联合省选 2024 模拟赛 Easy Ver.

题目名称	迷失深海	夜	前往宇宙末日的地球旅人
题目类型	传统型	传统型	传统型
输入文件名	`ocean.in`	`night.in`	`earth.in`
输出文件名	`ocean.out`	`night.out`	`earth.out`
提交源程序文件名	`ocean.cpp`	`night.cpp`	`earth.cpp`
每个测试点时限	2.0s	1.5 ~ 10.0s	3.0s
内存限制	512 MiB	32 MiB ~ 2.0 GiB	64 MiB
是否有子任务捆绑	否	是	是
测试点数目	20	13	6
测试点是否等分	是	否	否

注意事项（请仔细阅读）：

1. 文件名（程序名和输入输出文件名）必须使用英文小写。
2. C/C++ 中函数 `main()` 的返回值类型必须是 `int`，程序正常结束时的返回值必须是 0。
3. 若无特殊说明，结果的比较方式为全文比较（过滤行末空格及文末回车）。
4. 程序可使用的栈内存空间限制与题目的内存限制一致。
5. 提交的源程序文件大小不得超过 50KiB。
6. 没有搞很难的题，喧哗的同学请不要大声 AK。
7. 祝大家好运。

迷失深海 (ocean)

题目背景

深海鲛鳚属于鲛鳚鱼目，俗称灯笼鱼。鲛鳚头部上方形似小灯笼的肉状突出，是由鲛鳚鱼的第一背鳍逐渐向上延伸形成的。

鲴是鲤形目鲤亚目鲤科鲴属的鱼类动物，一般长 60 厘米，而其头长可达 20 厘米，故称“胖头鱼”，其英文名是 variegated carp，而不是下文出现的 fat head fish。

箱水母又叫“海黄蜂”。成年箱水母有足球那么大，蘑菇状，近乎透明。身体两侧各有两只原始的眼睛，可以感受光线的变化，身后托着 60 多条带状触须，它由体内喷出的水柱推动着身体旋转前进。其毒性非常可怕，是已知的地球上毒性最强的生物。

利维坦（Leviathan），传说中的海怪。它畅泳于大海之时，波涛亦为之逆流。它口中喷着火焰，鼻子冒出烟雾，拥有锐利的牙齿，身体好像包裹着铠甲般坚固。性格冷酷无情，暴戾好杀，它在海洋之中寻找猎物，令四周生物闻之色变。

题目描述

灯笼鱼游过一片茂密的珊瑚礁，发现了一只异常巨大而凶恶的怪兽——利维坦。它是海洋中的霸主，以毁灭和征服为乐。灯笼鱼深知自己无法单独对抗利维坦，她需要寻求帮助。

她从箱水母那里得到了一行代码，据箱水母们说这行代码可以打败利维坦。但是她无法理解这段代码的含义，她需要寻求帮助。

于是，灯笼鱼来到了鱼类界最负盛名的编程竞赛平台——Fishforces，寻找传说中的顶尖编程高手胖头鱼。胖头鱼是这个平台上第一个获得 LGM（Legendary Grandmaster）的高 Rating 鱼，他以其卓越的编码技巧和聪明才智而闻名。

灯笼鱼向胖头鱼讲述了她的困境，希望他能帮忙找到对抗利维坦的方法。胖头鱼仔细聆听，然后欣然接受了这个挑战。

胖头鱼专注地看着灯笼鱼提供的代码片段：

```
const int MOD = 998244353;
```

胖头鱼思考着如何通过这段代码来对抗利维坦。他知道，要解决这个问题，需要理解代码的含义和作用。

`const int` 表示这是一个常量整数，它的值无法改变。而 `MOD` 是常量的名称，它的值被指定为 998244353。接下来，胖头鱼明白了这个常量的作用。

胖头鱼解释道：“在编程中，`MOD` 经常用于表示模运算的模数。模运算是一种取余操作，它对于处理大数和减小计算结果的大小有很大帮助。这个特定的模数 998244353 是一个常见的选择，因为它是一个质数，而且是 NTT 模数，非常适用于数论和组合数学的问题。”

“胖头鱼，你告诉我了 `MOD` 的作用，但是这段代码的运行结果是什么？我们如何利用它对抗利维坦呢？”灯笼鱼好奇地问道。

胖头鱼微笑着回答：“噢，这是一个很好的问题。实际上，这段代码本身并不会产生明显的运行结果。它只是定义了一个常量，供其他代码在运行时使用。我们可以根据需要在程序中引用这个常量值，并进行各种计算和操作。在对抗利维坦的过程中，我们可能会使用到这个 `MOD` 常量来进行模运算，以确保计算结果不会超出一定的范围。”

灯笼鱼点点头，她开始理解胖头鱼的意图。通过使用这个特定的模数 998244353，他们可以在计算过程中限制结果的大小，避免数据溢出或失真。这对于处理海量数据和复杂计算是至关重要的。

灯笼鱼虽然对具体的技术细节不甚了解，但她明白胖头鱼正在努力寻找一种更强大的方式来对抗利维坦。她觉得她找到了希望，似乎胖头鱼是鱼族战胜利维坦的最后的希望。

思考了若干时间后，胖头鱼快速地写出了一段代码（如果你无法复制这份代码，那么可以参考选手目录下的 `ocean/ocean.cpp`）：

```
void my_favorite_animal_is_fat_head_fish() {
  int n, ans = 0; std::cin >> n;
  std::vector<int> arr(n);
  for (int i = 0; i < n; ++i) std::cin >> arr[i];
  for (int a = 0; a < n; ++a) for (int b = a; b < n; ++b)
    for (int c = b + 2; c < n; ++c) for (int d = c; d < n; ++d) {
      std::set<int> s;
      for (int i = b + 1; i ⩽ c - 1; ++i) s.insert(arr[i]);
      if (s.size() ≠ c - 1 - b) continue;
      bool flag = 1;
      for (int i = a; i ⩽ b; ++i) if (s.count(arr[i])) { flag = 0;
break; }
      for (int i = c; i ⩽ d; ++i) if (s.count(arr[i])) { flag = 0;
break; }
      if (flag) ans = (ans + 1) % MOD;
    }
  std::cout << ans << "\n";
}
```

最后得到的 ans 就是击败利维坦的钥匙。但无奈此程序运行的太慢，无法在利维坦攻来前运行出结果。

“能有一个更快的方式吗？”灯笼鱼问。

“当然，但是，我需要一些时间。”胖头鱼答道。

胖头鱼开始思考如何优化这段代码。他注意到，胖头鱼发现代码中存在一些不必要的循环次数。他决定优化这些循环，以减少程序的运行时间。一段时间后，胖头鱼编写出了一段新的代码：

```
[The data has been deleted.]
```

果不其然，这段代码很快就通过了 $n \leq 2 \times 10^5$ 的数据。灯笼鱼和胖头鱼都非常兴奋，这次他们一定能够击败邪恶的利维坦了。

你对这段高效实现的代码感到非常好奇。于是，你决定还原胖头鱼所写出的程序。

输入格式

从文件 `ocean.in` 中读入数据。

本题有多组输入数据。

第一行一个正整数 T，表示数据组数。

对于每组数据，第一行一个正整数 n，第二行 n 个整数表示 arr。

输出格式

输出到文件 `ocean.out` 。

一行一个整数，代表 ans 的值。

输入输出样例

样例 1 输入

```
3
3
1 2 3
5
1 2 1 2 3
10
1 1 2 1 2 3 1 2 3 4
```

样例 1 输出

```
1
4
33
```

样例 1 解释

将下发文件中的代码调用 T 次，输入【样例 1 输入】的内容，可以得到【样例 1 输出】的结果。

样例 2 输入 / 输出

见选手目录下的 `ocean/ocean2.in` 和 `ocean/ocean2.ans`。

除 $T = 500$ 外，该样例满足测试点 1 的性质。

样例 3 输入 / 输出

见选手目录下的 `ocean/ocean3.in` 和 `ocean/ocean3.ans`。

该样例满足测试点 $12 \sim 15$ 的性质。

数据范围

对于 100% 的数据，满足 $1 \leq T \leq 3, 1 \leq n \leq 2 \times 10^5, |arr_i| \leq -2 \times 10^9$。

测试点编号	$T =$	$n \leq$	$\|arr\| \leq$	特殊性质
1	3	20		
2	3	100		
$3 \sim 5$	3	500		
$6 \sim 9$	3	2×10^3		
$10 \sim 11$	1	10^5		所有 a 互不相同
$12 \sim 15$	1	10^5	5	
16	1	5×10^4		
$17 \sim 19$	1	10^5		
20	3			

没写就是没限制。

多测不清空，爆零两行泪。

月在读完故事后，默默地合上了古籍。她可能需要一些时间来沉淀所读之物，思考灯笼鱼和胖头鱼击败的利维坦的意义和价值。

夜 (night)

题目背景

> 知らないままで痛いけど
> 疼痛无知觉地蔓延
>
> ネオンなジャングルの向こう
> 面对霓虹灯的丛林
>
> 子供みたいな大人みたいな
> 既像个孩子又像个大人
>
> 夜に走り回りたい
> 在夜色中四处漫游
>
> エメラルドグリーンの粒ら光らせて
> 翡翠般闪耀着的光斑
>
> あなたの帰りをいつまでも
> 一直守望着你
>
> 待っている
> 归来的身影

月的记忆被时空大盗删除了。

题目描述

时间高塔是这个世界中最为神秘的存在之一，它拥有着无比强大的力量，可以帮助人们破解各种难题。

月行走在通往时间高塔的小路上，她希望能借助时间高塔的力量让她回忆起一些事情。

一段记忆可以被描述为一个长度为 n，值域在 $[m,n]$ 的序列 a。但由于高维空间的影响，一些时间线已经被彻底删除，月无法回忆起它们。

时间高塔能复原一个记忆，当且仅当它能构造一个排列 p，使得对于任意 $i\in[1,n]$，都有序列 $Q_i=[p_1,p_2,\cdots,p_{i-1},p_{i+1},\cdots p_n]$ 的前缀最大值个数为 a_i。

月需要你预测时间高塔能够复原的记忆个数，答案对 998244353 取模。

前缀最大值个数指的是 $\sum_{i=1}^{n-1}[Q_i=\max_{j=1}^{i}Q_j]$ 的值，也就是 Q_i 取前缀最大值后得到 Q'_i，然后求 $Q'_i=Q_i$ 的个数。

不形式化地，Q_i 的构造方式是将 p_i 从 p 中删除后得到的新序列。

输入格式

从文件 `night.in` 中读入数据。

一行两个正整数，表示 n,m。

输出格式

输出到文件 `night.out`。

一行一个自然数，表示答案。

输入输出样例

样例 1 输入

```
5 3
```

样例 1 输出

```
8
```

样例 1 解释

能被时间高塔复原的记忆是：$[4,4,4,4,4]$，$[3,3,3,4,4]$，$[3,3,4,4,3]$，$[3,3,3,3,4]$，$[3,4,4,3,3]$，$[3,3,3,4,3]$，$[4,4,3,3,3]$，$[3,3,4,3,3]$。$a=[3,3,3,4,4]$ 时对应的 p 可以是 $[1,2,3,5,4]$。

样例 2 输入

```
3 1
```

样例 2 输出

```
6
```

样例 3/4/5/6 输入 / 输出

见选手目录下的 `night/night[x].in` 和 `night/night[x].ans`。

数据范围

本题采用捆绑测试。

对于 100% 的数据，保证 $3 \le n \le 5 \times 10^3, 1 \le m \le n$。

子任务编号	$n \le$	$m \le$	时间限制	空间限制	子任务依赖	分值
1	3		2s	2.0 GiB		5
2	9		2s	2.0 GiB		12
3	18	1	2s	2.0 GiB		16
4	70	1	2s	2.0 GiB	3	7
5	70		2s	2.0 GiB	$1 \sim 4$	7
6	300	1	2s	2.0 GiB	$3, 4$	7
7	300		2s	2.0 GiB	$1 \sim 6$	7
8	2×10^3	1	3s	2.0 GiB		5
9	2×10^3		3s	2.0 GiB	8	5
10			10s	2.0 GiB		5
11			10s	32 MiB	10	5
12			1.5s	2.0 GiB	10	5
13			1.5s	32 MiB	$10 \sim 12$	14

没写就是没限制。

由于时空大盗在场，**请注意常数因子对程序效率和空间消耗的影响，同时注意本题特殊的时空限制。**

你可以尝试使用以下方式优化：

- 减少取模次数；
- 如果可以使用 `unsigned int`，那么不使用 `long long`。

「不要去了解我，也不要用因果报应咒骂我。」

「太多拘束可能，捱过破碎过程，让我重获新生。」

月成功计算出了记忆的数量，但是她只会记住其中的一个。

前往宇宙末日的地球旅人 (earth)

题目背景

> 我飞入宇宙
> 向末日漂流
> 原谅我的错
> 所有的荒谬
> 都消逝在那星云之后
>
> 我飞向宇宙
> 背对这洪流
> 松开你的手 什么都不带走
> 给自己一枪 最温柔的复仇
>
> 我飞入宇宙
> 能不能得救
> 在告别之后
> 自由和占有
> 最后摧毁得片甲不留
>
> 我飞入宇宙
> 向末日漂流
> 拯救我的错
> 所有的荒谬

> 都消逝在那星云之后
>
> 我飞向宇宙
> 背对这洪流
> 最后剩下我 我什么都没有
> 末日的飞船 载着我的骨头

“我们希望发现一个天堂，但却发现了更多的地狱。”

宇宙是一个广袤而空旷的地方，当中最常见的物质是暗物质和暗能量。除此之外，只有一座座地狱，很少能够遇见能活人的地方。

想要在宇宙中长期流浪，飞船上必须有完整的生态循环系统。但在目前看来，这一点无法做到。

因此，飞船上载着的不是人，而是骨头。

最终，会以何种方式结束？没人知道。在热寂之后，在大撕裂之后……一切原有的逻辑都将不复存在，一切都将归于虚无。

题目描述

我们曾去过那受光最多的天体 / 看到了回到人间的人 / 无法也无力重述的故事。

在某个星系中存在 n 个虫洞，它们按照一定的顺序构成了一个链状结构。每个虫洞都有一个特定的**时间编号**，第 i 个虫洞的时间编号是 p_i，且保证 p 是一个排列。

我们有 n 台时光机，分别标记为 $1 \sim n$。每台时光机会按照顺序依次停留在时间编号**大于等于**自身编号的**所有**虫洞上。比如初始 $p = [3, 4, 1, 2]$，选择时间编号为 2 的时光机从 p_1 驶向 p_4，其会依次经过时间编号为 $3, 4, 2$ 的虫洞。

考虑一个虫洞集合 S，其中包含了 $m(m > 0)$ 个虫洞，表示为 $S = \{S_1, S_2, \cdots, S_m\}$。如果我们可以对 S 中的所有虫洞的顺序进行重排，使得 p_{S_i} 构成一个**单调递增或者单调递减**的序列之后，**相邻两个**虫洞（即 S_i 和 $S_{i-1}(i > 0)$ 两个虫洞）可以使用**一台**时光机直达，且中途你**不会停留**在其它虫洞，那么我们称对于 S 的穿梭是**完美的**。

注意，你每到达 S 中的一个目标虫洞就可以选择更换另一台时光机（也可以不换），无需全程只乘坐一台时光机。以上所述内容的具体操作过程可以参考样例 1 解释。

时间变幻莫测，月需要你回答 q 次询问，每次询问给定 l_i, r_i，求出满足 $\forall k \in S, l_i \leq p_k \leq r_i$ 的完美虫洞集合 S 的个数。由于宇宙中的答案级别是几何级，所以你需要将最终答案对 998244353 取模后告诉月。

输入格式

从文件 `earth.in` 中读入数据。

第一行两个正整数 n, q。

第二行 n 个正整数表示 $p_{1,2,\cdots,n}$。

接下来 q 行，每行两个正整数 l_i, r_i，表示一次询问。

输出格式

输出到文件 `earth.out` 中。

共 q 行，每行一个非负整数，表示答案。

输入输出样例

样例 1 输入

```
5 3
2 4 5 1 3
3 5
1 5
1 4
```

样例 1 输出

```
5
12
6
```

样例 1 解释

对于第 1 次询问，完美的虫洞集合的**时间编号**集合（这里用这种方式描述更易于理解，只需要注意到以下集合的元素不是在排列中的位置而是位置所对应的**值**即可）如下面所述：

- $\{3\}, \{4\}, \{5\}$：显然满足。
- $\{3, 5\}$：它们的位置是 $5, 3$，选择 3 号时光机即可，可以达到相应的位置。
- $\{4, 5\}$：选择 4 号时光机即可，3 号时光机也是可以的。

$\{3, 4\}$ 无法成为答案，因为其会在路途中会停留在不应该停留的时间编号为 5 的虫洞。

对于第 2 次询问，完美的虫洞集合的**时间编号**集合如下面所述：
$\{2\},\{4\},\{5\},\{1\},\{3\},\{2,4\},\{4,5\},\{5,1\},\{1,3\},\{3,5\},\{2,4,5\},\{5,1,3\}$。其中 $\{5,1,3\}$ 是从时间编号为 1 的虫洞出发，选择 1 号时光机到达时间编号为 3 的虫洞，再选择 2 号时光机（也可以是 3 号，但不能是 1 号，因为中途会经过时间编号为 1 的虫洞）到达时间编号为 5 的虫洞。

对于第 3 次询问，完美的虫洞集合的**时间编号**集合如下面所述：
$\{2\},\{4\},\{1\},\{3\},\{2,4\},\{1,3\}$。

样例 2 输入 / 输出

见选手目录下的 `earth/earth2.in` 和 `earth/earth2.ans`。

该样例满足子任务 1 的性质。

样例 3 输入 / 输出

见选手目录下的 `earth/earth3.in` 和 `earth/earth3.ans`。

该样例满足子任务 2 的性质。

样例 4 输入 / 输出

见选手目录下的 `earth/earth4.in` 和 `earth/earth4.ans`。

该样例满足子任务 5 的性质。

数据范围

本题采用捆绑测试。

对于 100% 的数据，保证 $1 \le n, q \le 2 \times 10^5$，$p$ 是一个排列，$1 \le l_i \le r_i \le n$。

子任务编号	$n \le$	$q \le$	特殊性质	子任务依赖	分值
1	15	1000			10
2	1000	1000		1	20
3			$p_i = i$		5
4			排列 p 随机生成		25
5	5×10^4	5×10^4		1, 2	20
6				1, 2, 3, 4, 5	20

没写就是没限制。

请注意本题特殊的时空限制。

月来到了另一个宇宙，也许她永远无法打破这轮回。

“消亡”和“诞生”永远是交替守恒的，终点也同样是起点，当下即是永恒，因为看见黑暗，才知得光明可贵。