```cpp
#include <bits/stdc++.h>
constexpr int P = 998244353;
std::vector<int> rev, roots{0, 1};
int power(int a, int b) {
    int res = 1;
    for (; b; b >>= 1, a = 1ll * a * a % P)
        if (b & 1)
            res = 1ll * res * a % P;
    return res;
}
void dft(std::vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; ++i)
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
    }
    for (int i = 0; i < n; ++i)
        if (rev[i] < i)
            std::swap(a[i], a[rev[i]]);
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            int e = power(3, (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); ++i) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = 1ll * roots[i] * e % P;
            }
            ++k;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; ++j) {
                int u = a[i + j];
                int v = 1ll * a[i + j + k] * roots[k + j] % P;
                int x = u + v;
                if (x >= P)
                    x -= P;
                a[i + j] = x;
                x = u - v;
                if (x < 0)
                    x += P;
                a[i + j + k] = x;
            }
        }
    }
}
```

```
51  void idft(std::vector<int> &a) {
52      int n = a.size();
53      std::reverse(a.begin() + 1, a.end());
54      dft(a);
55      int inv = power(n, P - 2);
56      for (int i = 0; i < n; ++i)
57          a[i] = 1ll * a[i] * inv % P;
58  }
59  struct Poly {
60      std::vector<int> a;
61      Poly() {}
62      Poly(int a0) {
63          if (a0)
64              a = {a0};
65      }
66      Poly(const std::vector<int> &a1) : a(a1) {
67          while (!a.empty() && !a.back())
68              a.pop_back();
69      }
70      int size() const {
71          return a.size();
72      }
73      int operator[](int idx) const {
74          if (idx < 0 || idx >= size())
75              return 0;
76          return a[idx];
77      }
78      Poly mulxk(int k) const {
79          auto b = a;
80          b.insert(b.begin(), k, 0);
81          return Poly(b);
82      }
83      Poly modxk(int k) const {
84          k = std::min(k, size());
85          return Poly(std::vector<int>(a.begin(), a.begin() + k));
86      }
87      Poly divxk(int k) const {
88          if (size() <= k)
89              return Poly();
90          return Poly(std::vector<int>(a.begin() + k, a.end()));
91      }
92      friend Poly operator+(const Poly a, const Poly &b) {
93          std::vector<int> res(std::max(a.size(), b.size()));
94          for (int i = 0; i < int(res.size()); ++i) {
95              res[i] = a[i] + b[i];
96              if (res[i] >= P)
97                  res[i] -= P;
98          }
99          return Poly(res);
100     }
101     friend Poly operator-(const Poly a, const Poly &b) {
102         std::vector<int> res(std::max(a.size(), b.size()));
103         for (int i = 0; i < int(res.size()); ++i) {
104             res[i] = a[i] - b[i];
105             if (res[i] < 0)
```

```
            res[i] += P;
        }
        return Poly(res);
    }
    friend Poly operator*(Poly a, Poly b) {
        int sz = 1, tot = a.size() + b.size() - 1;
        while (sz < tot)
            sz *= 2;
        a.a.resize(sz);
        b.a.resize(sz);
        dft(a.a);
        dft(b.a);
        for (int i = 0; i < sz; ++i)
            a.a[i] = 1ll * a[i] * b[i] % P;
        idft(a.a);
        return Poly(a.a);
    }
    Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    Poly deriv() const {
        if (a.empty())
            return Poly();
        std::vector<int> res(size() - 1);
        for (int i = 0; i < size() - 1; ++i)
            res[i] = 1ll * (i + 1) * a[i + 1] % P;
        return Poly(res);
    }
    Poly integr() const {
        if (a.empty())
            return Poly();
        std::vector<int> res(size() + 1);
        for (int i = 0; i < size(); ++i)
            res[i + 1] = 1ll * a[i] * power(i + 1, P - 2) % P;
        return Poly(res);
    }
    Poly inv(int m) const {
        Poly x(power(a[0], P - 2));
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (2 - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m) const {
        return (deriv() * inv(m)).integr().modxk(m);
    }
    Poly exp(int m) const {
```

```
        Poly x(1);
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (1 - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    Poly sqrt(int m) const {
        Poly x(1);
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
        }
        return x.modxk(m);
    }
    Poly mulT(Poly b) const {
        if (b.size() == 0)
            return Poly();
        int n = b.size();
        std::reverse(b.a.begin(), b.a.end());
        return ((*this) * b).divxk(n - 1);
    }
    std::vector<int> eval(std::vector<int> x) const {
        if (size() == 0)
            return std::vector<int>(x.size(), 0);
        const int n = std::max(int(x.size()), size());
        std::vector<Poly> q(4 * n);
        std::vector<int> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = std::vector<int>{1, (P - x[l]) % P};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly
&num) {
            if (r - l == 1) {
                if (l < int(ans.size()))
                    ans[l] = num[0];
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
```

```cpp
    }
};
using i64 = long long;
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int t;
    std::cin >> t;
    while (t--) {
        int n;
        std::cin >> n;
        std::vector<int> a(n);
        for (int i = 0; i < n; i++) {
            std::cin >> a[i];
        }
        std::function<Poly(int, int)> solve = [&](int l, int r) {
            if (r - l == 1) {
                return Poly(std::vector{1, a[l]});
            }
            int m = (l + r) / 2;
            return solve(l, m) * solve(m, r);
        };
        auto p = solve(0, n);
        int ans = 0, bin = 1;
        for (int i = 1; i <= n; i++) {
            bin = i64(bin) * i % P * power(n - i + 1, P - 2) % P;
            ans = (ans + i64(bin) * p[i]) % P;
        }
        std::cout << ans << "\n";
    }
    return 0;
}
```