## ① Last Occurrence of a String

string = 'abcddedg'
0 1 2 3 4 5 6 7

Char x = 'd'

Op → 6 → last occurence of 'd'

Solution → Method -1, we can iterate over the string. Since we want the last occurence, we can start the iteration from right side.

Method -2 ⇒ We can use stl function → strrchr()

Method -3 ⇒ We will solve using recursion

Recursive Solution →

```cpp
#include <iostream>
using namespace std;

void lastOccLTR(string &s, char x, int i, int &ans){
    //base case
    if(i >= s.size()){
        return;
    }


    if(s[i] == x){
        ans = i;
    }
    lastOccLTR(s,x,i+1,ans);

}
int main()
{
    string s;
    cin>>s;
    char x;
    cin>>x;
    int ans = -1;
    lastOccLTR(s,x,0,ans);
    cout << ans << endl;

    return 0;
```

Left to Right

T.C → O(N)

S.C → O(N)

```cpp
#include <iostream>
using namespace std;

void lastOccLTR(string &s, char x, int i, int &ans){
    //base case
    if(i <0){
        return;
    }
    if(s[i] == x){
        ans = i;
        return;
    }
    lastOccLTR(s,x,i-1,ans);

}
int main()
{
    string s;
    cin>>s;
    char x;
    cin>>x;
    int ans = -1;
    lastOccLTR(s,x,s.size() -1,ans);
    cout << ans << endl;

    return 0;
}
```

Right to left

T.C → O(N)

S.C → O(N)

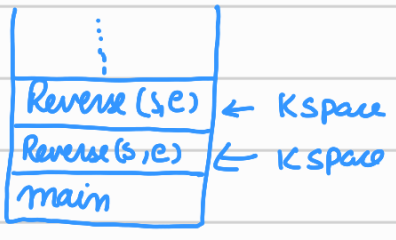
## ② Reverse a string

We will take 2 pointers one at the start & other at the end & we will swap the chars

start = 0, end = n-1,
Base case → start >= end

```cpp

#include <string>
#include <iostream>
using namespace std;
void reverse(string &s, int start, int end){
    //base case
    if(start >=end){
        return;
    }
    swap(s[start], s[end]);
    reverse(s, start+1, end-1);
}

int main()
{
    string s;
    cin >> s;
    reverse(s,0,s.size()-1);
    cout << "Ans is "<<s<<endl;
    return 0;
}
```

T.C → O(N)
S.C → O(N)



← K space
← K space

} total space of N/2+1 recursive call with each taking K space
So S.C = $O(K*(\frac{N}{2}+1)$ = O(N)

## ③ Adding Strings :-

https://leetcode.com/problems/add-strings/

We are given 2 non-negative integers num1 & num2 represented as string, return the sum of num1 & num2 as string.

I/p num1 = "11" & num2 = "123"
O/p = "134"

Solution → num1 = "456"
num2 = 77
533

↳ In normal addition, we start from One's place & carry forward the digits to

Solve the addition

We will take 2 pointers p1 & p2 which will points to single char of num1 & num2. We will also use a carry to keep track of the carry forward.

① First we will extract out integer from string as

```
int n1 = num1[p1] - 0; // "- 0" converts string into int
int n2 = num2[p2] - 0;
```

② If we look at our strings → 4 5 6 (p1) at position 2, '7' has nothing so we will add '0' there

7 7
1 0
↑ p2

Our $n_1$ & $n_2$ will become

```
int n1 = (p1 >= 0 ? num1[p1] : '0') - '0';
int n2 = (p2 >= 0 ? num2[p2] : '0') - '0';
```

Next we will find sum as

```
int csum = n1 + n2 + carry;
```

now we need the digit from csum (Ex→ 7+6 = 13, carry → 1, digit → 3)

```
int digit = csum % 10;
carry = csum / 10;
string ans = "";
ans.push_back(digit + '0');
```

T.C → O(N) , S.C → O(N) → see after → 27:00
↳ N is max length from both strings

```cpp
#include<string>
using namespace std;

class Solution {
public:

    void addRE(string &num1, int p1, string &num2, int p2, int carry, string &ans){
        //base case
        if(p1 < 0 && p2 <0){
            if(carry != 0){
               ans.push_back(carry + '0');
            }
            return;
        }

        int n1 = (p1 >=0 ? num1[p1] : '0') - '0';
        int n2 = (p2 >=0 ? num2[p2] : '0') - '0';
        int csum = n1 + n2 + carry;
        int digit = csum % 10;
        carry = csum /10;
        //string ans = "";
        ans.push_back(digit + '0');

        addRE(num1, p1-1, num2, p2-1, carry,ans);
        //return ans;

    }
    string addStrings(string num1, string num2) {

        string ans = "";
        addRE(num1, num1.size() -1, num2 ,num2.size() -1,0,ans);
        reverse(ans.begin(), ans.end());
        return ans;
    }
};
```

## (4) Palindrome Check

I/p → S = racecar

O/p → true as reverse of 'S' is also same.

Solution → we will take one pointer at start & one pointer at end & then compare if they are equal if equal increment start, decrement end. If at any point, we get inequality, then condition of palindrome is failed

```cpp
#include <iostream>
#include <string>
using namespace std;

bool isPalindrome(string &s, int start, int end){
    //base case
    if(start >= end){
        return true;
    }
    if(s[start] != s[end]){
        return false;
    }
    return isPalindrome(s, start+1, end-1);
}
int main()
{
    string s;
    cin >> s;
    cout<<isPalindrome(s,0,s.size()-1)<<endl;
    return 0;
}

```

T.C → O(N)

S.C → O(N)

## ⑤ Remove all occurences of a substring → "leetcode"

I/p → S = 'daabcbaabcbc" , part = 'abc'

O/p → S = 'daabcbaabcbc" → "dabaabcbc" → "dababc" → "dab"

(abc: remove; abc: remove; abc: remove)

Solution → we will follow 3 steps :-

1) Find part string position in 'S'.
2) Remove part from 's'.
3) Call again the function for new string

↳ First we will check whether substring is present in string or not using stl library func → Find.

```
int found = s.find(part)
```

→ If substring is not found 'nopos' will be returned.

```
if (found != string::nopos) { // if substr is found

}
```

→ If substring is found, remove substring.

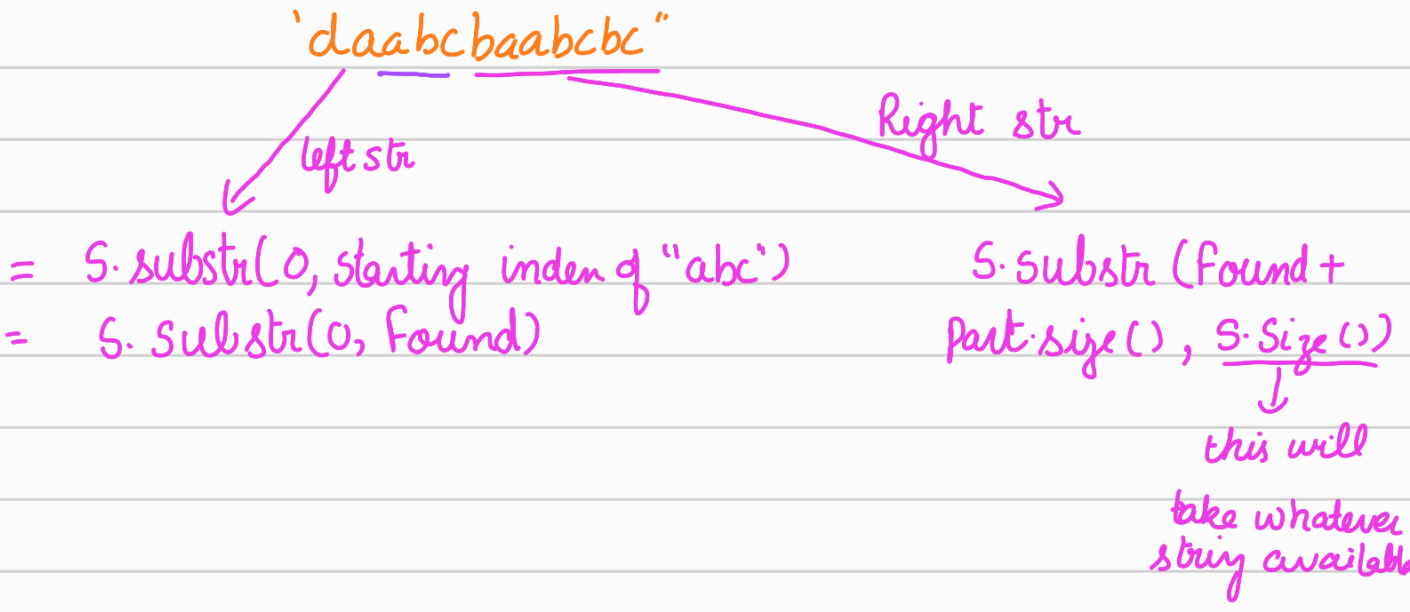
To remove substr, we can follow a method, we will take left & right string from substring & concat them

'daabcbaabcbc"

S = left string + Right string = dabaabcbc

To get left subst, we will use substr function.

String :: substr ( 0 , no. of char)
↓
start pos.
inden




Now our code will become :-

```
if ( found != String :: nopos ) {

    String left_part = S.substr (0, found);
    String right_part = S.substr (found + part.size(),
                                  S.size());

    S = left_part + right_part;

    removeOccre( s, part);
```

```cpp
#include<string>
using namespace std;

class Solution {
public:

    void removeOccre(string &s, string &part){
        int found = s.find(part);
   //find is stl function which will give us the start position of the substr

        if(found != string::npos){ //nopos means if the substr is not found

            //part string has been located
            //Please remove it

            string left_part = s.substr(0, found);
            string right_part = s.substr(found+part.size(), s.size());
            s = left_part + right_part;

            removeOccre(s, part);

        } else{
            //base case
            //all the occurences of part has been removed from s.
            return;
        }
    }
    string removeOccurrences(string s, string part) {

        removeOccre(s, part);
        return s;

    }
};
```

T.C → O(N)

S.C → O(N)

```cpp
#include<string>
using namespace std;

class Solution {
public:

    string removeOccre(string &s, string &part){
        int pos = s.find(part);
   //find is stl function which will give us the start position of the substr

         while(pos != string::npos){
             s.erase(pos,part.length());
             pos = s.find(part);
         }
         return s;
    }
    string removeOccurrences(string s, string part) {

        removeOccre(s, part);
        return s;

    }
};
```

## ⑤ Print all subarrays

I/p → arr = {1, 2, 3, 4, 5}

O/p →

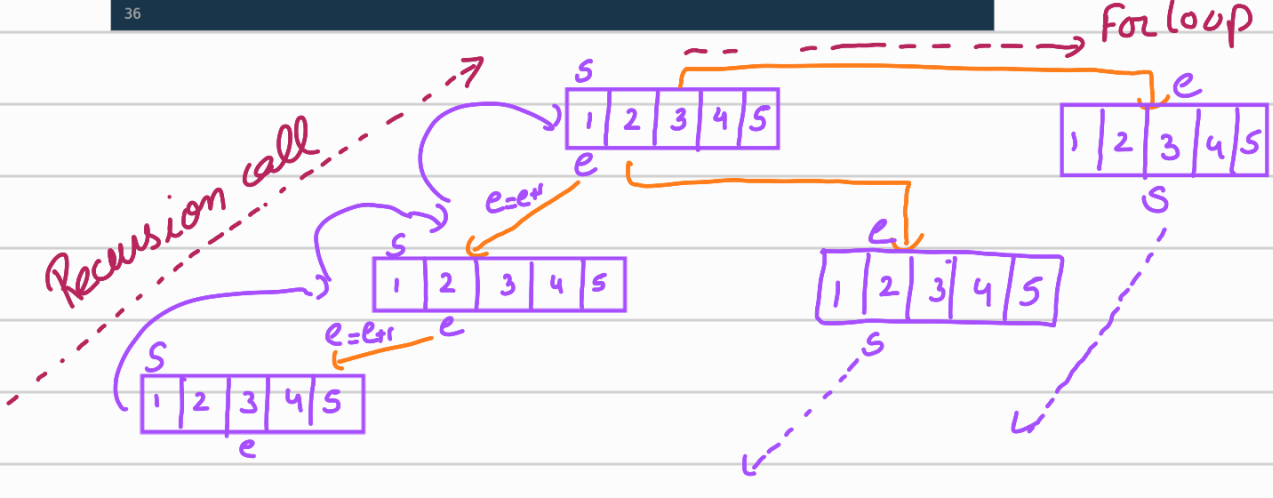
| | |
|---|---|
| 1 | 1, 2, 3, 4 ... |
| 1, 2 | 1, 2, 3, 4, 5 ... |
| 1, 2, 3 | 2, 3, 4, 5 ... |

Solution → we will take 2 pointers start and end which will change values as

① start = 0, end = start = 0
② start = 0, end = end+1 = 1
③ start = 0, end = end+2 = 2
④ start = 0, end+1 = 3

```cpp
#include <iostream>
#include <vector>

using namespace std;

void printSubarray_util(vector<int> &nums, int start, int end){
    //base

    if(end == nums.size()){
        return;
    }

    for(int i=start; i <=end;i++){
        cout << nums[i] << " ";
    }
    cout <<endl;

    printSubarray_util(nums, start, end+1);
}

void printSubarray(vector<int> &nums){
    for(int start=0; start<nums.size(); start++){
        int end = start;
        printSubarray_util(nums, start,end);
    }
}

int main()
{
    vector<int> num = {1,2,3,4,5};

    printSubarray(num);
    return 0;
}
```

T.C →



For loop will run for n times & in each time we will have n Recursive calls

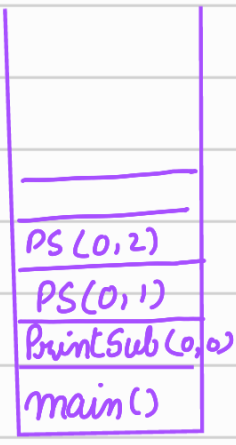
For $n=1$, $n$ Recursive call

$n=2$, $n-1$ R.C

$n=3$, $n-2$ R.C

So T.C = $O(n \times n) = O(n^2)$

Space Complexity →




## ⑥ Buy & Sell Stocks


- LeetCode

Can you solve this real interview question? - Level up your coding skills and quickly lan...

leetcode.com

I/p = prices = [7, 1, 5, 3, 6, 4]

O/p = 5 → Buy on day 2(1) & sell on Day 5(6),

profit = 6-1 = 5

Solution

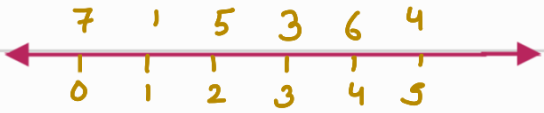
7 | 1 | 5 | 3 | 6 | 4

↓

Price[0]

↓

$i^{th}$ day → stock price




To maximise the profit, we will buy Stock when price is lowest & sell the stock when price is lower in future

We will take Variable → minPrice

7 | 1 | 5 | 3 | 6 | 4
0 1 2 3 4 5

Min price = INT_MAX , max profit = INT_MIN

if ( prices[i] < minPrice) { → If todays price less than minprice
minPrice = prices[i] update minPrice

}

if ( (prices[i] - minPrice) > maxProfit){ → If (today's price - minPrice)
maxProfile = prices[i] - minPrice; is greater than maxProfit

}

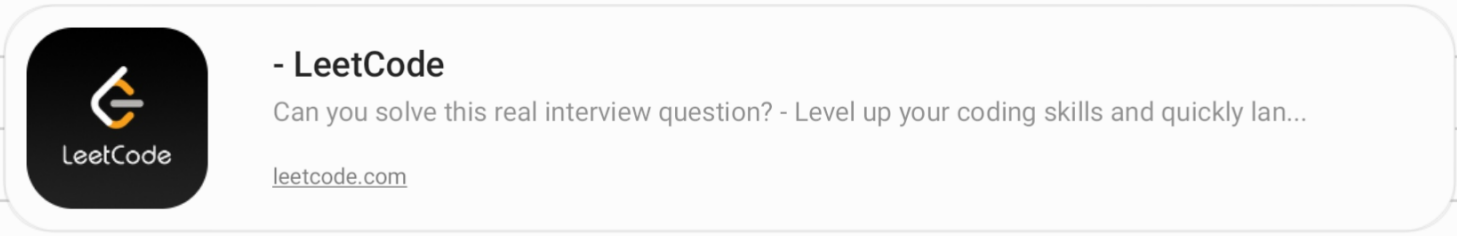
```cpp
#include<vector>

using namespace std;

class Solution {
public:

    void maxProfitFinder(vector<int>& prices, int i , int &minPrice, int &maxProfit){

        //base

        if(i == prices.size()){
            return;
        }

        if(prices[i] < minPrice){
            minPrice = prices[i];
        }

        int todaysProfit = prices[i] - minPrice;
        if(todaysProfit > maxProfit){
            maxProfit = todaysProfit;
        }

        maxProfitFinder(prices, i+1, minPrice, maxProfit);
    }
    int maxProfit(vector<int>& prices) {
        int minPrice = INT_MAX;
        int maxProfit = INT_MIN;
        maxProfitFinder(prices,0,minPrice, maxProfit);
        return maxProfit;

    }
};
```

T.C → O(N)

S.C → O(N)

Iterative solution is better one

# ⑦ → House Robbery Problem :-

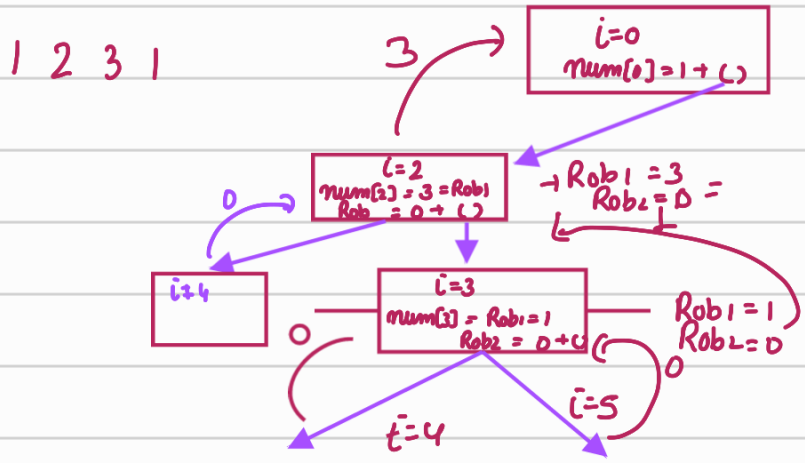



Input = nums = [1, 2, 3, 1]

Opp = 4. → Rob house 1 (money = 1) & then rob house 3 (money = 3). Total amount we can rob = 1+3 = 4

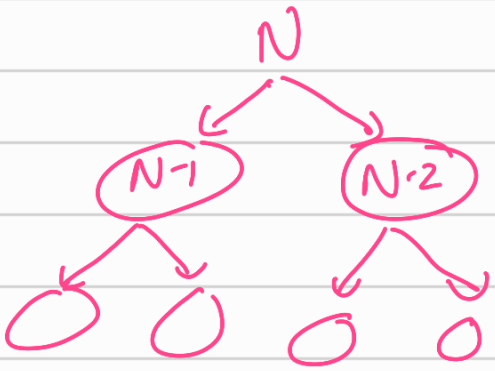
Solution → We can either rob current home or forget current home & move to next home

```cpp
#include <vector>
using namespace std;

class Solution {
public:

    int robHelper(vector<int>& nums , int i) {
        if(i >= nums.size()){
            return 0;
        }

        int robAmt1 = nums[i] + robHelper(nums, i+2); //chossing adjacent house
        int robAmt2 = 0 + robHelper(nums, i+1);

        return max(robAmt1, robAmt2);
    }
    int rob(vector<int>& nums) {
        return robHelper(nums, 0);

    }
};
```




The above solution will give us T.L.E as it's not optimized. To improve it we will use Dynamic programming

T.C →




→ Same as Fibonacci we are making 2 calls at a place.

So our T.C is $O(2^n)$

S.C → max-depth in our memory stack = $O(n)$

## ⑧ Integer to English word :-


- LeetCode

Can you solve this real interview question? - Level up your coding skills and quickly lan...

leetcode.com

Input → 123 → Op → 'One hundread twenty three'

Solution → The number system goes like

1 → one , 10 → ten , 100 → hundred

1000 - thousand , 10,000 → ten thousand

100000 → hundred thousand , 1000000 → one million

$10^9$ → Billion

Now if we have 123456

→ we will call this one hundread twenty three thousand

So we will need some base words also

1- one , 2- two , 3- three , 4- Four , 5- Five

. . . . . 10 - ten , 11 - eleven , 12 - twelve . . . . 19 - ninteen

20 - twenty , 30 - thirty , 40 , 50, 60, 70, 80, 90, 100 - hundread,

1000, $10^6$ - million , $10^9$ - Billion

So after the base words we can break any digit as

123 → 100 → One hundread
20 → twenty
3 → three

Now we will find out in the collection of base words which is smaller than 123 → it's 100
so we will divide 123/100 = 1.23 ≈ one.
so our first word will be one hundread

(123 % 100)
rem is 23 ⇒ 23 is just greater than 20, this time we will not divide by 20 (as it's smaller than 100)
so string is One hundread twenty
rem 23 % 20 ⇒ 3 → three

String → One hundread twenty three

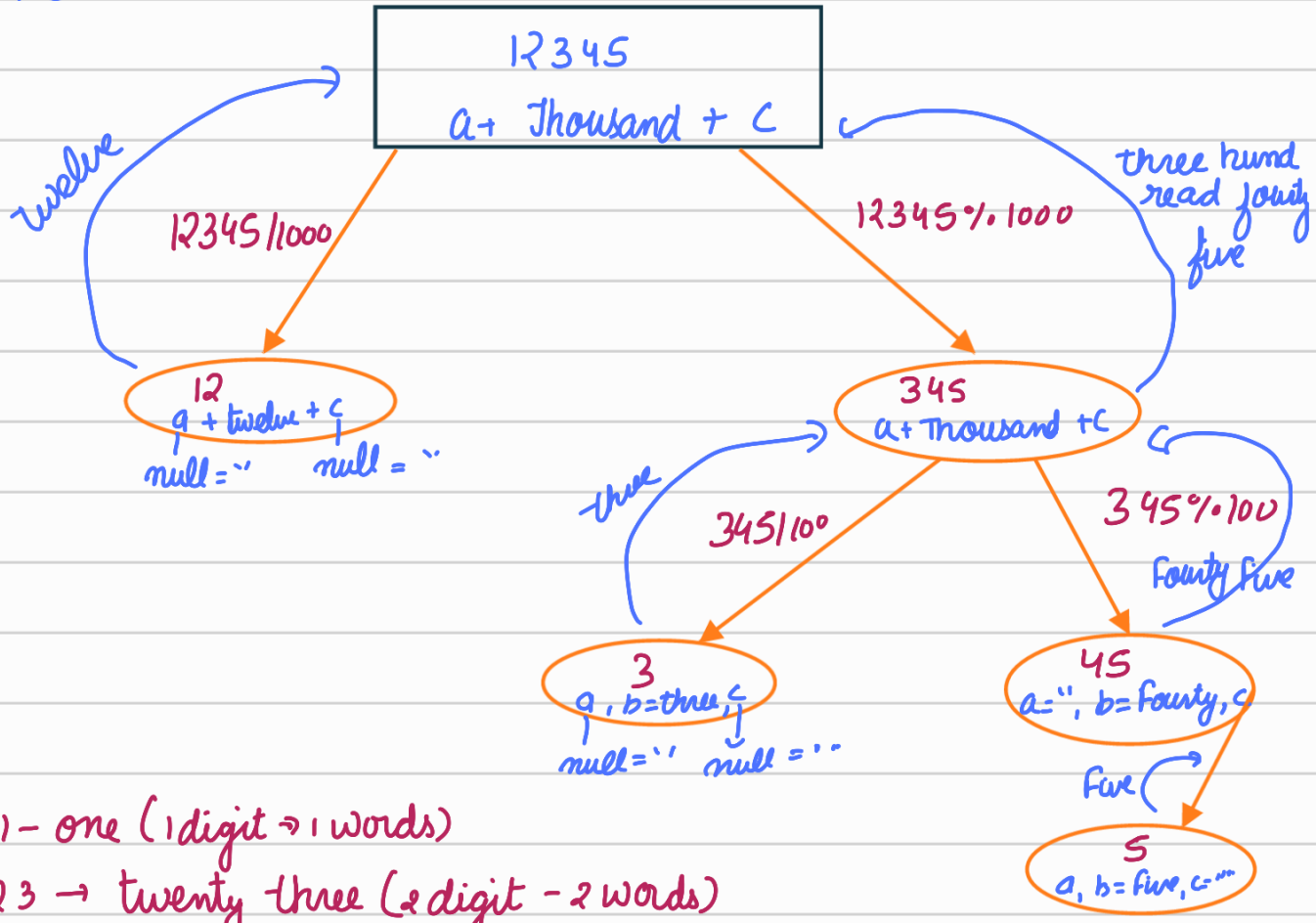
So the steps are :-

① Find N just >= In the map
(N >= map.Num)
if (N ≥ 100)
↳ How many map.Num = (N / map.num)

② N = N % Map.num
→ 123 % 100 = 23

```cpp
#include <vector>
#include <string>

using namespace std;

class Solution {
public:

    vector<pair<int, string>> mp = {{1000000000, "Billion"}, {1000000, "Million"}, {1000, "Thousa
  "}, {100, "Hundred"},{90, "Ninety"},{80, "Eighty"},{70, "Seventy"},{60, "Sixty"},{50, "Fifty"},{
   "Forty"},{30, "Thirty"},{20, "Twenty"},{19, "Nineteen"},{18, "Eighteen"},{17, "Seventeen"},{16,
  Sixteen"},{15, "Fifteen"},{14, "Fourteen"},{13, "Thirteen"},{12, "Twelve"},{11, "Eleven"},{10, "
  "},{9, "Nine"},{8, "Eight"},{7,  "Seven"},{6, "Six"},{5, "Five"},{4, "Four"},{3, "Three"},{2, "Tw
  },{1, "One"}};

    string numberToWords(int num) {
        if(num==0){
            return "Zero";
        }

        for(auto it:mp){
            if(num >= it.first){
                string a = "";

                if(num >= 100){
                    a = numberToWords(num/it.first) +" ";
                }
                string b = it.second;
                string c = "";
                if(num % it.first != 0){
                    c = " " + numberToWords(num % it.first);
                }
                return a+b+c;
            }
        }

        return "";
    }
};
```

En→ 12345 twelve thousand three hundread fourty five




T.C→ 1- one (1 digit → 1 words)

23 → twenty three (2 digit - 2 words)

123 → One hundread twenty three (3 digit = 4 words)

12345 = 5 digit = 6 words

TC ∝ No. of words , No. of words ∝ no. of digits

1 digit can be represented by < 2 words

⑨ Wild card matching :-


- LeetCode

Can you solve this real interview question? - Level up your coding skills and quickly lan...

leetcode.com

We have a string 's' & pattern 'p'. we have to match s with p.

'?' → denotes single char

'*' → denotes any no. of chars [0, ∞]

Eg → s = 'aa' , p = 'a'

s - a a
p - a ] → not same, return False

② s = 'aa' p = '*'

Here s = aa , p = * → Any no. of chars.

↳ return true

③ s = 'aa' p = '?a'

s - a a
p - ? a ] → ? single char which can be a → return true

④ S → aa , p = 'aa*'

S → [a] [a]

p → [a] [a] * → here '*' is extra & it can be any char so it can be empty, hence return true

⑤ S = **ab

p = ab → return true as ** can be empty

⑥ S = ab?d

p = abcc → a will match with a, b will match with b, ? & c will match as ? can be c, d & c will not match → return false

⑦ S = abcdefg

p = ab*fg → a match with a, b match with b, in S we have (cde) & p has (*) & * can be cde (any no. of chars), f match f, g match g → return true

In above example we have to take care of '*', as it can be any no. of 'chars', so, insted of cde, it can also be cdefg, then in that case, we wont have any char left in S to match with fg of p

S → [a] [b] [c d e f g]

p → [a] [b] [*] f g → no chars left in S to match → return false

Solution :-

Few rules we will follow :-

① Out of all comparisons
If any comparison return true, then ans will be true

S = a b c d e f g
P = a b * f g

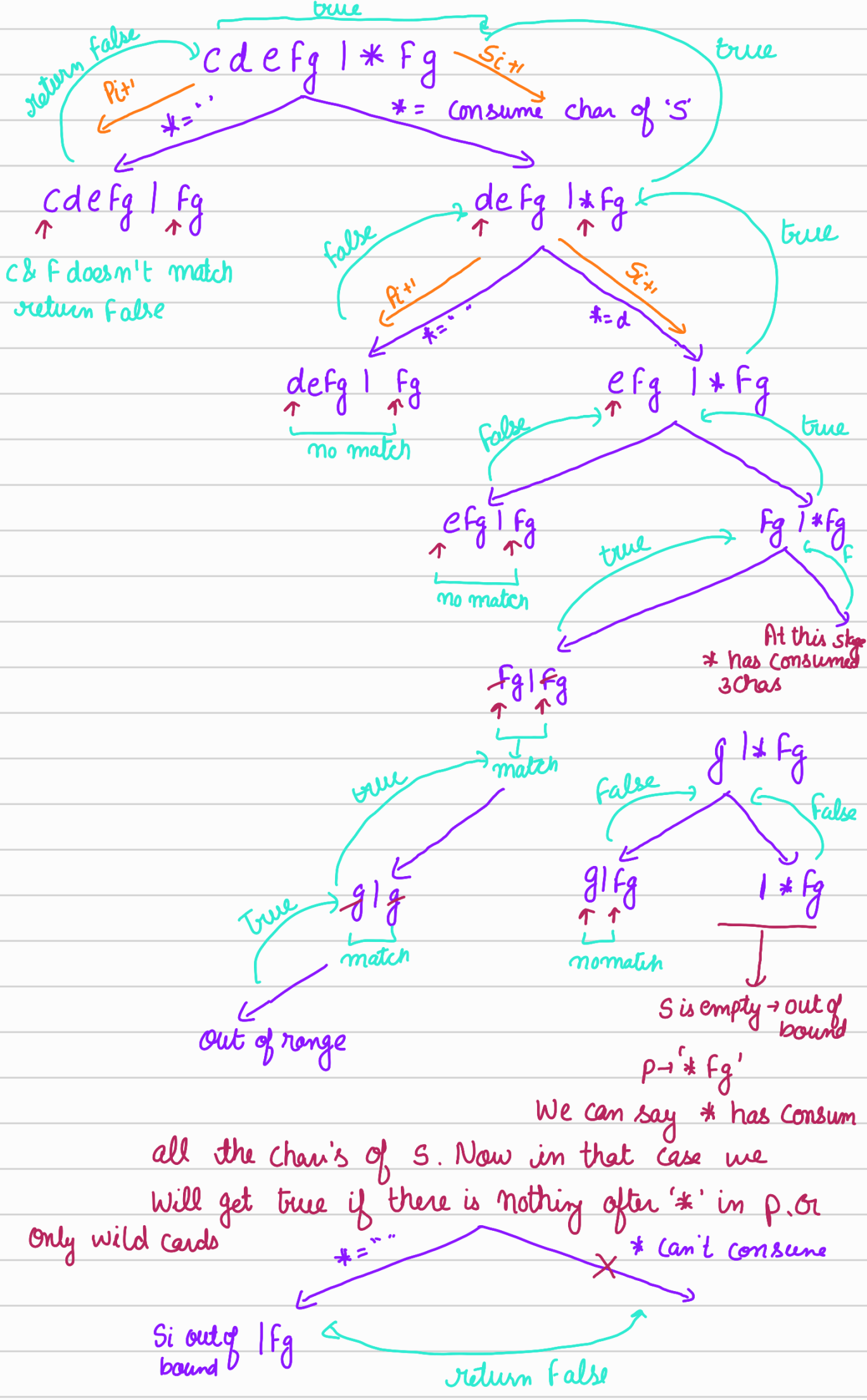
so if ( s[si] == p[pi] || p[pi] = '?' ) {
//Match
Call → to check pi+1, si+i
}

above condition is S[0] = a = p[0] ~ Matches
now check for i+1, S[1] = b = S[1]

if (p[i] == '*' ) {
bool caseA = isMatchHelper( s, si, p, pi+1);
bool caseB = isMatchHelper( s, si+1, p, pi);
return caseA || caseB;
}

We have cdefg | *fg
↑ s ↑ p

For 'c' & '*' we have two choises, either * is "" empty string or consume * with char of s ie 'c'


true
return false
cdefg | *fg
Si+1
Pi+1
*=''
* = consume char of 'S'
true
cdefg | fg
c & f doesn't match
return false
defg | *fg
false
Pi+1
*=''
Si+1
*=d
true
defg | fg
no match
efg | *fg
false
true
efg | fg
no match
fg | *fg
true
At this step * has consumed 3 chas
fg | fg
match
true
g | *fg
false
false
True
g | g
match
g | fg
no match
| *fg
Out of range
S is empty → out of bound
p → '* fg'
We can say * has consum
all the chan's of S. Now in that case we
Will get true if there is nothing after '*' in p. Or
Only wild cards
*=''
* can't consume
Si out of bound
| fg
return false

```cpp
#include <string>
using namespace std;

class Solution {
public:

    bool isMatchHelper(string &s, int si, string &p , int pi){
        //base
        if(si == s.size() && pi == p.size()){
            return true;
        }

        if(si == s.size() && pi < p.size()){
            while(pi < p.size()){
                if(p[pi] != '*') return false;
                pi++;
            }
            return true;
        }


        //single char matching
        if(s[si] == p[pi] || '?' == p[pi]){
            return isMatchHelper(s,si+1,p,pi+1);
        }

        if(p[pi] == '*'){
            //treat '*' as empty or null
            bool caseA = isMatchHelper(s,si,p,pi+1);

            //let '*' consume one char.
            bool caseB = isMatchHelper(s,si+1, p, pi);

            return caseA || caseB;
        }

        //char doesn't match
        return false;
    }
    bool isMatch(string s, string p) {
        int si = 0; //pointer index for s string
        int pi = 0; //pointer index for p string

        return isMatchHelper(s,si,p,pi);

    }
};
```

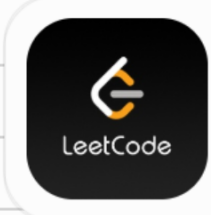
* Note → this will give us TLE in leet code as we have to use the Dynamic Programming

T.C → The recursion tree of the problem is same is Fibonacci series i.e exponential.

T.C → $O(2^m)$

SC → Max. depth of memory stack = $S = O(n)$

(6) → Perfect square sum :-




- LeetCode

Can you solve this real interview question? - Level up your coding skills and quickly lan...

leetcode.com

1, 4, 9, 16 are perfect squares → $\sqrt{4} = 2$, $\sqrt{9} = 3$, $\sqrt{16} = 4$

3 & 11 are not perfect sq → $\sqrt{3} \rightarrow 1.732$.

I/p = 12 → we will have to use perfect squares to reach 12

12 = 4 + 4 + 4 → no. of ps = 3

(Perfect square)

12 = 1 + 1 + 1 + 1 + - - 1 → no. of ps = 12

We need least no of ps = min(3, 12) = 3

O/p = 3

Ex-2 → I/p = 13

Case 1 → 1 + 1 + 1 ... + 1 = ps = 13

Case 2 → 4 + 4 + 4 + 1 → ps = 4

Case-3 → 4 + 1 + 1 + 1 ... + 1 = ps = 10

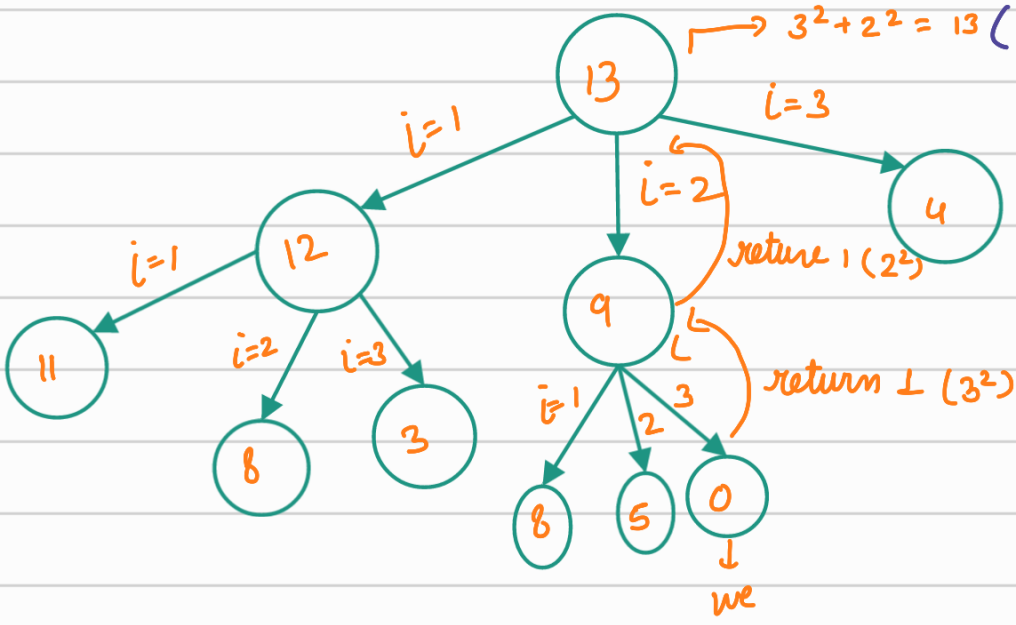
Case-4 → 4 + 9 = ps = 2

→ O/P = 2

Solution

```cpp
//LOVE BABBAR

#include <limits.h>
#include <cmath>
using namespace std;
class Solution
{
public:

    int numSquareHelper(int n){
          //base
        if(n == 0) return 1;

        if(n<0) return 0;

        int i =1;
        int ans = INT_MAX;
        int end = sqrt(n);
        while(i <= end){
            int perfectSquare = i*i;
            int noOfPerfectSquare = 1 + numSquareHelper(n - perfectSquare);
            if(noOfPerfectSquare < ans){
                ans = noOfPerfectSquare;
            }
            i++;
        }

        return ans;
    }
    int numSquares(int n) {

        return numSquareHelper(n) -1;
    }
};
```

+ will throw TLE error.




NumsquareHelper will return 3.

& Numsquare will return NumsquareHelper - 1 = 2 = ans

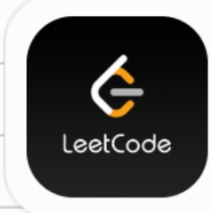
T.C → Each node is divided in square root of 'N'

$T.C \rightarrow (\sqrt{n})^n$

$O(N) = O(n^n)$

S.C → $O(N)$

(11) Minimum cost for tickets :-




- LeetCode

Can you solve this real interview question? - Level up your coding skills and quickly lan...

leetcode.com

Input : days - [1, 4, 6, 7, 8, 20] , cost = [2, 7, 15]

Output : 11

Explanation : On day 1, we can bought 1 day pass for 2$, cost[0] = 2

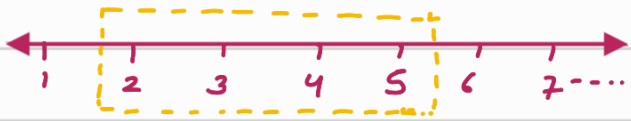
On day 3, we bought a 7-day pass for cost[1] = $7, which covers 3, 4, 5, 6, 7, 8, 9

On day 20, we bought 1 day pass = $2

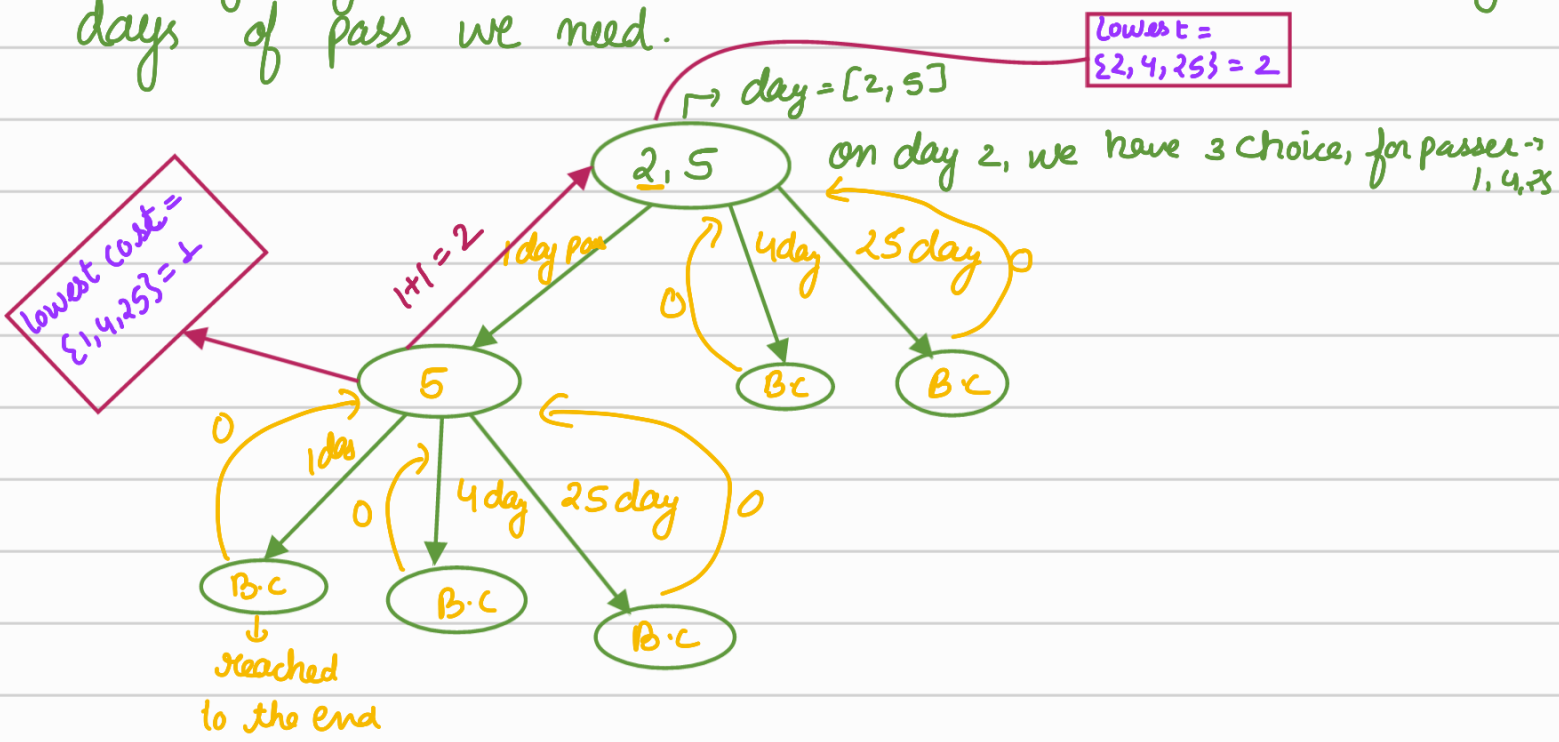
Total = 2 + 7 + 2 = 11

Solution → day - [2,5] | cost = [1, 4, 25]

day1 day2 day 30




at any day we will have choice to choose for how many days of pass we need.




```cpp
#include<vector>
#include<math.h>
using namespace std;

class Solution {
public:

    int mincostTickets_helper(vector<int>& days, vector<int> & costs,int i){
        //base case
        if(i >= days.size()) return 0;

        //sol for 1case
        //1 days pass taken
        int cost2 = costs[0] + mincostTickets_helper(days,costs,i+1);

        //7 Days pass taken
        int passEndDay = days[i] +7 -1;
        int j = i;
        while(j < days.size() && days[j] <= passEndDay){
            j++;
        }
        int cost7 = costs[1] + mincostTickets_helper(days,costs,j);

        //30 Days pass taken
        passEndDay = days[i] +30 -1;
        j = i;
        while(j < days.size() && days[j] <= passEndDay){
            j++;
        }
        int cost30 = costs[2] + mincostTickets_helper(days,costs,j);

        return min(cost2, min(cost7,cost30));

    }

    int mincostTickets(vector<int>& days, vector<int>& costs) {
       return mincostTickets_helper(days, costs,0);

    }
};
```

Will throw T.L.E error

Time Complexity → Since from one branch we are giving 3 branches so T.C will be $O(3^n)$

↳ exponential

S.C → $O(N)$

## (12) Number of Dice Roll with Target Sum :-


- LeetCode

Can you solve this real interview question? - Level up your coding skills and quickly lan...

leetcode.com

We have 'N' no. of dice with 'K' faces with targe sum(t)

$N = 3, \quad K = 6 \quad t = 12$

No. of ways we can get target sum as 12

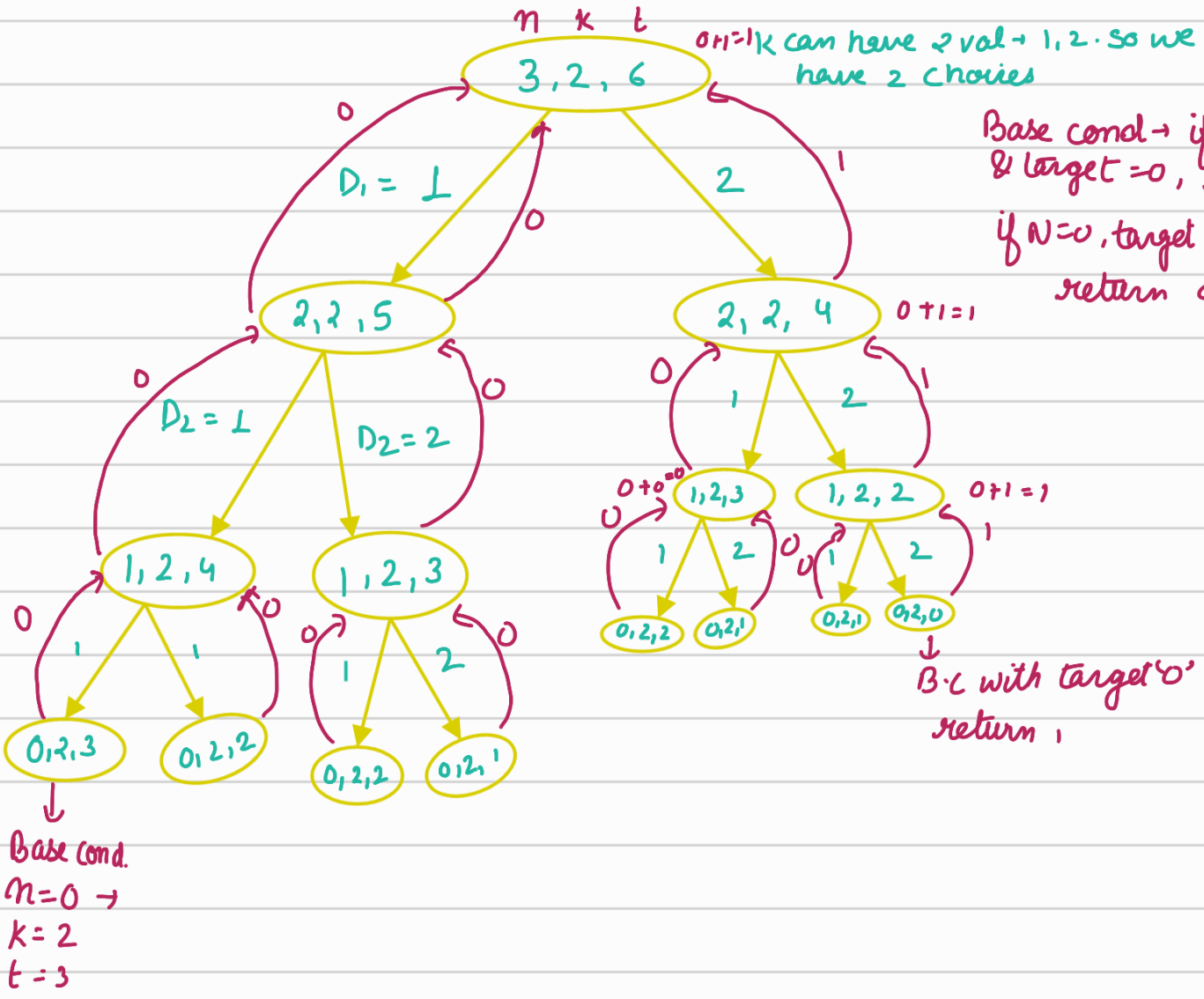
$D_1 = 1, \; D_2 = 5, \; D_3 = 6$

| | | |
|---|---|---|
| 2 | 5 | 5 |
| 3 | 6 | 3 |
| 2 | 6 | 4 |

Solution → Suppose $N=3, \quad K=2, \quad t=6$

(K=1,2)

| $D_1$ | $D_2$ | $D_3$ | Target Sum | |
|---|---|---|---|---|
| 1 | 1 | 1 | 3 | ✗ |
| 1 | 2 | 1 | 4 | ✗ |
| 2 | 2 | 2 | 6 | ✓ |




So We have 1 way for above recursive tree

```java
class Solution {
public:
    int numRollsToTarget(int n, int k, int target) {
        //base case
        if(target < 0) {
            return 0;
        }
        if(n==0 && target ==0) {
            return 1;
        }
        if(n==0 && target !=0) {
            return 0;
        }
        if(n !=0 && target ==0) {
            return 0;
        }
        int ans=0;
        for(int i=1; i<=k;i++){
            ans = ans + numRollsToTarget(n-1,k,target-i);
        }
        return ans;
    }
};
```

Will give us T.L.E

T.C → $O(k^n)$ , S.C → $O(n)$