Recursion :-

Problem 1: $\frac{abcde}{fghij} = N$, $a, b, c, \ldots i, j \in \{1, 2, \ldots, 9, 0\}$

We need to find the no. of 5 digits no.s that satisfies above condition if a, b, ..., i, j are distinct.

T.C: N = 62 } user input
2 solutions

$$\frac{79546}{01283} = 62$$

&

$$\frac{94736}{01528} = 62$$

Soln: We have 2 constraints

(i) $a, b, c, \ldots, i, j \in \{0, 1, 2, \ldots, 9\}$

(ii) $\frac{abcde}{fghij} = N$

app-1 { # we can find all 5-digit numbers (a,b,c,d,e) & (f,g,h,i,j) and do a check if all are distinct or not & do cnt++ accordingly.

app-2 { # abcde, fghij } then we will have permutation of 0-9. } check if $\frac{abcde}{fghij} = N$.

we can use next_permutation to generate all possible permutation.

time complexity : $O(10!) = O(3.63 \times 10^6)$

app.3 Instead of finding both abcde, fghij we will find only fghij → all values then find abcde = N * fghij. then check if all abcde and fghij are distinct or not.

time complexity : ${}^{10}C_5 \times 5! = {}^{10}P_5$

$O(10^5)$

These are basically brute force solutions.

```
void solution1 (int n)
{
   vector<int> a;
   for (int i = 0; i<n; i++){
    a.pushback(i)
    }

   do{
      for(int u: a){
          cout << u <<" ";
      }
      cout <<"\n";
   } while (next_permutation (a.begin(), a.end()));
```

→ To generate permutation of 'n' numbers.

}

Actual code :- (for approach 2)

```
int solve (int n) {
    vector<int> a;
    for (int i=0; i<10; i++) {
      a.push_back (i);
    }

    int ans = 0;

    do {
      // constrain 1
      int abcde = a[0]*10^4 + a[1]*10^3 + a[2]*10^2 + a[3]*10+
      int fghij = a[5]*10^4 + a[6]*10^3 + a[7]*10^2 + a[8]*10+a

      // contrain 2
      if (abcde == n*fghij) ans++;
    }
    while (next_permutation(a.begin(), a.end()));

    return ans;
}
```

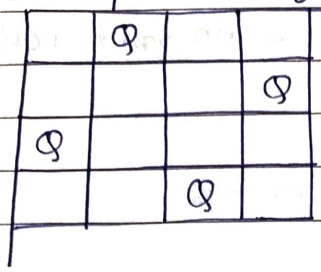
(for approach 3)

```
int solve (int n)
{
  int ans = 0;

  for(int fghij = 01234 ; fghij <= 98765/n ; fghij++)   // maxm value of abcde
  {
    // constraint 1
    int abcde = n * fghij;

    // constraint 2
    set<int> digits;

    int temp1 = abcde , temp2 = fghij ;
    for (int i=0;i<5;i++) {
    //while (temp1) {
      digits.insert (temp1 % 10);
      temp1 /= 10;
    }
    while (temp2) {
      digits.insert (temp2 % 10);
      temp2 /= 10;
    }
    if (digits.size == 10) ans++;
  }

  return ans;
}
```

we cannot use this can leading zeros will not be counted

N - Queens :

N = 4 } user input (no. of queens)



} → one valid placement.

N × N board.

Find the no. of places where queens can be placed without attacking each other.

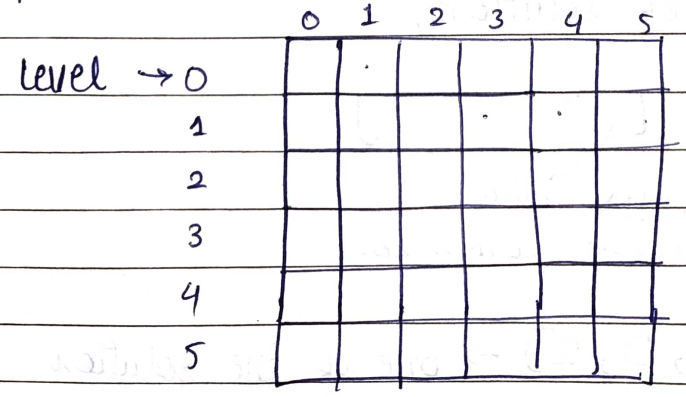
soln: each queen needs to have different row and column.

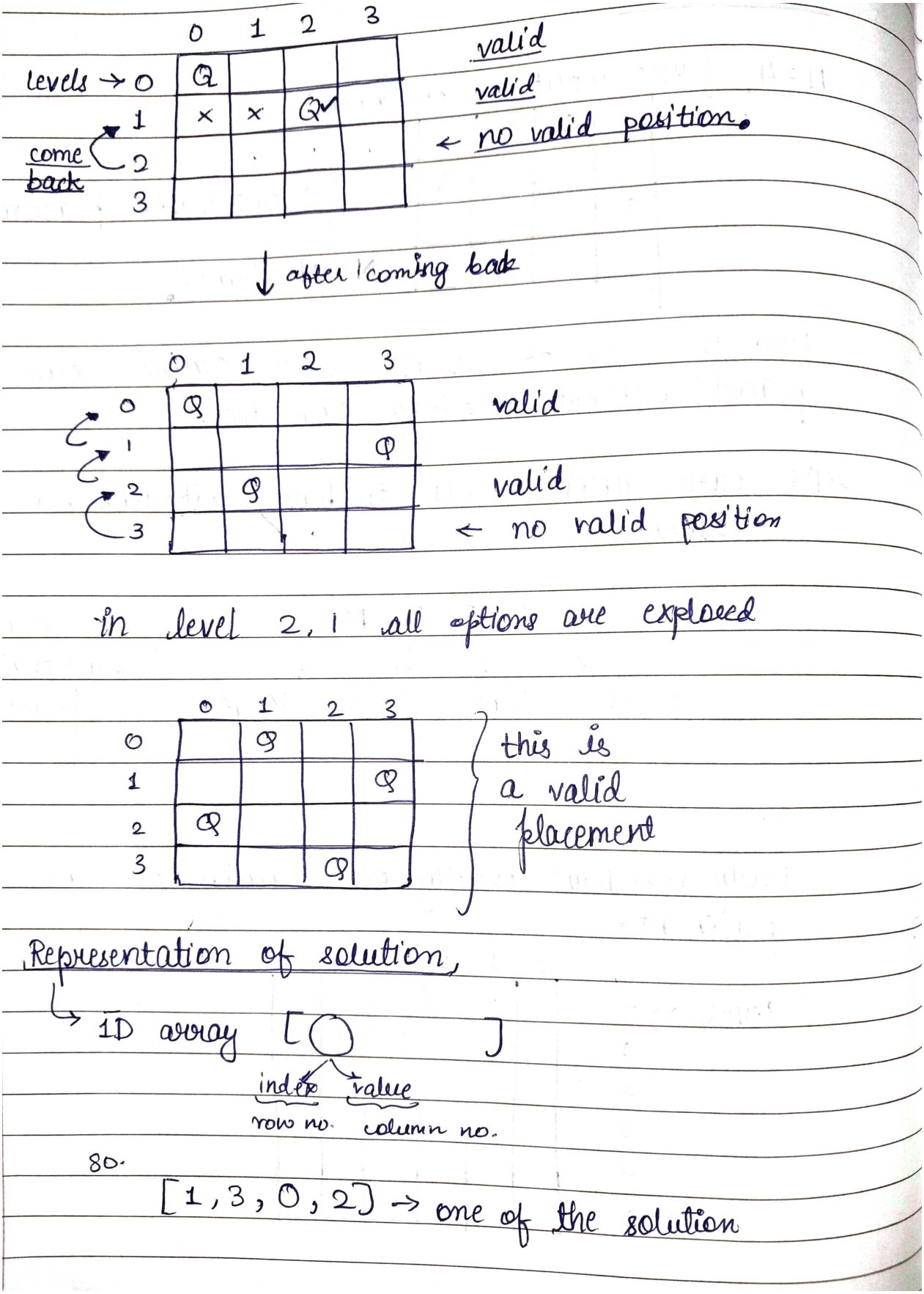
L C C M

level choice check Move

} for any backtracking problem.

Every row have exactly one queen in valid placement.





in level 2, 1 all options are explored

Representation of solution,

↳ 1D array [◯]

index → row no. value → column no.

80.

[1, 3, 0, 2] → one of the solution

[1, 3, -1, -1]

↑ no queen placed at that row (2) or (3)

~~void solve (int~~

Time complexity :- O(n!)

Inversion count using merge sort :-

Q1. Arr - N
↳ binary contains only 0/1

Find # subarrays such that (a) $\frac{\#0}{\#1} = \frac{P}{Q}$

(b) $\frac{\#0}{\#1} < \frac{P}{Q}$

(a)

Soln: Sample Input : 0 1 0 0 1 0 1

- Think of this problem as : $Q \cdot \#0 = P \cdot \#1$

Bruteforce : $O(n^2)$

```
int n,p,q;
cin >> n>>p>>q;

int arr[n];
int ans =0;
for(int i=0; i<n ; i++) {
        int cnt[] = {0,0};
        for (int j=i; j<n;j++) {
            cnt[arr[j]]++;
            if (p*cnt[1] == q*cnt[0]) {
            ans++
            }
        }
}
```

Divide and Conquer :

here we have,

$$q\#0 - p\#1 = 0$$

(a)

$$\#0(q) + \#1(-p) = 0$$

every zero contributes q to the sum
every one contributes $-p$ to the sum

q	-p	q	q	-p	q	-p

↳ find no. of subarrays such that sum of elements in it is 0.

sum_subarray(i,j)
pre[j] - pre[i-1] = 0

pre[j] = pre[i-1]

How many prefix sums are equal.

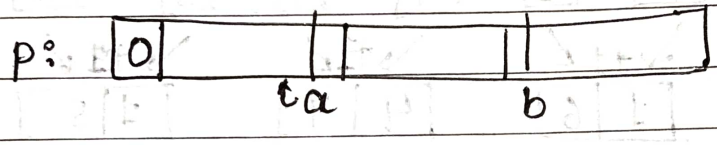
Divide and Conquer:

(b) Now

$\#0(Q) + \#1(-P) < 0$

$P[r] - P[l-1] < 0$

$P[r] < P[l-1]$



$P[b] < P[a]$

we want to find no. of pairs

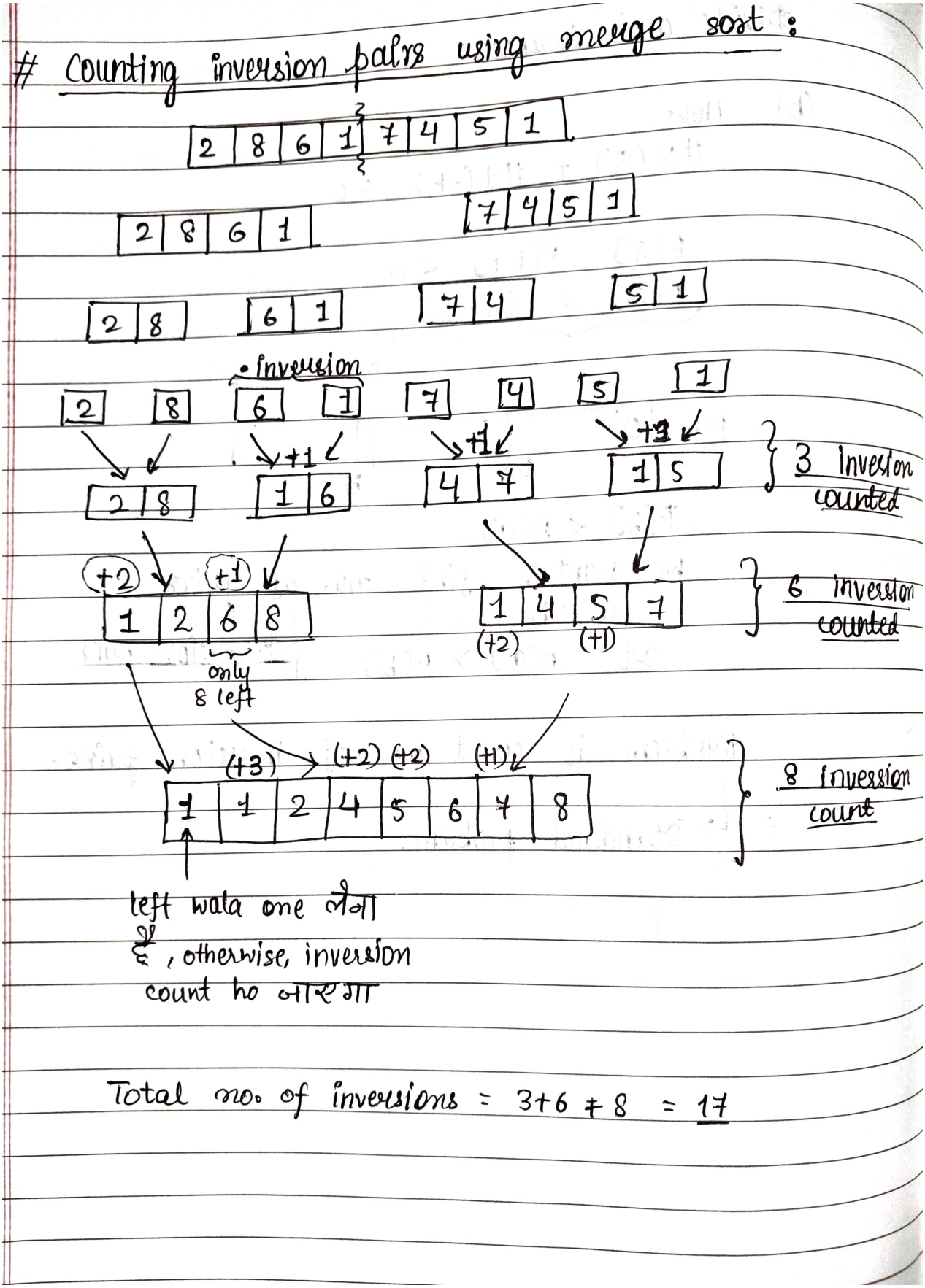
$a < b$

s.t $P[a] > P[b]$ } Inversion pair

Problem is count no. of inversion pairs.

↳ standard problem.

Counting inversion pairs using merge sort :



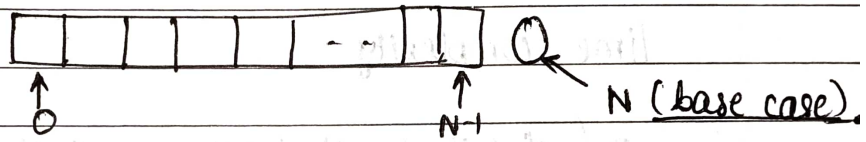
Total no. of inversions = 3+6+8 = 17

Framework for Backtracking

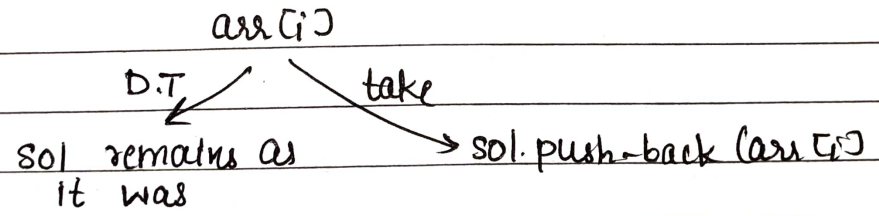
- level → things we can iterate on
- choice → Take / Don't take
- check → Whatever we take, is it valid
- move → changing the state.

eg 1: Generate all K sized subset of arr of size N
$K \leq N$.

soln: arr[]:



level → each arr[i] (0, 1, 2, .. N-1)
choice → take / not take
check → if #items ≤ k
move → change the element.



N-Queens :-

- Each row will have one queen
- Row is our level
- for each row, we have n choices
- for each choice, we will have to check
- then pass to other level. (move)

Time complexity

$$O(\underbrace{ch1 * ch2 * ch3 * \cdots * chn}_{N!} * \underbrace{(\text{Base case} + \textstyle\sum \text{checks})}_{(N + N^2)})$$

Date ___/___/_____ Basic examples:

eg:
```
F(string s){
    return (s.length() ≥ 2 && (s[0] == s[1] || F(s.substr(1))
}
```

Suppose s = a [b b] ← s.substr(1)
s[0] ≠ s[1]

F(bb) = true.

∴
F returns if any 2 consecutive characters are same or not.

eg:
```
F(x,y){
  if(x==0)
        return 0
  else
     return [F(x/y, y) + x/y];
}
```

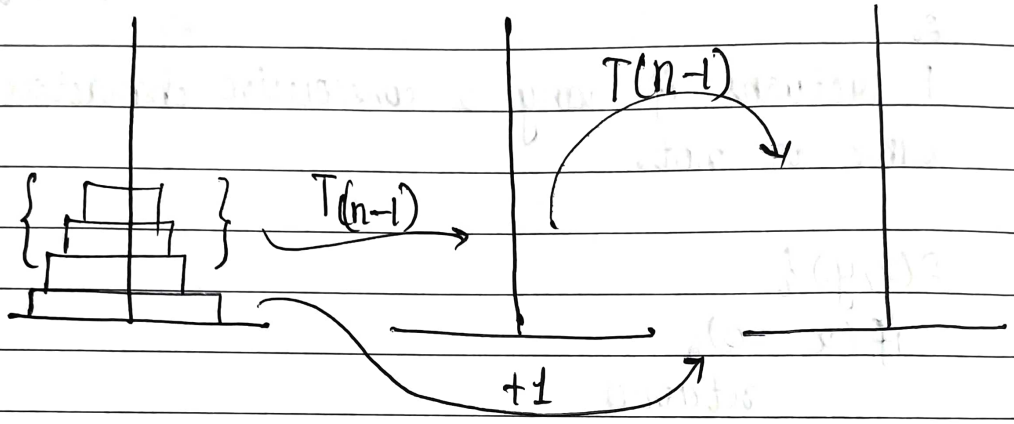
Ans = $\underbrace{\left[\frac{x}{y}\right] + \left[\frac{x}{y^2}\right] + \left[\frac{x}{y^3}\right] + \dots}$

Exponent of prime y in x!

Tower of Hanoi

1. Find seq. to move N(1 → 3)
2. Find k^{th} move
3. You have 4 peg.

N = 4



$$T(n) = 2T(n-1) + 1$$

$T(1) = 1$

$T(2) = 2(1) + 1 = 3$

$T(3) = 2(2+1)+1 = 2^2+2+1$

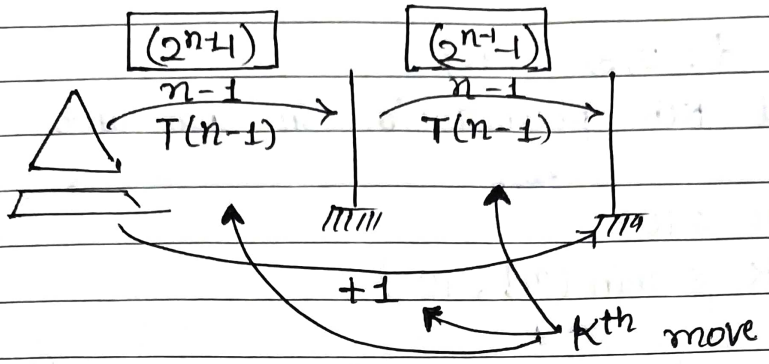
$T(4) = 2(2^2+2)+1)+1 = 2^3+2^2+2+1$

⋮

$T(n) = 2^{n-1} + 2^{n-2} + \cdots + 1$

$T(n) = 2^n - 1$

Find Kth move :



Kth Permutation

Q. $A = \{1, 2, 3, \ldots, n\}$
Find kth permutation in that list.

- $1 \le n \le 10^5$.
  $1 \le K \le \min(n!, 10^5)$

Solⁿ:

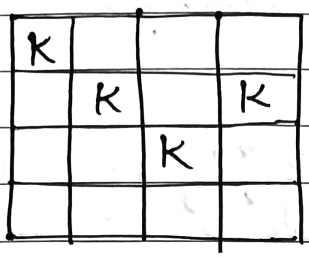
1 [_ _ _ _ _] } Class 1 — K^{th}
$(n-1)!$

2 [_ _ _ _] $(n-1)!$ } Class 2 — $K^{th} - (n$

3 [_ _ _ _ _] $(n-1)!$ } Class 3

Find the chunck 'K' lies in, then place that number & recursively find the others.

K Knights :-



N = 4
K = 4

Find # positions for which no 2 knights attack each other.

Use bruteforce, solve for all psbl placements of knights.

Karatsuba Multiplication :-

$$X = x_0 + x_1 * B + x_2 * B^2 + \ldots\ldots + x_n B^n$$
$$Y = y_0 + y_1 * B + y_2 * B^2 + \ldots\ldots + y_n B^n$$

$$Z = z_0 + z_1 B + z_2 B^2 + \ldots + z_{2n} B^{2n}$$

$$X_L + B^{\lfloor \frac{n}{2} \rfloor} X_R = X$$
$$Y_L + B^{\lfloor n/2 \rfloor} Y_R = Y$$

$$Z = (X_L + B^{\lfloor \frac{n}{2} \rfloor} X_R)(Y_L + B^{\lfloor \frac{n}{2} \rfloor} Y_R)$$

$$= Z_L + Z_m B^{\lfloor \frac{n}{2} \rfloor} + Z_R B^{(\frac{n}{2})(\frac{n}{2})}$$

$$Z_L = X_L \cdot Y_L$$
$$Z_M = X_L \cdot Y_R + X_R Y_L = (X_L + X_R)(Y_L + Y_R) - Z_L - Z_R$$
$$Z_R = X_R Y_R$$

Time complexity : $O(n^{1.585})$

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

Meet in the middle:

Q. For a set of N given numbers, find the numbers of subsets having sum of elements <=X for a given X. (N <= 20)

$N = \{1,2,3\}$ (2^N) →

{} → 0
{1} → 1
{2} → 2
{3} → 3
{1,2} → 3
{2,3} → 5
{1,3} → 4
{1,2,3} → 6

Empty subset sum = 0

X = 5, ans = 7

choice for each element = 2

soln: Bitmax (1 1 0 1)

(3) (2) (1) (0)

arr[3] + arr[2] + arr[0]

Coding the solution

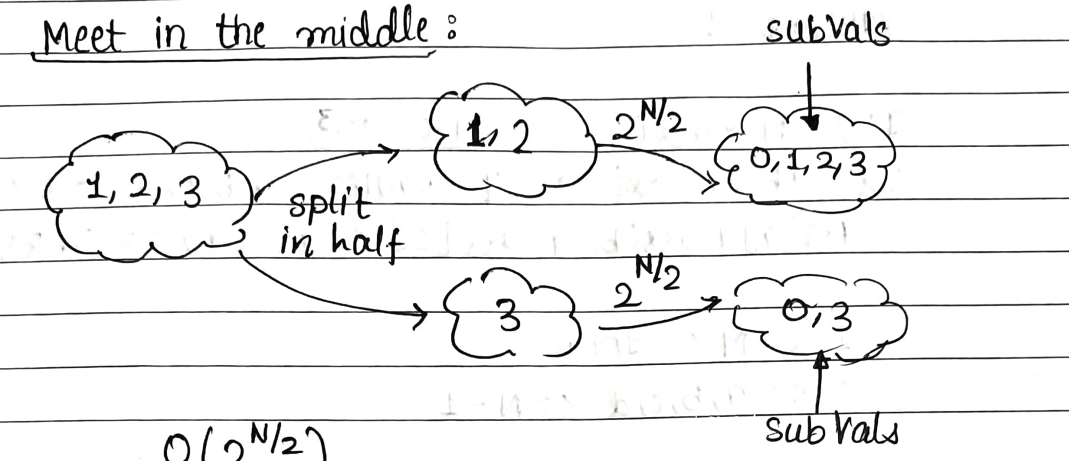
```
vector<int> generate (vector<int> arr){
    int n = arr.size();
    vector<int> subVals;

    for (int mask=0; mask<(1<<n); mask++) {
        int sum =0;

        for (int j=0; j<n; j++)
        {
            if ((mask >>j )& 1)  sum += arr[j];
        }
        subVals.push_back(sum);
    }
    sort (subvals.begin(), subvals.end()); } O(2^n log 2^n)

    return subVals;

}
```

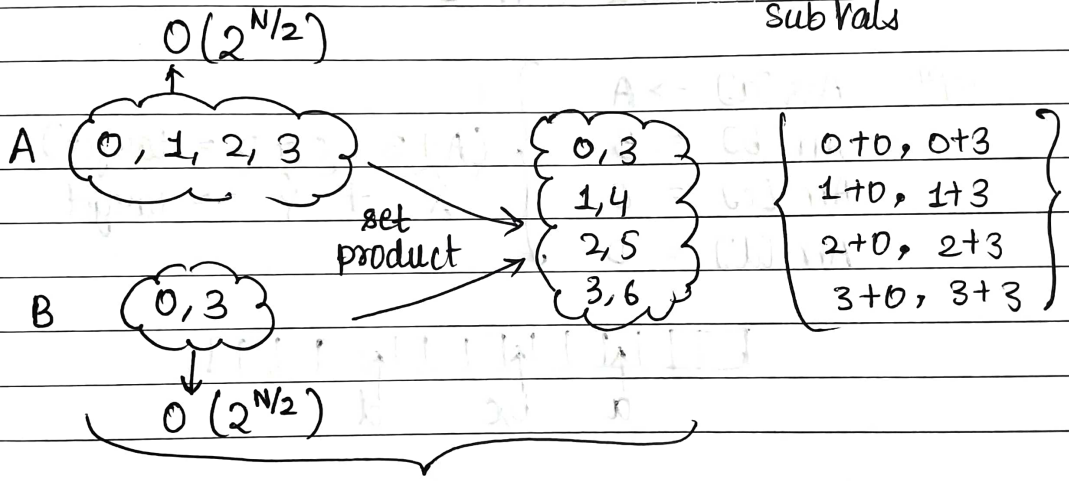
Time complexity : $O(N \cdot 2^N)$

this will pass for $N <= 20$.

$N <= 40$

Meet in the middle :





[we can use set A & set B, we can use STL to find # subset s.t sum <= k.

going through A and doing binary search on B.

Time complexity : $O(n\,2^{n/2})$

4 Sum :-

Variant 1

IP/ N : arr[] $N <= 10^3$

Does their exists 4 values a,b,c,d s.t:

arr[a] + arr[b] + arr[c] + arr[d] == TARGET

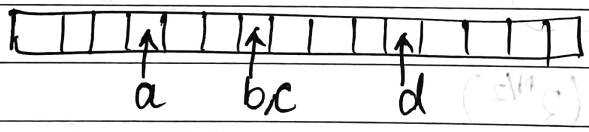
$1 <= N <= 1000$

$0 <= a,b,c,d <= N-1$

soln:

Arr[a] → A
Arr[b] = B
Arr[c] = C
Arr[d] = D

(A + B + C + D = target)
X + Y = target



Y = target − X

sum of 2 values of an array
set of a[i] + a[j] ; $0 <= i, j < N$

set of a[i]+a[j] } if sum of 2 vals in this is = target

cout << "Yes" << endl;

```
bool is4SUM_possible (vector<int> arr, int n, int target) {

    map<lli,lli> pos;      ]→ map of possibilities

    for (int i=0; i<n; i++)
    {                                          } creating map of
      for (int j=i; j<n; j++)                    all possible 2
      {                                          sums.
          pos [a[i]+a[j]] = 1;
      }
    }

    // check if 2sums can sum upto target

    for (int i=0; j<n; i++)
    {
      for (int j=0; j<n; j++)
      {
          if (pos[target-a[i]-a[j]]) return 1;
      }
    }

    return 0;
}
```

Time complexity : $O(n^2 \log n)$

Variant 2

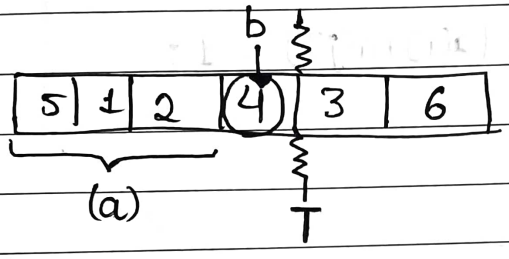
arr[a] + arr[b] + arr[c] + arr[d] == target

$4 \le N \le 1000$

$0 \le a < b < c < d \le N-1$

a, b, c, d are distinct

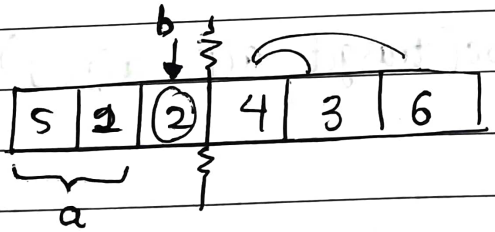
soln:

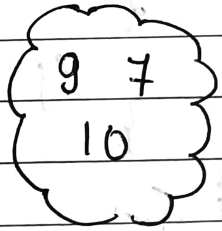


c,d pair sums

loop → check in map → if [T - (a+b)] is present or not

Shift the partition





c,d pair sums

(1) Consider b, loop A, check [T-(a+b)]

(2) c = b, loop D, add (C+D)

```
bool is4SUMpossible (vector<int> arr, int n, int target)
{
    map <lli, lli>  pos ;
    int b;
    for (b = n-2; b>= 1 ; b--)
    {
        for (a=b-1 ; a>=0 ; a--)
        {
            if ( pos [target - arr(a] - arr(b]]) {
                    return 1;
            }
        }
        int  c= b;
        for (int  d = c+1 ;  d<n ; d++)
        {
            pos[arr (c]+ ars[d] ] =1;
        }
}
 return 0;
}
```

Time complexity : $O(n^2 \log n)$

Variant 3

We can use the same process to find the 4 values as well.

Just make map<lli, pair<lli,lli>> pos;
* pos[arr[c] + arr[d]] = {c,d};
* pair<int,int> ans = pos[target - arr[a] - arr[b]];
(c,d)

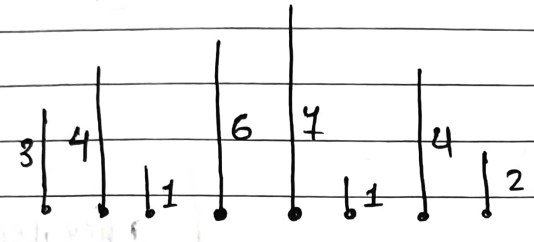
Variant 4

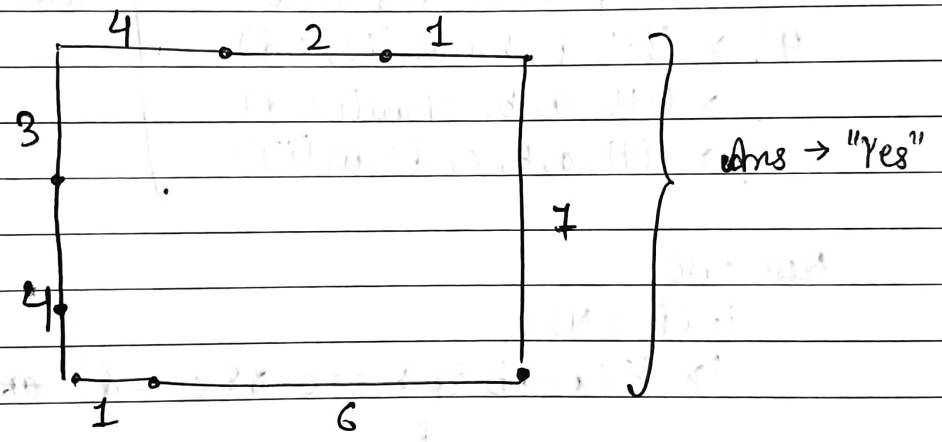
Find #(a,b,c,d) st. $\sum arr[a]$ = Target

$0 <= a < b < c < d <= N-1$

Rod Problem:

Given N rods with their lengths in array Arr[]. Can you use all the rods to combine them into bigger rods and form a square?

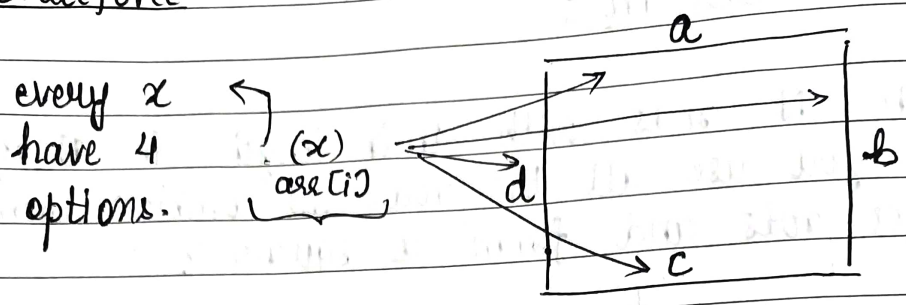




soln: obs: (1) on same side, order does not matter
(2) sq. we form, side length = total / 4.

Bruteforce

every x have 4 options.



generate (i, a, b, c, d)

- 4 ↳ (i+1, a + arr[i], b, c, d)
- ↳ (i+1, a, b + arr[i], c, d)
- ↳ (i+1, a, b, c + arr[i], d)
- ↳ (i+1, a, b, c, d + arr[i])

} generates all (a, b, c, d)

Base case

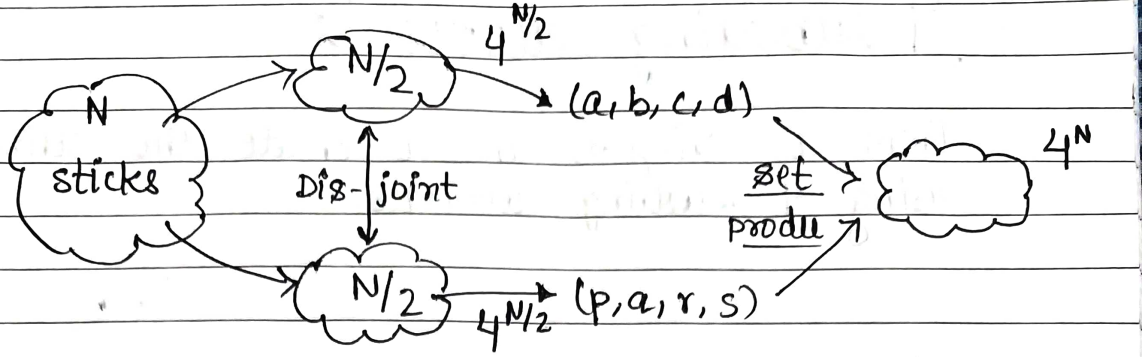
if (i == N)

↳ if (a == b && b == c && c == d) ~~an~~ return 1;

⇓

Square

But, this is $O(4^N)$



set X

(a, b, c, d) $4^{N/2}$

set Y

(p, q, r, s) $4^{N/2}$

$\left[\frac{total}{4}, \frac{total}{4}, \frac{total}{4}, \frac{total}{4}\right]$

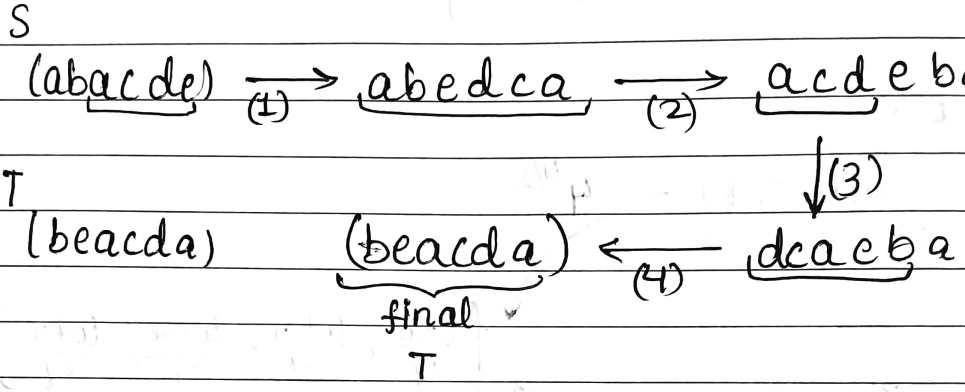
Iterate through set X & find if $\left[\frac{I}{4}-a, \frac{I}{4}-b, \frac{I}{4}-c, \frac{I}{4}-d\right]$ is present in set Y or not.

Time complexity: $O(4^{N/2} \log 4^{N/2})$

$O(N \cdot 4^{N/2})$

4 substring reversals

From a string S, can we create the string T using 4 substring reversals.

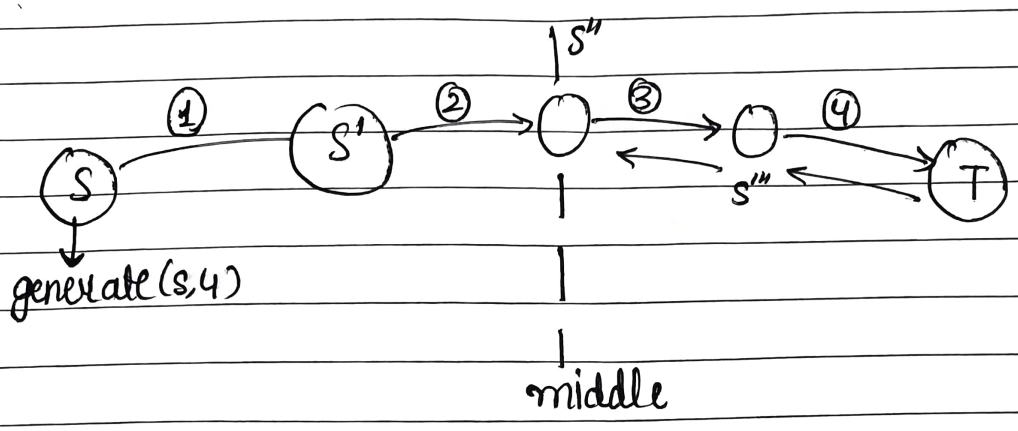



```
generate (string s, int k)

  if (k == 0)
        add s to gen;
        return;
  for all (l, r) in [1, n]:
        s' = reverse(s, l, r)
        generate (s', k-1)
```

generating all pssbl substrings.

Time complexity: $O(n^9)$

Thinking in Terms of Meet in the middle :



make move (4) & (3) on T to get s''

- create set X of strings that can be made using S
- " " Y " " " " " " " " T

(but in 2 moves)

if there is such a element exists which is there in both X and Y, then "Yes" else "No".

Now,

Time complexity $O(N^{2K} \cdot \underbrace{KN}_{\text{for reversals.}} \log N)$

$(K = 2)$.

$O(N^5 \log N)$