたり O(NA) にまずなる → 時間制限に間に合わない

1 最適な操作

- まず最適な区切り方について考える

- 総和が K になったら切っていい
- 0 になっても切っていい

- これでシュミレーションができる形に
- クエリあたり O(NA) にまずなる → 時間制限に間に合わない

- よく考えるとランレングス圧縮した後の区間 1 つ分は同時に処理できる

1 シュミレーション

- 総和が K になったら切っていい
- 0 になっても切っていい

- よく考えるとランレングス圧縮した後の区間 1 つ分は同時に処理できる

- ans = 0, rem = 0
- for a in A:
  - rem ← max(0, rem + a)
  - ans ← ans + rem / K
  - rem ← rem % K

1 シュミレーション

- 総和が K になったら切っていい
- 0 になっても切っていい

- よく考えるとランレングス圧縮した後の区間 1 つ分は同時に処理できる

- これでクエリあたり O(N) になった

- O(NQ) で解けました

1 今後について

- このシミュレーションを考えると，A_1, A_3, ... については K で割ったあまりに変えていいことが分かる
- 商の分は後で累積和として計算すれば良い
- A_1, A_3, ... は K 未満と帰着することができる
- 同様に考えると A_2，A_4, ... も K 以下とできる
- 以降この形で考えます

Subtask 2

$K \leqq 2$

2 考察

- K = 1 と K = 2 が解ければ良い

- K = 1：甘い味の数を数えれば良い
- 累積和を使えば O(N + Q)

- K = 2 を解く 先ほどのシュミレーションを思い返す

- ans = 0, rem = 0
- for a in A:
  - rem ← max(0, rem + a)
  - ans ← ans + rem / K
  - rem ← rem % K

2 K = 2

- K = 2 を解く 先ほどのシュミレーションを思い返す

- ans = 0, rem = 0
- for a in A:
  - rem ← max(0, rem + a)
  - ans ← ans + rem / K
  - rem ← rem % K

- 負の a を処理する前の rem は 0 か 1
- つまり負の a を処理した後の rem は必ず 0

2 解法

- K = 2 を解く 先ほどのシュミレーションを思い返す

- よって辛い味のものを跨ぐ必要はない
- したがって，甘い味の a について，a / 2 の総和を取れば OK

- これも累積和で O(N + Q)

- 以上より，どちらの K でも O(N + Q) で解けました

Subtask 3

$K \leqq 10$

3 シュミレーションの復習

- シュミレーションを振り返る

- ans = 0, rem = 0
- for a in A:
  - rem ← max(0, rem + a)
  - ans ← ans + rem / K
  - rem ← rem % K

- これの初期値の ans が 0 ではなく x でなると最終結果の ans はどう変わる？
- 単に x 足すだけ

3 シュミレーションの復習

- シュミレーションを振り返る
- つまり，大事なのは rem の値
- ある区間について i = 0, 1, ..., K - 1 について，(ans, rem) の初期値が (0, i) としてシュミレーションを走らせると (ans, rem) は最終的にどうなるかを求めることを考える
- これは Segment Tree に乗る形になる
- ノードのマージも O(K) でできる

3 セグメント木

- したがって，O((N + Q log N) K) などで解くことができる
- 関数合成を Segment Tree に乗せるという考え方が近いと思う
- ういろう(Uiro) の小課題 5 などが同じ考え方

3 休憩用スライド

- 話疲れてそうなのでちょっと休憩
- Dango in USA の写真を貼る

Subtask 4

A の総和 $\leqq$ 500 000

4 前準備

- ランレングス圧縮してあるのをバラす

- Sample 1 なら 2, 1, 2, 4, 3 なので
- +1, +1, -1, +1, +1, -1, -1, -1, -1, +1, +1, +1 という形

- これにおける区間クエリが解ければ良い

- A の総和を S とする

4 シュミレーションの復習

- シュミレーションを振り返る

- ans = 0, rem = 0
- for a in A:
  - rem ← max(0, rem + a)
  - ans ← ans + rem / K
  - rem ← rem % K

- これの初期値の ans が 0 ではなく x でなると最終結果の ans はどう変わる？
- 単に x 足すだけ

4 シュミレーションの復習

- シュミレーションを振り返る
- 最初に rem が 0 になるところが分かったとする
- つまり，そこで区切りますよという場所
- すると，l からスタートして r に行くまで何回区切りますかと見ることができる

4 シュミレーションの復習

- シュミレーションを振り返る
- 最初に rem が 0 になるところが分かったとする
- つまり，そこで区切りますよという場所
- すると，l からスタートして r に行くまで何回区切りますかと見ることができる
- これは**ダブリング**！

4 ダブリング

- シュミレーションを振り返る
- 最初に rem が 0 になるところを求めよう
- シミュレーションの動きを考えると最初に 0 以下 or K 以上に到達する場所であるとみなせる
- これは prefix の [min, max] が [1, K - 1] をはみ出すのはいつかということになる
- Segment Tree で二分探索すれば全体で O(S log S)

4 具体例

- Sample 2 の K = 3, A = (2, 1, 2, 4, 3) で考える
- +1, +1, -1, +1, +1, -1, -1, -1, -1, +1, +1, +1 にバラす

- 左から始めると
- +1, +1, -1, +1, +1 / -1 / -1 / -1 / -1 / +1, +1, +1
- という感じに

- よって答えは 2 ですねということになる
- 実装上は -1 からスタートする区間は飛ばすと楽だと思う

4 ダブリング

- シュミレーションを振り返る
- 最初に rem が 0 になるところが O(S log S) で求まった
- あとはこれでダブリングをすれば良い 前準備は O(S log S)
- クエリあたりは O(log S) になる
- 全体で O((S + Q) log S) で解けました

4 おまけ

- 小課題 1 で説明した通り A_i ≦ K とできるので, これをすることで O((NK + Q) log (NK)) などで解くことができる

- 小課題 2 はいけそう
- 小課題 3 はギリギリな気がする メモリ制限の話もありそう

Subtask 6

追加制約なし

6 小課題 5 と 6 の話

- 小課題 5 と 6 は 2.5 倍ぐらいの差なので，定数倍が悪かったりどこかで余計な log や √ がついていると小課題 5 までしか通らないと思います
- ここでは A_2, A_4, ... を -1 倍するのは一旦やめたとします

6 ダブリングの思考

- 先ほどのダブリングを A_i が大きくても使えるようにしたい
- A_i を足して mod K をして，A_{i + 1} を引いて，... を繰り返して最初に 0 以下になるところを求めたとする (A_pos とする)
- その間でいくつ取ることができるかを考える
- [A_i, A_pos) の区間で何個作れますかという話になる
- これは A_i から A_{pos - 1} までの総和を K で割った商になる
- 貪欲をすれば良い

6 具体例

- Sample 2 の K = 3, A = (2, 1, 2, 4, 3) で考える

- A_1 からスタートする
- +2 → +1 → +3 (mod 3 をして 0 に) → -4 となる

- よって A_1 からスタートしたら [A_1, A_3] の間はこれの総和を K で割った 1 個取れると言える，加えて A_4 以降(実際には A_4 は負の部分なので A_5 以降) と独立して考えられる

6 条件整理

- 結局のところ A_i が大きくなっても最初に 0 以下になる場所を探したいということになる
- これができればあとはダブリング
- ただ難しいのが，mod K を取ったりするところ
- 条件の形としては，各 i について，(A_i - A_{i + 1} + A_{i + 2} - … + A_{j - 1}) mod K ≦A_j なる最小の j を知りたいことに

6 余談

- 結局のところ A_i が大きくなっても最初に 0 以下になる場所を探したいということになる
- これができればあとはダブリング
- 愚直にやると O(N^2) になってしまう
- 実は O(NK) での評価は可能なので小課題 2, 3 はこれでも通る
- 高速に求める方法を紹介

6 式変形

- 各 i について，(A_i - A_{i + 1} + A_{i + 2} - ... + A_{j - 1}) mod K ≦ A_j なる最小の j を知りたい
- A_i の交互和の累積和 s_1, s_2, ..., を考えると上の式は以下に
- (s_{j - 1} - s_{i - 1}) mod K ≦ A_j
- j ≧ x というのは (s_{x - 1} - s_{i - 1}) mod K ≦ A_x が x = i, i + 1, ..., j で成り立つかということになる

6 式変形

- 各 i について，(A_i - A_{i + 1} + A_{i + 2} - ... + A_{j - 1}) mod K ≦A_j なる最小の j を知りたい
- j ≧ x かというのは (s_{x - 1} - s_{i - 1}) mod K ≦ A_x が x = i, i + 1, ..., j で成り立つかということになる

- i が大きい順に j を求めることにする
- すると，s_{i - 1} が mod K でこの区間にあるような i については j はこの値以下みたいな制約が増えていく形になる

- 座圧をすれば 区間更新 / 一点取得になる

6 2 つの方針

- i が大きい順に j を求めることにする
- すると，s_{i - 1} が mod K でこの区間にあるような i については j はこの値以下みたいな制約が増えていく形になる
- (s_{x - 1} - s_{i - 1}) mod K ≦ A_x 由来の制約を制約 x とする

- 座圧をすれば 区間更新 / 一点取得になる

- 2 つの方針がある

- ① set で頑張る方法
- ② データ構造を書く方法

6 ① set で頑張る

- まだ j が未定の (i, s_i) の集合を管理しておく

- (s_{x - 1} - s_{i - 1}) mod K ≦ A_x 由来の制約を制約 x とした
- これを x = 1, 2, ... の順に追加していく
- (i, s_i) もこの順に追加していく

- 制約 x を追加したとき，それで確定する i (つまり，j = x で初めて制約 x に違反して j = i となる i) は set の lower_bound などで高速に取得できる

- O(N log N) などでできる

6 ② データ構造を書く

- 区間更新 / 一点取得の形になった
- これは x が大きい方から制約 x を足していく

- まず Lazy Segment Tree で解ける
- O(N log N)

- でももっと楽に書ける
- オーダーは同じだけど定数倍がよくなる

6 ② データ構造を書く

- 区間更新 / 一点取得の形になった
- これは x が大きい方から制約 x を足していく

- まず Lazy Segment Tree で解ける
- O(N log N)

- でももっと楽に書ける
- オーダーは同じだけど定数倍がよくなる

- その名は **Dual Segment Tree** (双対セグメント木)！

6 ② データ構造を書く

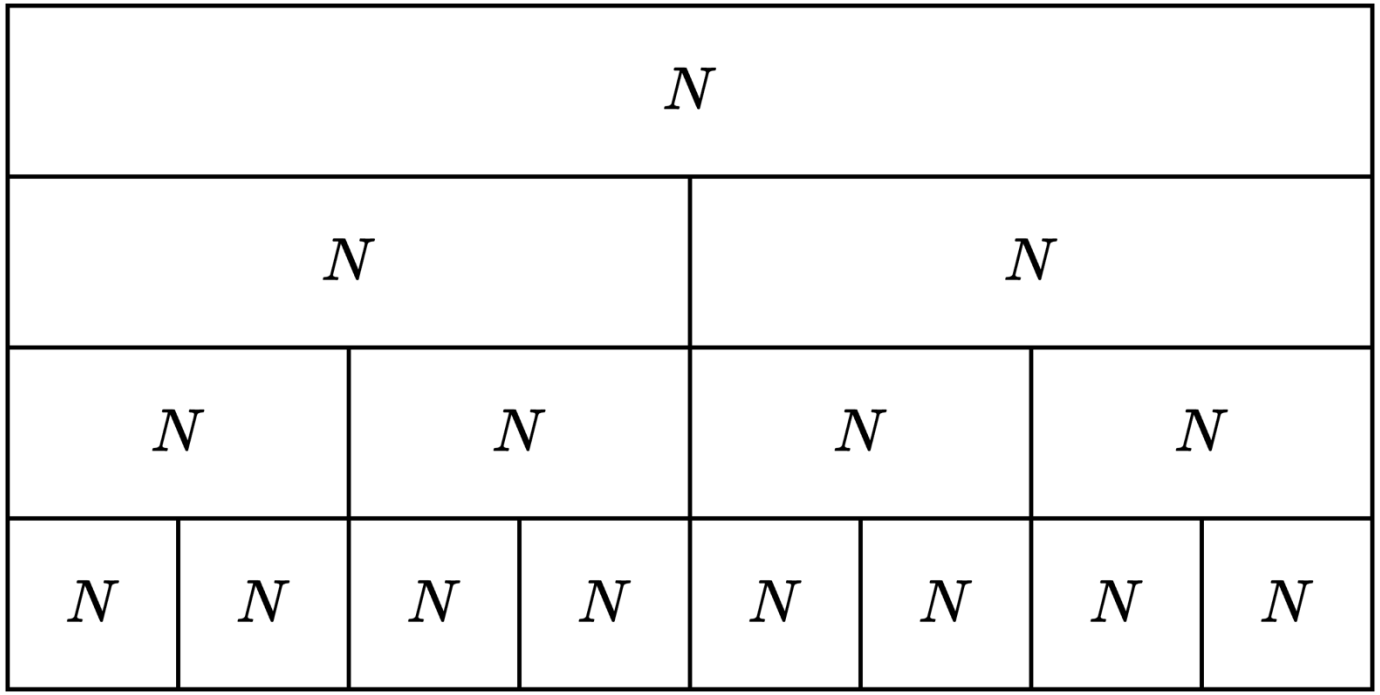
- **Dual Segment Tree** とは？

- チューターの tatyam さんの記事を見るととても参考に
- https://hackmd.io/@tatyam-prime/DualSegmentTree

- ザックリいうと，Segment Tree の一点更新と区間の prod 取得を逆にしたよということ

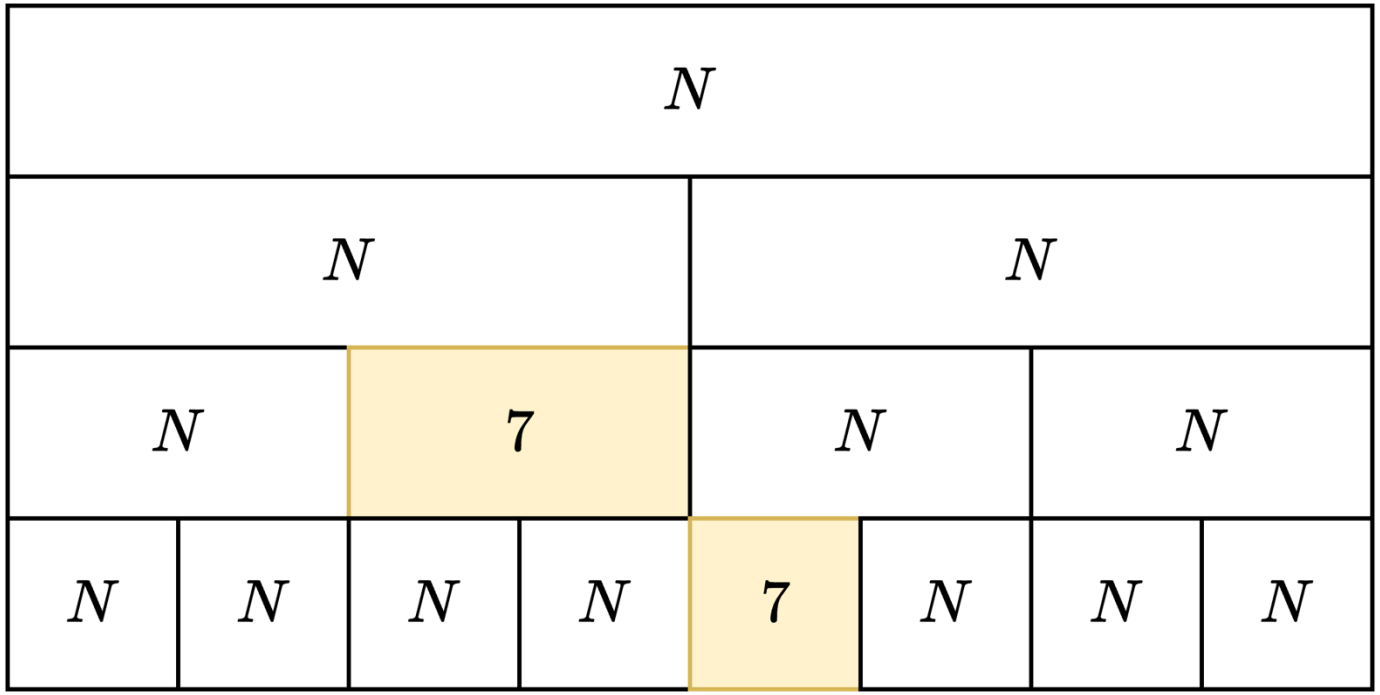
6 ② データ構造を書く

- N = K = 8 で具体例を説明
- 下は Segment Tree でよくある図 (0-indexed)
- まだ何の制約もないという状態



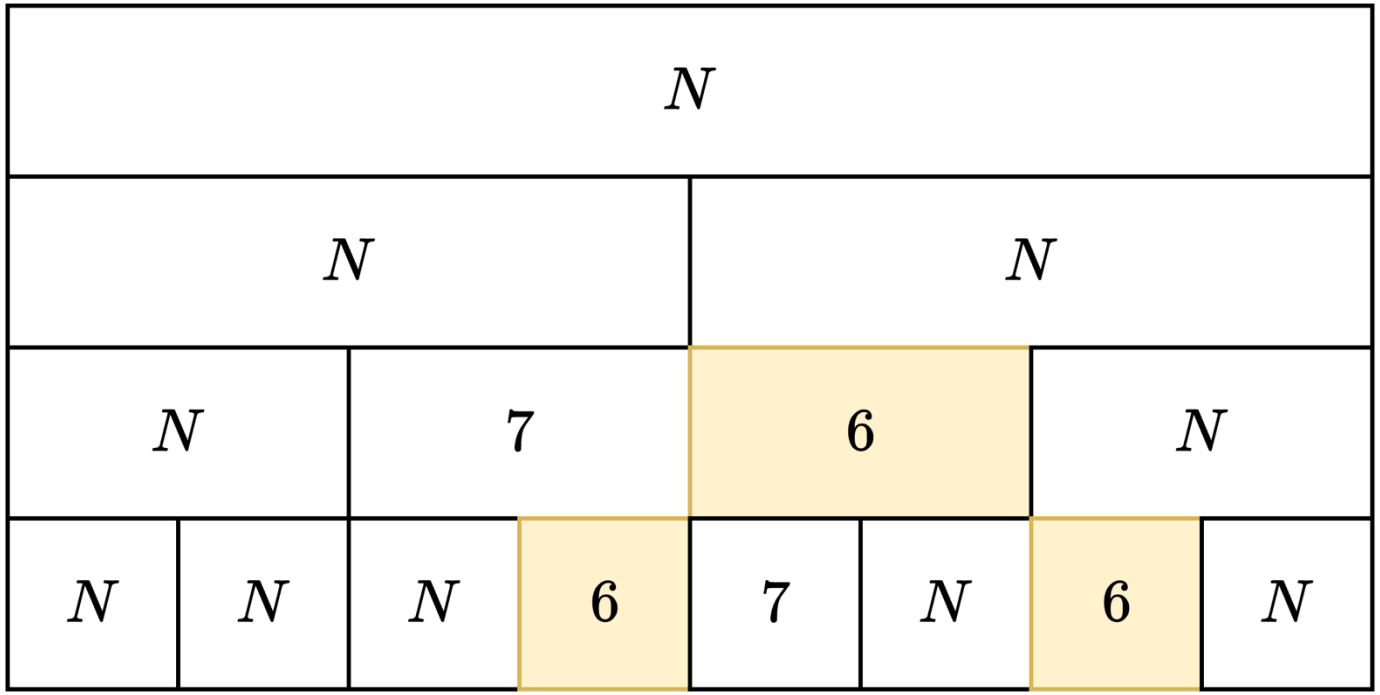
6 ② データ構造を書く

- mod K で [2, 5) なら 7 (= N - 1) で引っ掛かりますという制約を追加
- 普通の Segment Tree で [2, 5) の区間クエリを求める感じ



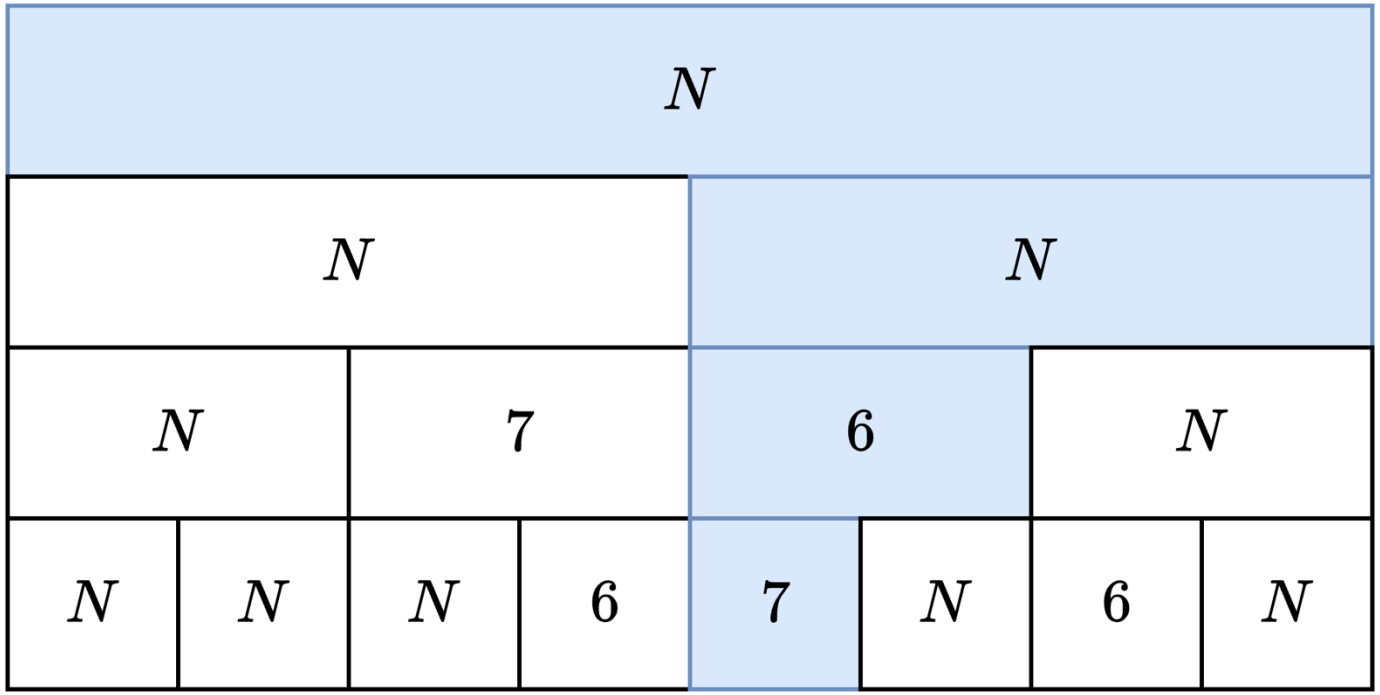
6 ② データ構造を書く

- mod K で [3, 7) なら 6 (= N - 2) で引っ掛かりますという制約を追加
- 普通の Segment Tree で [3, 7) の区間クエリを求める感じ



6 ② データ構造を書く

- mod K で 4 なら今いくつかを知りたい
- 普通の Segment Tree で 4 の一点更新をする感じでできる！



6 ② データ構造を書く

- これが操作あたり O(log K) なのは Segment Tree の計算量を知っていれば理解できると思う

- 今回は K が大きいので座圧が必要
- 座圧したらサイズは N になる

- よって操作あたり O(log N) になる

- 全体で O(N log N)

6 解法

- とりあえず $O(N \log N)$ で一手先が分かった
- 別の方法として平方分割がありうるが，小課題 5 までかも
- あとはダブリングすれば OK
- 全体で $O((N + Q) \log N)$ となる

Step 7

得点分布

7 得点分布

- 理想:

点数	1	2	3	4	5	6	人数	累計
100	o	o	o	o	o	o	30	30

- 選抜という意味では差がつかないな

7 得点分布

- 現実:

点数	1	2	3	4	5	6	人数	累計
100	o	o	o	o	o	o	4	4
56	o	o	o	o			2	6
38	o	o		o			2	8
29	o	o	o				7	15
27				o			1	16
～11	こ	の	辺				14	30