```cpp
/*
    | Divide & Conquer DP Optimization |
    Desc: Optimizing DP transitions in the form of

        dp[i][j] = min(dp[i-1][k-1] + C(k,j)) for (0 ≤ k ≤ j)

    where C(k,j) is a cost function.

    lets define opt(i,j) be the value of k that optimize dp[i][j].
    DnC ONLY APPLIES IF:

        opt(i,j) ≤ opt(i,j+1)

    One case of where this condition holds is when cost function C(k,j) satisfy the Quadrangle Inequality:

        C(a,c) + C(b,d) < C(a,d) + C(b,c) for a ≤ b ≤ c ≤ d. (Note that "<" indicates more optimal)

    Runs in O(n*log(n)*C) with C is time to compute Cost function C(k,j)

    Source: KawakiMeido
    State: Untested lmao
*/

void DnC (int k){
    deque<pair<int,pair<pii,pii>>> dq;
    int lvl = 0;
    dq.push_back({1,{{k,m},{k,m}}});
    while (!dq.empty()){
        auto in = dq.front();
        int curlvl = in.fi;
        int l = in.se.fi.fi;
        int r = in.se.fi.se;
        int optl = in.se.se.fi;
        int optr = in.se.se.se;
        dq.pop_front();

        if (curlvl≠lvl){
            lvl = curlvl;
            BIT.Init(m);
        }

        int mid = (l+r)/2;

        pii best = {0,-INF};

        for (int i = optl; i≤min(mid,optr); i++){
            int sum = dpPrev[i-1] - precalc[mid][i-1] + precalc[mid][mid];

            if (sum > best.se){
                best.fi = i;
                best.se = sum;
            }
        }
        dp[mid] = best.second;
        int opt = best.first;

        if (l<=mid-1) dq.push_back({lvl+1,{{l,mid-1},{optl,opt}}});
        if (mid+1<=r) dq.push_back({lvl+1,{{mid+1,r},{opt,optr}}});
    }
}
```