

```

1  #include <bits/stdc++.h>
2  constexpr int P = 998244353;
3  std::vector<int> rev, roots{0, 1};
4  int power(int a, int b) {
5      int res = 1;
6      for (; b >= 1, a = 1ll * a * a % P)
7          if (b & 1)
8              res = 1ll * res * a % P;
9      return res;
10 }
11 void dft(std::vector<int> &a) {
12     int n = a.size();
13     if (int(rev.size()) != n) {
14         int k = __builtin_ctz(n) - 1;
15         rev.resize(n);
16         for (int i = 0; i < n; ++i)
17             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
18     }
19     for (int i = 0; i < n; ++i)
20         if (rev[i] < i)
21             std::swap(a[i], a[rev[i]]);
22     if (int(roots.size()) < n) {
23         int k = __builtin_ctz(roots.size());
24         roots.resize(n);
25         while ((1 << k) < n) {
26             int e = power(3, (P - 1) >> (k + 1));
27             for (int i = 1 << (k - 1); i < (1 << k); ++i) {
28                 roots[2 * i] = roots[i];
29                 roots[2 * i + 1] = 1ll * roots[i] * e % P;
30             }
31             ++k;
32         }
33     }
34     for (int k = 1; k < n; k *= 2) {
35         for (int i = 0; i < n; i += 2 * k) {
36             for (int j = 0; j < k; ++j) {
37                 int u = a[i + j];
38                 int v = 1ll * a[i + j + k] * roots[k + j] % P;
39                 int x = u + v;
40                 if (x >= P)
41                     x -= P;
42                 a[i + j] = x;
43                 x = u - v;
44                 if (x < 0)
45                     x += P;
46                 a[i + j + k] = x;
47             }
48         }
49     }
50 }

```

```

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)

```

```

106         res[i] += P;
107     }
108     return Poly(res);
109 }
110 friend Poly operator*(Poly a, Poly b) {
111     int sz = 1, tot = a.size() + b.size() - 1;
112     while (sz < tot)
113         sz *= 2;
114     a.a.resize(sz);
115     b.a.resize(sz);
116     dft(a.a);
117     dft(b.a);
118     for (int i = 0; i < sz; ++i)
119         a.a[i] = 1ll * a[i] * b[i] % P;
120     idft(a.a);
121     return Poly(a.a);
122 }
123 Poly &operator+=(Poly b) {
124     return (*this) = (*this) + b;
125 }
126 Poly &operator-=(Poly b) {
127     return (*this) = (*this) - b;
128 }
129 Poly &operator*=(Poly b) {
130     return (*this) = (*this) * b;
131 }
132 Poly deriv() const {
133     if (a.empty())
134         return Poly();
135     std::vector<int> res(size() - 1);
136     for (int i = 0; i < size() - 1; ++i)
137         res[i] = 1ll * (i + 1) * a[i + 1] % P;
138     return Poly(res);
139 }
140 Poly integr() const {
141     if (a.empty())
142         return Poly();
143     std::vector<int> res(size() + 1);
144     for (int i = 0; i < size(); ++i)
145         res[i + 1] = 1ll * a[i] * power(i + 1, P - 2) % P;
146     return Poly(res);
147 }
148 Poly inv(int m) const {
149     Poly x(power(a[0], P - 2));
150     int k = 1;
151     while (k < m) {
152         k *= 2;
153         x = (x * (2 - modxk(k) * x)).modxk(k);
154     }
155     return x.modxk(m);
156 }
157 Poly log(int m) const {
158     return (deriv() * inv(m)).integr().modxk(m);
159 }
160 Poly exp(int m) const {

```

```

161     Poly x(1);
162     int k = 1;
163     while (k < m) {
164         k *= 2;
165         x = (x * (1 - x.log(k) + modxk(k))).modxk(k);
166     }
167     return x.modxk(m);
168 }
169 Poly sqrt(int m) const {
170     Poly x(1);
171     int k = 1;
172     while (k < m) {
173         k *= 2;
174         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
175     }
176     return x.modxk(m);
177 }
178 Poly mult(Poly b) const {
179     if (b.size() == 0)
180         return Poly();
181     int n = b.size();
182     std::reverse(b.a.begin(), b.a.end());
183     return ((*this) * b).divxk(n - 1);
184 }
185 std::vector<int> eval(std::vector<int> x) const {
186     if (size() == 0)
187         return std::vector<int>(x.size(), 0);
188     const int n = std::max(int(x.size()), size());
189     std::vector<Poly> q(4 * n);
190     std::vector<int> ans(x.size());
191     x.resize(n);
192     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
193         if (r - l == 1) {
194             q[p] = std::vector<int>{1, (P - x[l]) % P};
195         } else {
196             int m = (l + r) / 2;
197             build(2 * p, l, m);
198             build(2 * p + 1, m, r);
199             q[p] = q[2 * p] * q[2 * p + 1];
200         }
201     };
202     build(1, 0, n);
203     std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly
&num) {
204         if (r - l == 1) {
205             if (l < int(ans.size()))
206                 ans[l] = num[0];
207         } else {
208             int m = (l + r) / 2;
209             work(2 * p, l, m, num.mult(q[2 * p + 1]).modxk(m - l));
210             work(2 * p + 1, m, r, num.mult(q[2 * p]).modxk(r - m));
211         }
212     };
213     work(1, 0, n, mult(q[1].inv(n)));
214     return ans;

```

```

215     }
216 };
217 using i64 = long long;
218 int main() {
219     std::ios::sync_with_stdio(false);
220     std::cin.tie(nullptr);
221     int t;
222     std::cin >> t;
223     while (t--) {
224         int n;
225         std::cin >> n;
226         std::vector<int> a(n);
227         for (int i = 0; i < n; i++) {
228             std::cin >> a[i];
229         }
230         std::function<Poly(int, int)> solve = [&](int l, int r) {
231             if (r - l == 1) {
232                 return Poly(std::vector{1, a[l]});
233             }
234             int m = (l + r) / 2;
235             return solve(l, m) * solve(m, r);
236         };
237         auto p = solve(0, n);
238         int ans = 0, bin = 1;
239         for (int i = 1; i <= n; i++) {
240             bin = i64(bin) * i % P * power(n - i + 1, P - 2) % P;
241             ans = (ans + i64(bin) * p[i]) % P;
242         }
243         std::cout << ans << "\n";
244     }
245     return 0;
246 }

```