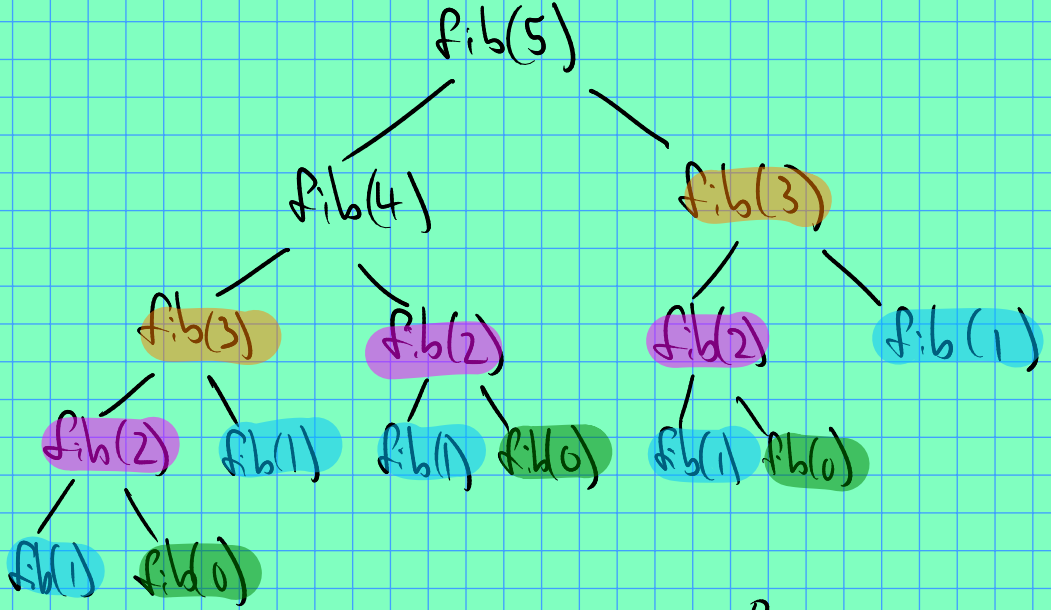
Let's Revisit Fibonacci:

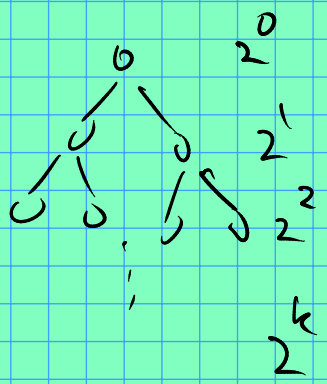
$$a_n = \begin{cases} 1 & \text{if } n < 2 \\ a_{n-1} + a_{n-2} & \text{else} \end{cases}$$

1 1 2 3 5 8 ...

```
int fib (int n)
{
    if(n < 2) return 1;
    return fib(n-1) + fib(n-2);
}
```

Let's trace this (draw recursion tree)





$$\sum_{i=0}^{n} 2^i = \frac{(2^0 + 2^1 + 2^2 + \dots + 2^n)(1-2)}{(1-2)}$$

$$= \frac{(2^0 - 2^1) + (2^1 - 2^2) + (2^2 - 2^3) + \cdots + (2^n - 2^{n+1})}{1 - 2}$$

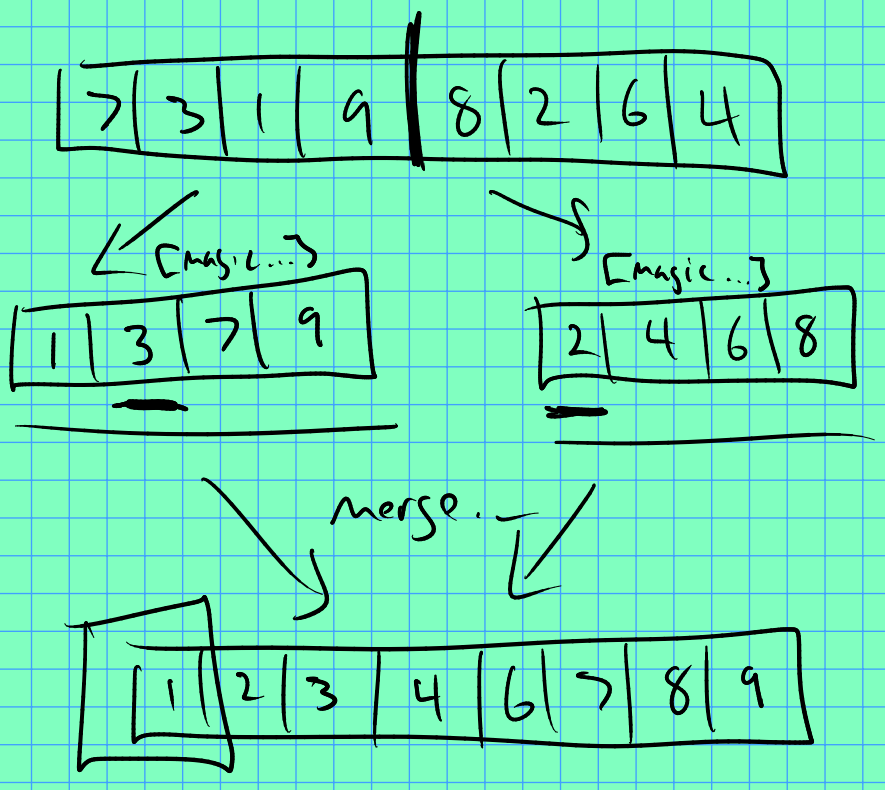
$$= \frac{1 - 2^{n+1}}{1 - 2} = 2^{n+1} - 1$$

$$\sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}$$

So size of recursion tree for f.b

$\approx 2^n \cdot (T - T)$

Example: sorting (merge sort)



```
void merge(int* L, int nL,
           int* R, int nR,
           int* S)
{
    // we know L, R are sorted.
    // nL = size of L, nR = size of R
    // S is at least nL+nR elements.
    // Then we will fill S with contents
    // of L, R in sorted order.

    int iL=0; iR=0; iS=0;
    while (iL < nL && iR < nR) { // both still have elems
        if(L[iL] < R[iR])
            S[iS++] = L[iL++];
        else
            S[iS++] = R[iR++];
    }
    // one ran out.
    while(iL < nL) S[iS++] = L[iL++];
    while(iR < nR) S[iS++] = R[iR++];
}
```

```
void mergeSort(int* A, int n, int* Aux){
    // base case: n<2 ⇒ nothing to do!
```

```
    // pick an index mid = n/2.
    // Recursively sort A[0,...,mid] and
    // A[mid+1, ..., n-1].
    // Note: for the second call, use (A+mid+1)
    // as the array! And be careful to set
    // the size right!

    // Finally, merge the two sub-arrays
    // together (using Aux) so that the
    // result is completely sorted.

}
```