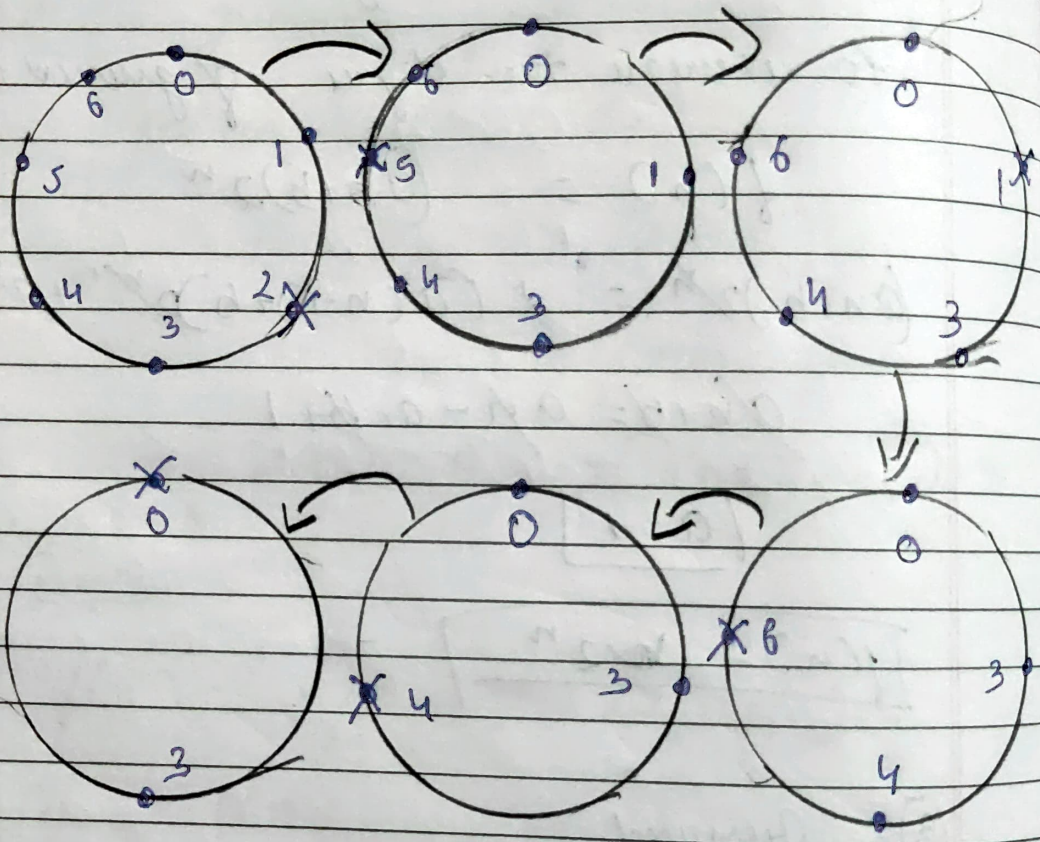
8/03/22

Recursion

Josephus problem

I/P = n=7, K=3

output = 3



Final survivor : 3

```
int jos (int n, int K)
{ if (n==1)
    return 0
  else
    return (jos(n-1), K)+k)%n
}
```

$T(n) = T(n-1) + c^0$

$\Theta(n)$

$\Rightarrow f(n) = \alpha^n$

$\alpha^n = \alpha^{n-1}$

$\alpha^n - \alpha^{n-1} = 0$

$\alpha - 1 = 0$

$\boxed{\alpha = 1}$

$\Rightarrow f(n) = c_1 \alpha_1^n + c_2 \cdot n\alpha_2^n$

$f(n) = c_1 + c_2 n\alpha^n$

$f(1) = 0$

$f(0) = 0$

$0 = c_1 + c_2\alpha$

$c_1 = 0$

$c_2 = 0$

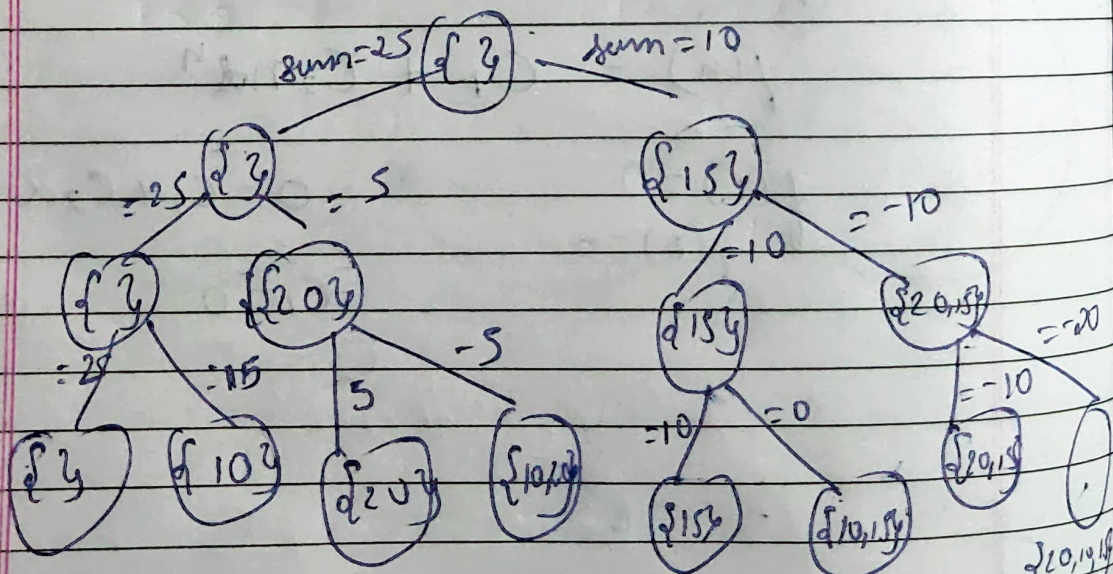
Subset sum problem

Return the number of subsets where sum is equal to a given value.

I/p : {10, 5, 2, 3, 6}
sum = 8
O/p = 2.

Subsets = $2^n = 2^5$ subsets are produced.
{5, 3}, {6, 2}

Example {10, 20, 15}, sum = 25.



```
int countsubsets (int arr[], int n, int sum)
{
    if (n==0)
        return (sum==0) ? 1:0;

    return countsubsets (arr, n-1, sum) +
           countsubsets (arr, n-1, sum-arr[n-1]);
}
```

$$T(n) = \Theta(2^n) //$$

→ Disadvantages of recurrence :

* Takes a lot of time as all the functions remain in the stack until the base condition of the func is reached.

* Greater time requirements because of functional calls and returns overhead.

→ Advantages :

* Clean and simple way to write code.

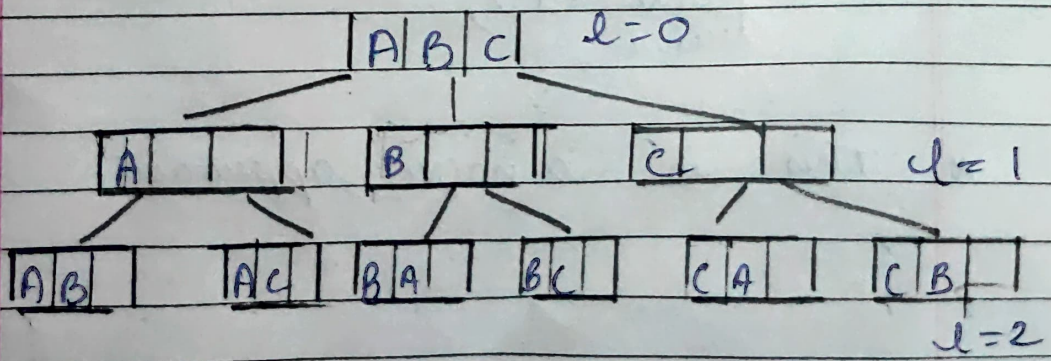
Backtracking

Q. Given a string, Print all the permutation which do not contain "AB" as a substring.

I/P: Str = "ABC"
O/P: ACB, BAC, BCA, CBA.

So basically, backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

Naive approach for the above mentioned question would be to generate all the possible set of combinations and then selecting the ones which are needed.



(l = starting index)
(r = ending index)

```
void permutation (string str, int l, int r)
{
  if (l==r)
    cout<<a<<endl;

  else
{
  for(int i=l; i<=r; i++)
{
    swap(a[l], a[i]);
    permute(a, l+1, r);
    swap(a[l], a[i]);
}
}
}
```

To print only at substring, i++

```
if (str.find("AB")==string::npos)   // not found.
    print(str);
    return
```

* This is a naive approach.

⇒ Adding backtracking solution to the naive solution

```
bool issafe(String str, int l, int i)
{
if (l!=0 && str[l-1] == 'A' && str[i] == 'B')
    return false;

if (r==l+1 && str[i] == 'A' && str[l] == 'B')
    return false;

}
```

This function can be used to enclose the lines of code in the for-loop to make a safe check if "AB" occurs or not.

⇒ Time Complexity

Depth of the recursive tree * no. of permut - calls

= n * n!

$T(n) = O(n * n!)$ => naive.

Backtracking is better than naive.