universal declaration

```
MergeSort(A):
  if len(A) > 1:
    [S_L = mergesort(L half of A)
    S_R = mergesort(R half of A)]
    return merge(S_L, S_R)
  else:
    return A
```

$\frac{n}{2}$

inductive hypothesis (pf: assume claim holds for smaller instances)

assume mergesort works on any array smaller than A

base case

```
merge(sorted arrays B and C):
  S = []
  while B, C not empty:
    add smaller of B and C's first element to S
    remove said element from B or C
```

with groups:

For arrays B, C that are length n:

merge(B, C) runs in $\Theta(n)$ in best case

merge(B, C) runs in $\Theta(n)$ in worst case

B = [5 6 10 11 15] C = [4 7 9 12]

S = [0 1 2 ·] ✓

$\Theta(1)$ "constant"

$\Theta(n)$ "linear"

idea: let's write the runtime of mergesort recursively!

```
MergeSort(A):
  if len(A) > 1:
    [ S_L = mergesort(L half of A)
      S_R = mergesort(R half of A)
      return merge(S_L, S_R) ]   <- n/2
  else:
    return A
```

$T(n)$ = runtime of MergeSort on an input of size n

if $n = 1$:

$T(n) = 1$

if $n > 1$:

$T(n)$ = [time to make 2 recursive calls to mergesort] + [time to merge] $\Theta(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

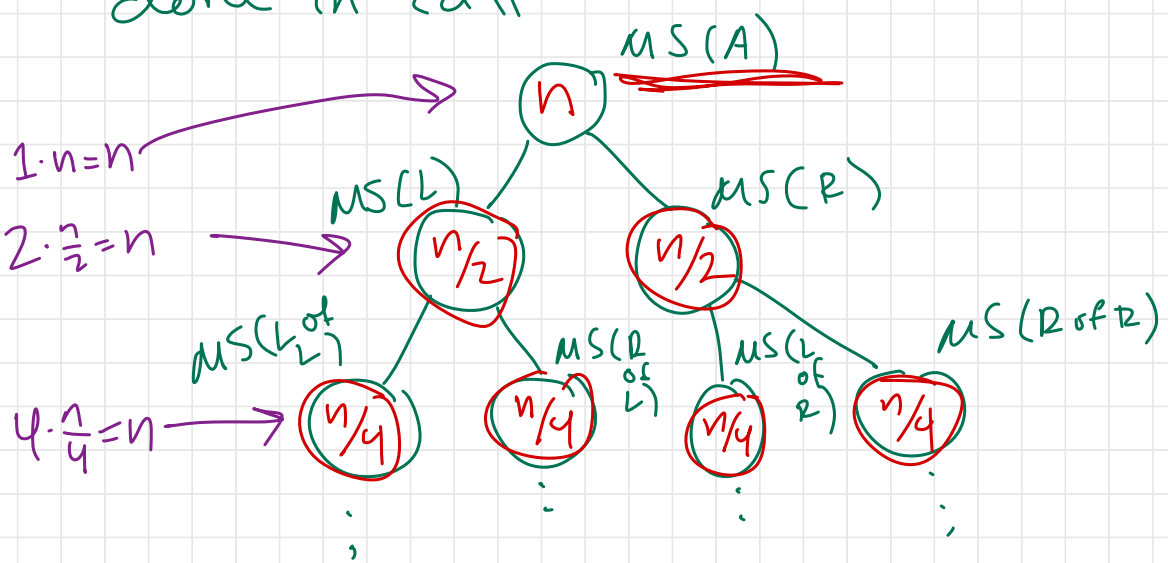
recurrence ✓

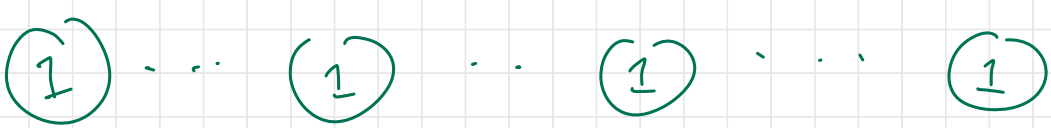
want closed-form version of $T(n)$...

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(\frac{n}{2}) + n & \text{if } n > 1 \end{cases}$$

Recursion Trees

- rooted tree with one node per call ✓
- value of node is non-recursive work done in call





$T(n)$ = sum of all node values

= sum of ~~each~~ all layers

$= \underline{\log_2 n} \quad n$