Relation to Induction:
Prove: $2^0 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1$
Base Case:
k=0, $2^0 = 2^1 - 1$
Inductive Hypothesis:
If $2^0 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1$ is true,
then we can prove that $2^0 + 2^1 + 2^2 + \ldots + 2^{k+1} = 2^{k+2} - 1$
Inductive Step:
$2^0 + 2^1 + 2^2 + \ldots + 2^{k+1} = (2^0 + 2^1 + 2^2 + \ldots + 2^k) + 2^{k+1} = 2^{k+1} - 1 + 2^{k+1} = 2^{k+2} - 1$

Think about what is the subproblem: recursive step
Think about the base case

Splitting the problem into the last and the rest:
Factorial

```
int fact(int n)
{
    if (n == 0) return 1;
    return n * fact(n-1);
}
```

Splitting the problem into the first and the rest:

```
void printArrayInOrder(const double a[], int n)
{
   if (n == 0)
         return;
   cout << a[0] << endl;
   printArrayInOrder(a + 1, n-1);
}
```

Splitting the problem in halves:
MergeSort
a = {4, 3, 5, 2, 1, 8, 7, 6} MergeSort(a, 0, 8)
MergeSort(a, 0, 4) MergeSort(a, 4, 8) Merge(a, 0, 4, 8)
MergeSort(a, 0, 2) MergeSort(a, 2, 4) Merge(a, 0, 2, 4)
MergeSort(a, 4, 6) MergeSort(a, 6, 8) Merge(a, 4, 6, 8)

Creating and destroying stack frames: More time consuming.
Limited addresses for stack calls. Stack overflow
Use iterative approach if possible.
Iterative solutions come from recursive ones.
Iterative merge sort. But the idea comes from the recursion.
Dynamic programming in CS180. Memoization

The way of choosing a subproblem matters.
exponential

```
int expon(int a, int b)
{
    if (b == 0) return 1;
    return a * expon(a, b-1);
}

int expon_fast(int a, int b)
{
    if (b == 0) return 1;
    if (b % 2 == 0){
        int m = expon_fast(a, b/2);
        return m * m;
    }else{
        return a * expon_fast(a, b-1);
    }
}
```

Recursion help you to come up with a solution

Fibonacci:
fib(n) = fib(n-1) + fib(n-2)
Time consuming
Iteration or Memorization

Step:
Go either 1 or 2 steps. How many ways to go n steps?
step(n) = step(n-1) + step(n-2)

```
int solveParade(int n)
{
    if(n == 1) return 2;
    if(n == 2) return 3;
    return solveParade(n-1) + solveParade(n-2)
}
P(n) = F(n)+B(n)
F(n) = P(n-1)
B(n) = F(n-1)

A::print() B::print()  A::print()
A::print() B::print()  B::print()
ABBABB
```