CSE424 Optimization

HW6 REPORT

Alperen Erdem 161044027
Jan 02

Contents

PROBLEM DEFINITION

Given n cases that store k different products. Cases are represented as list which elements are representing count of products.

And there are m orders that store k different products like cases. We need find a subset of cases which satisfy sum of all orders with minimum excess product.

DECISION VARIABLES

$x = [x_1, x_2, x_3, x_4, x_5 \ldots . x_n]$ n=product number

DOMAIN

$k \leq n\,.\ \forall k \rightarrow x_k \rightarrow D_1 = [0,1]$

(For every decision variable's domain is [0,1])

CONSTRAINTS

1- $\forall p \rightarrow \sum_{i=0}^{n} C_i(p) * x_i \geq \sum_{k=0}^{m} O_k(p)$

(For every product, the chosen cases must have more or equal number of product than all orders.)

OBJECTIVE FUNCTION

$$\min \left(\sum_{i=0}^{n} C_i\big(sum(p)\big) * x_i - \sum_{k=0}^{m} O_k(sum(p))\right.$$

(minimize the difference between sum of products in all chosen cases and sum of products in all orders)

EXPLAIN OF TEST CODE

I'm reading given txt file for making list of test objects.

All test objects are creating one order class for all orders (summing them) while reading also creating list of cases.

Then I'm calling given methods with this test objects.

GENETIC ALGORITHM(SGA) WAY(HW6)

HOW TO RUN

If you run the main it will work for Genetic Algoritm, it will try to solve all test cases in sga way. Algorithm finished all tests in 5 min.

```
Genetic Way -------- Test Case 0: 6 7 8  with excess of 329
Genetic Way -------- Test Case 1: 1 2 6 7 10 12  with excess of 6004
Genetic Way -------- Test Case 2: 1 2 4 10  with excess of 1455
Genetic Way -------- Test Case 3: 2 3 5 6  with excess of 8036
Genetic Way -------- Test Case 4: 1 2 3 4  with excess of 8609
Genetic Way -------- Test Case 5: 1 3 4 5 8  with excess of 17325
Genetic Way -------- Test Case 6: 1 2 3 4 6  with excess of 40822
Genetic Way -------- Test Case 7: 2 4 5 8  with excess of 24245
Genetic Way -------- Test Case 8: 3 5 8 9  with excess of 70660
Genetic Way -------- Test Case 9: 4 7 8 11  with excess of 107546
Genetic Way -------- Test Case 10: 2 3  with excess of 12620
Genetic Way -------- Test Case 11: 8 10 11  with excess of 27393
Genetic Way -------- Test Case 12:  with excess of
Genetic Way -------- Test Case 13: 1 2 10  with excess of 207459
Genetic Way -------- Test Case 14: 3 6 8  with excess of 61713
Genetic Way -------- Test Case 15: 1 3 4 11  with excess of 51232
Genetic Way -------- Test Case 16: 2 4 5 6  with excess of 103940
Genetic Way -------- Test Case 17: 1 5 6 9 10  with excess of 155801
Genetic Way -------- Test Case 18: 1 3 8 10  with excess of 169840
Genetic Way -------- Test Case 19: 2 7 10  with excess of 141055
Genetic Way -------- Test Case 20: 1 2 3 9  with excess of 56519
Genetic Way -------- Test Case 21: 1 2 9  with excess of 106897
Genetic Way -------- Test Case 22: 4 5 9  with excess of 340396
Genetic Way -------- Test Case 23: 2 4 8  with excess of 141340
Genetic Way -------- Test Case 24: 4 5 6 10  with excess of 256140
Genetic Way -------- Test Case 25: 1 10  with excess of 3483
Genetic Way -------- Test Case 26: 4 6 8 14  with excess of 12576
Genetic Way -------- Test Case 27: 8 9  with excess of 24465
Genetic Way -------- Test Case 28: 5 13 20  with excess of 19599
Genetic Way -------- Test Case 29: 2 4 5 7 18 19  with excess of 6858
Genetic Way -------- Test Case 30: 2 16 18 21  with excess of 36097
Genetic Way -------- Test Case 31: 5 6 7 12 13 15  with excess of 42123
Genetic Way -------- Test Case 32: 3 4 8 13 14  with excess of 69693
Genetic Way -------- Test Case 33: 13 20  with excess of 47151
Genetic Way -------- Test Case 34: 2 8  with excess of 80003
Genetic Way -------- Test Case 35: 2 5 7 16  with excess of 58358
Genetic Way -------- Test Case 36: 3 14 17 20  with excess of 60719
Genetic Way -------- Test Case 37: 2 11 13 14  with excess of 54436
Genetic Way -------- Test Case 38: 1 9 10 12  with excess of 248967
Genetic Way -------- Test Case 39: 3 4 11 12 17  with excess of 154192
Genetic Way -------- Test Case 40: 1 3 10 17  with excess of 75741
Genetic Way -------- Test Case 41: 4 10  with excess of 148922
Genetic Way -------- Test Case 42: 3 11 13 21 23  with excess of 218335
Genetic Way -------- Test Case 43: 4 5 6 8 11  with excess of 117794
Genetic Way -------- Test Case 44: 10 15 16  with excess of 262758
Genetic Way -------- Test Case 45: 4 5 11  with excess of 103928
Genetic Way -------- Test Case 46: 3 5 8 18  with excess of 252203
Genetic Way -------- Test Case 47: 2 4 6 7 11 13  with excess of 347960
Genetic Way -------- Test Case 48: 1 6 9 10 11  with excess of 545335
Genetic Way -------- Test Case 49: 5 16  with excess of 478362
Genetic Way -------- Test Case 50: 5 14 30 32 34  with excess of 6095
Genetic Way -------- Test Case 51: 2 11 12 14 15 24 25  with excess of 2263
Genetic Way -------- Test Case 52: 11 28  with excess of 17595
Genetic Way -------- Test Case 53: 9 11 30  with excess of 32632
Genetic Way -------- Test Case 54: 11 20 23  with excess of 49446
Genetic Way -------- Test Case 55: 2 4 9 13 15 18 22 25  with excess of 42684
Genetic Way -------- Test Case 56: 8 24 26  with excess of 95473
Genetic Way -------- Test Case 57: 4 19 35 36  with excess of 162394
Genetic Way -------- Test Case 58: 6 12 18  with excess of 248065
Genetic Way -------- Test Case 59: 15 17 25 29 33  with excess of 187071
Genetic Way -------- Test Case 60: 17 21 34  with excess of 157500
Genetic Way -------- Test Case 61: 19 23  with excess of 99646
Genetic Way -------- Test Case 62: 3 5 8 18 45  with excess of 306734
Genetic Way -------- Test Case 63: 3 18 27  with excess of 424894
Genetic Way -------- Test Case 64: 6 22 23 32 34  with excess of 928502
Genetic Way -------- Test Case 65: 8 10 13 22 33 37  with excess of 209684
Genetic Way -------- Test Case 66: 6 7 9 12 13 20 40  with excess of 362725
Genetic Way -------- Test Case 67: 8 9 32  with excess of 375188
Genetic Way -------- Test Case 68: 20 29 33 36  with excess of 878591
Genetic Way -------- Test Case 69: 4 9 11 14 36  with excess of 1202679
Genetic Way -------- Test Case 70: 10 13 24 27  with excess of 160048
Genetic Way -------- Test Case 71: 17 18 22 23 37  with excess of 429460
Genetic Way -------- Test Case 72: 2 8 18 37  with excess of 242374
Genetic Way -------- Test Case 73: 5 9 24 27  with excess of 687864
Genetic Way -------- Test Case 74: 18 31  with excess of 1390845
Genetic Way -------- Test Case 75: 26 30  with excess of 12270
Genetic Way -------- Test Case 76: 14 25  with excess of 27017
Genetic Way -------- Test Case 77: 27 31 41 50  with excess of 72007
Genetic Way -------- Test Case 78: 27 50 57  with excess of 55222
Genetic Way -------- Test Case 79: 17 39  with excess of 61425
Genetic Way -------- Test Case 80: 20 23 56 65 86  with excess of 47456
Genetic Way -------- Test Case 81: 5 35 49  with excess of 122068
Genetic Way -------- Test Case 82: 4 10 20 50 56  with excess of 243799
Genetic Way -------- Test Case 83: 17 34 47  with excess of 255159
Genetic Way -------- Test Case 84: 11 17 57  with excess of 263422
Genetic Way -------- Test Case 85: 44 72 85 92  with excess of 105886
Genetic Way -------- Test Case 86: 2 27 46 52 55  with excess of 448018
Genetic Way -------- Test Case 87: 22 36 48 53 63 64 70  with excess of 580425
Genetic Way -------- Test Case 88: 23 30 65 85  with excess of 1197154
Genetic Way -------- Test Case 89: 2 13 33 37  with excess of 1257127
Genetic Way -------- Test Case 90: 22 23 36 46 71  with excess of 418318
Genetic Way -------- Test Case 91: 16 64 78 82  with excess of 205803
Genetic Way -------- Test Case 92: 29 47 52  with excess of 621672
Genetic Way -------- Test Case 93: 4 6 10 25 31 48 54 55  with excess of 1324125
Genetic Way -------- Test Case 94: 4 10 22 47 52 81  with excess of 1399821
Genetic Way -------- Test Case 95: 4 13 17 46  with excess of 304176
Genetic Way -------- Test Case 96: 23 40 41 42 48  with excess of 1156855
Genetic Way -------- Test Case 97: 6 34 44  with excess of 1339783
Genetic Way -------- Test Case 98: 17 24 66  with excess of 4858521
Genetic Way -------- Test Case 99: 24 28 35 91  with excess of 4015014
```

Results(0-100) on the picture.

If you want to see specific test, just change this at main

```
x=0
    while(x<100):
```

x and number in while condition.

For example, if you want to see case 37, x=37 number in condition is=38

COMPLEXITY

N=case number

M=order number

K=product number

Finding total number of products in all orders = $O(m * k)$

Number Of Iterations in First Loop (Generation Count)= 150

Built in sort function(python) = Its nlogn but n is 20(population count). So its $20\log 20 \approx 85$

Selection Function = $20 * 18$

Mating Function = $N * K$

First Loop = $O(150) = O(1)$

Sort = $O(85) = O(1)$

Selection= $O(n)$

Mating = $O(n * k)$

Complexity= $\boldsymbol{O(n * k)}$

Worst Case

Worst case is equal to average case.

Best Case

There is no best case, it will always iterate in average complexity.

ALGORITHM (Genetic Algorithm)

Step 1- Find 19 random initial solution and one from greedy algorithm

Step 2- Loop until 150 Times

Step 3- Sort solutions

Step 4- Make selection list according to solutions excess value(for example if its not valid solution, don't take it. If its best solution take it more than one time depend of frequency)

Step 5- Save last two best solution.

Step 6- Generate 18 new solutions with mating(crossing over and mutations) function. Merge it with last best two solution from previous generation.

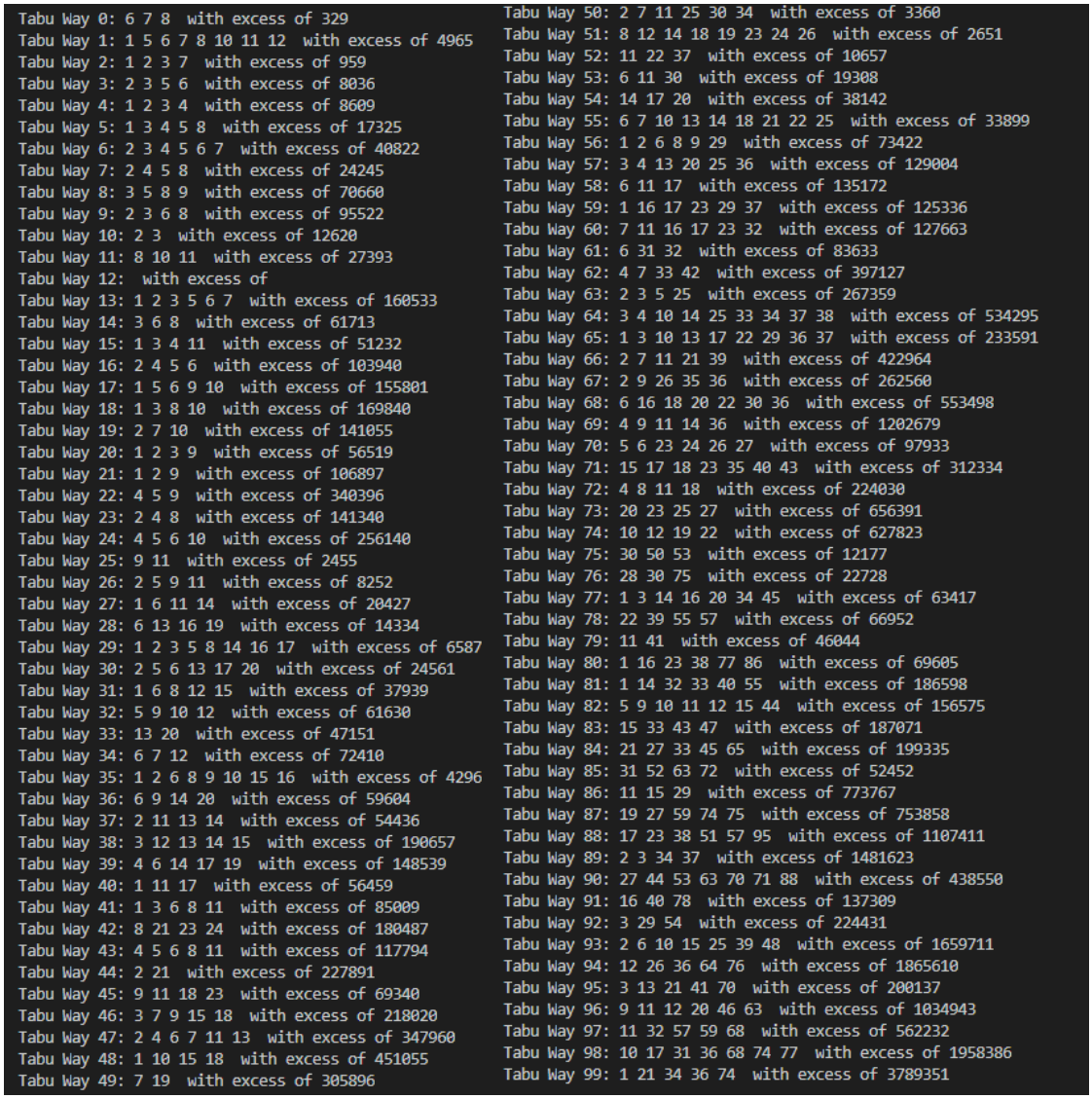
Step 7- Continue loop with 20 solutions.

Step 8- Return binary representation of solution and excess value (If not a single subset satisfies conditions, it will return -1, the initial value of index of satisfied subset)

TABU SEARCH WAY(HW5)

HOW TO RUN

If you run the main it will work for Tabu Search, it will try to solve all test cases in Tabu way. Algorithm finished all tests in 7 min 46 seconds.



```
Tabu Way 0: 6 7 8  with excess of 329
Tabu Way 1: 1 5 6 7 8 10 11 12  with excess of 4965
Tabu Way 2: 1 2 3 7  with excess of 959
Tabu Way 3: 2 3 5 6  with excess of 8036
Tabu Way 4: 1 2 3 4  with excess of 8609
Tabu Way 5: 1 3 4 5 8  with excess of 17325
Tabu Way 6: 2 3 4 5 6 7  with excess of 40822
Tabu Way 7: 2 4 5 8  with excess of 24245
Tabu Way 8: 3 5 8 9  with excess of 70660
Tabu Way 9: 2 3 6 8  with excess of 95522
Tabu Way 10: 2 3  with excess of 12620
Tabu Way 11: 8 10 11  with excess of 27393
Tabu Way 12:  with excess of
Tabu Way 13: 1 2 3 5 6 7  with excess of 160533
Tabu Way 14: 3 6 8  with excess of 61713
Tabu Way 15: 1 3 4 11  with excess of 51232
Tabu Way 16: 2 4 5 6  with excess of 103940
Tabu Way 17: 1 5 6 9 10  with excess of 155801
Tabu Way 18: 1 3 8 10  with excess of 169840
Tabu Way 19: 2 7 10  with excess of 141055
Tabu Way 20: 1 2 3 9  with excess of 56519
Tabu Way 21: 1 2 9  with excess of 106897
Tabu Way 22: 4 5 9  with excess of 340396
Tabu Way 23: 2 4 8  with excess of 141340
Tabu Way 24: 4 5 6 10  with excess of 256140
Tabu Way 25: 9 11  with excess of 2455
Tabu Way 26: 2 5 9 11  with excess of 8252
Tabu Way 27: 1 6 11 14  with excess of 20427
Tabu Way 28: 6 13 16 19  with excess of 14334
Tabu Way 29: 1 2 3 5 8 14 16 17  with excess of 6587
Tabu Way 30: 2 5 6 13 17 20  with excess of 24561
Tabu Way 31: 1 6 8 12 15  with excess of 37939
Tabu Way 32: 5 9 10 12  with excess of 61630
Tabu Way 33: 13 20  with excess of 47151
Tabu Way 34: 6 7 12  with excess of 72410
Tabu Way 35: 1 2 6 8 9 10 15 16  with excess of 4296
Tabu Way 36: 6 9 14 20  with excess of 59604
Tabu Way 37: 2 11 13 14  with excess of 54436
Tabu Way 38: 3 12 13 14 15  with excess of 190657
Tabu Way 39: 4 6 14 17 19  with excess of 148539
Tabu Way 40: 1 11 17  with excess of 56459
Tabu Way 41: 1 3 6 8 11  with excess of 85009
Tabu Way 42: 8 21 23 24  with excess of 180487
Tabu Way 43: 4 5 6 8 11  with excess of 117794
Tabu Way 44: 2 21  with excess of 227891
Tabu Way 45: 9 11 18 23  with excess of 69340
Tabu Way 46: 3 7 9 15 18  with excess of 218020
Tabu Way 47: 2 4 6 7 11 13  with excess of 347960
Tabu Way 48: 1 10 15 18  with excess of 451055
Tabu Way 49: 7 19  with excess of 305896
Tabu Way 50: 2 7 11 25 30 34  with excess of 3360
Tabu Way 51: 8 12 14 18 19 23 24 26  with excess of 2651
Tabu Way 52: 11 22 37  with excess of 10657
Tabu Way 53: 6 11 30  with excess of 19308
Tabu Way 54: 14 17 20  with excess of 38142
Tabu Way 55: 6 7 10 13 14 18 21 22 25  with excess of 33899
Tabu Way 56: 1 2 6 8 9 29  with excess of 73422
Tabu Way 57: 3 4 13 20 25 36  with excess of 129004
Tabu Way 58: 6 11 17  with excess of 135172
Tabu Way 59: 1 16 17 23 29 37  with excess of 125336
Tabu Way 60: 7 11 16 17 23 32  with excess of 127663
Tabu Way 61: 6 31 32  with excess of 83633
Tabu Way 62: 4 7 33 42  with excess of 397127
Tabu Way 63: 2 3 5 25  with excess of 267359
Tabu Way 64: 3 4 10 14 25 33 34 37 38  with excess of 534295
Tabu Way 65: 1 3 10 13 17 22 29 36 37  with excess of 233591
Tabu Way 66: 2 7 11 21 39  with excess of 422964
Tabu Way 67: 2 9 26 35 36  with excess of 262560
Tabu Way 68: 6 16 18 20 22 30 36  with excess of 553498
Tabu Way 69: 4 9 11 14 36  with excess of 1202679
Tabu Way 70: 5 6 23 24 26 27  with excess of 97933
Tabu Way 71: 15 17 18 23 35 40 43  with excess of 312334
Tabu Way 72: 4 8 11 18  with excess of 224030
Tabu Way 73: 20 23 25 27  with excess of 656391
Tabu Way 74: 10 12 19 22  with excess of 627823
Tabu Way 75: 30 50 53  with excess of 12177
Tabu Way 76: 28 30 75  with excess of 22728
Tabu Way 77: 1 3 14 16 20 34 45  with excess of 63417
Tabu Way 78: 22 39 55 57  with excess of 66952
Tabu Way 79: 11 41  with excess of 46044
Tabu Way 80: 1 16 23 38 77 86  with excess of 69605
Tabu Way 81: 1 14 32 33 40 55  with excess of 186598
Tabu Way 82: 5 9 10 11 12 15 44  with excess of 156575
Tabu Way 83: 15 33 43 47  with excess of 187071
Tabu Way 84: 21 27 33 45 65  with excess of 199335
Tabu Way 85: 31 52 63 72  with excess of 52452
Tabu Way 86: 11 15 29  with excess of 773767
Tabu Way 87: 19 27 59 74 75  with excess of 753858
Tabu Way 88: 17 23 38 51 57 95  with excess of 1107411
Tabu Way 89: 2 3 34 37  with excess of 1481623
Tabu Way 90: 27 44 53 63 70 71 88  with excess of 438550
Tabu Way 91: 16 40 78  with excess of 137309
Tabu Way 92: 3 29 54  with excess of 224431
Tabu Way 93: 2 6 10 15 25 39 48  with excess of 1659711
Tabu Way 94: 12 26 36 64 76  with excess of 1865610
Tabu Way 95: 3 13 21 41 70  with excess of 200137
Tabu Way 96: 9 11 12 20 46 63  with excess of 1034943
Tabu Way 97: 11 32 57 59 68  with excess of 562232
Tabu Way 98: 10 17 31 36 68 74 77  with excess of 1958386
Tabu Way 99: 1 21 34 36 74  with excess of 3789351
```

Results(0-100) on the picture.

If you want to see specific test, just change this at main

```
x=0
    while(x<100):
```

x and number in while condition.

For example, if you want to see case 37, x=37 number in condition is=38

COMPLEXITY

N=case number

M=order number

K=product number

Finding total number of products in all orders = $O(m * k)$

Number Of Iterations in First Loop= 150

Number Of Iterations in Second Loop (Finding Neighborhood(one bit change)) = N

Number Of Iterations in Second Loop (Finding Neighborhood(one bit change)) = N^2

First Loop = $O(150) = O(1)$

Second Loop = $O(N)$

Check neighbor and take excess from case list= $O(n)$

Check all neighbors with two-bit difference = $O(n^2 * k)$

Check excess difference between order and neighbor = $O(k)$

Complexity= $\boldsymbol{O(n^2 * (n + k))}$

Worst Case

Worst case is equal to average case.

Best Case

If there is 30 bad move or not better solution , inner loop have a break.

Best Case Complexity= $\boldsymbol{O(n * (k))}$

ALGORITHM(Tabu Search)

Step 1- Find initial solution from greedy algorithm

Step 2- Loop until 150 Times

> **Step 3- Find Neighborhood (one-bit change possible, two-bit change if there is no solution in one-bit change neighborhood)**
>
> **Step 4- Check every neighbor in neighborhood**
>
> > **Step 5- Find local best solution in local neighborhood which is not in tabu list**
> >
> > **Step 6- Add local best to tabu list**
> >
> > **Step 7- Compare local best and best solution, if its better than best change best.**

Step 8- Return binary representation of solution and excess value (If not a single subset satisfy conditions, it will return -1, the initial value of index of satisfied subset)

SIMULATED ANNEALING WAY(HW4)

HOW TO RUN

If you run the main it will work for Simulated Annealing, it will try to solve all test cases in SA way. Algorithm finished all tests in 2 min 48 seconds.

```
SA Way --Test Case 0: 1 4  with excess of 878
SA Way --Test Case 1: 1 5 6 7 8 10 11 12  with excess of 4965
SA Way --Test Case 2: 4 6 11  with excess of 1364
SA Way --Test Case 3: 3 4 7 8  with excess of 12001
SA Way --Test Case 4: 2 3 4 8  with excess of 13737
SA Way --Test Case 5: 1 3 4 5 8  with excess of 17325
SA Way --Test Case 6: 2 3 4 5 6 7  with excess of 40822
SA Way --Test Case 7: 5 7 8  with excess of 32348
SA Way --Test Case 8: 4 5 7 9  with excess of 88292
SA Way --Test Case 9: 1 4 6 7  with excess of 140518
SA Way --Test Case 10: 2 12  with excess of 15481
SA Way --Test Case 11: 3 9 11  with excess of 34756
SA Way --Test Case 12:  with excess of
SA Way --Test Case 13: 2 3 4 5 6 8  with excess of 171098
SA Way --Test Case 14: 3 6 8  with excess of 61713
SA Way --Test Case 15: 2 7 9  with excess of 65109
SA Way --Test Case 16: 3 5 6  with excess of 153648
SA Way --Test Case 17: 1 5 6 9 10  with excess of 155801
SA Way --Test Case 18: 6 8 9 11  with excess of 213035
SA Way --Test Case 19: 3 10  with excess of 162151
SA Way --Test Case 20: 1 2 3 9  with excess of 56519
SA Way --Test Case 21: 1 2 9  with excess of 106897
SA Way --Test Case 22: 1 5 7 9  with excess of 342912
SA Way --Test Case 23: 1 4 8  with excess of 183793
SA Way --Test Case 24: 4 7 8  with excess of 440754
SA Way --Test Case 25: 4 10  with excess of 3688
SA Way --Test Case 26: 1 7 8 12  with excess of 8765
SA Way --Test Case 27: 8 9  with excess of 24465
SA Way --Test Case 28: 6 7 9 17 19  with excess of 36520
SA Way --Test Case 29: 3 5 8 11 15 17 18  with excess of 9058
SA Way --Test Case 30: 2 5 6 13 17 20  with excess of 24561
SA Way --Test Case 31: 5 6 11 12 13  with excess of 45807
SA Way --Test Case 32: 5 12 14 15  with excess of 63989
SA Way --Test Case 33: 5 11 20  with excess of 75868
SA Way --Test Case 34: 4 6 8  with excess of 84855
SA Way --Test Case 35: 3 7 8 17  with excess of 83250
SA Way --Test Case 36: 2 14 17  with excess of 67462
SA Way --Test Case 37: 7 11 13  with excess of 163972
SA Way --Test Case 38: 1 3 7 8 9 11  with excess of 230789
SA Way --Test Case 39: 2 6 11 19 20  with excess of 182815
SA Way --Test Case 40: 3 10 14  with excess of 86067
SA Way --Test Case 41: 13 22  with excess of 146535
SA Way --Test Case 42: 5 10 21 23  with excess of 265507
SA Way --Test Case 43: 4 5 6 8 11  with excess of 117794
SA Way --Test Case 44: 12 21  with excess of 265296
SA Way --Test Case 45: 5 10 11 18  with excess of 85772
SA Way --Test Case 46: 3 7 9 11 18  with excess of 293371
SA Way --Test Case 47: 4 6 8 11  with excess of 454904
SA Way --Test Case 48: 1 5 9 17  with excess of 640260
SA Way --Test Case 49: 1 20  with excess of 631046
SA Way - Test Case 50: 9 11 21 28 29 34  with excess of 8843
SA Way - Test Case 51: 5 6 8 9 10 12 14 15 22  with excess of 4585
SA Way - Test Case 52: 4 14 18 21  with excess of 14145
SA Way - Test Case 53: 12 17 27  with excess of 36796
SA Way - Test Case 54: 16 19 23  with excess of 60951
SA Way - Test Case 55: 2 4 9 15 20 25  with excess of 41447
SA Way - Test Case 56: 2 9 20 21 27  with excess of 105250
SA Way - Test Case 57: 6 10 18 21 25 27 37  with excess of 214363
SA Way - Test Case 58: 15 21  with excess of 196741
SA Way - Test Case 59: 1 3 4 23  with excess of 325844
SA Way - Test Case 60: 2 4 7 11 16 19 23  with excess of 119413
SA Way - Test Case 61: 4 12  with excess of 92048
SA Way - Test Case 62: 12 16 17 39 40 41 44  with excess of 418003
SA Way - Test Case 63: 20 24  with excess of 530701
SA Way - Test Case 64: 3 7 14 17 31 34  with excess of 729627
SA Way - Test Case 65: 1 3 10 20 22 26 27 29  with excess of 199171
SA Way - Test Case 66: 17 25 34  with excess of 324418
SA Way - Test Case 67: 2 29 35 39  with excess of 265135
SA Way - Test Case 68: 1 2 6 27 32 36  with excess of 1063966
SA Way - Test Case 69: 2 11 15 20 23  with excess of 1074401
SA Way - Test Case 70: 2 15  with excess of 200193
SA Way - Test Case 71: 3 15 17 23 35 40 43  with excess of 324449
SA Way - Test Case 72: 18 22 23 25  with excess of 353924
SA Way - Test Case 73: 4 5 6 9  with excess of 840399
SA Way - Test Case 74: 2 10 14 19 36  with excess of 994214
SA Way - Test Case 75: 43 64 65  with excess of 7768
SA Way - Test Case 76: 45 63  with excess of 34521
SA Way - Test Case 77: 6 13 14 20 45 51  with excess of 71578
SA Way - Test Case 78: 25 46 48  with excess of 97804
SA Way - Test Case 79: 44 49  with excess of 68703
SA Way - Test Case 80: 13 30 74 75 76  with excess of 88247
SA Way - Test Case 81: 3 9 41 49  with excess of 141221
SA Way - Test Case 82: 9 15 28 47 53 54  with excess of 247556
SA Way - Test Case 83: 15 19 28 32  with excess of 329734
SA Way - Test Case 84: 16 17 21 32  with excess of 281787
SA Way - Test Case 85: 9 50  with excess of 132932
SA Way - Test Case 86: 23 29 41 42 49 55 61  with excess of 364297
SA Way - Test Case 87: 18 19 42 63 76  with excess of 654614
SA Way - Test Case 88: 1 6 12 15 44 53  with excess of 782504
SA Way - Test Case 89: 2 4 8 35 40  with excess of 1249281
SA Way - Test Case 90: 31 32 33 44 63 73  with excess of 445980
SA Way - Test Case 91: 16 40 78  with excess of 137309
SA Way - Test Case 92: 4 17 54 56  with excess of 331015
SA Way - Test Case 93: 2 6 18 31 43 54  with excess of 1503645
SA Way - Test Case 94: 4 37 38 52 64 65  with excess of 1262228
SA Way - Test Case 95: 21 72 76  with excess of 401894
SA Way - Test Case 96: 9 23 37 61 67  with excess of 640978
SA Way - Test Case 97: 4 11 30 36  with excess of 887657
SA Way - Test Case 98: 1 10 36 60 68 69  with excess of 1060736
SA Way - Test Case 99: 19 42 58 66  with excess of 4509154
```

Results(0-100) on the picture.

If you want to see specific test, just change this at main

```
x=0
    while(x<100):
```

x and number in while condition.

For example, if you want to see case 37, x=37 number in condition is=38

COMPLEXITY

N=case number

M=order number

K=product number

Temperature= Excess product from greedy0.0001/Temperature ^1/5*M

$$\text{Cooling Ratio} = \left(\frac{0.0001}{\text{Temperature}}\right)^{\left(\frac{1}{5*M}\right)}$$

Number Of Iterations in First Loop(Cooling)= 5*M

Number Of Iterations in Second Loop(Checking Neighbors Randomly) = M

Finding total number of products in all orders = $O(m * k)$

First Loop = $O(5n) = O(n)$

Second Loop = $O(n)$

Check neighbor and take excess from case list= $O(n)$

Check excess difference order and neighbor = $O(k)$

Complexity= $\boldsymbol{O(n^2 * (n + k))}$

Worst Case

Worst case is equal to average case.

Best Case

If there is 6 bad move in a row with help of temperature, inner loop have a break.

Best Case Complexity= $\boldsymbol{O(n^2 * (k))}$

ALGORITHM(Simulated Annealing)

Step 1- Find initial solution from greedy algorithm

Step 2- Loop until 5* Case Number

Step 3- Loop until Case Number

Step 4- Check randomly one neighbor

Step 5- If there is better solution, change it

Step 6- If there is not better solution, check temperature.

Step 7- If temperature fits the solution, accept it and change solution. Else don't change

Step 8- Return binary representation of solution and excess value (If not a single subset satisfy conditions, it will return -1, the initial value of index of satisfied subset)

VNS WAY(HW3)

HOW TO RUN

If you run the main it will work for VNS, it will try to solve all test cases in VNS way. Algorithm finished up to test 60 in five second. And it finished all tests 5min 40 seconds.

```
VNS Way ------------ Test Case 0: 1 4  with excess of 878
VNS Way ------------ Test Case 1: 1 2 3 6 7 8 11 12  with excess of 7912
VNS Way ------------ Test Case 2: 4 7 11  with excess of 609
VNS Way ------------ Test Case 3: 2 3 5 6  with excess of 8036
VNS Way ------------ Test Case 4: 1 2 3 4  with excess of 8609
VNS Way ------------ Test Case 5: 3 4 7 8  with excess of 20681
VNS Way ------------ Test Case 6: 2 3 4 5 6 7  with excess of 40822
VNS Way ------------ Test Case 7: 3 6 7  with excess of 48500
VNS Way ------------ Test Case 8: 2 6 8 9  with excess of 89969
VNS Way ------------ Test Case 9: 4 7 8 11  with excess of 107546
VNS Way ------------ Test Case 10: 2 3  with excess of 12620
VNS Way ------------ Test Case 11: 6 11  with excess of 35776
VNS Way ------------ Test Case 12:  with excess of
VNS Way ------------ Test Case 13: 1 2 10  with excess of 207459
VNS Way ------------ Test Case 14: 1 4  with excess of 104211
VNS Way ------------ Test Case 15: 4 6 9  with excess of 66809
VNS Way ------------ Test Case 16: 3 5 6  with excess of 153648
VNS Way ------------ Test Case 17: 1 2 4 10  with excess of 209849
VNS Way ------------ Test Case 18: 3 4 6  with excess of 183723
VNS Way ------------ Test Case 19: 3 10  with excess of 162151
VNS Way ------------ Test Case 20: 2 4 8  with excess of 67315
VNS Way ------------ Test Case 21: 6 9 10  with excess of 129010
VNS Way ------------ Test Case 22: 1 2 9  with excess of 401028
VNS Way ------------ Test Case 23: 2 6  with excess of 288174
VNS Way ------------ Test Case 24: 1 4 9  with excess of 477435
VNS Way ------------ Test Case 25: 1 10  with excess of 3483
VNS Way ------------ Test Case 26: 1 6 11 13  with excess of 9210
VNS Way ------------ Test Case 27: 8 9  with excess of 24465
VNS Way ------------ Test Case 28: 5 13 20  with excess of 19599
VNS Way ------------ Test Case 29: 3 5 9 14 15 18  with excess of 5227
VNS Way ------------ Test Case 30: 2 16 18 21  with excess of 36097
VNS Way ------------ Test Case 31: 1 2 10  with excess of 60831
VNS Way ------------ Test Case 32: 6 10 11  with excess of 81122
VNS Way ------------ Test Case 33: 13 20  with excess of 47151
VNS Way ------------ Test Case 34: 4 6 16  with excess of 83838
VNS Way ------------ Test Case 35: 7 11 17  with excess of 82252
VNS Way ------------ Test Case 36: 6 10 18  with excess of 86589
VNS Way ------------ Test Case 37: 1 3  with excess of 133238
VNS Way ------------ Test Case 38: 2 12 13 14  with excess of 286061
VNS Way ------------ Test Case 39: 2 10 23  with excess of 222637
VNS Way ------------ Test Case 40: 3 10 14  with excess of 86067
VNS Way ------------ Test Case 41: 4 10  with excess of 148922
VNS Way ------------ Test Case 42: 8 12 21  with excess of 270801
VNS Way ------------ Test Case 43: 5 10 13  with excess of 231261
VNS Way ------------ Test Case 44: 7 10  with excess of 394345
VNS Way ------------ Test Case 45: 18 20  with excess of 171432
VNS Way ------------ Test Case 46: 4 6 9  with excess of 468906
VNS Way ------------ Test Case 47: 1 9 12 13  with excess of 516400
VNS Way ------------ Test Case 48: 5 11 15  with excess of 657991
VNS Way ------------ Test Case 49: 7 19  with excess of 305896
VNS Way ------------ Test Case 50: 3 5 11 32 34  with excess of 5073
VNS Way ------------ Test Case 51: 1 3 6 14 18 23 24  with excess of 3271
VNS Way ------------ Test Case 52: 11 21  with excess of 12748
VNS Way ------------ Test Case 53: 6 11 30  with excess of 19308
VNS Way ------------ Test Case 54: 18 19 26  with excess of 52431
VNS Way ------------ Test Case 55: 5 12 16 19 23  with excess of 52491
VNS Way ------------ Test Case 56: 8 15 26  with excess of 93589
VNS Way ------------ Test Case 57: 4 19 35 36  with excess of 162394
VNS Way ------------ Test Case 58: 2 24  with excess of 248711
VNS Way ------------ Test Case 59: 22 28  with excess of 357487
VNS Way ------------ Test Case 60: 9 18 21  with excess of 142764
VNS Way ------------ Test Case 61: 19 23  with excess of 99646
VNS Way ------------ Test Case 62: 2 3 42  with excess of 353944
VNS Way ------------ Test Case 63: 5 15 26  with excess of 449787
VNS Way ------------ Test Case 64: 4 26 31 34  with excess of 832623
VNS Way ------------ Test Case 65: 2 13 23 36  with excess of 259339
VNS Way ------------ Test Case 66: 9 25 36  with excess of 406514
VNS Way ------------ Test Case 67: 21 32  with excess of 643987
VNS Way ------------ Test Case 68: 25 30 34  with excess of 1174193
VNS Way ------------ Test Case 69: 14 36 37  with excess of 1430749
VNS Way ------------ Test Case 70: 17 22  with excess of 214811
VNS Way ------------ Test Case 71: 33 35 42  with excess of 758806
VNS Way ------------ Test Case 72: 11 35  with excess of 726873
VNS Way ------------ Test Case 73: 12 19 26  with excess of 1478103
VNS Way ------------ Test Case 74: 17 27  with excess of 1848158
VNS Way ------------ Test Case 75: 1 30  with excess of 8880
VNS Way ------------ Test Case 76: 30 84  with excess of 23399
VNS Way ------------ Test Case 77: 27 31 41 50  with excess of 72007
VNS Way ------------ Test Case 78: 42 55 60  with excess of 49057
VNS Way ------------ Test Case 79: 11 41  with excess of 46044
VNS Way ------------ Test Case 80: 4 38 60 73  with excess of 60246
VNS Way ------------ Test Case 81: 5 33 40  with excess of 187197
VNS Way ------------ Test Case 82: 4 30 41 51  with excess of 244942
VNS Way ------------ Test Case 83: 6 15 21  with excess of 387611
VNS Way ------------ Test Case 84: 13 60 68  with excess of 322677
VNS Way ------------ Test Case 85: 25 66  with excess of 134170
VNS Way ------------ Test Case 86: 11 15 29  with excess of 773767
VNS Way ------------ Test Case 87: 18 22 75  with excess of 607061
VNS Way ------------ Test Case 88: 23 42 51  with excess of 1192323
VNS Way ------------ Test Case 89: 19 37 40  with excess of 1352475
VNS Way ------------ Test Case 90: 4 11 77  with excess of 686423
VNS Way ------------ Test Case 91: 27 32  with excess of 475742
VNS Way ------------ Test Case 92: 34 54  with excess of 847268
VNS Way ------------ Test Case 93: 24 37 39  with excess of 2359775
VNS Way ------------ Test Case 94: 13 76 90  with excess of 2221527
VNS Way ------------ Test Case 95: 9 20  with excess of 335415
VNS Way ------------ Test Case 96: 10 63 65  with excess of 1002237
VNS Way ------------ Test Case 97: 6 34 59  with excess of 1008434
VNS Way ------------ Test Case 98: 44 48 74  with excess of 1862757
VNS Way ------------ Test Case 99: 7 36 94  with excess of 4303681
```

Results(0-100) on the picture.

If you want to see specific test, just change this at main

```
x=0
    while(x<100):
```

x and number in while condition.

For example, if you want to see case 37, x=37 number in condition is=38

COMPLEXITY

It's very hard to say exact complexity of VNS. Because it can stop at the first solution if there are not better solutions, or it can find the furthest solution in $\mathbf{2^n}$ subset. So I'll calculate complexities for best and worst cases.

N=case number

M=order number

K=product number

Worst Case

Finding total number of products in all orders = $O(m * k)$

Check all neighbors of initial subset to furthest one = $O(n)$

Check all neighbors with one-bit difference = $O(n * k)$

Check all neighbors with two-bit difference = $O(n^2 * k)$

Worst Case= $\boldsymbol{O(m * k + n^3 * k)}$

Best Case

Finding total number of products in all orders = $O(m * k)$

Find best neighbor in the first try = $O(1)$

Check all neighbors with one-bit difference = $O(n * k)$

Check all neighbors with two-bit difference = $O(n^2 * k)$

Best Case = $\boldsymbol{O(m * k + n^2 * k)}$

ALGORITHM(VNS)

Step 1- Find initial solution from greedy algorithm

Step 2- Loop until there are no better solutions in neighbors

> **Step 3- Check all neighbors with one bit difference, if there are better solution change it**
>
> **Step 4- If there are not better solution with one bit change, check all neighbors with two bit difference**
>
> **Step 5- If there is no change in step 3 and step 4, stop loop.**

Step 5- Return binary representation of solution and excess value (If not a single subset satisfy conditions, it will return -1, the initial value of index of satisfied subset)

GREEDY WAY(HW2)

HOW TO RUN

If you run the main with uncommenting greedy, it will try to solve all test cases in greedy way. Algorithm finished in one second.

```
Greedy Way - Test Case 0: 0 3 with excess -878
Greedy Way - Test Case 1: 0 1 2 5 6 7 10 11 with excess -7912
Greedy Way - Test Case 2: 4 5 6 10 with excess -9428
Greedy Way - Test Case 3: 2 3 5 6 with excess -17736
Greedy Way - Test Case 4: 1 4 5 6 with excess -18262
Greedy Way - Test Case 5: 3 6 7 8 with excess -30256
Greedy Way - Test Case 6: 1 2 3 4 5 6 with excess -40822
Greedy Way - Test Case 7: 2 5 6 with excess -48500
Greedy Way - Test Case 8: 0 1 2 5 8 with excess -178460
Greedy Way - Test Case 9: 2 5 6 7 10 with excess -168279
Greedy Way - Test Case 10: 4 10 with excess -29743
Greedy Way - Test Case 11: 1 5 with excess -73973
Greedy Way - Test Case 12:
Greedy Way - Test Case 13: 0 1 7 9 with excess -341042
Greedy Way - Test Case 14: 3 4 8 with excess -252805
Greedy Way - Test Case 15: 5 7 8 with excess -99915
Greedy Way - Test Case 16: 2 4 6 with excess -179684
Greedy Way - Test Case 17: 1 3 7 10 with excess -330803
Greedy Way - Test Case 18: 1 6 11 with excess -393300
Greedy Way - Test Case 19: 3 8 with excess -289996
Greedy Way - Test Case 20: 5 6 8 with excess -118898
Greedy Way - Test Case 21: 3 4 9 with excess -248665
Greedy Way - Test Case 22: 0 1 2 with excess -503933
Greedy Way - Test Case 23: 0 4 5 with excess -460726
Greedy Way - Test Case 24: 0 2 6 with excess -689935
Greedy Way - Test Case 25: 0 13 with excess -4878
Greedy Way - Test Case 26: 2 4 5 7 with excess -31155
Greedy Way - Test Case 27: 7 8 with excess -24465
Greedy Way - Test Case 28: 8 9 12 19 with excess -73041
Greedy Way - Test Case 29: 3 8 12 13 14 17 with excess -20497
Greedy Way - Test Case 30: 1 15 17 20 with excess -36097
Greedy Way - Test Case 31: 1 9 15 with excess -82706
Greedy Way - Test Case 32: 3 9 10 14 with excess -157656
Greedy Way - Test Case 33: 2 6 16 with excess -304660
Greedy Way - Test Case 34: 0 12 14 with excess -213165
Greedy Way - Test Case 35: 10 11 12 13 with excess -159805
Greedy Way - Test Case 36: 4 15 21 with excess -207452
Greedy Way - Test Case 37: 3 4 with excess -343604
Greedy Way - Test Case 38: 1 5 6 13 with excess -418029
Greedy Way - Test Case 39: 0 7 9 22 with excess -537531
Greedy Way - Test Case 40: 8 11 17 with excess -217876
Greedy Way - Test Case 41: 11 14 17 with excess -498341
Greedy Way - Test Case 42: 11 14 21 with excess -565087
Greedy Way - Test Case 43: 0 6 9 with excess -699813
Greedy Way - Test Case 44: 2 19 with excess -1169353
Greedy Way - Test Case 45: 14 19 with excess -278641
Greedy Way - Test Case 46: 3 5 11 13 with excess -980212
Greedy Way - Test Case 47: 0 4 10 11 with excess -838889
Greedy Way - Test Case 48: 1 4 14 15 with excess -1351529
Greedy Way - Test Case 49: 8 21 with excess -1474416
Greedy Way - Test Case 50: 4 9 18 31 33 with excess -16100
Greedy Way - Test Case 51: 0 2 11 13 17 23 25 with excess -19166
Greedy Way - Test Case 52: 10 18 27 with excess -58359
Greedy Way - Test Case 53: 9 10 29 with excess -44508
Greedy Way - Test Case 54: 2 5 6 with excess -183686
```

Results(0-54) on the picture.

If you want to see specific test, just change this at main

```
x=0
    while(x<100):
```

x and number in while condition.

For example if you want to see case 37, x=37 number in condition is=38

COMPLEXITY

First I'm sorting the cases according to products

I'm finding maximum of product counts in given order, then I'm picking according to this product which have closest number(if possible bigger) from cases. Until it satisfies all products, it repeats.

N=case number

M=order number

K=product number

Finding total number of product in all orders = $O(m * k)$

Sorting cases according to products (python sorted function= $O(n * log_n)$)= $O(m * n * log_n)$

Loop until cases finished or satisfied products = (worst case cases finished) $O(n)$

Pick best case which have most close number of required product = $O(n)$

Subtract chosen case from order and check if it satisfies constraint= O(k)

Total Complexity= $\boldsymbol{O(m * (k + n * log_n) + (k * n^2))}$

ALGORITHM(GREEDY)

Step 1- Sort cases in several ways according to product counts(5 product mean 5 sorted list)

Step 2- Loop until cases finished or constraints satisfies

> **Step 3- Find maximum count product of given products in order list**
>
> **Step 4- Pick best case which have most close number of required product(max required one)**
>
> **Step 5- Subtract chosen case from order then upgrade order, check if it every product is enough. If not, loop until cases finish or a solution is found.**

Step 6- Return list representation of subset and excess values for future compares (If not a single subset satisfy conditions, it will return empty subset, the initial value of index of satisfied subset)

BRANCH AND BOUND(HW2)

HOW TO RUN

If you run the main, it will try to solve all test cases in brute force. But I waited 5 minute and I only saw 55 test(it was 38 for 30 minutes in brute force)

```
BranchAndBound Way - Test Case 0: 6 7 8
BranchAndBound Way - Test Case 1: 1 5 6 7 8 10 11
BranchAndBound Way - Test Case 2: 4 7 11
BranchAndBound Way - Test Case 3: 2 3 5 6
BranchAndBound Way - Test Case 4: 1 2 3 4
BranchAndBound Way - Test Case 5: 1 3 4 5 8
BranchAndBound Way - Test Case 6: 2 3 4 5 6 7
BranchAndBound Way - Test Case 7: 2 4 5 8
BranchAndBound Way - Test Case 8: 3 5 8 9
BranchAndBound Way - Test Case 9: 2 3 6 8
BranchAndBound Way - Test Case 10: 2 3
BranchAndBound Way - Test Case 11: 8 10 11
BranchAndBound Way - Test Case 12:
BranchAndBound Way - Test Case 13: 1 2 3 5 6 7
BranchAndBound Way - Test Case 14: 3 6 8
BranchAndBound Way - Test Case 15: 1 3 4 11
BranchAndBound Way - Test Case 16: 2 4 5 6
BranchAndBound Way - Test Case 17: 1 5 6 9 10
BranchAndBound Way - Test Case 18: 1 3 8 10
BranchAndBound Way - Test Case 19: 2 7 10
BranchAndBound Way - Test Case 20: 1 2 3 9
BranchAndBound Way - Test Case 21: 1 2 9
BranchAndBound Way - Test Case 22: 4 5 9
BranchAndBound Way - Test Case 23: 2 4 8
BranchAndBound Way - Test Case 24: 4 5 6 10
BranchAndBound Way - Test Case 25: 9 11
BranchAndBound Way - Test Case 26: 2 5 9 11
BranchAndBound Way - Test Case 28: 6 13 16 19
BranchAndBound Way - Test Case 29: 2 4 8 13 14 19
BranchAndBound Way - Test Case 30: 2 5 6 13 17 20
BranchAndBound Way - Test Case 31: 6 7 15 16
BranchAndBound Way - Test Case 32: 2 7 13 14
BranchAndBound Way - Test Case 33: 13 20
BranchAndBound Way - Test Case 34: 6 7 12
BranchAndBound Way - Test Case 35: 1 2 6 8 9 10 15 16
BranchAndBound Way - Test Case 36: 6 9 14 20
BranchAndBound Way - Test Case 37: 2 11 13 14
BranchAndBound Way - Test Case 38: 3 12 13 14 15
BranchAndBound Way - Test Case 39: 4 6 14 17 19
BranchAndBound Way - Test Case 40: 1 11 17
BranchAndBound Way - Test Case 41: 1 3 6 8 11
BranchAndBound Way - Test Case 42: 8 21 23 24
BranchAndBound Way - Test Case 43: 4 5 6 8 11
BranchAndBound Way - Test Case 44: 2 21
BranchAndBound Way - Test Case 45: 9 11 18 23
BranchAndBound Way - Test Case 46: 3 7 9 15 18
BranchAndBound Way - Test Case 47: 2 4 6 7 11 13
BranchAndBound Way - Test Case 48: 1 10 15 18
BranchAndBound Way - Test Case 49: 7 19
BranchAndBound Way - Test Case 50: 2 6 13 29 34
BranchAndBound Way - Test Case 51: 5 7 8 10 14 16 17 23 25
BranchAndBound Way - Test Case 52: 14 25 37 40
BranchAndBound Way - Test Case 53: 11 15 26
BranchAndBound Way - Test Case 54: 14 17 20
```

Results (0-54) on the picture.

If you want to see specific test, just change this at main

```
x=0
    while(x<100):
```

x and number in while condition.

For example, if you want to see case 37

x=37 number in condition is=38

COMPLEXITY

Its very hard to say exact complexity of brand and bound. It can traverse all nodes but it can traverse only from one branch too. So I'll calculate complexities for best and worst cases.

N=case number

M=order number

K=product number

Worst Case

Finding total number of product in all orders = $O(m * k)$

Traverse all subsets of cases = $O(2^n)$

Finding total number of product in all subsets = $O(n * k)$

Comparing product numbers with order product numbers= O(n)

Worst Case= $\boldsymbol{O(m * k + 2^n * n * k)}$

Best Case

Finding total number of product in all orders = $O(m * k)$

Traverse one node in subsets of cases = $O(log_n)$

Finding total number of product in subset = $O(n * k)$

Comparing product numbers with order product numbers= O(n)

Best Case = $\boldsymbol{O(m * k + log_n * n * k)}$

ALGORITHM (BRANCH AND BOUND)

Step 1- Find bound from greedy algorithm

Step 2- right recursion is include, left is exclude (all subsets as binary representation)

Step 3- find excess value of satisfied products in current subset of node

Step 4- compare excess value with bound

> **If it satisfies all products and better than bound, upgrade bound, return it to top recursion**
>
> **If it doesn't satisfy all products, check its current excess value of satisfied products, if its worse don't branch this node(kill this node)**

Step 5- Return binary representation of subset (If not a single subset satisfies conditions, it will return -1, the initial value of index of satisfied subset)

BRUTE FORCE(HW1)

HOW TO RUN

If you run the main, it will try to solve all test cases in brute force. But I waited 30 minute and I only saw test 38

Results(0-38) on the next picture.

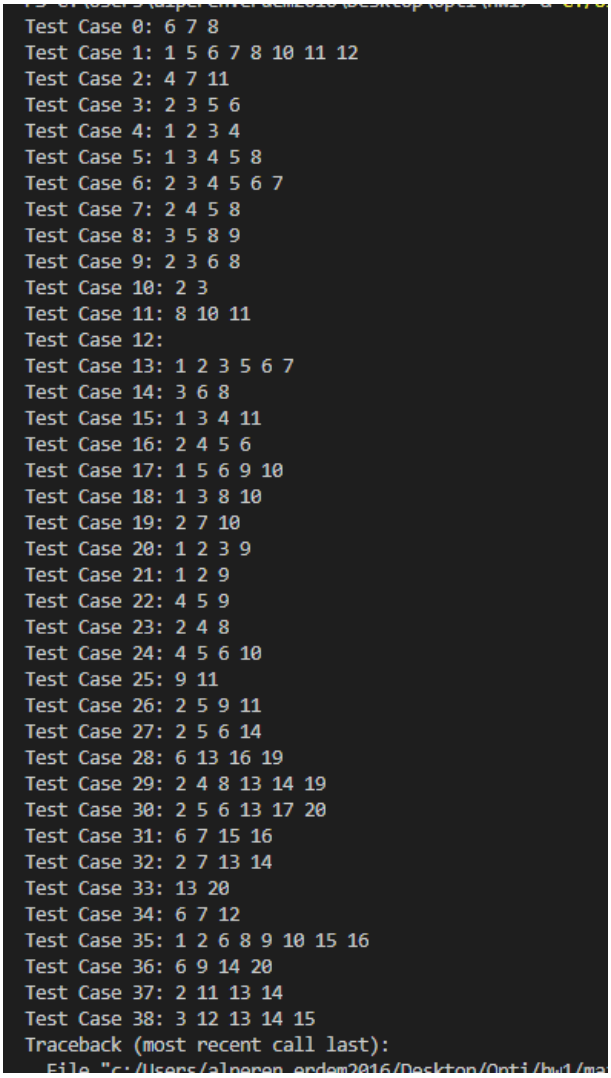
If you want to see specific test, just change this at main

```
x=0
    while(x<100):
```

x and number in while condition.

For example if you want to see case 37

x=37 number in condition is=38



COMPLEXITY

N=case number

M=order number

K=product number

Finding total number of product in all orders = $O(m * k)$

Loop in subsets of cases = $O(2^n)$

Finding total number of product in all subsets = $O(n * k)$

Comparing product numbers with order product numbers= O(n)

Total Complexity= $\boldsymbol{O(m * k + 2^n * n * k)}$

EXPLAIN OF CODE and ALGORITHM

I'm reading given.txt file for making list of test objects.

All test objects are creating one order class for all orders (summing them) while reading also creating list of cases.

Then I'm calling bruteforce method with this test objects.

ALGORITHM (BRUTE FORCE)

Step 1- find size of subsets of cases

Step 2- represent all subsets as binary (1=include 0=exclude)

Step 3- find sum of products in chosen subset

Step 4- compare products of subset with products of order

If it satisfies all products, compare with last satisfied subset

If it has less excess product, save subset, total excess product and its index

Step 5- Do step 3 with all subsets

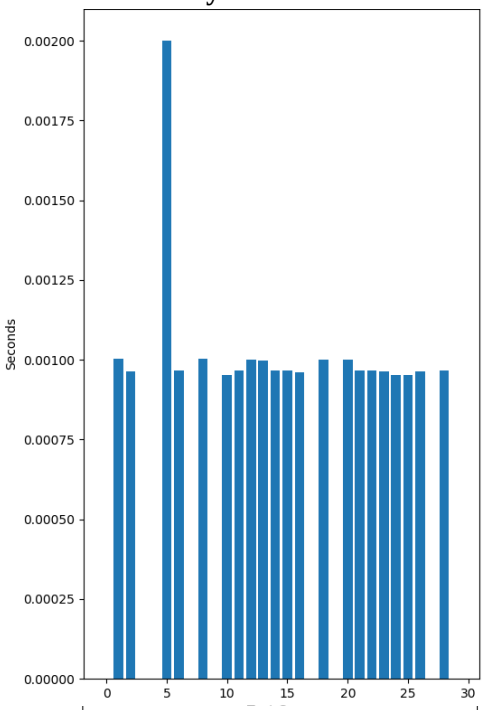
Step 6- Return binary representation of subset (If not a single subset satisfy conditions, it will return -1, the initial value of index of satisfied subset)

COMPARE BRUTEFORCE-BRANCH&BOUND-GREEDY METHODS

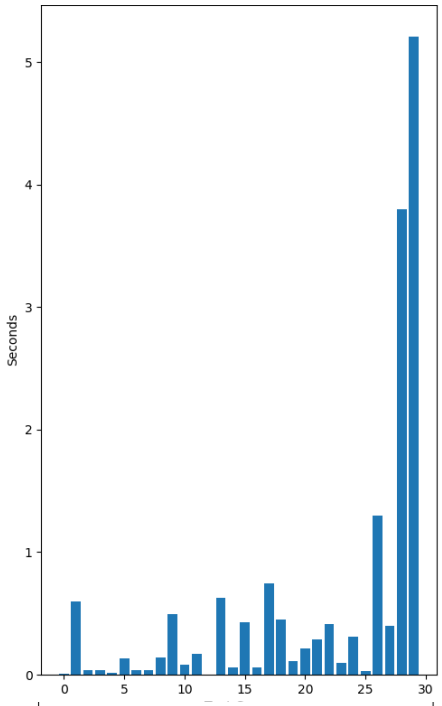
Because of brute force working so slow, I only compare for first 30 test case.

EXECUTION TIME

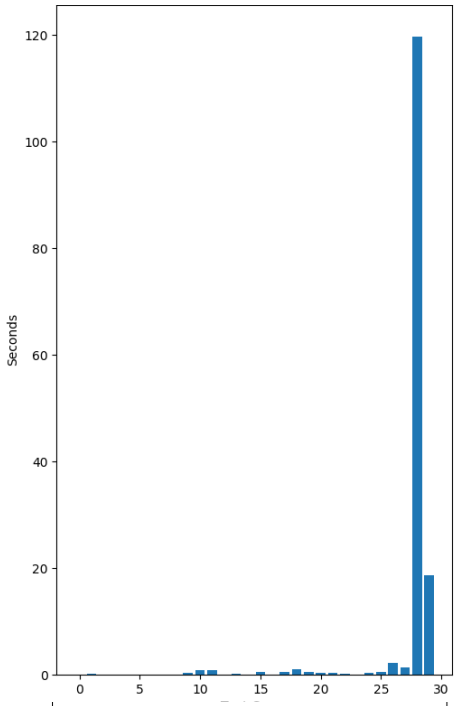
Greedy

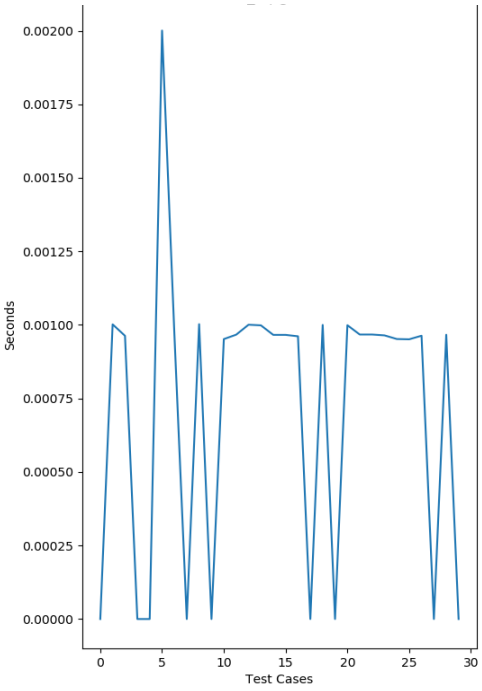


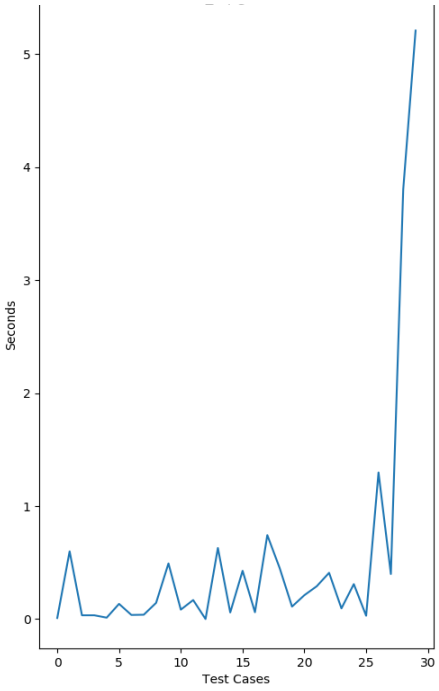
Branch And Bound

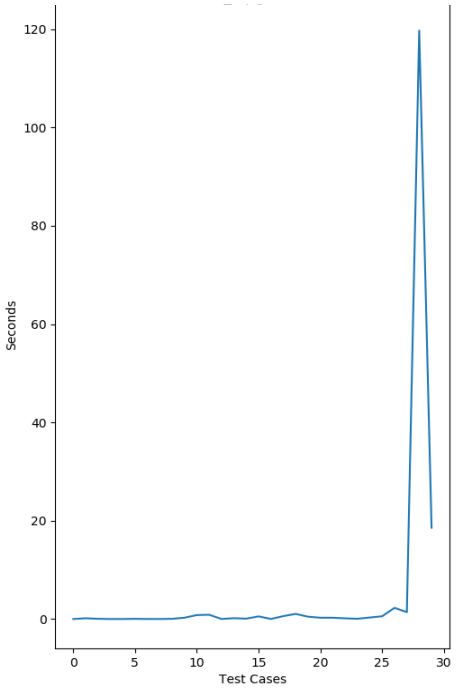


Brute-Force







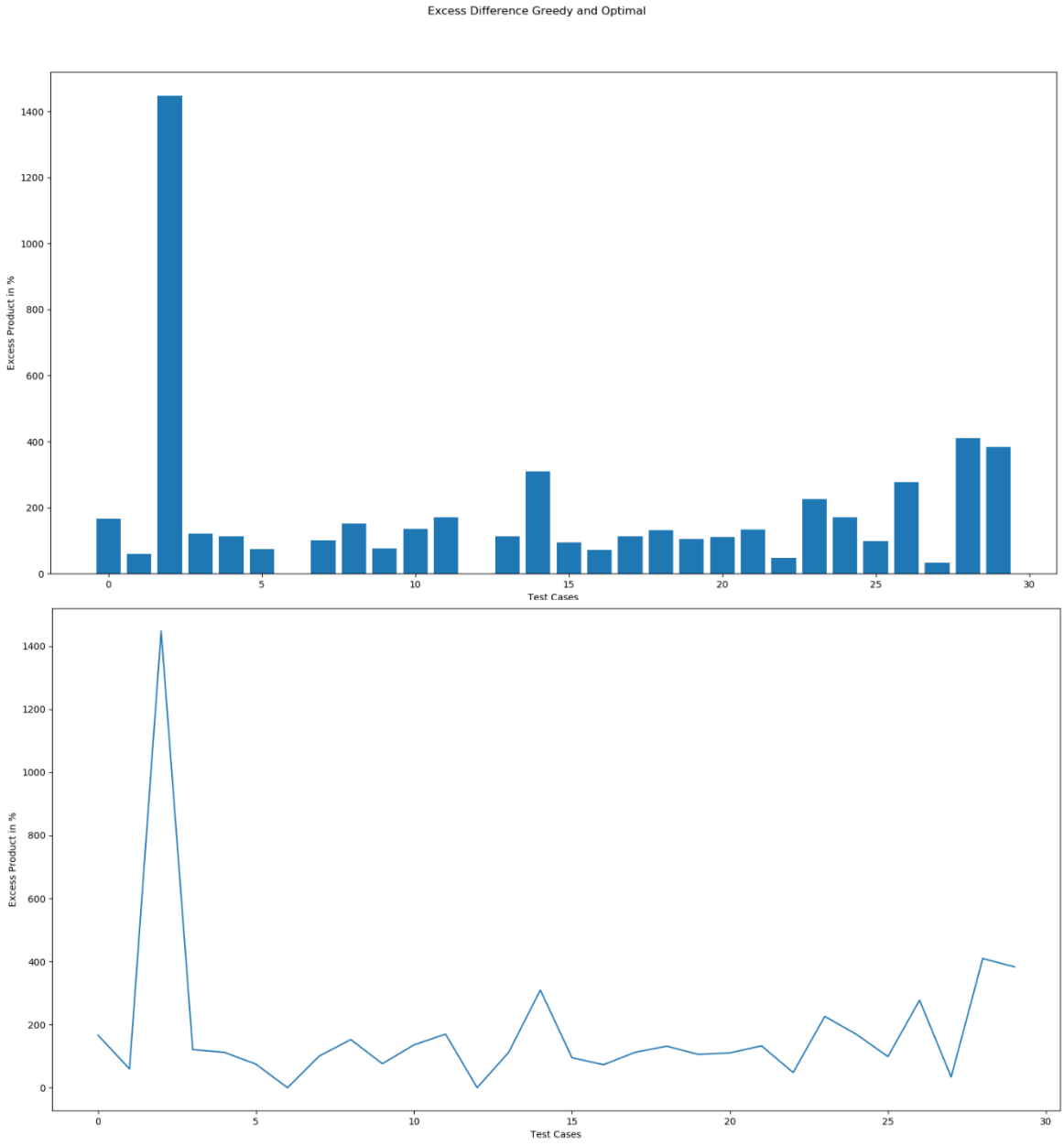


Excess Product (Greedy vs Optimal)

Both brute force and branch-bound methods are giving most optimal solution. So I'll compare excess product between greedy and branch-bound, because branch and bound is faster.

My method for compare

$$y = \frac{\text{ExcessGreedy} - \text{ExcessOptimal}}{\text{ExcessOptimal}} * 100$$



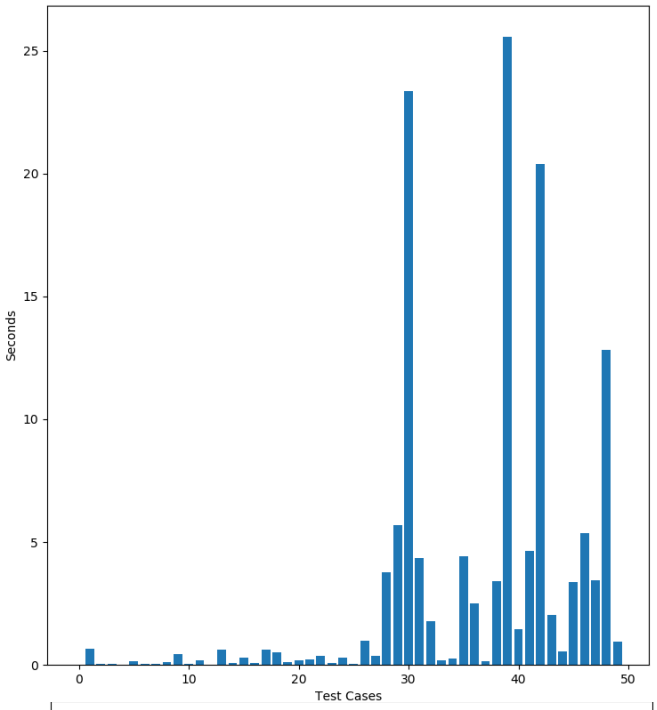
According to graphs, greedy is faster but find average of 2.5 times worse solution. But sometimes like test case 2, it finds like 14times worse solution which is never acceptable.

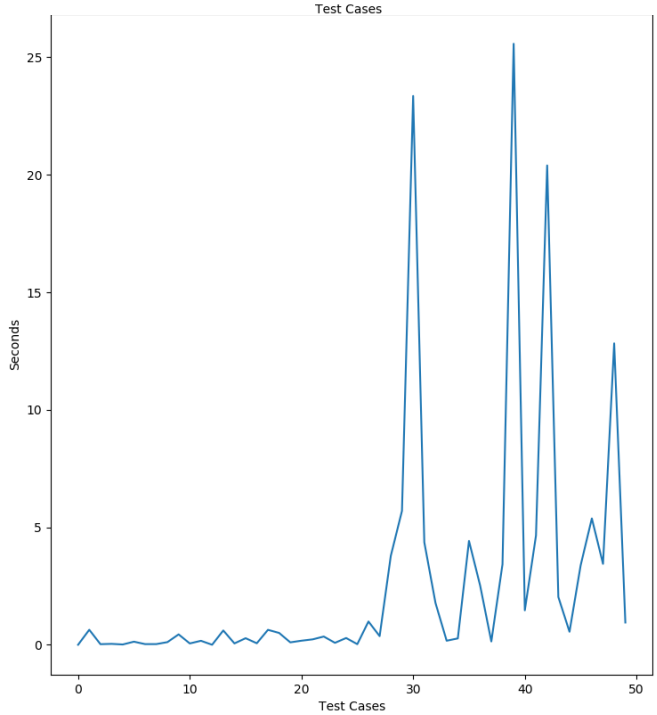
COMPARE VNS AND BRANCH&BOUND METHODS

Because of branch and bound working little slow, I only compare for first 50 test case.

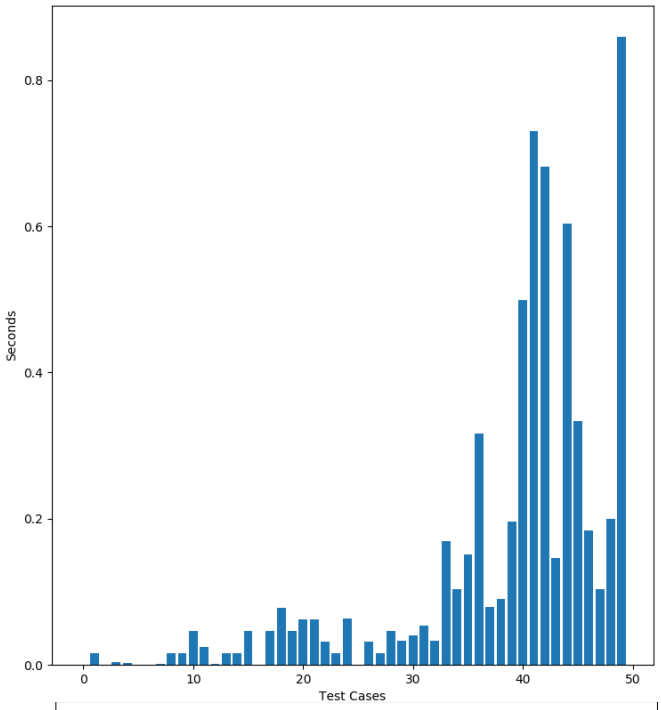
EXECUTION TIME

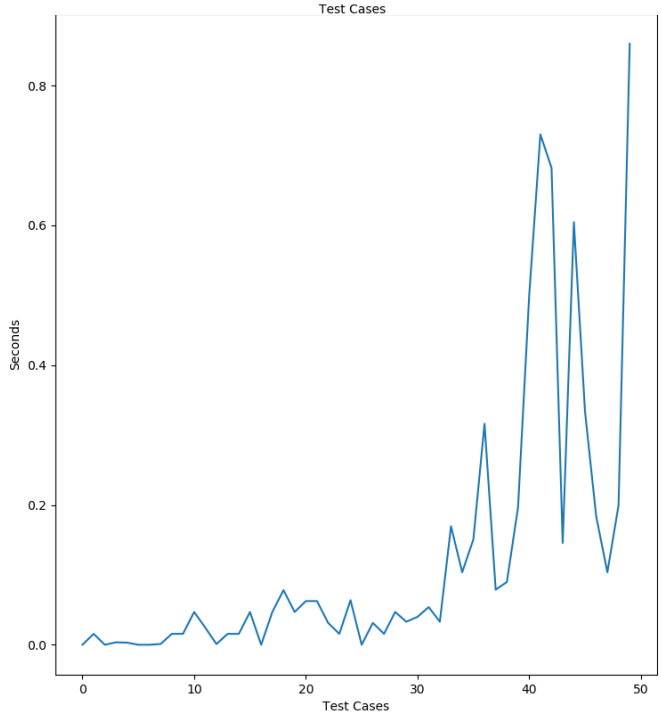
Branch And Bound





Variable Neighborhood Search



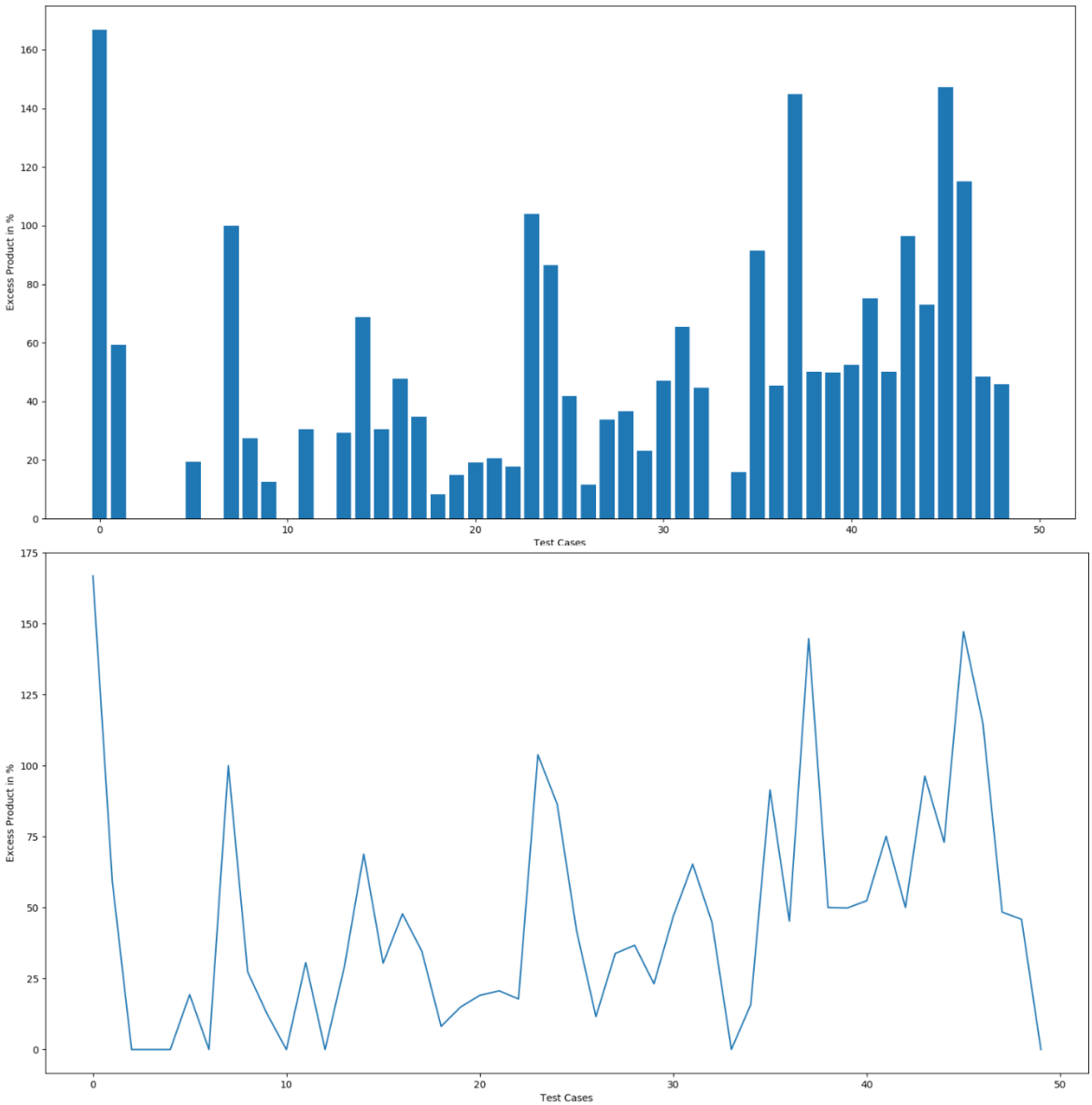


Excess Product (VNS vs Optimal)

Both brute force and branch-bound methods are giving most optimal solution. So, I'll compare excess product between **VNS** and branch-bound, because branch and bound is faster.

My method for compare

$$y = \frac{\text{ExcessGreedy} - \text{ExcessOptimal}}{\text{ExcessOptimal}} * 100$$



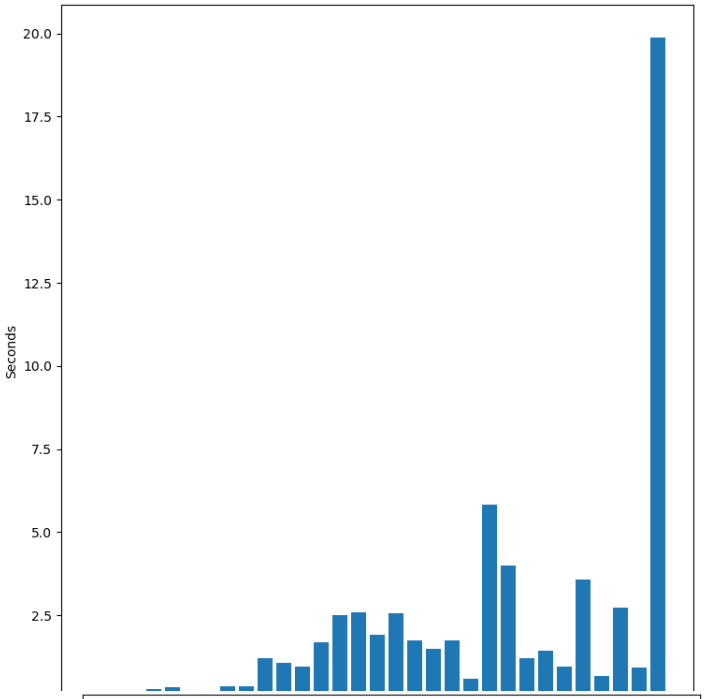
In old comparisons, we found that that Branch-Bound is optimal but longer, but now with VNS method we can execute all test cases in 5min while in Branch-Bound it lasts hours and VNS is giving average of 0.6 times worse solution. I think its in acceptable range so we can say VNS is acceptable method for bigger test cases.

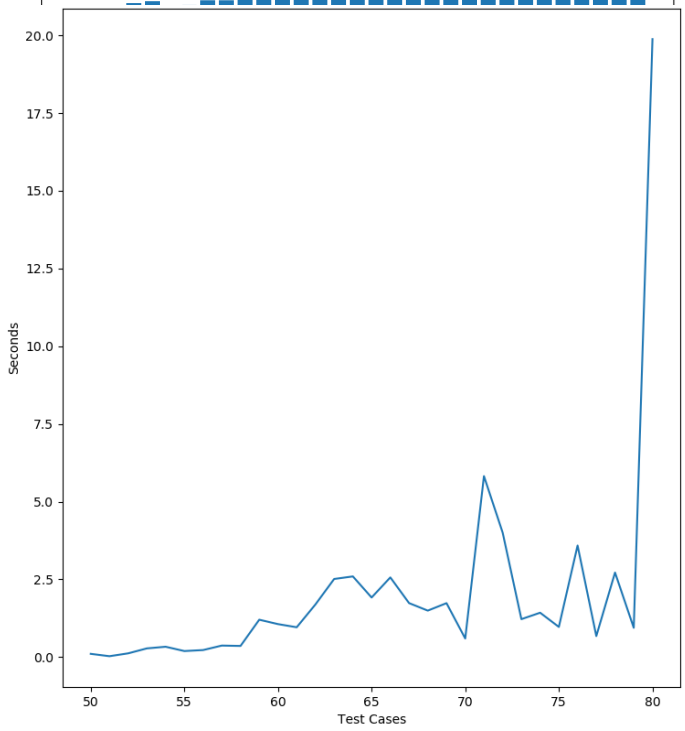
COMPARE SIMULATED ANNEALING AND VNS METHODS

From test case 50 to 80

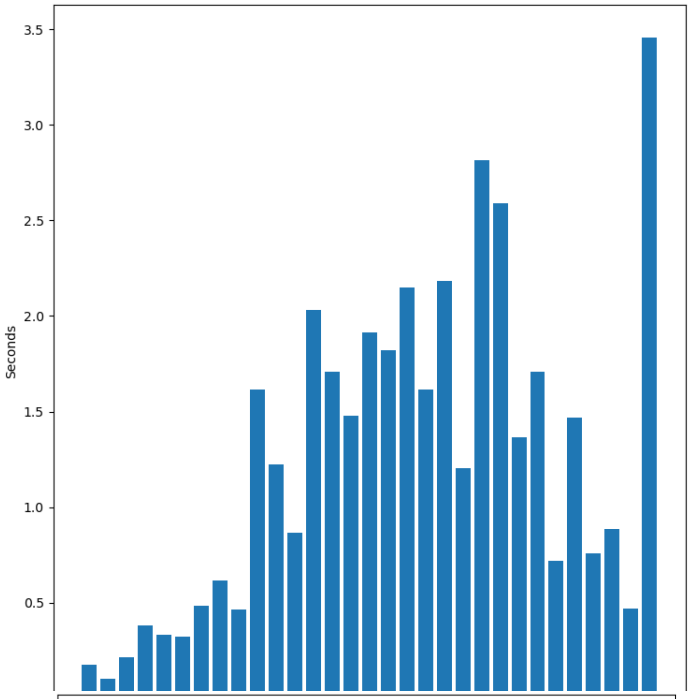
EXECUTION TIME

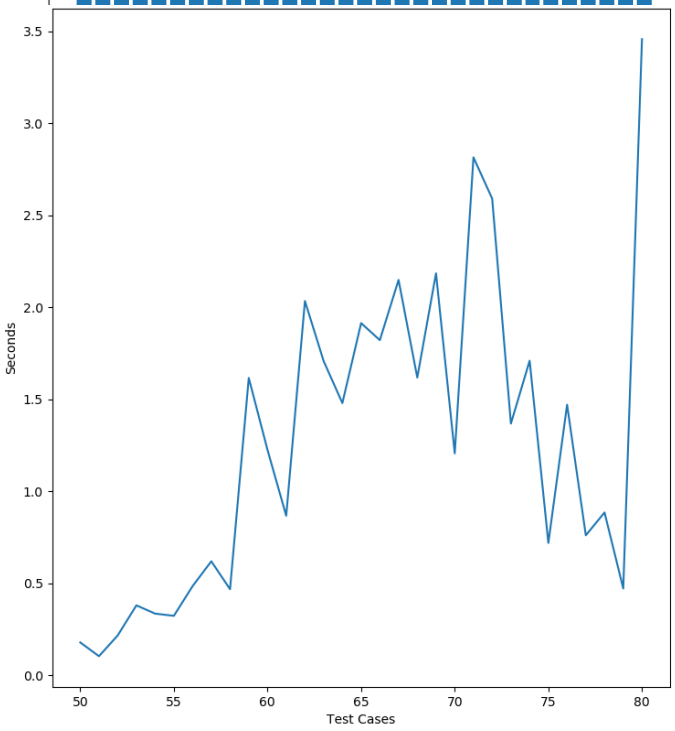
Variable Neighborhood Search





Simulated Annealing



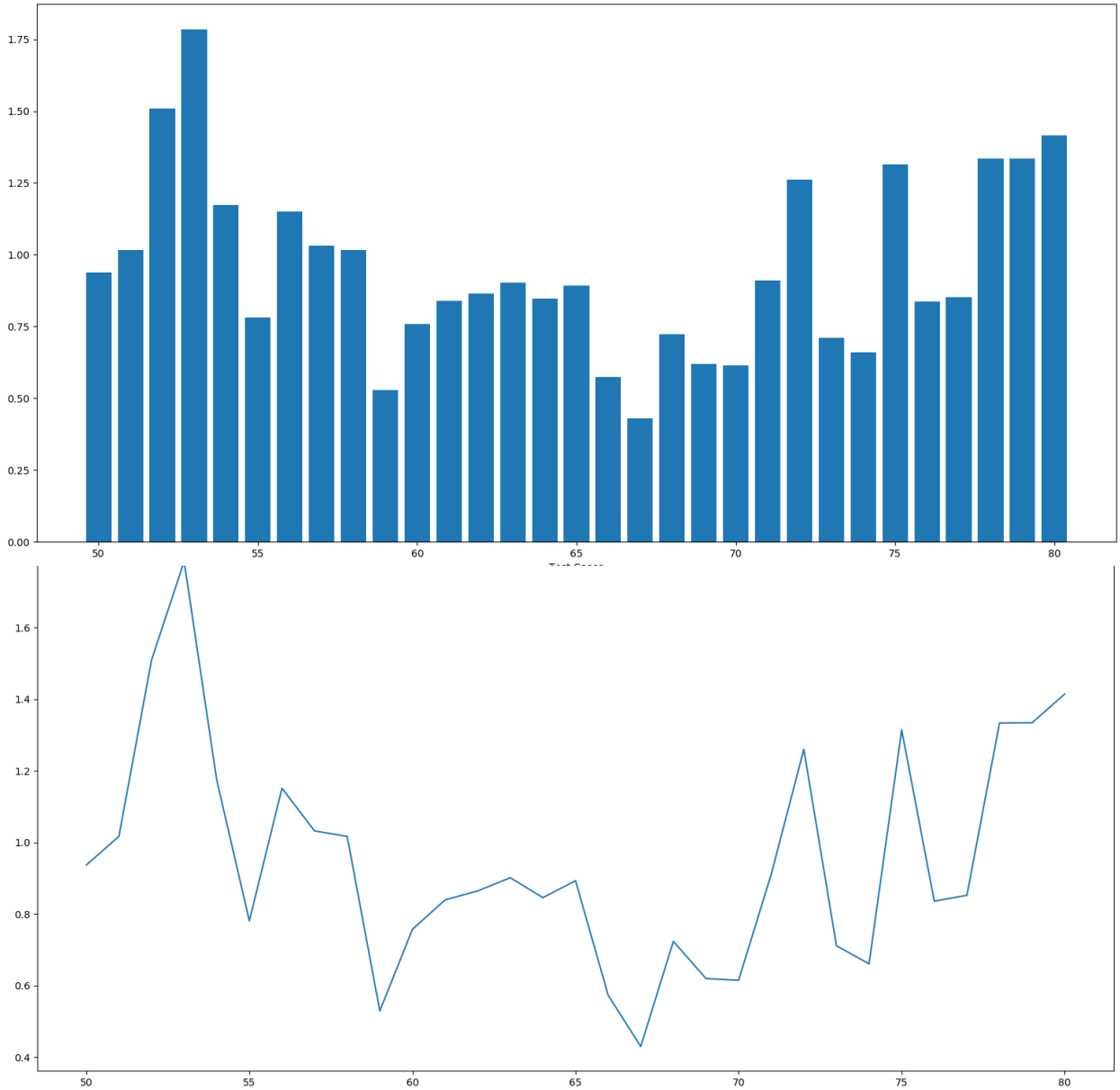


Excess Product (VNS vs Simulated Annealing)

I'll compare how many times Simulated Annealing is better than VNS with simple excess divide.

My method for compare

$$y = \frac{\text{ExcessVNS}}{\text{ExcessSimulatedAnnealing}}$$



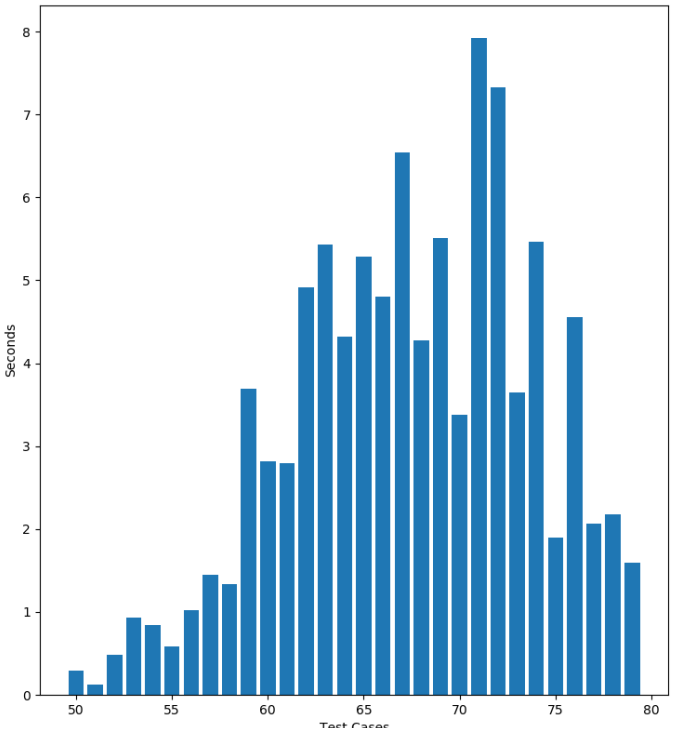
In old comparisons, we found that that VNS is better than other methods, not giving best solution but giving ok solutions for average. But for larger test cases, it still works slow. In the average, SA is working faster than VNS, but VNS giving average of 0.9 times better results. For larger tests we can use SA. Or we can choose better initial values for SA, so it will work slower but will find better solutions.

COMPARE TABU SEARCH AND SIMULATED ANNEALING

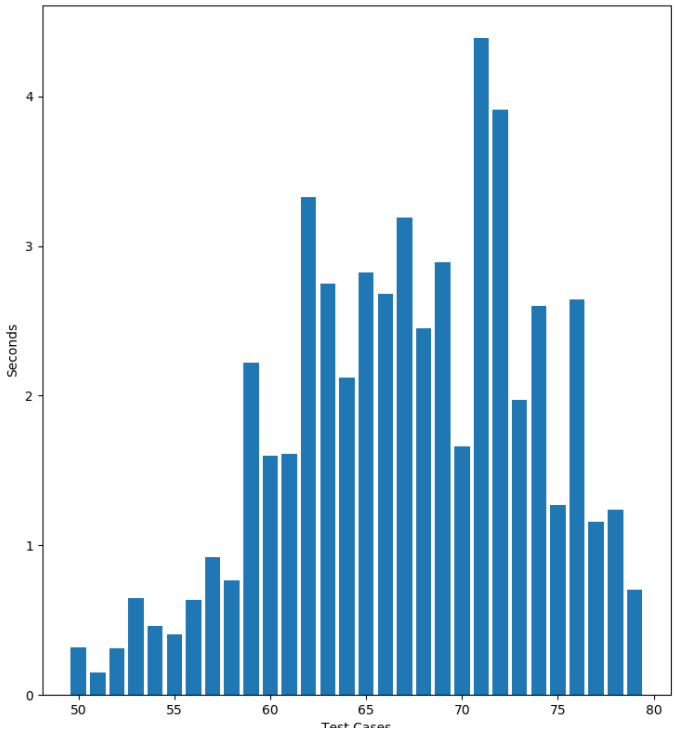
From test case 50 to 80 and 20 to 50

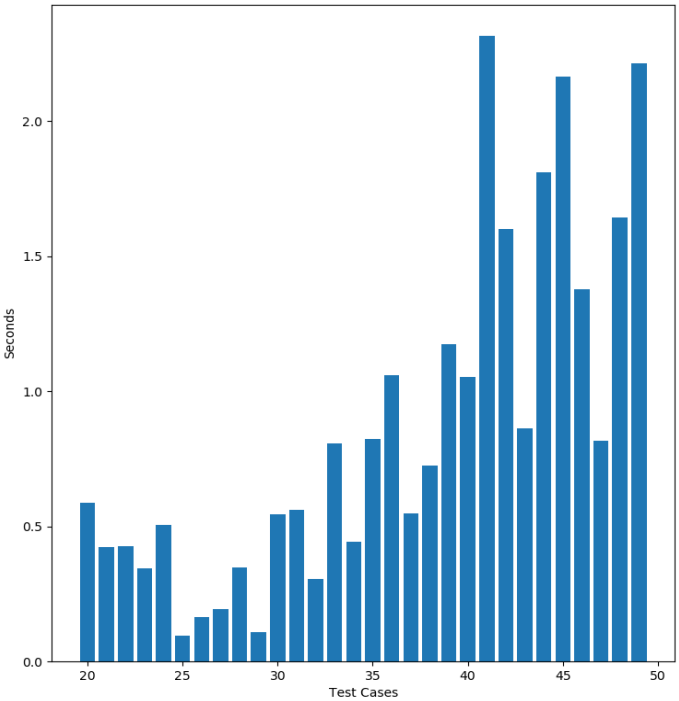
EXECUTION TIME

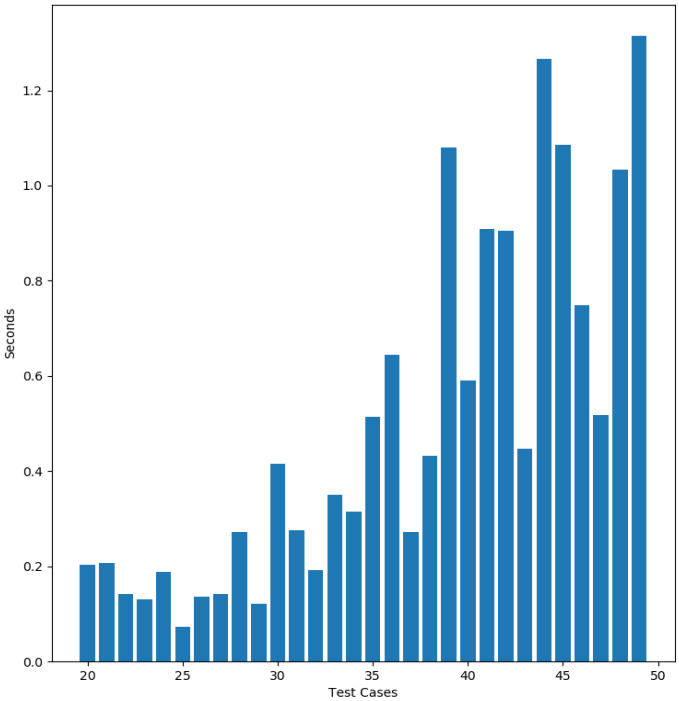
Tabu Search



Simulated Annealing



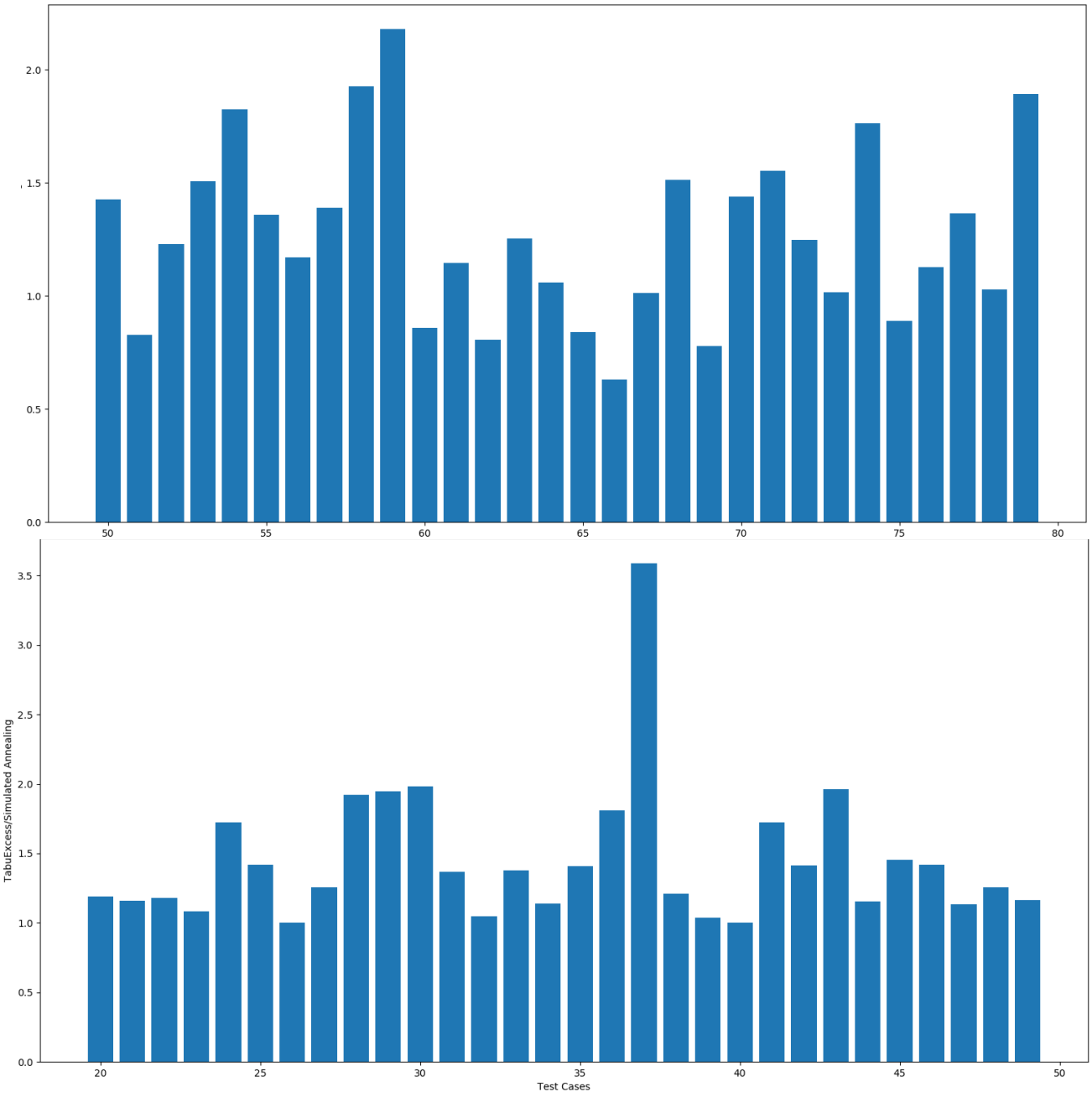




Excess Product (Tabu Search vs Simulated Annealing)

I'll compare how many times Simulated Annealing is better than VNS with simple excess divide.

My method for compare $y = \frac{\text{ExcessSimulatedAnnealing}}{\text{ExcessTabuSearchs}}$



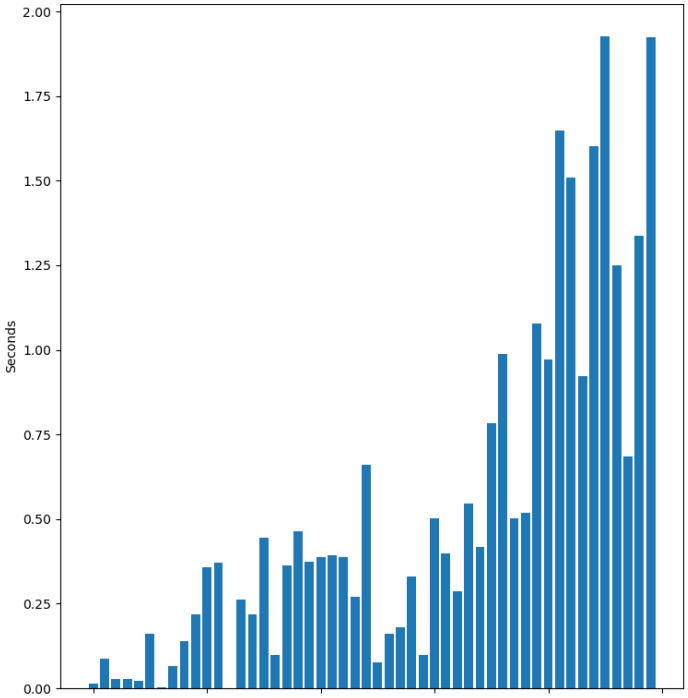
It looks like my simulated annealing method working 2 times faster for these test cases but giving worse methods than tabu search. Tabu search giving average of 2 times of better results. Unless seconds of time is important, implemented tabu search is better than simulated annealing for this problem.

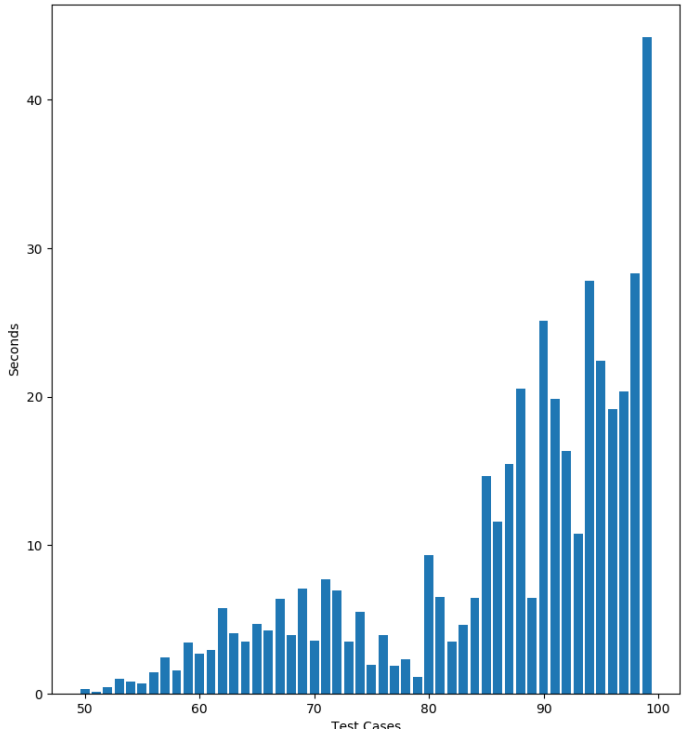
COMPARE TABU SEARCH AND GENETIC ALGORITHM

From test case 0 to 49 and 50 to 99

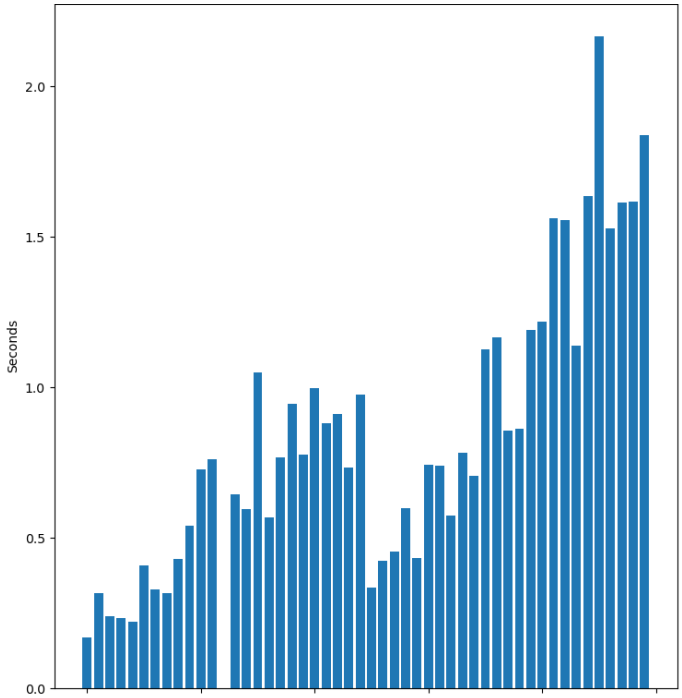
EXECUTION TIME

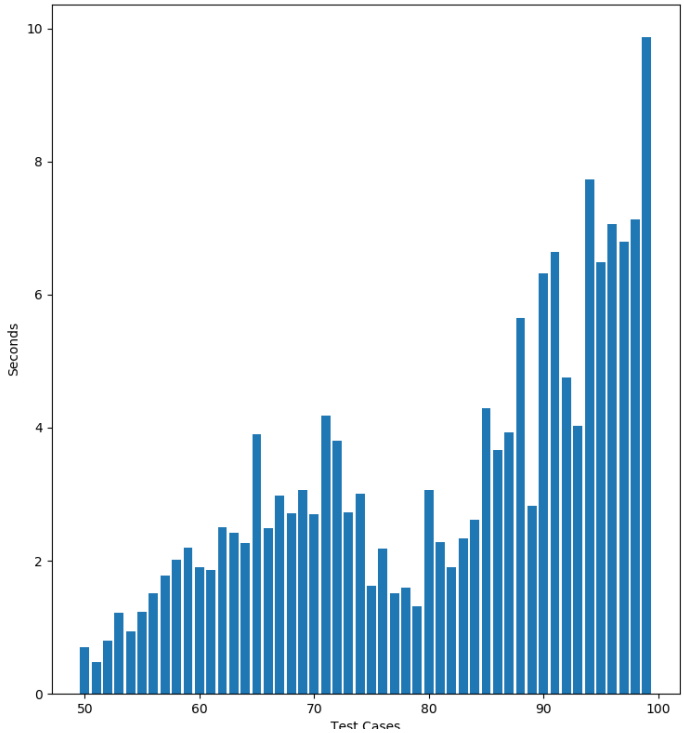
Tabu Search





Genetic Algorithm

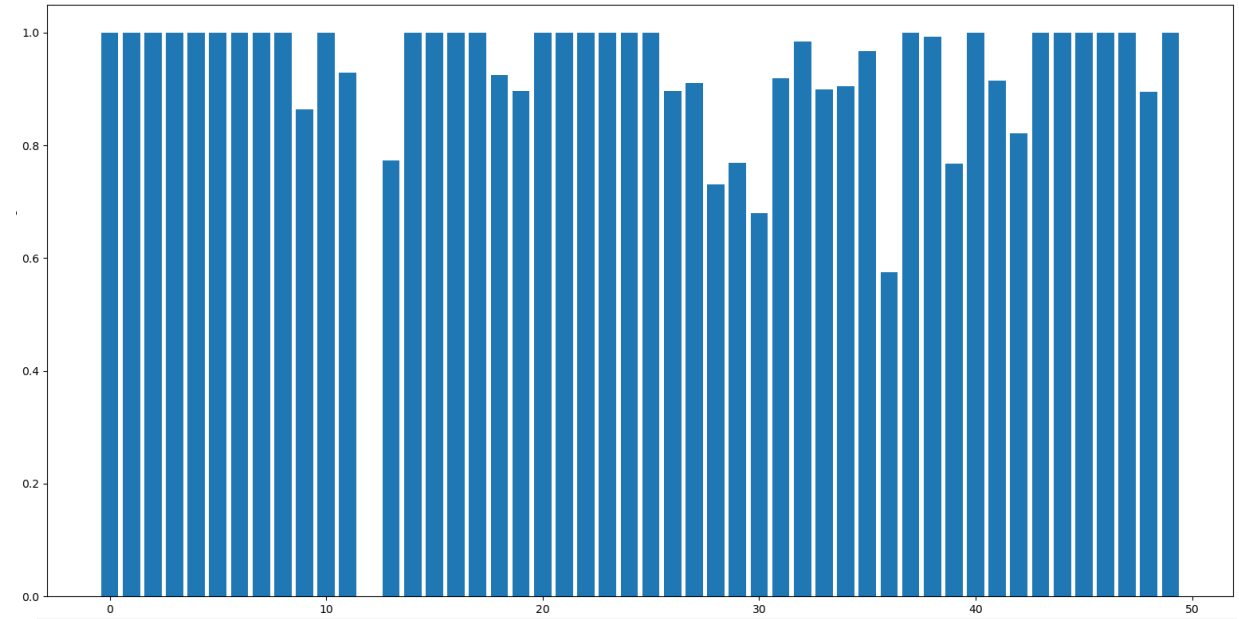


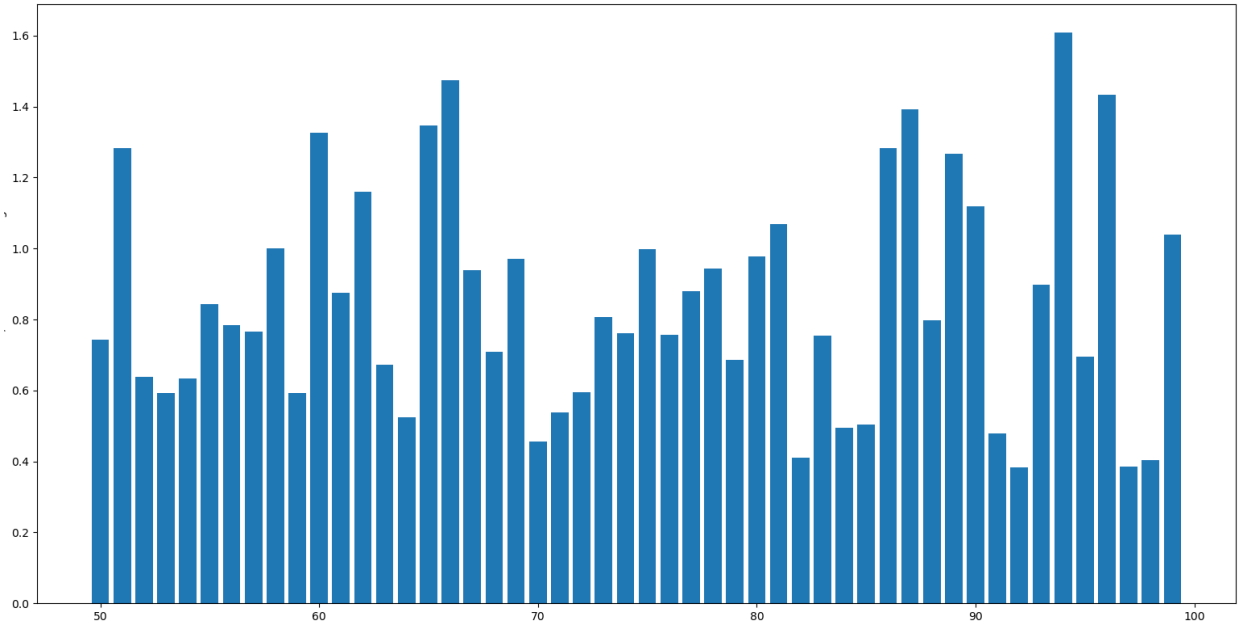


Excess Product (Tabu Search vs Simulated Annealing)

I'll compare how many times Simulated Annealing is better than VNS with simple excess divide.

My method for compare $y = \frac{\text{ExcessTabuSearchs}}{\text{ExcessGeneticAlgorithm}}$





Its easier to see from graphs, my genetic algorithm working faster, especially in last 50 test it works 4 times faster. But in average tabu search giving 1.5 times better results. Maybe with changing numbers of generation and population, we can achieve better results with longer time. But taking "not bad" results maximum at 10 seconds makes my implemented genetic algorithm preferable.