SPOJ ADAPLANT

Ada and Plants

Prof. Edson Alves – UnB/FGA

Problema

Ada the Ladybug has grown many plants. She was trying to grow all plants with equal size. Now she is wondering about the biggest difference between heights of two plants which are near each other. Plants are near each other, if there are at most K plants between them.

Entrada e saída

Input

The first line contains T, the number of test-cases. The first line of each test-case will contain $N, K, 1 < N \leq 10^5, 0 \leq K \leq 10^5$ where N indicates number of plants. Next line will contain N integers $0 \leq hi \leq 10^9$ indicating height of i-th plant.

Sum of all N among all test-cases won't exceed 3×10^6.

Output

For each test-case, print exactly one number – the biggest difference of plants near each other (biggest $h_i - h_j$ such that $|i - j| - 1 \leq K$).

Exemplo de entradas e saídas

Sample Input

```
3
5 0
1 2 3 5 6
4 6
1 10 2 9
10 1
1 7 8 9 19 11 21 8 11 0
```

Sample Output

```
2
9
13
```

Solução com complexidade $O(N \log N)$

- Segundo o texto do problema, quaisquer duas plantas cujos índices estejam em um intervalo $[L, R)$, com $R - L = K + 2$, estão próximas o suficiente
- A cada intervalo $[L, R)$, a maior diferença ocorrerá entre os elementos com índices neste intervalo e que tenham a maior e a menor altura, respectivamente
- Logo, o problema pode ser resolvido usando a técnica de dois ponteiros com janela móvel (*sliding window*)
- Para obter, de forma eficiente, os valores da maior e da menor altura no intervalo, há duas formas
- Uma delas é manter as alturas dos elementos do intervalo em um `multiset`: os ponteiros `rbegin()` e `begin()` apontarão para os elementos desejados

Solução com complexidade $O(N \log N)$

- A inserção e remoção de um elemento no `multiset` é feita em $O(\log K)$
- Cada elemento será inserido ou removido do `multiset` no máximo uma vez, logo a solução terá complexidade $O(N \log K)$
- Observe que o valor de K pode exceder N, de modo que ele deve ser tratado com cuidado
- Também é possível resolver este problema em $O(N)$: basta usar a estratégia de dupla pilha para manter os valores do menor e do maior elemento do intervalo
- A implementação é mais longa do que a que utiliza o `multiset`, porém tem melhor tempo de execução e usa menos memória

Solução AC com complexidade $O(N \log K)$

```cpp
#include <bits/stdc++.h>

using namespace std;

int solve(int N, int K, const vector<int>& xs)
{
    K = min(N - 1, K + 1);

    int L = 0, R;
    multiset<int> s;

    for (R = 0; R <= K; ++R)
        s.insert(xs[R]);

    int ans = *s.rbegin() - *s.begin();

    while (R < N) {
        auto it = s.find(xs[L]);
        s.erase(it);
        s.insert(xs[R]);
```

Solução AC com complexidade $O(N \log K)$

```
22          ans = max(ans, *s.rbegin() - *s.begin());
23          ++L;
24          ++R;
25      }
26
27      return ans;
28  }
29
30  int main()
31  {
32      ios::sync_with_stdio(false);
33
34      int T;
35      cin >> T;
36
37      while (T--)
38      {
39          int N, K;
40          cin >> N >> K;
```

Solução AC com complexidade $O(N \log K)$

```
42          vector<int> xs(N);
43
44          for (int i = 0; i < N; ++i)
45              cin >> xs[i];
46
47          auto ans = solve(N, K, xs);
48
49          cout << ans << endl;
50      }
51
52      return 0;
53  }
```

Solução AC com complexidade $O(N)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;

const int oo { 1'000'000'010 };

struct StackM {

    function<int(int, int)> op;
    stack<ii> in, out;

    void add(int x) { add(x, in); }

    int top()
    {
        if (in.empty())
            return out.top().second;
        else if (out.empty())
            return in.top().second;
```

Solução AC com complexidade $O(N)$

```
22          return op(in.top().second, out.top().second);
23      }
24
25      void pop()
26      {
27          move();
28          out.pop();
29      }
30
31      void move()
32      {
33          if (out.empty())
34          {
35              while (not in.empty()) {
36                  auto x = in.top().first;
37                  in.pop();
38                  add(x, out);
39              }
40          }
41      }
```

Solução AC com complexidade $O(N)$

```
43      void add(int x, stack<ii>& s)
44      {
45          int m = s.empty() ? x : op(s.top().second, x);
46          s.emplace(x, m);
47      }
48  };
49
50  int solve(int N, int K, const vector<int>& xs)
51  {
52      K = min(N - 1, K + 1);
53
54      auto min_of = [](int x, int y) { return min(x, y); };
55      auto max_of = [](int x, int y) { return max(x, y); };
56
57      StackM smin { min_of, {}, {} };
58      StackM smax { max_of, {}, {} };
59
60      int L = 0, R;
```

Solução AC com complexidade $O(N)$

```
62      for (R = 0; R <= K; ++R) {
63          smin.add(xs[R]);
64          smax.add(xs[R]);
65      }
66
67      int ans = smax.top() - smin.top();
68
69      while (R < N)
70      {
71          smin.pop();
72          smax.pop();
73
74          smin.add(xs[R]);
75          smax.add(xs[R]);
76
77          ++L;
78          ++R;
79
80          ans = max(ans, smax.top() - smin.top());
81      }
```

Solução AC com complexidade $O(N)$

```
83      return ans;
84  }
85
86  int main()
87  {
88      ios::sync_with_stdio(false);
89
90      int T;
91      cin >> T;
92
93      while (T--)
94      {
95          int N, K;
96          cin >> N >> K;
97
98          vector<int> xs(N);
99
100         for (int i = 0; i < N; ++i)
101             cin >> xs[i];
```

Solução AC com complexidade $O(N)$

```
103         auto ans = solve(N, K, xs);
104
105         cout << ans << '\n';
106     }
107
108     return 0;
109 }
```