**Solutions**

**Problem: CF 426B**

Origin: http://codeforces.com/contest/834/submission/29027867

Learned: sweeps can be easier done with ordered map and set which erases.

**Problem: CF 426C (Math)**

Origin: http://codeforces.com/blog/entry/53567

Learned: use math before brute forcing solutions, number theory problems are hard, they almost never involve calculating divisors… math with cubic roots and non cubic roots binary search on 10^9, for math problems look at tricks!

**Contest Summary 426C: Please think of doing it with a trick before using brute force methods**

**Problem: CF 427B**

Origin: <http://codeforces.com/blog/entry/53588>

Learned: Simplify and do it with less coding if you can 20 minutes too long

**Problem: CF 427C (Partial Sums)**

Origin: <http://codeforces.com/blog/entry/53588>

Learned: psums implementation good, coding fast, thinking too long go faster!

2) Alternate way of implementing with rectangles = psum[x2][y2] – psum[x1-1][y2] – psum[x2][y1-1] + psum[x1-1][y1-1]

**Problem: CF 427D (DP Palindromes)**

Origin: <http://codeforces.com/contest/835/problem/D>

<http://codeforces.com/submissions/Ali.Pi/contest/835> (hashing solution)

Learned: 1) Learned how to do Hashing in O(n) 2) Fix hashing it has problems in it later 3) Can do DP on palindromes in O(n^2) time and memory 4) use recursive relationships for everything to find the mark and f[][] 5) sum it all together!

**Contest Summary CF 427**: Think of DP before using hashing, learned hashing with palindromic substrings, code faster = good -> great

Segment Tree Practice

**Problem: Haybales USACO Platinum December 2015 RMQ, Lazy, Sum Seg Tree**

Origin: http://www.usaco.org/current/data/sol\_haybales\_platinum\_dec15.html

Learned: Practice with rmq and sum lazy propagation. Argh long long be careful bc 1LL << 60 not 1 << 60 (long long) error make sure to fix. Argh forgot to update both in lazy propagation. Bug! Very tricky

**Problem: Promotion Counting USACO Platinum January 2017**

Origin: <http://www.usaco.org/current/data/sol_promote_platinum_jan17.html>

Learned: 1) Tree and segment / bit tree problem clearly n = 100000 2) tree + segment/bit always involves a certain preorder/inorder traversal which puts the objects in a proper ordering 3) preorder traversal = simplifies tree problem into an array problem 4) then sort the states in decreasing order then use bit to update the result.

**Problem: CF 424D (greedy)**

Origin: <http://codeforces.com/contest/831/problem/D>

Learned: 1) Make observations first, does not have to be DP problem even if it looks like DP 2) Make observation that all the segments have to be in order 2) DP Solution also correct 3) memorization may TLE so do not use 4) DP solution just make sure you initialize with big enough integer 0x3f to small needs INT\_MAX as INF

**Problem: CF 424A (brute force)**

Origin: <http://codeforces.com/contest/831/problem/A>

Learned: 1) took 20 minutes too long 2) could have done it with one for loop sweep.

**Problem: CF 424C (prefix sums)**

Origin: <http://codeforces.com/contest/831/status/C>

Learned: 1) you can also use prefix sums to solve this problem

**Contest Summary CF 423:** Good:Coding really fast in first two problems! Nice job! Last problem Problem C = got stuck because TLE bc reading in 2 million strings Bad: Chuck you need to remember to use fast i/o when you are switching to cin/cout. Switch to cin/cout

**Problem: CF 423C (string problem, union find problem)**

Origin: <http://codeforces.com/contest/828/status/C>

Learned: 1) switch to cin / cout with ios::sync\_with\_stdio(false) 2) can do it with other method of sorting it and then processing strings too 3) can also use the union-find get\_next method to solve it even faster (cool) 4) whenever there is a continuous line segment and you are updating intervals with same value you should think of union - find

**Problem: CF 423D (greedy graph problem)**

Origin: http://codeforces.com/contest/828/problem/D  
  
Learned: 1) make a greedy observation that the tree should just be a star – graph which is optimal. 2) coding is really simple.

**Problem: CF 275D (BIT + segment + prefix sums)**

Origin: <http://codeforces.com/blog/entry/14417>

Learned: 1) release and do it for and with God!! 2) rest!!! :D :D 10x more productive at least 3) work in spurts and work hard!!! 4) make cool observations about and that all 1’s have to be 1’s and all 0’s have to have at least 1 0 in interval 5) can split it up into individual bits to solve 6) then realize you can process all the 1’s first to evaluate and make sure the 1’s condition is satisfied 7) then with bit shift operators you can verify if 0’s situation can be satisfied 8) then you solve!

**Problem: CF 242E (Segment Tree + XOR + Lazy Updates)**

Origin: <http://codeforces.com/contest/242/status/E/page/4?order=BY_CONSUMED_TIME_ASC>

Learned: 1) when you xor you are doing it bit by bit so you can split it into the 20 bits of the xor, tree[x][i] stands for the xor of the value at node x (which represents a range of values) and at bit i 2) after you decide to update bit by bit, you realize that summing is really easy with lazy propagation and so is building 3) only update is a little tricky and requires you to realize that you can only update when there value you are pushing has a bit at this bit value. 4) write helper push function to update.

**Problem: CF 422C (Sorting + Query Updates)**

Origin: http://codeforces.com/contest/822/status/C

Learned: 1) Important observation is that you don’t need to check both sides!!! You can only check if ex < sy because the other situation is covered 2) Then you can process it in forms of queries with l and r, queries. 3) You can solve it with sorting, with pq, and with two pointer.

**Contest Summary CF 422**: Good: Got my highest rank 601 Solved Problem D with guess and check Bad: Was not very focused, not present, kind of gave up on solving problem C LOL, should have focused on finding a different sort instead of debugging, I need to think a little bit more before I code, find the trick (don’t just jump on a solution when you see one), 2 improvements, 1) be present + 10 min 2) don’t give up after you get WA, 3) work on finding the trick/optimal solution – 10 min 4) be greedy when you can 5) for sorting questions/query questions always think psums, two pointers, binary search. Also, think pq solution and sorting according to l and r index/intervals. Maybe getf() solution also. Finally, DP

**Problem: KQuery SPOJ (Offline Query w/Segment + BIT Tree) (#2 Vector Segment Tree)**

Origin: <http://www.spoj.com/problems/KQUERY/>

Learned: 1) switch to scanf and AC in one go 2) coding very fast (but could be faster) 3) got the general idea of segment tree problems 4) basic = straight implementation of seg tree 5) other = preprocess/ don’t preprocess and sort in some arbitrary order to get the valid ordering then, use segment tree to update values in increasing/decreasing order. Harder problems = figure out when to sort and when to add stuff.

**Problem: CF 182D (Offline Query + Sieve)**

Origin: <http://codeforces.com/contest/301/problem/D>

Learned: 1) Really hard but nice problem 2) have to make the observation that we use offline queries starting from l-1 with bit updates to get the answer… sweep from right to left3) find the update/transition graph with sieve of erasthonas 4) sweep and update to find answer. Bit indexing a little tricky.

**Problem: CF 216E (Offline Query + BIT)**

Origin: http://codeforces.com/contest/369/problem/E

Learned: 1) Got the transformation that I need to solve for pl+1, pr-1. 2) Solve by sorting in a specific order. 3) Use offline query again!!!! If hard to do online -> offline query 4) After realizing that I can use offline to query I switch to processing one by one in left 5) use bit tree to update! 6) 10^6 index bounds usually indicates that you have to sweep across the index somehow.

**Problem: CF 225C (Segment Tree + Tree)**

Origin: <http://codeforces.com/contest/383/problem/C>

Learned: 1) Bit update on intervals tricky!!! 2) +val, -val, tricky 3) solve by keeping two arrays of even and odd height level bits 4) After this, and dfs, it becomes simple lazy segment/bit tree problem 5) updating ranges of consecutive elements by adding intervals. Cool way http://codeforces.com/blog/entry/10476

**Problem: CF 200D (Segment Tree + Tree)**

Origin: <http://codeforces.com/contest/343/problem/D>  
  
Learned: 1) You cannot just update ranges with one segment tree in a naïve manner. 2) I got the idea but implementation was all wrong 3) Just use two segment trees one to keep track of the last subtracted point and one to keep track of the last updated range. 4) use two range maximum query, one normal, and one with range updates

**Problem: CF 169E (Segment Tree + BIT + Tree)**

Origin: <http://codeforces.com/contest/276/status/E>

Learned: 1) I got the right idea I just missed an implementation detail where you are double counting when you update both the root and the tree 2) tried different ways to alleviate this issue, cute trick of not updating the tree all the way helps 3) write classes for LCA, segment tree, sparse table etc. helps a lot in coding contest when you have to have multiple seg trees 4) got the problem idea where you update the tree and the root separately 5) another way of solving it involves breaking the tree into chains and updating each chain/bit correspondingly using bit range updates +l +x, + r+1 –x etc. really cool!

**Problem: CF 428B Greedy**

Origin: <http://codeforces.com/blog/entry/53815>

Learned: 1) Greedy solution very tricky 2) made observation that 7, 6, 5, 4 all should use 4 because only they can use all of 4 3) missed greedy observation that 3 should also use 4 because of the fact that 3 can only be split up into 2 2’s or a 4 4) this leads to only 2 cases left to consider (where I had 3 which made it impossible to do) greedy!!!! 2 and 1’s 5) 2’s should obviously use the two side 2’s and then 4 + 1 while 1’s have to be greater than 2 \* 4 + 1 + 2 left

**Contest Summary CF 428:** This contest was really bad for me… performed at 1400 rating again. 1) May need to consider taking more contests… we will see on Wednesday 2) Biggest problem was that I slept too late only got like 5 hours of sleep and yeah was so tired when I woke up… started at 7:35 barely could read problem statements 3) Vow: not to come back next summer and have to hang out ugh 4) Vow: sleep earlier and treat contests more seriously + 7 hours of sleep before 5) Good did not waste time after renap 6) When school starts = treat life + sin + happiness + contests + everything more seriously… this is last time this contest is not legit. 7) Only got one problem C right ☹ 8) at least 1600 next round treat seriously = sleep

Learned: 1) read the question carefully ugh. Number one was so easy, if got right rating = + 100 2) Problem B greedy was tricky! 3) clique + gcd/number theory/math relearn!

**Problem: CF 232E Bit/Segment + Tree**

Origin: http://codeforces.com/contest/396/status/C

Learned: 1) h[v] to update and h[u] to update if each level is different from the rest 2) two bit trees/segment trees to update ranges 3) the TLE will happen if you use lazy segment tree because of the fact that segment tree build takes a lot of time 4) can do it with normal segment tree and switching query and update 5) can also do it with bit tree. Range update and single element query 6) use bit tree and also use height to account for different level differences. 7) Gosh darn it wasted so much time debugging and it was just a simple mistake… need to learn how to debug better 8) think twice code once 9) should have gotten the height implementation and implemented with bit tree always 10) in coding think twice code once – 20 minutes per implementation… (5 plan 15 coding, timer) 11) debug: look through problem 3-> logic 5 -> code 2-> debug 10 12) debug pretty fast with template + binary search

**Problem: CF 52C Segment Tree Lazy 8/15**

Origin: http://codeforces.com/contest/52/problem/C

Learned: 1) RMQ implementation took way too long 2) tiny tiny bug in order of push which heavily skewed results… 3) coding speed = fast enough but very inaccurate = slow down. Think twice code once. 4) sscanf and gets() 5) circle -> array 6) rmq structure built and fully functional now. 7) better coding. And debugging = more focused

**Problem: CF 145E Segment Tree Lazy 8/15**

Origin: <http://codeforces.com/contest/145/problem/E>

Learned: 1) Looking at 5 solutions good pretty much does same thing, learned more about classes and using struct, and also building segment tree iteratively + N etc. 2) I got most of it except for key observation that you use an opposite… like opposite increasing value… keep that in mind that for these problems when you flip something it might all become opposite so you might have to store opposite. 3) try more examples to test 4) ask yourself what does each node store to solve this problem 5) got the cool math formulas by contradiction why each case must be so 6) timer good 7) implement 12 minutes :D 8) write helper functions to push, modify, and swap 9) build, query mx[1], if within range swap, if out of range then push, query, and then merge. 10) can use the nodes as classes to update to make life easier 11) push function messes up because you can’t swap the nodes and then swap the children too (should just swap children). 12) gave up lol 13) UGHHHHHHH no the previous comments were wrong. The real reason why it did not work is because of Array out of bounds fk. My range update does update the ranges, and I push at top so it should be fine, and more importantly I combine and merge correctly so it should be fine. What is wrong is index out of bounds. For large test case error look into index out of bounds etc. large problems. Basically code more carefully Chuck.

**Problem: CF 312E Segment Tree + Lazy + 26 counting sort 8/16**

Origin: <http://codeforces.com/contest/558/problem/E>

Learned: 1) Can do it using just vector and upper/lower bound 2) Another cool idea to use is counting sort… did not see that 3) whenever sorting is involved and only numbers/digits/letters use counting sort 4) counting sort + segment tree with range updates = super fast. Cool idea. 5) implementation tricky be careful 15 minutes careful implementation + 5 minutes check 6) implementation in 40 minutes pretty good little debugging 1 try AC good! Nice focus! Good focus! Implementation 40 minutes could be faster!

**Problem: CF 240F ACM ICPC Segment Tree + Lazy + 26 letters update 8/16**

Origin: <http://codeforces.com/contest/240/status/F>

Learned: 1) use arrays to implement if you don’t want to make a class 2) build can also be replaced with range updates 3) solve can also be replaced with queries 4) somehow code does not work on first try bad implementation 5) most other implementations use same idea of using 26 segment trees cool 6) implement solving palindromes and incrementing counter with lazy propagation. 7) Logical error that the odd value has to have size one but it could have size 7 or 9 so it does not have to be middle just where l and r are…. Must update at end.

**Problem: CF 91E (Segment Tree + Range Update + min Q)**

Origin: <http://codeforces.com/contest/121/problem/E>

Learned: 1) This problem… it’s okay not solving it… way too hard in implementation and idea. 2) Got the bare bones of how you can set your array as a positive integer and check if it goes down to 0 for all 30 lucky numbers 3) To do that you need to keep track of the minimum value, the count of the minimum value, how to subtract all the minimum values, ways to reset the number if the minimum value goes to 0, by keeping track of the left most minimum value 4) implementation with rmq does not do all of that in the solutions I see but they do keep track of min values and then update based off of that. They also use dfs to find the lucky numbers, then use dist to find the distance between the min values. Then in the update functions they always have a pushup and a pushdown function to merge the two values. Then finally they have ways to merge the results together that is super confusing lol.5) can also solve with bit tree since time constraint is big and lol super cheap way 6) it’s okay to ditch stupid hard problems like these 7) just remember dfs and idea that you can update according to distance and do rmq on it. (tricky)

**Problem: CF 250D (Segment Tree + Range Update)**

Origin: <http://codeforces.com/contest/438/problem/D>

Learned: 1) wow this problem has an easy implementation 2) you can just keep max and sums and mod the max and sums at their roots. solves for update 3 3) you can also sum with just range sums 4) update with normal updates 5) the key to this problem is realizing this is fast 6) fast implementation 7) Onlogn log ai) because you every time you mod you are reducing it by two or you are not reducing it at all = return; mod = use max’s; side cases = O(nlognlogai) + O(nlognlogai)

**Problem: Dividing Apples (Sorting + Math)**

Origin: <https://www.hackerearth.com/practice/algorithms/sorting/heap-sort/practice-problems/algorithm/divide-apples/editorial/>

Learned: 1) Could not solve this problem initially 2) Got the idea that you need to find the average and maybe involves some sorting 3) cool math proof that the first guy who pushes to the right then, the next guy must give X1 + k (you can give negative coins), then the next guy gives X1 + X2 + k, … etc… 4) this means that the product is |k| + |x1 + k| + |x2 + x1 + k| … 5) solve this problem by then finding the minimum value of k. which turns out to be the middle value of x1 , x2 , x3 , x4 etc. this is because you are trying to minimize the absolute value so at least n/2 + 1 and n/2 – 1.

**Problem: Horrible Queries SPOJ (BIT Tree range update)**

Origin: <http://www.spoj.com/problems/HORRIBLE/>

Learned: 1) Use long long for casting or just change to long long to make life easier 2) bit range update good wrote a good code template/struct 3) bit updates can do x++ to make life easier. Yeah we will do 1 – n

**Problem: Can you Answer these Queries I & III (SPOJ Segment Tree + Merging + Node Updates)**

Origin: <http://www.spoj.com/problems/GSS1/>

<http://www.spoj.com/problems/GSS3/>

Learned: 1) Should have realized that this was a segment tree problem so I need to consider what each node stores (may be different), how to merge each node (may be weird), how to push each node at the top, and how to query, also how to sort/process queries efficiently/if sorting is required. 2) keep your eyes on the prize. 3) with that observation I realize that best = max(l.best, r.best, l.suffix + r.prefix) 4) then I realize I need to keep prefix and suffix. Which are made by doing prefix = max(l.prefix, l.sum + r.prefix) sum = l.sum + r.sum. 5) then with that merge function in the node written, I can easily write the build and query function and eventually the update function and more functions on range queries!!! :D 6) update is the same idea you just do a normal update with merge function… :D easy peasy #logn

**Problem: Can you Answer these Queries V (SPOJ)**

Origin: <http://www.spoj.com/problems/GSS5/>

Learned: 1) solved problem too fast, got the right idea but missed the edge cases and implementation details 2) Good ideas that you have to keep track of the sums and prefix’s and best sums still 3) merge idea still good 4) die -> no feeling -> no thinking -> no mind chatter -> focus on problem -> clear -> 100% excited -> working hard 5) mind chatter gone = stop drawing circles in head/on paper start thinking about what this problem is asking me. 6) how can I use stuff I know, merge function etc. to solve this problem again. 7) solve this problem by doing and using the merge functions and queries that I already wrote. Then I realize that there are really only one portion of the graph to consider if y1 < x2. Which is then I must do suffix1 + sum\_middle + prefix 2 8) else we can portion the graph into the three queries x1 x2, x2 y1, y1, y2. -> this leads to the best answer being three queries of suffix’s and sums which I got.

**Problem: KGSS Maximum Sum SPOJ**

Origin: <http://www.spoj.com/problems/KGSS/>

Learned: 1) solved it really fast and then I realized I solved the wrong problem. 2) solved harder variant where x, y could extend beyond where you were at -> segment tree keep track of id, count, and max value pretty good. 3) this solution is so easy with just nodes! Merge function and build function 4) literally everything in segment tree remains the same except when you are pushing just make sure to add everything together 5) store the two max values sort and then solve.

**Problem: SPOJ POSTER (Segment Tree Range Update)**

Origin <http://www.spoj.com/problems/POSTERS/>

Learned: 1) got the idea! 2) poster problem solve with timer incremental + range update (not sum range update) 3) use compression cool 4) count the size of the poster at the end. 5) redo? 6) tricky in implementation 7) pseduocode better (never code without pseudocode) 8) for (auto &x: mp) &x is important to change the second value 9) other mistake = add more maxn – 1, ranges because that is what the node is from. If you can. 10) finally idk why but its range between compress[i].ff, compress[i].ss not (compress[i].ff, compress[i].ss – 1), if WA try both… but why 11) solved it by doing/returning -1, and i +1 for all values 12) why f, s not f, s – 1???

**Problem: SPOJ BRACKETS (Segment Tree + Brackets)**

Origin: <http://www.spoj.com/problems/BRCKTS/>

Learned: 1) Did not implement + missed idea 2) Was too careless be more careful, think clearly about the problem, do it like I’m actually solving it, write out clear pseudocode in fast 3) Missed idea that you cannot just add brackets -1 and + 1 bc of cases like )) (( 4) must do l = left.l + right.l – min(left.l, right.r) (the number of merged brackets) 5) update is just normal update with merge 6) solve by checking if the tree[1].l == tree[1].r == 0; 7) pretty easy problem 10 minute problem

**Problem: Light Switching! (Segment Tree + Range Update)**

Origin: <http://www.spoj.com/problems/LITE/>

Learned: 1) Solved it in 5 minutes 2) simple range update query which you can do to update the ranges 1 or 0 3) and query is very similar 4) use ^= operator for lazy updates. ^= 1 ^= 0

**Problem: KQuery Online (Segment + Vector)**

Origin: <http://www.spoj.com/problems/KQUERYO/>

Learned: 1) can do with bit tree 2) can also do with vectors with back inserter which is cool 3) redo?

**Problem: Multiple of 3 SPOJ Segment Tree**

Origin: <http://www.spoj.com/problems/MULTQ3/>

Learned: 1) Basically lazy update with 3 variables + mod 3 2) got the idea right not sure about specifics 3) basically range updates but you keep track of 3 variables that can alternate! 4) mod %=3 every value. And range update push

**Problem: IOPC 1207 (3D Segment Tree + PIE)**

Origin: <http://www.spoj.com/problems/IOPC1207/>

Learned: 1) Lazy update on all 3 regions 2) realized the realm was too big so must keep track of 3 ranges separately 3) update like light switching with ^= 1 operator as pushing 4) got the general gist 5) make less errors by reading problem + 1 min being creative in thinking/stop drawing circles = + 5 min writing pseudo-code + 3 minutes checking work + 1 minute = + 10 minutes. Per every hard problem – debugging – 5 minutes, CA + 20% :D 6) Missed the ^= 1 operator update 7) when solving problems like this easy illusion is that I understand without actually understanding = resolve + write pseudocode + check code line by line 45 min solve. 8) PIE can do two ways add middle circle + x1 ^ not (x2 U x3) etc. or you can do it with classical PIE. 9) cannot do it with the other PIE way

**Summary of 20 problems review:** 1) reduce the mental chatter, don’t force yourself, remind yourself to be excited and focus 100% on the present and solve the problem @ hand 2) I really liked doing review and going on tangents to learn new things! It was fun! 3) Good @ Segment Tree easy problems at least for now. 4) Solve in ACM? 5) Main takeaways in segment tree: lazy update = very powerful, tools: offline query, tree, sorting, bit tree, multiple segment trees storing multiple identities, separate node/pp classes to make merging and pushing easier, lazy propagation 6) key idea: long long, storing multiple variables in each node, calculating the merge function. Main question: is it segment tree? What do we store in each node? What is the trick? How do we calculate the merge function? The push function if it requires lazy propagation? 7) have fun!!! You are doing great – 4 hours today and still got most of work done. Glory to glory

8) Better schedule: Morning = contest 2 hour practice (Crucial) (or review for contest)

Then 2 hour implementation 30 min old problems 3 30 minute new problems. (30 minute sessions timed like a contest) (crucial deliberate practice in debugging + implementation)

Then 4 hours of 30 minutes or 45 minute sessions of solving problems = (6 problems) (most important, expose to new ideas)

Then 1 hour of review or 1 hour of reading CS stuff.

**Problem: USACO January 2014 Building A Ski Course (DP!) 8/21**

Origin: <http://www.usaco.org/index.php?page=viewproblem2&cpid=383>

Learned: 1) If you just do it! Through the fear. You can overcome fear. Fun Focus on the present 2) Have fun :D 3) Doing good! 4) Do it fast 5) Contest = round in shape Chuck. No time to be worried 6) Starting DP!

This problem is super hard. 1) Make tricky observation of starting in the back and checking 2) Make observation that you don’t have to use binary search 3) You can make another greedy observation that you only need to know the minimum of the maximum square per query = O(MN^2) solution 4) then you use dp to check /find the minimum and update the r, and c matrices 5) Finally you win by doing this in a while loop and establishing the break condition as when all the r’s == c’s. 6) dp and next (swap to save space) both represent the max square.

**Problem: Codeforces 597C (LIS + Segment Tree + DP)**  
Origin: <http://codeforces.com/contest/597/problem/C>

Learned: 1) fun and contest and focused!!! 2) solved problem pretty well 3) learned that it is simplification -> DP -> segment tree 4) dp / segment tree on the a[i] range 1 – n (can use compression) 5) k is small so we can use segment tree 6) implementation sucked lol, long long error then index out of bounds we need 11 segment trees so index has to be 12 (be careful less debugging saved 30 minutes) 7) contest more like = more focus 8) got the idea though! 9) Can also use simplifications in DP to save less work = 0 update with value 1 at beginning, 1 – n fine in segment tree, 1 – n in range, end = sum + = tree[k].range (0, n), keep everything 1 indexed! 10) Other people used BIT tree same idea cool 11) DP -> BIT/segment

**Problem: Codeforces 271E (Segment Tree + DP)**

Origin: <http://codeforces.com/contest/474/problem/E>

Learned: 1) DP -> Segment Tree 2) Coordinate compression idea, sweep through h+, and h- to find the maximum value. 3) tricky implementation problem -> 10^9 compress to O(n) 4) can use bit tree to do rmq 5) I am too lazy to do that to find rmq. 6) dp tricks = find the lowest height and highest height = upper and lower bounds 7) after that establish the previous and then solve! 8) C++ 14 switch good trick other bug = fixing node to x 9) laying out really helps me establish what code I need to write 20 min debugging.

**Contest Summary Codeforces Educational 27:** 1) Started late, was not super focused, ended up only solving 2 problems out of 5 potential solved 2) 3 changes 1) have fun, it’s supposed to be hard, focus on one problem, know that God is in charge so you do the best you can do and let God do the rest #sovereignty, work hard 3) not contest mode ☹ could have gotten 4 4) worked too fast, for number 2, obfuscated implementation, not enough thinking and then checking 5) bfs/dfs for numbers please learn how to do better, dfs global array 6) misread/could not understand number 5, p bad English tbh but could have guessed what they were asking so close (time crunch) 7) complicated D just because it was D, remember to use not just bsearch but also just stack if needed too.

**Problem: USACO Open 09 Job 8/21 (greedy, pq, union find)**

Origin: <file:///C:/Users/chuck/Downloads/job_soln.pdf>

Learned: 1) Map<int, int> union find structure learn better2) greedy + sorting solution, greedy/observation + work scheduling + pq solution 3) edge cases with adding 0 as a special case 4) such a clean solution with priority que and curTime

**Problem: Codeforces Educational 27B 8/22 (dfs, simulation)**

Origin: <http://codeforces.com/contest/845/problem/B>

Learned: 1) I solved it with back inserter + merging, a little bit too complicated 2) bounds are big enough so I thought of bfs 3) I can just simplify by iterating through with for loop 1 million -> 0 and solving for each case. Keep Bounds in mind 4) Or even better I can just use dfs and solve the problem, dfs with global array = clean. BFS too complicated since for each node I have to store an array. 8) dfs with string

**Problem: Codeforces Educational 27C (greedy, scheduling)**

Origin: <http://codeforces.com/contest/845/problem/C>

Learned: 1) Read the problem wrong 2) tried two wrong implementations 3) solved the problem of seeing if an interval is a valid tv situation though 4) could not solve this problem/variant because misunderstood question 5) got the idea now that it is just trying to see if you can find two continuous, non-intersecting tv sets. 6) solved it 7) ranges a[l] + 1, a[r] -1 :D

**Problem: Codeforces Educational 27D (implementation)**

Origin: <http://codeforces.com/contest/845/problem/D>

Learned: 1) Just because it is O(nlogn) does not mean you have to use sorting or binary search 2) implementation pretty easy 3) simulation problem with stack 4) tricky bug where the 70 speed limit comes after the 100 so you always have to check if the speed is greater than the speed limit. Fun problem.

Problem: Codeforces 52E (Segment Tree, Stack, DP)

Origin: <http://codeforces.com/contest/56/problem/E>

Learned: 1) DP on previous ranges -> Segment Tree/PQ 2) 1 hour implementation because of bugs 3) could have done it in 30 minutes -> in contest 30 minutes! 4) AC in two tries good! 1 hour problem. 5) can do with stack also 6) seg tree represents maximum value that this dominoe can go to. Rmq then the total dominoes is domino x – I; 5) also can do with union-find, get next

Problem: Codeforces 343D (Segment Tree, DP)

Origin: <http://codeforces.com/contest/629/problem/D>

Learned: 1) DP -> segment tree / pq 2) 30 min implementation 3) rmq can solve really easily just update and query 4) use a map to solve it too because map has lower\_bound function which can find all the lower values 5) update map with erase function = fun 6) level D dp = good 7) cout << setprecision(17) << double << endl;

**Problem: Codeforces 353E (Segment Tree, DP)**

Origin: <http://codeforces.com/contest/675/problem/E>

Learned: 1) Missed the DP need to practice DP 2) DP was store the max value of each sum in the end and process by n – 1 to 0 3) too tired 4) work! 5) debug fun! 6) dp[i] = use greedy observation that you choose the maximum value of a[i] always that can be reached (p obvious why) 7) from that DP you can use a rmq/sparse table (no updates) to find the max value of the m 8) then with PIE and careful indexing you get the dp[i] = dp[m] + n – 1 – i + a[i] – m 9) solve the equation 10) hard dp, missed + greedy, -> out greedy + dp = good 11) implementation too slow, just implement with care but no over thinking 12) debugging good just print + evaluate. 13) coding = print + evaluate be careful you may be wrong 14) use pair to store index, chuck please a[i] – m not other way around

**Problem: SPOJ Rent your Airplane (DP + BinarySearch)**

Origin: <http://www.spoj.com/problems/RENT/>

Learned: 1) Binary search + DP easy 10 minutes 2) no segment tree 3) solve with DP :D 4) fast 10 minutes.

**Contest Summary Codeforces 429:** I did really well in this contest place 80 + 199 points wow 2) 3 things that helped 3) before contest 10 minutes = chill 4) going fast in beginning good but slow + accurate = good. (could have gotten like 10th place if I was going for speed but no speed yet) 5) slow + steady = good 6) not overthinking though only check twice. (speed + accuracy = good) 7) slow and steady got the problems in good time, 45 min done with problem C 1 hour 40 done with problem D. continue on like this. 8) DFS practice good 9) debugging good! And final check good! 10) continue doing like this = speedy but accurate = read problem, proof, slow, 50 min problem C, 1 hour 30 problem D, check work before submit. If wrong dwai -> wheee win!

**Problem: CF 429D (dfs)**

Origin: <http://codeforces.com/contest/841/problem/D>

Learned: 1) Debug good, stuck -> reread question then find error in algorithm -> debug 2) got it in like 45 min good dfs 3) realized how -1 can be used positively whenever 4) subset of graph dp/recursion/induction 5) thus, we can always win with -1, and with even sum 6) mst idea too 8) dfs and then check if used / edge -> solve!

**Problem: CF 429C (math, inequalities)**

Origin: <http://codeforces.com/contest/841/problem/C>

Learned: 1) clearly saw the cheap way to do this but not the real way 2) learned how to do it with the hockey-stick identity 3) n c k = n-1 c k -1 + n-2 c k -1 .. k -1 c k -1 (draw out the table) 4) use it twice to get the f(n,k) = n+1/k + 1 5) do math quickly! 10 minutes -> 6) rearrangement inequality cool 7) don’t know proof but know that if we want to maximize sum we pair large with large, and minimize = small with large. We can do this yay! Then -> we want to pair large fraction with large number, large fraction = small number so solve!

**Problem: CF 429E (DP, Combinatorics)**

Origin: <http://codeforces.com/contest/841/problem/E>

Learned: 1) This is a really hard DP problem 2) look at codechef link 3) had make like 4 observations 1) that perfect squares that multiply each other should be grouped -> transforms into the counting number of ways to organize digits such that no distinct digit is next to each other -> processing dp group by group (with factorial since each permutation/digit is different) -> processing dp in a very cool manner -> processing dp with keeping track of which group you are on and how many bad cases there are 5) combinatorics for counting -> solve. 6) first time solving all 5 probles 7) hard dp like usaco gold! I am excited I can do it!

**Problem: CF 11D (DP, Bitmask)**

Origin: <http://codeforces.com/contest/11/problem/D>

Learned: 1) check for simple cycle really hard 2) \_\_builtin\_ffz(x) 3) tricky ideas involved of dp with three digits 4) don’t know why it has to be in order? 5) other than that pretty standard dp, + divide by 2;

**Problem: Hamiltonian Paths (Bitmask DP)**

Origin: <https://www.hackerearth.com/practice/algorithms/dynamic-programming/bit-masking/tutorial/>

Learned: 1) got the O(n2^n) implementation of checking if a cycle exists, use the bitmask twice 2) cool idea of bitmask in checking if cycle exists 3) double for loop to check last value to continue the dp. 4) tsp also reviewed. Easy 5) adding cycle is pretty easy. 6) bug in implementation for one ☹ 7) only trick/hard is finding the number of cycles.   
  
**Problem: Assignments (SPOJ, Bitmask DP)**

Origin: <http://www.spoj.com/problems/ASSIGN/>

Learned: 1) know idea super easy 2) 25 minutes implement and solve good 3) did bitwise dp 4) focused and like contest solve 5) bitwise dp with one dimension n in middle and transition to next stage easier. 6) stop caring so much, but Chuck focus and with this you can think more creatively = not over planning = perfect coding! 6) can solve variant of minimum cost of assignment.

**Problem: Largest Area Rectangle (Segment Tree, Divide and conquer, stack)**

Origin: <http://www.geeksforgeeks.org/largest-rectangle-under-histogram/>

Learned: 1) find the minimum trick and then solve. 2) divide and conquer + rmq = o(nlogn) 3) also with that observation there is an O(n) solution with trick by popping out larger values and keeping smaller values.

**Problem: Fredo’s Coins/Party (Hamiltonian Path)**

Origin: <https://www.hackerearth.com/practice/algorithms/graphs/hamiltonian-path/practice-problems/algorithm/fredos-crush-2/editorial/>

Learned: 1) Hamiltonian path review, can do both ways 2) easy 3) stop doing these easy problems already got it! 4) pretty fast review 1 hour done.

**Problem: Fetching (Codechef Hamiltonian Path)**

Origin: <https://www.codechef.com/problems/TOOLS>

Learned: 1) Can turn it into a Hamiltonian path problem 2) Extra constraint = even vertices must be visited after odd vertices, complete graph, can do before or after, initialize points in graph, also two tools = count of 01 must be <= 2 3) coding fast like contest

**Problem: Picture (USACO Training Pages Line Sweep review)**

Origin: <https://github.com/geeeqie/codes/tree/master/USACO/5.5.1%20Picture>

Learned: 1) Line sweep review good 2) Can use coordinate compression to make it faster 3) also a very slow way of O(n \* coordinates) is to just sweep for left and right planes of coordinates. :D and add counters 4) solve like it is a contest # no stress # let’s go #10 minutes focus 5) basically you have to sweep twice through the array tricky.

**Problem: Islands (USACO Platinum, DFS, BFS, Hamiltonian path)**

Origin: http://www.usaco.org/index.php?page=jan13problems

Learned: 1) Did it like a contest problem good 2) timer :D good 3) last Hamiltonian path problem pretty good 4) confused on implementation errors! 5) basically pretty simple, dfs -> number islands, bfs -> construct graph, run Hamiltonian path 6) one of them has bug idk where 7) easier implementation by shifting everything up 1 and reading in like grid[i][1] :D

**Problem: CF 391D (Bitmask DP)**

Origin: <http://codeforces.com/problemset/problem/757/D>

Learned: 1) contest like very fast and very good, do the best you can do right now 2) slow and steady 3) learned about approximation max number is 20 so perfect for dp 4) dp[75][1 << 20] is okay if memory limits \*= 2; 5) top-down approach in dp very good too use/practice memorization 6) bottom – up relearn pretty good too 7) tricky – implementation 8) got idea but too hard to implement #looked at someone elses 9) implementation involves, writing a good[values] of consecutive subsets 111 -> n = n << 1 | 1 or n & (n + 1) == 0 || \_\_builtin\_popcount(n + 1) == 1 && n != 0 to check for consecutive subsets 10) another termination case = i == n; (nice dp realization of mask and ith string). 11) dfs on all i points 12) final termination is when you have visited this point before. 13) dp with for loop that adds i 14) important keep in mind that a <= 20; to go to next points 15) really good bitmask dp problem

**Problem: CF 247D (Bitmask DP + Binary Search)**

Origin: <http://codeforces.com/contest/431/problem/D>

Learned: 1) get more sleep… way too tired 2) sleep @ 11 work like contest fun 3) this was such a cool problem, these bitmask dp problems are super hard 4) wow I am challenged by these bitmask dp problems lol too hard 5) uncertain don’t have to do it all. 6) prove that function is monotonically increasing because n +1 and 2 \* (n + 1) have same number of bits and 2 \* (n + 1) + 1 has > 0 bits 7) binary search on the answer after that 8) f(2 \* x) – f(x) where f(x) stands for number of bits from 1 – x which has k bits. 9) you can calculate the combinations first 10) recurrence = for all bits 63 -> 0 iterate downwards, if bit (x, i) == 0 then it has to be 0 or else the number is > than the x, if bit(x, i) == 1. Then you can set this current bit as 0 where you would then have i bits left c k choices for the k bits which are all smaller, or else then you recurrence and have k – 1 bits left to fill and keep on going down loop. Also, you have to add back edge case of adding where bitcount(x) == k then ans + 1; 16) built\_in\_popcountll(m) 17) comb[0][0] = 1… 18) idk why this is true but you have to add bitcount for the answer…

**Problem: CF 226D (Bitmask DP)**

Origin: <http://codeforces.com/contest/385/problem/D>

Learned: 1) bitmask dp standard of what you choose and what you don’t choose 2) also added a computational geometry factor but you have to account for when the angle goes suddenly out of range 3) can do with pure geometry + angles 4) cleaner ways to do it involves rotating angles 5) arc tan 6) law of cosines 7) missed simplification of -=l always 8) convert to radians 9) got in 45 minutes 10) plan two super fun!!! :D

**Problem: CF Educational 13D (Bitmask DP, greedy, observation)**

Origin: <http://codeforces.com/contest/678/problem/E>

Learned: 1) j != k and I & 1 << j and I & 1 << k dp[i][j] = dp[i ^ 1 << j][k] \* p[j][k] 2) greedy observation that the sith must be in the back clearly, or else he is just multiplying the chance he will die. 3) then you can start from the back and do dp 4) transition states = p[0][1] = 1; all p[0][i] = false. 5) iterate forward dp. Tricky! (not fully understood yet) 6) really good probability and maximize problem / greedy problem 7) work smart -> consistent -> hard 8) stop wasting time not enjoying programming #fun #rest -> release -> run!!! 5 problems a day! Good enough! Less is more! Easy does it, diffuse + focus: Always contest, always hard, always focused, always fun, always released, always rest, always -> working hard! 9) the basic idea is that dp[mask][j] stores the maximum probability that you can win with if j is the current sith on the table and there are mask siths left 10) reverse trick calculation 11) greedy that this sith has to be last -> 12) calculating reverse 13) each sith transition from dp[mask][j] = max(dp[mask & 1 << i][j] \* pr[j][i] + dp[mask ^ 1 << j][i] \*pr[i][j]) .. probability that j beats i \* remainder prob + prob I beats j \* remainder bitmask probability 14) can prove by induction/greedy this is correct.

**Problem: CF 302C (Bitmask DP)**

Origin: <http://codeforces.com/problemset/problem/543/C>

Learned: 1) not fully understood but it is okay 2) have fun 3) basically idea is you can dp on both parts of substring, losing control yay, two operations you can switch all the letters in this column or you can change one bit in this column. 4) do it on the lowest bit for some reason… 5) I think it has something to do with m and how it doesn’t change the result

**Problem: HackerEarth DigIT (Digit DP)**

Origin: https://www.hackerearth.com/problem/algorithm/digit/editorial/

Learned: 1) Focus on the moment, rule over problem, treat like a contest, clear your mind, stop being anxious, and just don’t do too much, clarity 2) Digit DP basic idea is very simple just start from the end and iterate over and over to the beginning 3) Learned DP approach, next topic/implementation = memorization approach 4) DP approach = base cases/initialization steps = start with N and start generating digits from N – N - strlength 5) then base cases = dp[0][0][0][0] = 1; trust that the base case will evaluate to 0 until necessary 6) final cases = dp[N][i][0][0] + dp[N][i][0][1]; 7) iterations = always going from position i to i+1 in digit dp. Make sure to account for both conditions in transitions properly. Keeping track of sum and mod is pretty easy. 7) iteration is string position, sum, mod (interchangeable sum and mod), and finally digit last. 8) dp[pos][sum][mod][under] 1 = under, 0 = not under, pos = what pos/prefix you are starting from 0 – i. string goes from k – N; (inclusive). sum = sum, mod = mod. Dp stores the number of prefixes that have this sum and mod and are under or not under and at this position inclusive. (please do not make read in errors…) 9) reduce space complexity by shifting between modulos (only need last position) so modulo 2; 10) whatever could not get final case. 11) long long verses int be very careful 12) final implementation problems = code faster 2 hours too long 13) finally get it though -> be careful in all aspects -> work hard -> learned digit dp -> initialization, greedy, solve from each bit on, and digit dp, and state space reduction.

**Problem: Light OJ 1068 (Digit DP + recursion + naïve)**

Origin: <http://lightoj.com/volume_showproblem.php?problem=1068>

Learned: 1) used digit dp + recursion to solve this 2) had to write the naïve solution too because of memory limit exceeded 20 million integer usually too much. 3) naïve solution implementation easy 4) non-naïve solution implementation was fun too 5) implementing 0 was good too 6) noted what each state stood for then simple recursion dfs = pass 7) also noted that n is the only one current @ pos everything else is n – 1 8) for loop + Boolean algebra = easy money (recursive best, then -> dp, then that weird dp that I saw)

**Problem: USACO Odometer (2014 US Open Silver Digit DP, DP)**

Origin: <http://www.usaco.org/index.php?page=viewproblem2&cpid=435>

Learned: 1) Hard problem! Finally done! 2) Good at both dp and recurrence formulas of digit DP! 3) a couple of tricks first solve(b) – solve(a-1). 3) also in solving in dp you have to remember you might double count 555666 (where both are counted) so write mutual function that can take in two targets!!! 4) Your dp states are dp[pos][under][k][is0] trick where k is 25. Can go up and down 5) base case is dp[0][0][25][1] is0 is true because you are at 0 right now. 6) write extra conditions -> don’t linger on them -> write the functions -> write the getter functions -> write base case and final cases -> write dp. 7) final states = f[25] and f[I < 25]…. 8) final cases / transition states! Dp relation = pretty easy to calculate run the whole dp thing and get it! (fun dp problem)

**Problem: Codeforces Educational Round 8D (Digit DP recursion)**

Origin: <http://codeforces.com/contest/628/problem/D>

Learned: 1) Easy digit dp problem 2) learned how with recursion keeping two states under and over = solve for a and b in one go. 4) can solve for all cases! 5) can also do it with dp pretty easy do usaco to learn dp way! (fun recursion problem)

**Problem: Palindromic Numbers Light OJ 1205 Berkeley Programming Contest (Digit DP Recursion Zero)**

Origin: <http://www.cnblogs.com/mengfanrong/p/4050252.html>

Learned: 1) Recursion based digit dp with palindromes (l + 1, r – 1) 2) solve of b, a – 1 3) go which iterates each step of the I I = len -> I = 1; 3) dp ( (last digit), l, r, under, valid palindrome so far) is your state 4) got most of it just forgot about the leading 0 extra condition which is tricky to deal with and needs an extra for loop 5) can store it all in 4D array or can use memorization to precompute only the under values dp[l][len] when under! Easy money 6) very similar to the approach seen on codeforces blogs when calculating with extra is0 7) recursion = ending case l < r 8) recurrence steps = for (I -> N) keep on iterating next value max, limit, a[i] or 9 9) special if condition to consider 0 and finally special conditions to keep the under and the limits. Etc. go! (fun recursion + is0 problem)

Problem: Codechef (recurrence)

Problem: Codechef Recurrence + New Type