

Improved Recursion for Computing Powers

```
def power(x, n):
```

```
    if n == 0:
```

```
        return 1
```

```
    else:
```

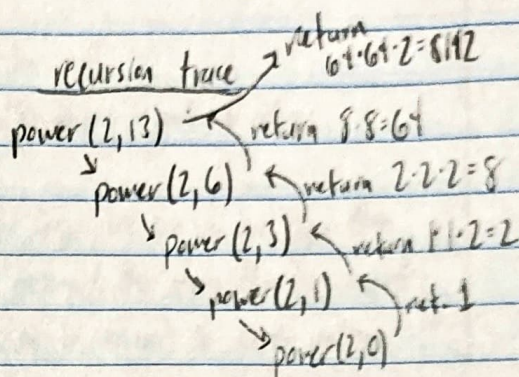
```
        partial = power(x, n // 2)
```

```
        result = partial * partial
```

```
        if n % 2 == 1:
```

```
            return * = x
```

```
        return result
```



→ complexity: exponent in each recursive call of the function is at most half of the preceding exponent.

The number of times we can divide n in half before getting to one or less is $O(\log n)$. Thus, this algorithm uses $O(\log n)$.

Binary Recursion

→ when a function makes 2 recursive calls

```
def binary-sum(S, start, stop):
```

```
    if start >= stop:
```

```
        return 0
```

```
    elif start == stop - 1:
```

```
        return S[start]
```

```
    else:
```

```
        mid = (start + stop) // 2
```

```
        return binary-sum(S, start, mid) + binary-sum(S, mid, stop)
```

uses $O(\log n)$ additional space

