2.

DIVIDE AND CONQUER

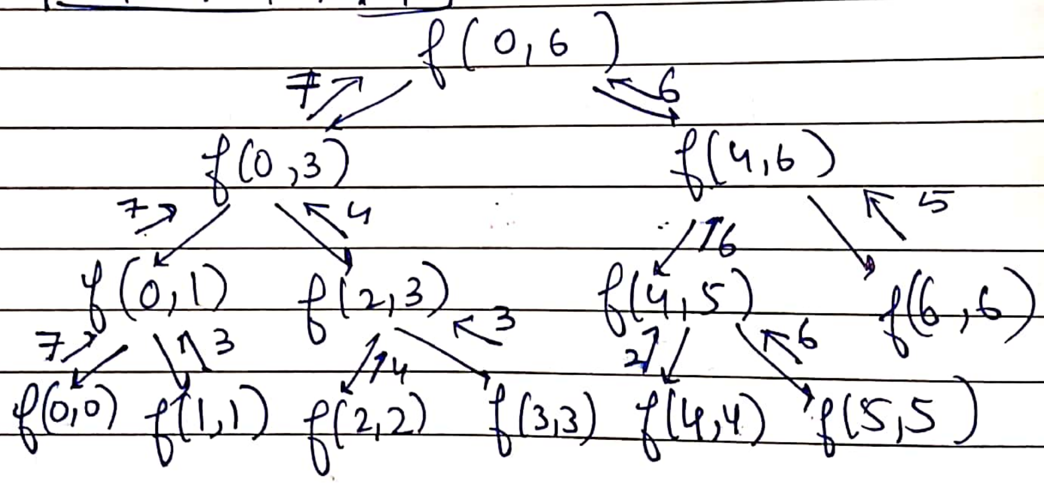
Divide a problem into subparts and then moving further.

Exp find max in array. using Recursion.

```
max (int i, int j)
{
    if (i == j ){f(i,j) = arr[i];}

    f(i,j) = max (f(1, mid ), f(mid+1, j)),
```

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



Date / /

```
int findMax( int i, int j, int arr[])
{
    if(i==j)
        return arr[i];

    int m= (i+j)/2;

    int m1= findmax( i, m, arr[]) ;
    int m2= find max( m+1, j, arr[]);

    return max(m1, m2);
```

Time Complexity $= O(N)$

At each function call $O(1)$ work is done i.e $max(m_1, m_2)$; and at last it is return i.e $O(1)$ & number of function calls is the total

$$N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} \dots \approx 2*N \quad \text{i.e } O(N)$$

Question Given N, K find N^K

```
int ans=1
for (I <- 0 to K-1)
    ans = ans* K;
ret ans;
```

func(int n, int k)

x= fun(n, k/2) fu(n, k/2)

Same

so calculate once

& do x*x

N^k : $N^{K/2} * N^{K/2} \rightarrow$ even

termination cond = K=1,0

$N * N^{K/2} * N^{K/2} \rightarrow$ odd

edge case (if even.) ↳ don't need to calc.

if odd, then not

N^{20} ↑ $N^{10} * N^{10}$
↓ ↑ $N^5 * N^5$
N^{10}
↓ ↑ $N * N^2 * N^2$
N^5
↓ ↑ $N * N$
N^2
↓ ↑ $1 \times 1 \times N$
N^1
↓ ↑ 1
N^0

Time complexity $= O(\log_2 k)$

at each func. there is $O(1)$ work done.

```
int pow(int n, int k)
{
    if (k==0) ret 1;
    int x = pow (n, k/2);
    if (k % 2 = = 0) ret x*x;
    else    return x* x* N;
}
```

if $(k \% 3 == 0)$

N^k →
- $N/3 * N/3 * N/3$
- $N/3 * N/3 * N/3 * N$
- $N/3 * N/3 * N/3 * N * N$

```
if (k==0) ret 1
int x = pow(n, k/3);
~~else~~ if
if (k % 3 = = 0)
        ret x*x*x;
elsif ( = =1) ret  ~~x*x*~~x * N;
else            ret  " " " * N*N;
```

Here number of levels will be $\log_3 k$

no. of levels will be less than $\log_2 k$.
But here at each level more multiplications are happening so for higher values of k. Then
$\log_3 k * 2 > \log_2 k$

Breaking into 2 parts is always better hence.

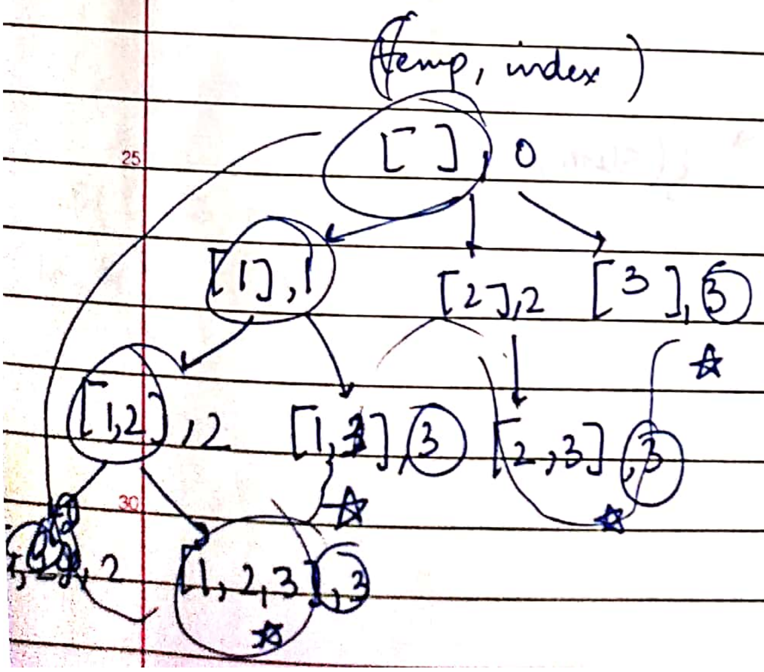
Ques All subsets of set. print in lexicographic order.

[1, 2, 3]

[ ], [1], [1, 2], [1, 2, 3], [1, 3] [2] [2,3] [3]

[ ] < [1] < [2] < [3]
^
[1, 2]
^
[1, 3]

whenever you're fixing current element 'i', Then all other elements should come from i+1, n-1



no. of func. calls = 2^N

Date / /

```
void lexicSS( temp [], sz , i )
{
    print temp ; \n.

    if (i == n)
        return;

    for( j = i → N-1 )
    {
        temp [sz] = arr[j] ;
        lexicSS( tmp, sz+1, j+1 );
    }
```

Ques

Arr [N] int SUM

Count no. of subsets with a given sum

for [2, 3, 4, 1] ↳ 4, (3, 1)

op 2

sum → req. sum

$$f(sum, 0)$$

$$\xleftarrow{Y} \qquad \xrightarrow{N}$$

$$f(sum - a[0], 1) \qquad\qquad f(sum, 1)$$

remainingSum //start with original sum.

```
int func (remSum, index)
{
        if (index == n)
        {
                if (index == 0)
                        return 1;
                else  return 0;

        x = func (remSum-a[index], index+1);
        y = func (remSum, index+1);
        return x+y;
}
```

Question: Int Arr[N] , distinct elements +ve

any element can be taken any number of times

SUM → Given

find how many different combinations will be there with sum == SUM

Example: Arr [1,2]

Sum : 4

combinations → 1, 1, 1, 1

1, 1, 2

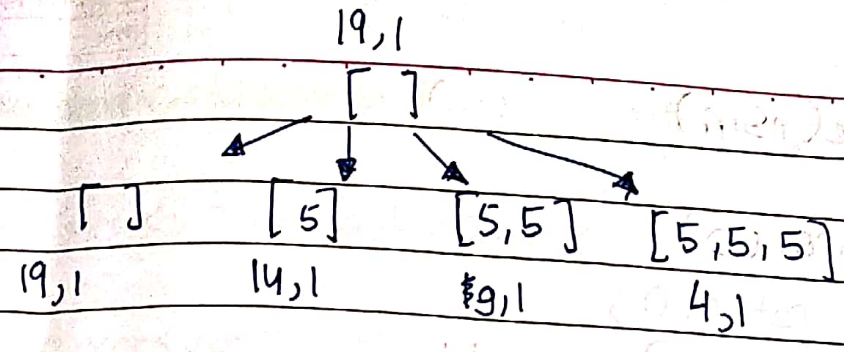
2, 2

Sum = 19   [5 | 1 | ...]

no. of times 5 could be included =

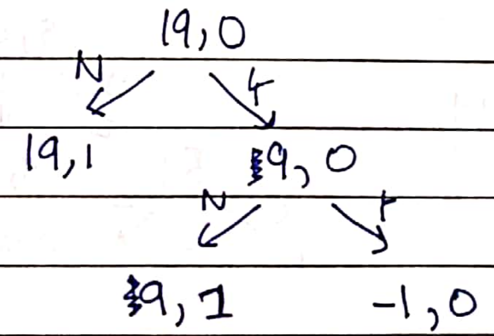
SUM/a[i]   for 5 → 19/5 = 3

{0, 1, 2, 3 times



int func (remSum, index)
{
    if (index == N)
    {   if (remSum == 0) return 1;
        else return 0;
    }
    int ~~ans = 0~~; Sum int ans = 0;
    for (j = 0 to j ≤ (remSum/arr[5]))
    {
        ~~// sum = sum + arr[5]~~;
        val = func (remSum - ~~sum~~ j*a[i], index+1);
        ans += val;   j*arr[i]
    }
    return ans;
}

Different Approach



if you take an element change the sum & index is same, if you donot take it sum will remain same & index will be incremented.

```
int func(rs, i)

    if ( rs < 0)
        return 0;
    if ( i == N )
    {
        if (rs == 0)
        {   return 1 }
        retun 0
    }
    func ( rs, i+1);
    func( rs - ar[i], i);
```

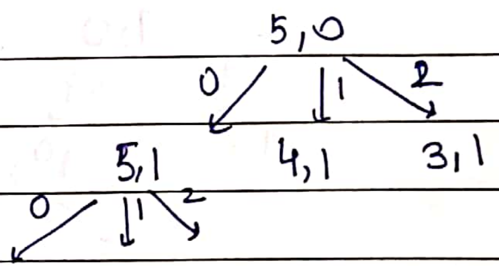
Ques Find no. of distinct combinations SUM*
arr[N] has multiple enteries

[2, 2, 2, 1, 3]   combinations = [2,2] , [2,2] [2,2]...
[1,3], [2,2].

OUTPUT → 2.

[2 1 2 1 3 4 3]

[1 1 2 2 3 3 4] sort



create freq array

ele	freq
1	2
2	2
3	2
4	1

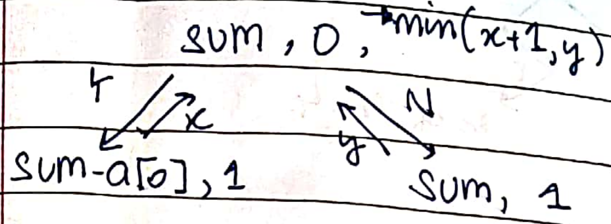
Becomes same as previous

ques Length of smallest subset whose sum = Given sum

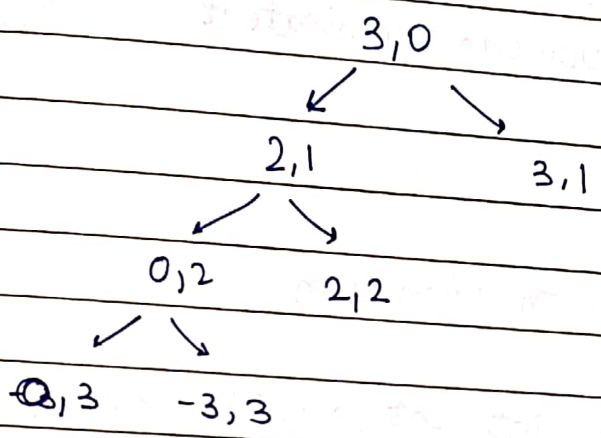
I/P = [5 1 0.4 0.9 1.7 -1]

sum = 4

O/P = 2 → [5, -1]



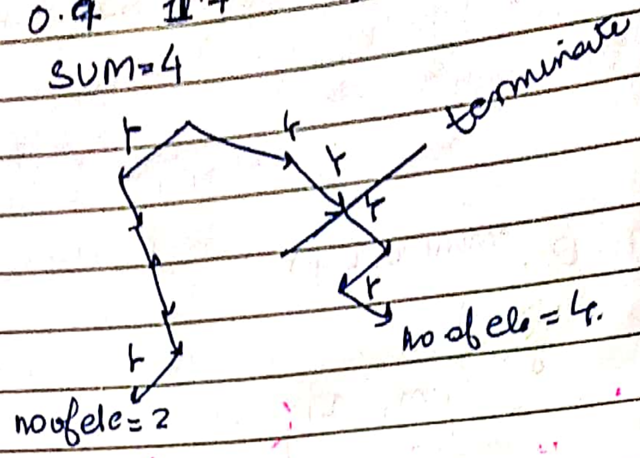
Example: arr = [1, 2, 3]   sum = 3




```
long minss (rS, i)
{
    if (i==N)
    {
        if (rS != 0) return integer.MAX_VALUE
        return 0;
    }
    long x = minss(rs-ar[i], i+1);
    long y = minss (rs, i+1);
    return min(x+1, y);
}
main()
{
    if (ans >= integer.MAXVALUE)  no. subset
}
```

[5 1 0.4 0.9 1.7 -1]

SUM=4



you can keep a track of current best ~~element~~ answer so whenever the another branch is over That answer then you can terminate it

→ Recursion with Pruning ←

Ques- Smallest subset with given sum

```
void
func ( int &ans , int cnt , vec<int>&vec ,int i,
                                    int rsum)
{
      if ( cnt >= ans) return;
      if (i == vec.size())
      {
           if (rsum != 0) ret;
           else {
                ans = cnt; return; } }
      }
inclusion →  func( ans, count+1, vec, i+1, rsum- vec[i]) ;
exclusion →  func( ans , count , vec, i+1, rsum);
}
```