Problem
You have just finished cooking for some diners at the Infinite House of Pancakes. There are S stacks of pancakes in ll, and you have arranged them in a line, such that the i-th stack from the left (counting starting from 1) has Pi panca es.

Your supervisor was about to bring out the stacks to the customers, but then it occurred to her that a picture of the st cks might make for a good advertisement. However, she is worried that there might be too many stacks, so she inten s to remove the L leftmost stacks and the R rightmost stacks, where L and R are nonnegative integers such that L + ≤ S - 3. (Notice that at least 3 stacks of pancakes will remain after the removal.)

Your supervisor also thinks the remaining stacks will look aesthetically pleasing if they have the pyramid property. sequence of N stacks of heights H1, H2, ... , HN has the pyramid property if there exists an integer j (1 ≤ j ≤ N) such that H1 ≤ H2 ≤ ... ≤ Hj-1 ≤ Hj and Hj ≥ Hj+1 ≥ ... ≥ HN-1 ≥ HN. (It is possible that this sequence might not look mu h like a typical "pyramid" — a group of stacks of the same size has the pyramid property, and so does a group in whi h the stack heights are nondecreasing from left to right, among other examples.)

Note that the sequence of stacks remaining after your supervisor removes the L leftmost and R rightmost stacks mig t not yet have the pyramid property... but you can fix that by adding pancakes to one or more of the stacks! The pyra idification cost of a sequence of stacks is the minimum total number of pancakes that must be added to stacks to give the sequence the pyramid property.

While your manager is carefully deciding which values of L and R to choose, you have started to wonder what the s m of the pyramidification costs over all valid choices of L and R is. Compute this sum, modulo the prime 109+7 (10 0000007).

Input
The first line of the input gives the number of test cases, T. T test cases follow. Each begins with one line containing one integer S: the number of stacks of pancakes. Then, there is one more line containing S integers P1, P2, ..., PS. T e i-th of these is the number of pancakes in the i-th stack from the left.

Output
For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is th sum of the pyramidification costs over all valid choices of L and R, modulo the prime 109+7 (1000000007).

Limits
Time limit: 30 seconds per test set.
Memory limit: 1GB.
1 ≤ T ≤ 100.
1 ≤ Pi ≤ 109, for all i.

Test set 1 (Visible)
S = 3000, for up to 20 test cases.
3 ≤ S ≤ 500, for all remaining cases.

Test set 2 (Hidden)
S = 106, for up to 1 test case.
S = 105, for up to 3 test cases.
3 ≤ S ≤ 10000, for all remaining cases.

Sample

Input

Output

3
3
2 1 2
5
1 6 2 5 7
4
1000000000 1 1 1000000000

Case #1: 1
Case #2: 16
Case #3: 999999991

In Sample Case #1, your supervisor must choose L = 0 and R = 0, so that is the only scenario you need to consider. he optimal strategy for that scenario is to add a single pancake to the middle stack. Although the resulting sequence f stacks looks flat, notice that it has the pyramid property; in fact, any index will work as the j value.

In Sample Case #2, here are all possible choices of L and R, the corresponding remaining stacks, and what you shou d do in each scenario.

L = 0, R = 0: H = [1 6 2 5 7]. The optimal solution is to add four pancakes to the third stack and one pancake to the f urth stack. Then we have [1 6 6 6 7], which has the pyramid property with j = 5.
L = 0, R = 1: H = [1 6 2 5]. The optimal solution is to add three pancakes to the third stack. Then we have [1 6 5 5], hich has the pyramid property with j = 2.
L = 0, R = 2: H = [1 6 2]. This already has the pyramid property with j = 2.
L = 1, R = 0: H = [6 2 5 7]. The optimal solution is to add four pancakes to the second stack and one pancake to the t ird stack. Then we have [6 6 6 7], which has the pyramid property with j = 4.
L = 1, R = 1: H = [6 2 5]. The optimal solution is to add three pancakes to the second stack. Then we have [6 5 5], w ich has the pyramid property with j = 1.
L = 2, R = 0: H = [2 5 7]. This already has the pyramid property with j = 3.
So the answer is (5 + 3 + 0 + 5 + 3 + 0) modulo (109 + 7), which is 16.

In Sample Case #3, we only need to add extra pancakes to create the pyramid property when L = 0 and R = 0. In that case, it is optimal to add 999999999 pancakes to each of the second and third stacks. (We hope the diners are hungry) So the answer is (999999999 + 999999999) modulo (109 + 7) = 999999991.

Solution:

```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <bits/stdc++.h>

using namespace std;
```

```
typedef long long int64;
typedef unsigned long long uint64;
#define two(X) (1<<(X))
#define twoL(X) (((int64)(1))<<(X))
#define contain(S,X) (((S)&two(X))!=0)
#define containL(S,X) (((S)&twoL(X))!=0)
const double pi=acos(-1.0);
const double eps=1e-11;
template<class T> inline void ckmin(T &a,T b){if(b<a) a=b;}
template<class T> inline void ckmax(T &a,T b){if(b>a) a=b;}
template<class T> inline T sqr(T x){return x*x;}
typedef pair<int,int> ipair;
#define SIZE(A) ((int)A.size())
#define LENGTH(A) ((int)A.length())
#define MP(A,B) make_pair(A,B)
#define PB(X) push_back(X)
#define FOR(i,a,b) for(int i=(a);i<(b);++i)
#define REP(i,a) for(int i=0;i<(a);++i)
#define ALL(A) A.begin(),A.end()
using VI=vector<int>;

template<typename base_type, base_type _MOD>
class IntMod
{
public:
 static const int INVERSE_CACHE_SIZE = (1 << 20);
 static base_type MOD;
 static void set_mod(base_type new_mod) { MOD = new_mod; }

 base_type n;

 IntMod(long long d = 0) { n = (d >= 0 ? d % MOD : (d % MOD + MOD) % MOD); }
 virtual ~IntMod() = default;

 IntMod operator-() const { return build(n == 0 ? 0 : MOD - n); }
 IntMod& operator+=(IntMod a) { n = (n >= MOD - a.n ? n - MOD + a.n : n + a.n); return *this; }
 IntMod& operator-=(IntMod a) { n = (n >= a.n) ? n - a.n : n - a.n + MOD; return *this; }
 IntMod& operator*=(IntMod a) { *this = *this * a; return *this; }
 IntMod& operator/=(IntMod a) { *this = *this / a; return *this; }

 static IntMod build(base_type n) { IntMod r; r.n = n; return r; }

 static base_type inverse_cache[INVERSE_CACHE_SIZE];
 static bool inverse_cache_ready;
 friend IntMod inverse(IntMod n) { return build(inverse_internal(n.n)); }
 static base_type inverse_internal(base_type n)
 {
  if (!inverse_cache_ready)
  {
   inverse_cache_ready=true;
   inverse_cache[0] = 0;
   inverse_cache[1] = 1;
```

```
   for (int n = 2; n < INVERSE_CACHE_SIZE; ++n) inverse_cache[n] = (MOD - (base_type)((long long)inverse_ca
he[MOD % n] * (MOD / n) % MOD));
  }
  return n < INVERSE_CACHE_SIZE ? inverse_cache[n] : MOD - (base_type)((long long)inverse_internal(MOD
 n) * (MOD / n) % MOD);
 }

 friend bool operator==(IntMod a, IntMod b) { return a.n == b.n; }
 friend bool operator!=(IntMod a, IntMod b) { return a.n != b.n; }
 friend IntMod operator+(IntMod a, IntMod b) { return build(a.n >= MOD - b.n ? a.n - MOD + b.n : a.n + b.n); }
 friend IntMod operator-(IntMod a, IntMod b) { return build(a.n >= b.n ? a.n - b.n : a.n - b.n + MOD); }
 friend IntMod operator*(IntMod a, IntMod b) { return build(static_cast<base_type>(static_cast<long long>(a.n) * b
n % MOD)); }
 friend IntMod operator/(IntMod a, IntMod b) { return a * inverse(b); }
 friend IntMod pow(IntMod p, long long e)
 {
  if (e <= 0) return IntMod(1);
  IntMod r = IntMod(1);
  while (1) { if (e & 1) r *= p; e /= 2; if (e) p = p * p; else break; }
  return r;
 }
};
template<typename base_type, base_type _MOD> base_type IntMod<base_type, _MOD>::inverse_cache[INVERS
_CACHE_SIZE];
template<typename base_type, base_type _MOD> bool IntMod<base_type, _MOD>::inverse_cache_ready;
template<typename base_type, base_type _MOD> base_type IntMod<base_type, _MOD>::MOD = _MOD;

#define MOD (1000000007)
using Int = IntMod<int,MOD>;

int main()
{
#ifdef _MSC_VER
 freopen("input.txt","r",stdin);
#endif
 std::ios::sync_with_stdio(false);
 int testcase;
 cin>>testcase;
 for (int case_id=1;case_id<=testcase;case_id++)
 {
  int n;
  cin>>n;
  VI a(n);
  REP(i,n) cin>>a[i];
  VI prev(n);
  VI q;
  REP(i,n)
  {
   for (;SIZE(q)>0 && a[q.back()]<a[i];q.pop_back());
   prev[i]=(SIZE(q)==0?-1:q.back());
   q.push_back(i);
  }
```

```
  VI next(n);
  q.clear();
  for (int i=n-1;i>=0;i--)
  {
   for (;SIZE(q)>0 && a[q.back()]<=a[i];q.pop_back());
   next[i]=(SIZE(q)==0?n:q.back());
   q.push_back(i);
  }
  Int ret=0;
  vector<Int> s(n+1);
  s[0]=0;
  REP(i,n) s[i+1]=s[i]+Int(a[i]);
  REP(i,n)
  {
   /*
   int p1=i-1;
   for (;p1>=0 && a[p1]<a[i];--p1);
   int p2=i+1;
   for (;p2<n && a[p2]<=a[i];++p2);
   assert(p1==prev[i]);
   assert(p2==next[i]);
   */
   int p1=prev[i];
   int p2=next[i];
   if (p2<n && p2-i>1)
   {
    ret+=(Int(a[i])*Int(p2-i-1)-s[p2]+s[i+1])*Int(i-p1)*Int(n-p2);
    // for (int k=i+1;k<p2;k++) ret+=Int(a[i]-a[k])*Int(i-p1)*Int(n-p2);
   }
   if (p1>=0 && i-p1>1)
   {
    ret+=(Int(a[i])*Int(i-p1-1)-s[i]+s[p1+1])*Int(p2-i)*Int(p1+1);
    // for (int k=i-1;k>p1;k--) ret+=Int(a[i]-a[k])*Int(p2-i)*Int(p1+1);
   }
  }
  printf("Case #%d: %d\n",case_id,ret.n);

 }
 return 0;
}
```