1- Generate All Binary Strings of Length N

0 & 1

→ 2 → 010

→ 3 → 011

→ 5 → 101

n=3

2^3 ⇒ 8 2^3-1 ⇒ 7

0 → 000 ✓
1 → 001 ✓
2 → 010 ✓
3 → 011 ✓
4 → 100 ✓
5 → 101 ✓
6 → 110 ✓
7 → 111 ✓

~~8 →~~ 1000 bits

n=4

2^4-1 ⇒ 15

2^5-1

31

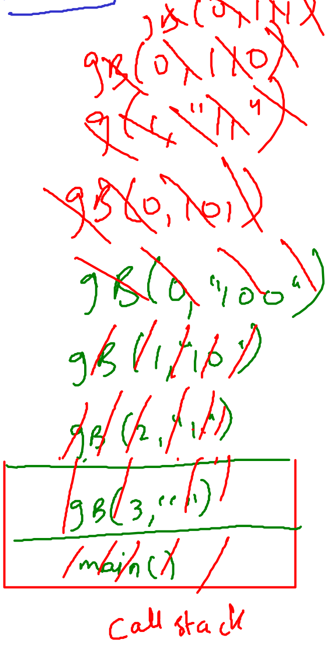
① Base condition →

(0 1)

n=3
2
1
0 → return output

0 & 1

gB(n-1, "0").
gB(n-1, "1")



gB(3,"")

```
void generateBinary(int n,string out){
  if(n==0){
     cout<<out<<endl;
     return;
  }
 generateBinary(n-1,out+"0");
 generateBinary(n-1,out+"1");
}
```

2, "1"

1, "11"

1, 10

4/10

```
void generateBinary(int n,string out){
  if(n==0){
     cout<<out<<endl;
     return;
  }
 generateBinary(n-1,out+"0");
 generateBinary(n-1,out+"1");
}
```

1, "11"

0 101

0 100

0 110

```
void generateBinary(int n,string out){
  if(n==0){
     cout<<out<<endl;
     return;
  }
  generateBinary(n-1,out+"0");
  generateBinary(n-1,out+"1");
}
```

0 110

0 "100"

0, 101

0, 111

```
void generateBinary(int n,string out){
  if(n==0){
     cout<<out<<endl;
     return;
  }
  generateBinary(n-1,out+"0");
  generateBinary(n-1,out+"1");
}
```

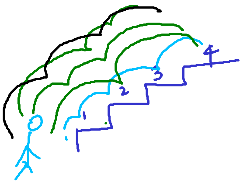
output
000
001
010
001
100
101
110
111

not

2- Count Ways to Climb Stairs

. 1 step

2 step .



3 4

1 & 2 .

n=4

1+1+1+1

2+2

1+1+2

2+1+1

1+2+1

5 combinations

① Base case

(n=0 , n=1) return 1

cw(n-1) + cw(n-2)

↓ 1step ↓ 2step

n=4 n=0

4

```
int countWays(int n){
   if(n==0 || n==1)
      return 1;
   return countWays(n-1)+countWays(n-2);

}
```

3 cw(2)

3+2

3

3

```
int countWays(int n){
   if(n==0 || n==1)
      return 1;
   return countWays(n-1)+countWays(n-2);

}
```

cw(1)

2+1

2 .

2

```
int countWays(int n){
   if(n==0 || n==1)
      return 1;
   return countWays(n-1)+countWays(n-2);

}
```

cw(1) cw(0)

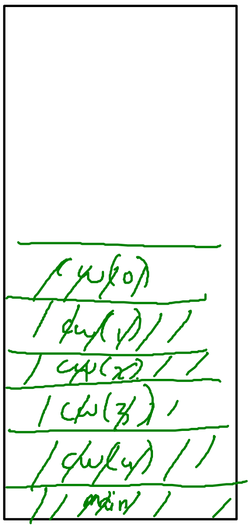
2 1 + 1

0

```
int countWays(int n){
   if(n==0 || n==1)
      return 1;✓
   return countWays(n-1)+countWays(n-2);

}
```

5



Call Stack

3- Print All Balanced Parentheses Combinations

() → BP

f() ∝ BP

n=3

()()()

(())()

(()())

()(()())

((()))

5 combinatis

((( )))

0 0 0 cc c

(3—3)

Balanced P

open>0 => (

close>open => ()

(())

3 3

3>0✓ (

2>0✓ ((

1>0✓ (((

0>0✗

3>0 ((()

2>0 ((())

1>0 $\boxed{((()))}$ → Base → return //

✗0>0

logical

bp(3,3,"")

3 3 ""

```
void balancedParentheses(int open, int close,string out){
 if(open==0 && close==0){
   cout<<out<<endl;
   return;
 }
if(open>0) balancedParentheses(open-1,close,out+"(");
 if(close>open) balancedParentheses(open,close-1,out+")");
}
```

3>0✓ bp(2,3,"(")

2 3 "("

```
void balancedParentheses(int open, int close,string out){
 if(open==0 && close==0){
   cout<<out<<endl;
   return;
 }
if(open>0) balancedParentheses(open-1,close,out+"(");
 if(close>open) balancedParentheses(open,close-1,out+")");
}
```

2>0 bp(1,3,'((')

1 3 ((

```
void balancedParentheses(int open, int close,string out){
 if(open==0 && close==0){
   cout<<out<<endl;
   return;
 }
if(open>0) balancedParentheses(open-1,close,out+"(");
if(close>open) balancedParentheses(open,close-1,out+")");
}
```

1>0 bp(0,3, "(((")

(3>1)✓ bp(1,2,(()) –

0 3 (((

```
void balancedParentheses(int open, int close,string out){
 if(open==0 && close==0){
   cout<<out<<endl;
   return;
 }
if(open>0) balancedParentheses(open-1,close,out+"(");
if(close>open) balancedParentheses(open,close-1,out+")");
}
```

0>0✗

3>0✓ bp(0,2,((())

0 2 ((()

```
void balancedParentheses(int open, int close,string out){
 if(open==0 && close==0){
   cout<<out<<endl;
   return;
 }
if(open>0) balancedParentheses(open-1,close,out+"(");
if(close>open) balancedParentheses(open,close-1,out+")");
}
```

0>0

✓2>0 bp(0,1,"((())")

0 1 ((())

```
void balancedParentheses(int open, int close,string out){
 if(open==0 && close==0){
   cout<<out<<endl;
   return;
 }
if(open>0) balancedParentheses(open-1,close,out+"(");
if(close>open) balancedParentheses(open,close-1,out+")");
}
```

0>0

1>0✓ bp(0,0,"((()))")

5 combinations

```
void balancedParentheses(int open, int close,string out){
 if(open==0 && close==0){
   cout<<out<<endl;
   return;
 }
 if(open>0) balancedParentheses(open-1,close,out+"(");
 if(close>open) balancedParentheses(open,close-1,out+")");
}
```

```
void balancedParentheses(int open, int close,string out){
 if(open==0 && close==0){
   cout<<out<<endl;
   return;
 }
 if(open>0) balancedParentheses(open-1,close,out+"(");
 if(close>open) balancedParentheses(open,close-1,out+")");
}
```

Output

((()))

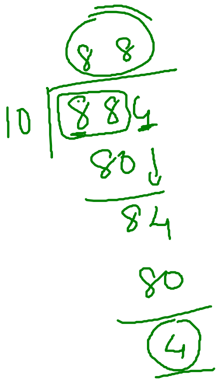
→ 4

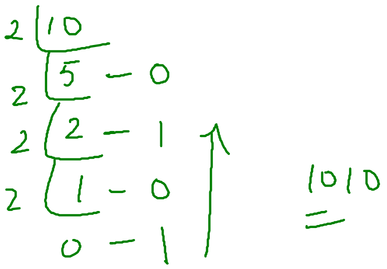
4- Convert a Decimal Number to Binary Using Recursion ✓ Lecture

2,3,4, - . 0 & 1

10 => 1010

4 -> 100





num = 10

num => num/10

rem => num%10

n==0 return

1) BC

2) CTF ✓ Tail recursion.

3) P

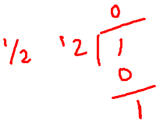
dB(10)

```
void decimalToBinary(int n){
   if(n==0) return;
   decimalToBinary(n/2);   dB(5)
   cout<<n%2;
}
```

```
void decimalToBinary(int n){
   if(n==0) return;
   decimalToBinary(n/2);   dB(2)
   cout<<n%2;
}
```

```
void decimalToBinary(int n){
   if(n==0) return;
   decimalToBinary(n/2);  dB(1)
   cout<<n%2;
}
```

```
void decimalToBinary(int n){
   if(n==0) return;
   decimalToBinary(n/2);  dB(0)
   cout<<n%2;
}
```



```
void decimalToBinary(int n){
   if(n==0) return;
   decimalToBinary(n/2);
   cout<<n%2;
}
```

```
void decimalToBinary(int n){
   if(n==0) return;
   decimalToBinary(n/2);
   cout<<n%2;
}
```

1010 //

Lecture

Number

BTD , DTB

HW : BTD

100 => 4

5- Count Number of Ways to Express N as Sum of 1, 3, and 4

n = 5

1) 1+1+1+1+1

2) 1+1+3

3) 1+4

4) 3+1+1

5) 1+4

6) 1+3+1

6 combinations.

2nd

n=0 (1)

n<0 (0)

$cw(n-1) + cw(n-3) + cw(n-4)$.

6

```
int countSumWays(int n){        5
   if(n==0) return 1;
   if(n<0) return 0;              4 + 1. + 1.
   return countSumWays(n-1)+countSumWays(n-3)+countSumWays(n-4);
}              cw(4)                  cw(2)              cw(1)
```

main => 6

4

```
int countSumWays(int n){        4
   if(n==0) return 1;
   if(n<0) return 0;              2 + 1 + 1
   return countSumWays(n-1)+countSumWays(n-3)+countSumWays(n-4);
}              cw(3)                  cw(1)    +   cw(0)
```

2

```
int countSumWays(int n){        3
   if(n==0) return 1;
   if(n<0) return 0;            1  + 1 + 0
   return countSumWays(n-1)+countSumWays(n-3)+countSumWays(n-4);
}            cw(2)
```

1

```
int countSumWays(int n){        2
   if(n==0) return 1;
   if(n<0) return 0;              1 + 0 + 0
   return countSumWays(n-1)+countSumWays(n-3)+countSumWays(n-4);
}            cw(1)             cw(-1)              cw(-2)
                                  0
```

1

```
int countSumWays(int n){        1
   if(n==0) return 1;
   if(n<0) return 0;
   return countSumWays(n-1)+countSumWays(n-3)+countSumWays(n-4);
}         (1) cw(0)      (0) cw(-2)         cw(-3)   0
```

0

```
int countSumWays(int n){       -3  0  -2
   if(n==0) return 1; ✓
   if(n<0) return 0;
   return countSumWays(n-1)+countSumWays(n-3)+countSumWays(n-4);
}
```

0

6- Count Number of Digits in a Number

I/p num => 1045
o/p => 4

num = 10
o/p => 2

cD(1045)

1+(n/10)

4

```
int countDigits(int n){
  if(n==0) return 0;
  return 1+countDigits(n/10);
}
```

1045

1+ cD(104) => 3

3

```
int countDigits(int n){
  if(n==0) return 0;
  return 1+countDigits(n/10);
}
```

104

1+ cD(10)

4 //

2

```
int countDigits(int n){
  if(n==0) return 0;
  return 1+countDigits(n/10);
}
```

10

1+ cD(1)

1

```
int countDigits(int n){
  if(n==0) return 0;
  return 1+countDigits(n/10);
}
```

1

1 + cD(0)

1+0 = 1

0

```
int countDigits(int n){
  if(n==0) return 0;
  return 1+countDigits(n/10);
}
```

0

7-Sum of Digits of a Number

(cs)

n: 1234
1+2+3+4 => 10 //

n = 123$\boxed{4}$

$\boxed{\%10}$

n => n/10

recursion

n = = 0

n = 123
rem = n%10 => 3
sum = 0 + $\boxed{3}$
n = n/10 (12)

n = 12
rem = 2
sum = 5
n = 1

n = 1
rem = 1
sum = 6
n = 0 return 0

127

```
int sumDigits(int n){
   if(n==0) return 0;
   return (n%10)+sumDigits(n/10);
}
```

7+ sum (12)

.3

3

12

```
int sumDigits(int n){
   if(n==0) return 0;
   return (n%10)+sumDigits(n/10);
}
```

2+ sum (1)

.1

10/1

127 1

1

1

```
int sumDigits(int n){
   if(n==0) return 0;
   return (n%10)+sumDigits(n/10);
}
```

1 + sum (0)

.0

0

0

```
int sumDigits(int n){
   if(n==0) return 0;
   return (n%10)+sumDigits(n/10);
}
```

8-Product of Digits of a Number -- Homework

n=123

1*2*3 = $\boxed{6}$ //

9- Reverse a Number Using Recursion

(C³)

n=1234

o/p:) 4321

rem:) n%10 => 4

rev => rev×10+ rem

=> 0+4 => 4 //

n=n/10 (123)

n=123

rem= 3

rev => 4×10+ 3

=> 40+3=43

n=12

n: 12

rem=2

rev = 43×10+ 2

=> 430+2

=> 432

n=1

n=1

rem=1

rev =432×10+1

$\boxed{\Rightarrow 4321}$

n=0

BC.

1234

```
int reverseNumber(int n,int reverse=0){
   if(n==0) return reverse;
   return reverseNumber(n/10,reverse*10+(n%10));
}
```

rN (123, 4)

123 4

```
int reverseNumber(int n,int reverse=0){
   if(n==0) return reverse;
   return reverseNumber(n/10,reverse*10+(n%10));
}
```

rN (12, 43)

12 43

```
int reverseNumber(int n,int reverse=0){
   if(n==0) return reverse;
   return reverseNumber(n/10,reverse*10+(n%10));
}
```

rN (1, 432)

1 432

```
int reverseNumber(int n,int reverse=0){
   if(n==0) return reverse;
   return reverseNumber(n/10,reverse*10+(n%10));
}
```

rN (0, 4321)

0 4321

```
int reverseNumber(int n,int reverse=0){
   if(n==0) return reverse;
   return reverseNumber(n/10,reverse*10+(n%10));
}
```

=> 4321

10-Count Zeros in a Number

n = 102040

o/p. 3//

Dry run -> H/w.

int count = (n%10 == 0) ? 1 : 0

count + cZ(n/10)

11- Count Number of Times a Digit Occurs in a Number -> H/w

12-Check if a Number is a Power of 2 Using Recursion

n=0 false

n=1 -true

n=4 -> true

n=3 -> false

2^0 -> 1

2^1 -> 2

2^2 -> 4

2^3 -> 8

n==0 false

n==1 true

(n%2 != 0) false n = 2 / 2

n = 16 / 2

n = 8 / 2

n = 4 / 2

n = 1 -> true

n=17 => 1

(n%2 != 0) ✓ return 1

8 7

```
bool isPowerof2(int n){
  if(n==0) return false;
  if(n==1) return true;
  if(n%2!=0) return false;
  return isPowerof2(n/2);
}
```

p(4)

4

```
bool isPowerof2(int n){
  if(n==0) return false;
  if(n==1) return true;
  if(n%2!=0) return false;
  return isPowerof2(n/2);
}
```

p(2)

2

```
bool isPowerof2(int n){
  if(n==0) return false;
  if(n==1) return true;
  if(n%2!=0) return false;
  return isPowerof2(n/2);
}
```

r(1)

true

8

1

```
bool isPowerof2(int n){
  if(n==0) return false;
  if(n==1) return true;
  if(n%2!=0) return false;
  return isPowerof2(n/2);
}
```

```
bool isPowerof2(int n){
  if(n==0) return false;
  if(n==1) return true;
  if(n%2!=0) return false;
  return isPowerof2(n/2);
}
```

```
bool isPowerof2(int n){
  if(n==0) return false;
  if(n==1) return true;
  if(n%2!=0) return false;
  return isPowerof2(n/2);
}
```

14- Print sum from 1 to n.

n=5

1+2+3+4+5 ⇒ 15 //

(n-1) (n-1) n

$\boxed{n+(n-1)}$

4+(n-1)

15

⇒

```
int sumOfN(int n){
  if(n==1) return 1;
  return n+sumOfN(n-1);
}
```

5 + sum(4)

10

6

```
int sumOfN(int n){
  if(n==1) return 1;
  return n+sumOfN(n-1);
}
```

3+ sum(2)

1

```
int sumOfN(int n){
  if(n==1) return 1;
  return n+sumOfN(n-1);
}
```

10

```
int sumOfN(int n){
  if(n==1) return 1;
  return n+sumOfN(n-1);
}
```

4+ sum(3)

6

3

```
int sumOfN(int n){
  if(n==1) return 1;
  return n+sumOfN(n-1);
}
```

2 + sum(1)

```
int sumOfN(int n){
  if(n==1) return 1;
  return n+sumOfN(n-1);
}
```

14- Calculate nth Fibonacci number

1st 2nd

1 1 2 3 5 8 13 21 34 . . . . .

3 4 5 6 7 8 9

n=1
n=2 } 1

$\boxed{(n-1)+(n-2)}$

n=8

21

6 → 8

8

```
int fibonacci(int n){
  if(n==1 || n==2) return 1;
  return fibonacci(n-1)+fibonacci(n-2);
}
```

f(5)

5 + 3

3

```
int fibonacci(int n){
  if(n==1 || n==2) return 1;
  return fibonacci(n-1)+fibonacci(n-2);
}
```

f(3) 2 + 1

1 1

```
int fibonacci(int n){
  if(n==1 || n==2) return 1;
  return fibonacci(n-1)+fibonacci(n-2);
}
```

5

```
int fibonacci(int n){
  if(n==1 || n==2) return 1;
  return fibonacci(n-1)+fibonacci(n-2);
}
```

f(4)

3 + 2

2

```
int fibonacci(int n){
  if(n==1 || n==2) return 1;
  return fibonacci(n-1)+fibonacci(n-2);
}
```

f(2) + f(1)

1+1

```
int fibonacci(int n){
  if(n==1 || n==2) return 1;
  return fibonacci(n-1)+fibonacci(n-2);
}
```