# Lattice MST editorial

**disclaimer**: i will have a lot of off by ones in this editorial... refer to implementation for those :)

## The base case

So I assume you have already read the statement.. if not, go do that now. So now try to solve it for $d = k = 2$, and $D[1] * D[2] \leq 10^6$. (i want a linear sol). The important observation here is that the furthest point from any given point is one of the corners. so if we have our corners then we can loop over each point and draw an edge to the furthest point from it (which is one of 4 points) in $O(D[1] * D[2])$ time. Note that there is actually an edge case since the edges between corners are duplicated, but we will handle that later. So lets define $S$ as the sum of the distances between all nodes and their furthest corners.

so,

$$S = \sum_{i=1}^{n} \sum_{j=1}^{m} \max(i, n-i)^2 + \max(j, m-j)^2$$

Now we can evaluate $S$ in $O(D[1] * D[2])$ time. Note that i didn't actually loop over the corners, but instead just took the largest direction it can go and squared it. Now the observation is, "oh wait i can seperate the sigmas" and, yes, you can. so we can rewrite $S$

$$S = m \cdot \sum_{i=1}^{n} max(i, n-i)^2 + n \cdot \sum_{j=1}^{m} max(j, n-j)^2$$

This is now evaluating $S$ is $O(D[1] + D[2])$ time. now comes the second observation, which is "oh wait, i dont need the sigmas... i can use sum of squares to find this sum", and, yes, you are right. with some casework on when n is odd and even, the sigma expands to something like $n^2 + (n-1)^2 + (n-2)^2 \cdots (n/2+1)^2 + (n/2)^2 + (n/2+1)^2 \cdots (n-2)^2 + (n-1)^2 + (n)^2$ or it goes down to $(n/2)^2$ and back up again. so if we have $sumsq(n) = \frac{(n)(n+1)(2n+1)}{6}$ or our favorite sum of squares formula, the answer looks something like

$$S = m \cdot (sumsq(n) - sumsq(n/2)) \cdot 2 + n \cdot (sumsq(m) - sumsq(m/2)) \cdot 2$$

Now we have $S$ evaluated in $O(1)$ time. I (chessbot), initially proposed this problem with $d = k = 2, D[1] * D[2] \leq 10^6$ and PurpleCrayon made the above observations in around 10 minutes, and you can blame him for having to generalize this problem to $d$ dimenions and $k$ powers for distance :).

## $k$ powers

Now comes the first of two fun parts, generalizing the distance formula. So now lets keep $d = 2$ but $k$ can now be $\leq 500$ and $D[1], D[2] \leq 10^9$. Now we can define $kthsum$.

$$kthsum(n, k) = \sum_{i=1}^{n} i^k$$

So we can see that $kthsum(n, k)$ and be found in $O(n \log k)$ time by simply looping over $n$ and using binary exponentiation. Note that $O(n)$ time with $O(n)$ memory is also doable with seive :). But in this context, $n$ is $D[i]$ which is $10^9$ and then we tle. So the idea is to make a $k + 1$ -degree polynomial that uniquely defines the set of points $\{(1, 1^k), (2, 1^k + 2^k), (3, 1^k + 2^k + 3^k), \cdots, (k + 2, 1^k + 2^k + \cdots (k + 2)^k)\}$. We know that the sum of $k$ powers can be expressed as a polynomial that grows relateive to $n^{(k+1)}$ (proof is left as an exercise to the reader), and we also know that an $n$-degree polynomial is unique defined by $n + 1$ points. Now we can find a $k + 1$-degree polynomial that goes through the first $k + 2$ points of $i, kthsum(i, k)$. This can be done in $O(k \log mod)$ or $O(k)$ depending on the implementation. this blog gives an excellent approach to Lagrange interpolation, the technique used here and this blog is literally the editorial of finding $kthsum(n, k)$ (problem F).

## d dimensions

Now lets define a new function, $eval(n, k)$. which is basically $\sum_{i=1}^{n} max(i, n - i)^k$ but evaluated faster since it looks more like

$$eval(n, k) = 2 * (kthsum(n, k) - kthsum(n/2, k))$$

with some edge cases concerning even/odd $n$, but $eval(n,k)$ can be evaluted in $O(k \log mod)$ time. Now lets try to generalize or 2d approach into 3d. Consider the contribution of $eval(D[1],k))$ for some $D[1]$ and some $k$. If $d=2$ then the contribution is just $D[2]$ since there are $D[2]$ "rows" that use the sum of $eval(D[1],k)$. Now lets say $d=3$ , the new contribution is just $D[2]*D[3]$ since there are that many "rows" in the 3d figure. Try to draw it out on paper, it will help. Now with this, let's redefine $S$.

$$S=\sum_{i=1}^{d}(\prod_{j=1,j\neq i}^{d}D[j])\cdot eval(D[i],k)$$

or the product of all the dimensions save $D[i]$ multiplied by $eval(D[i],k)$. The important observation here is that $S$ can be evaluated in $O(d\cdot k \log mod)$ time. which is enough to fit within the limits. So are we done? no, we stil have not handled the case for corners sharing edges.

## corners

Now we have $S$ but it turns out $S$ is wrong as the answer to our MST. So for the most basic case, we will solve the case of corners for $d=2$. Let's define $n=D[1], m=D[2]$. So we have two edges from $(1,1)\rightarrow(n,m)$ and two from $(1,m)\rightarrow(n,1)$ since the corners are the furthest nodes from each other. Now we should subtract $2*(n-1)^2*(m-1)^2$ or the weights of two of the corner edges and add one more edge in that is legal. Now i suggest you grab a sheet of paper since it will help. Imagine a $4\times4$ lattice. the edges will look something like edges from each corner to the nodes near the opposite corner. So edges out of $(1,1)$ will end at nodes close to $(4,4)$. But how close? around $4/2$ close or just $2$ nodes close. Since any further and it would be just as (if not more) optimal to draw your edge out from another corner. More specifically, the edges from $(1,1)$ go to the nodes between $(n/2,m/2)$ and $(n,m)$ with some off by ones concerning rounding $n/2$ and $m/2$. So the furthest node from $(1,1)$ that isn't directly counted is either node $(n/2,m)$ or node $(m/2,n)$ (proof is left as an exercise to the reader). So in the end if we have $S$ already, we should subtract $(n-1)^2+(m-1)^2$ twice and add $\max((n/2)^2+m^2,(m/2)^2+n^2)$. note there are off by ones. Now lets try this on 3d, imagine a $3\times3\times2$ grid. It gets a little scuffed, but on a $3$ dimensional grid, the furthest point not already connect is $\max((D[i]/2)^k+\sum_{j=1,j\neq i}^{d}D[j]^k)$ across all i. Proof is left as an exercise to the reader. Now try to see by always connecting the furthest legal edge doesn't work. So on the $3\times3\times2$ board, we have that the best edge is between corners on a $3\times3$ face. But cant draw more than $2$ of them before a cycle appears. So you have to take $1$ of the second best legal edge. So the result is a formula that works for $d$ dimensions. Let's define $C$ as the sum of all of the "legal" edges we are taking after removing the duplicate corners

pair edges. If $D$ is sorted in least to greatest order

$$C = \sum_{i=1}^{d-1} 2^{i-1} \cdot ((D[i]/2)^k + \sum_{j=1, j \neq i}^{d} D[j]^k)$$

which is taking as many of the largest legal edge as possible, then as many of the second largest. This formula can be proven with something like kruskal's. So the our end answer is

$$S - C - (2^{d-1} * (\sum_{i=1}^{d} D[i]^k))$$

So we have $S$ and we subtract $C$ and the last term is the amount of duplicate diagonal or edges between pairs of corners multipled by the weight of such a corner edge. This entire thing is evaluated in $O(d \cdot k \cdot mod)$ time or $O(d \cdot k)$ depending on the implementation. Thank chessbot for the good problem and blame purple for the bad problem. below is my code, i yoinked benq's mint and the CF editorial's interpolation...

```
//gyrating cat enthusiast
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <string>
#include <utility>
#include <cassert>
#include <algorithm>
#include <vector>
#include <random>
#include <chrono>
#include <queue>
#include <set>

#define ll long long
#define lb long double
#define pii pair<int, int>
#define pb push_back
#define mp make_pair
#define ins insert
#define cont continue
```

```cpp
#define siz(vec) ((int)(vec.size()))

#define LC(n) (((n) << 1) + 1)
#define RC(n) (((n) << 1) + 2)
#define init(arr, val) memset(arr, val, sizeof(arr))
#define bckt(arr, val, sz) memset(arr, val, sizeof(arr[0]) * (sz+5))
//#define uid(a, b) uniform_int_distribution<int>(a, b)(rng)
#define tern(a, b, c) ((a) ? (b) : (c))
#define feq(a, b) (fabs(a - b) < eps)
#define abs(x) tern((x) > 0, x, -(x))

#define moo printf
#define oom scanf
#define mool puts("")
#define orz assert
#define fll fflush(stdout)

const lb eps = 1e-9;
const ll mod = 1e9 + 7, MOD = 1e9 +7, ll_max = (ll)1e18;
const int MX = 1e6 +10, int_max = 0x3f3f3f3f;

using namespace std;
//mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
//------------------------------------------
//this is benq's short mint template
//https://github.com/bqi343/USACO/blob/master/Implementations/content/number-theory%20(11.1)/Modular%20Arithmetic/ModIntShort.h
struct mi {
  int v; explicit operator int() const { return v; }
  mi() { v = 0; }
  mi(ll _v):v(_v%MOD) { v += (v<0)*MOD; }
};
mi& operator+=(mi& a, mi b) {
  if ((a.v += b.v) >= MOD) a.v -= MOD;
  return a; }
mi& operator-=(mi& a, mi b) {
  if ((a.v -= b.v) < 0) a.v += MOD;
  return a; }
```

```
mi operator+(mi a, mi b) { return a += b; }
mi operator-(mi a, mi b) { return a -= b; }
mi operator*(mi a, mi b) { return mi((ll)a.v*b.v); }
mi& operator*=(mi& a, mi b) { return a = a*b; }
mi Pow(mi a, ll p) { assert(p >= 0); // asserts are important!
  return p==0?1:Pow(a*a,p/2)*(p&1?a:1); }
mi inv(mi a) { assert(a.v != 0); return Pow(a,MOD-2); }
mi operator/(mi a, mi b) { return a*inv(b); }
//------------------------------------------

vector<mi> sums;
int D[MX];

int kthsum(int n, int k){ //sum of 1^k, 2^k, ... x^k
  sums.clear(); sums.pb(mi());

  mi co(1), ans, t;

  for(int i = 1; i<=k+1; i++){
    t += Pow(i, k); sums.pb(t);
  }
  if(n < siz(sums)) return int(sums[n]);
  for(int i = 1; i<siz(sums); i++){
    co *= mi(n - i);
    co = co / mi(0 - i);
  }
  for(int i = 0; i<siz(sums); i++){
    ans += co * sums[i];
    if(i + 1 >= siz(sums)) break;
    co *= mi(n - i)/mi(n - (i + 1));
    co *= mi(i - (siz(sums) - 1))/mi(i + 1);
  }
  return int(ans);
}


ll pOw(ll a, ll k){
  ll ans = 1;
```

```
  for(ll i = 1; i <= k; i *= 2ll){
    if(k&i){
      (ans *= a) %= mod;
    }
    (a *= a) %= mod;
  }
  return ans;
}

ll eval(int x, int k){
  return (((ll)kthsum(x, k) - (ll)kthsum(x/2, k))*2ll + (ll)tern(x%2, 0ll, pOw(x/2, k)))%mod;
}


int main(){
  cin.tie(0) -> sync_with_stdio(0);
  int T; cin >> T;
  while(T--){
    ll d, k; cin >> d >> k;
    for(int i = 0; i<d; i++){
      cin >> D[i];
    }

    sort(D, D + d);

    mi P(1), ans(0), pfull(0), phalf(0);
    ll V = (1ll << d); //number of corners

    for(int i = 0; i<d; i++){
      P *= mi(D[i]);
      D[i]--; orz(D[i] >= 0);
      pfull += Pow(mi(D[i]), k);
    }

    for(int i = 0; i<d; i++){
      ans += mi(eval(D[i], k)) * (P/mi(D[i] + 1));
    }
```

```
    //ans = S
    //mst without corners done

    for(ll i = 0, j = V/4ll; j; i++, j >>= 1ll){
      ans += mi(j) * (pfull + Pow(D[i]/2, k) - Pow(D[i], k));
      //note D[i] is not arbitrary, i sorted it earlier
    }
    //ans = S - C
    ans -= pfull*mi(V/2ll); //v/2 duplicate diagonals
    //ans = S - C - the last term

    cout << int(ans) << "\n";
  }
  return 0;
}

```

if my template is ignored, the code is quite short for such a proof heavy problem :). Good luck upsolving!