5.5 Solution

A

> 难度估计：入门

简单题。模拟即可。

> 如果你对密码学有了解或有兴趣的话，你会注意到，这事实上就是省去了加密步骤的 CBC 模式。具体相关内容可以参考 Wikipedia - Block cipher of operation ，或者自行搜索相关资料。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pll pair<ll, ll>
int from_hex(char c) {
  if (c >= '0' && c <= '9')
    return c - '0';
  if (c >= 'A' && c <= 'F')
    return c - 'A' + 10;
  if (c >= 'a' && c <= 'f')
    return c - 'a' + 10;
  return -1;
}
int main() {
  int tmp;
  string in, out;
  ios::sync_with_stdio(false);
  cin >> tmp >> in;
  for (unsigned i = 0; i < in.length(); i += 2) {
    int x = (from_hex(in[i]) * 16 + from_hex(in[i + 1]));
    out.push_back(x ^ tmp);
    tmp = x;
  }
  cout << out << endl;
  return 0;
}
```

B

难度估计：普及-

简单的贪心。

首先判断是否有矛盾。如果有矛盾意味着不存在可行解。如果不存在矛盾，且 $\exists i$, s.t. $b_i \neq -1$ ，则必然有唯一解，输出 1 即可。否则，只需枚举可能的 s 即可。容易证明，可能的 s 的数量为 $k - \max a_i + \min a_i + 1$ 。因此输出即可。具体实现参考代码。

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pll pair<ll, ll>
const ll MAXN = 200010;
ll n, k;
ll a[MAXN], b[MAXN];
void solve() {
  cin >> n >> k;
  for (int i = 0; i < n; i++)
    cin >> a[i];
  for (int i = 0; i < n; i++)
    cin >> b[i];
  int s = -1;
  for (int i = 0; i < n; i++) {
    if (b[i] != -1) {
      if (s == -1)
        s = a[i] + b[i];
      else {
        if (s != a[i] + b[i]) {
          cout << 0 << '\n';
          return;
        }
      }
    }
  }
  if (s == -1) {
    int mx = *max_element(a, a + n) - *min_element(a, a + n);
    cout << k - mx + 1 << '\n';
    return;
  }
```

```
  for (int i = 0; i < n; i++) {
    if (a[i] > s || s - a[i] > k) {
      cout << 0 << '\n';
      return;
    }
  }
  cout << 1 << '\n';
}
int main() {
  ll test_cases;
  ios::sync_with_stdio(false);
  cin >> test_cases;
  while (test_cases--) {
    solve();
  }
  return 0;
}
```

C

难度估计：普及/提高-

LIS 模板题。注意 100 pts 需要使用 $O(n \log n)$ 的解法。二分树状数组均可。具体解答自行搜索。

二分版本：

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pll pair<ll, ll>
const ll MAXN = 100010;
int n;
int val[MAXN];
int tail[MAXN], cur;
int main() {
  ios::sync_with_stdio(false);
  cin >> n;
  for (int i = 1; i <= n; ++i) {
    cin >> val[i];
  }
  tail[++cur] = val[1];
```

```
  for (int i = 2; i <= n; ++i) {
    if (val[i] > tail[cur]) {
      tail[++cur] = val[i];
    } else {
      int p = lower_bound(tail + 1, tail + cur + 1, val[i]) - tail;
      tail[p] = val[i];
    }
  }
  cout << cur << endl;
  return 0;
}
```

树状数组版本:

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pll pair<ll, ll>
const ll MAXN = 100010;
int n;
int val[MAXN], tmp[MAXN], tot;
int bit[MAXN];
int dp[MAXN];
int lowbit(int x) { return x & (-x); }
void add(int x, int v) {
  for (; x <= n; x += lowbit(x)) {
    bit[x] = max(bit[x], v);
  }
}
int sum(int x) {
  int ans = 0;
  for (; x; x -= lowbit(x)) {
    ans = max(ans, bit[x]);
  }
  return ans;
}
int main() {
  ios::sync_with_stdio(false);
  cin >> n;
  for (int i = 1; i <= n; ++i) {
    cin >> val[i];
```

```
    tmp[i] = val[i];
  }

  sort(tmp + 1, tmp + n + 1);
  tot = unique(tmp + 1, tmp + n + 1) - tmp - 1;
  for (int i = 1; i <= n; ++i) {
    val[i] = lower_bound(tmp + 1, tmp + tot + 1, val[i]) - tmp;
  }

  for (int i = 1; i <= n; ++i) {
    dp[i] = sum(val[i] - 1) + 1;
    add(val[i], dp[i]);
  }
  int ans = 0;
  for (int i = 1; i <= n; ++i) {
    ans = max(ans, dp[i]);
  }
  cout << ans << endl;
  return 0;
}
```

D

难度估计：普及 +/提高

首先不考虑插入任何元素的做法。容易想到贪心策略：每当扫到 `a` 中的一个元素时，若它不小于下一个 `b` 中的元素，就取出这个元素。如果这个贪心策略能完整地运行结束，意味着最终答案为 0 。

那么，我们开始考虑插入元素会导致什么。这个时候可以有一个想法：**插入一个元素，实际上相当于在 `b` 中跳过了一个元素**。这是因为，贪心地讲，插入的这一个元素必须改变原来的贪心策略下选中的元素的集合，而如果 k 本身没有被选中，那么就相当于没有插入这个元素。从而，k 会占据选出的集合中的一位，相当于删除了 $\{b_i\}$ 中对应位置的元素。这个时候就有一个显然的想法：**枚举所有可能的 i ，删去 b_i 后贪心计算**。这么做预期可以得到 `60pts` 的分数。

接下来，考虑如何优化。可以想到一个常见的预处理策略：将这个问题拆分成前缀和后缀上的部分处理。考虑 s_i 和 t_i ，其中 s_i 是最小的下标 j ，使得 $b_1 \cdots b_i$ 可以在 $a_1 \cdots a_j$ 上执行此前所说的贪心策略；t_i 是最大的下标 j ，使得 $b_i \cdots b_m$ 可以在 $a_j \cdots a_n$ 上运行这个贪心策略。这有点类似二分 LIS 中 `tail` 数组的思路。

那么，只需要预处理出 s，t 两个数组，然后扫一遍，若 i 满足 $s_{i-1} < t_{i+1}$ ，则意味着可以通过删去 b_i 来使得贪心策略成立。特别地，对于 $i = 1$ 只需 $t_2 > 0$ ，对于 $i = m$ 只需 $s_{m-1} < n$ 。接着，取所有满足条件的 i 中 b_i 最小的一个即可。如果不存在，则无解。

代码如下:

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pll pair<ll, ll>
const ll INF = 1ll << 62;
void solve() {
  ll n, m;
  cin >> n >> m;
  vector<ll> a(n), b(m);
  for (ll i = 0; i < n; i++)
    cin >> a[i];
  for (ll i = 0; i < m; i++)
    cin >> b[i];
  vector<ll> backwards_match(m);
  ll j = n - 1;
  for (ll i = m - 1; i >= 0; i--) {
    while (j >= 0 && a[j] < b[i])
      j--;
    backwards_match[i] = j--;
  }
  vector<ll> forwards_match(m);
  j = 0;
  for (ll i = 0; i < m; i++) {
    while (j < n && a[j] < b[i])
      j++;
    forwards_match[i] = j++;
  }
  if (forwards_match.back() < n) {
    cout << 0 << endl;
    return;
  }
  ll ans = INF;
  for (ll i = 0; i < m; i++) {
    ll match_previous = i == 0 ? -1 : forwards_match[i - 1];
    ll match_next = i + 1 == m ? n : backwards_match[i + 1];
    if (match_next > match_previous) {
      ans = min(ans, b[i]);
    }
```

```
  }
  cout << (ans == INF ? -1 : ans) << "\n";
}
int main() {
  ll test_cases;
  ios::sync_with_stdio(false);
  cin >> test_cases;
  while (test_cases--) {
    solve();
  }
  return 0;
}
```