Algorithm

An **algorithm** is a step-by-step method of solving a problem.

- Each step should be basic and $\in O(1)$
- the procedure must terminate

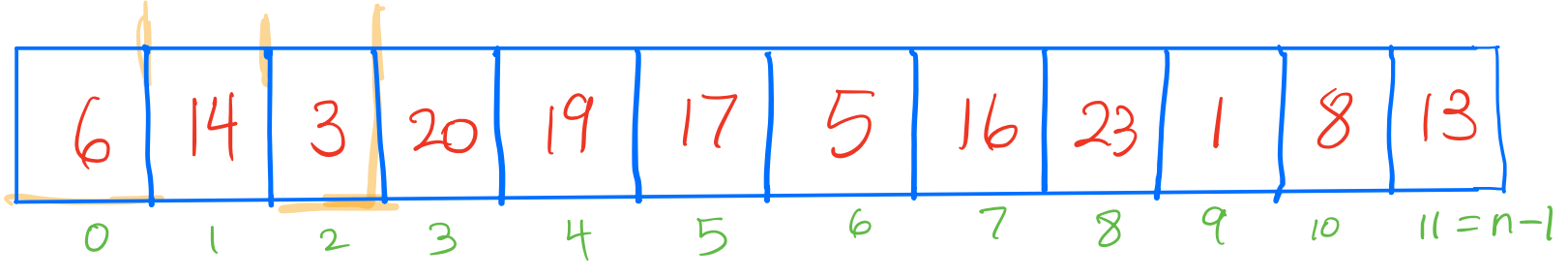
> Aside: Algs have much in common with proofs.
> - each step of your proof should be easily seen to follow from axioms (theorems) and earlier statements you have already proved.

How do we come up with algorithms to solve problems?

- There are a handful of Algorithmic Paradigms that are good to know; the majority of algorithms use these approaches in one form or another.

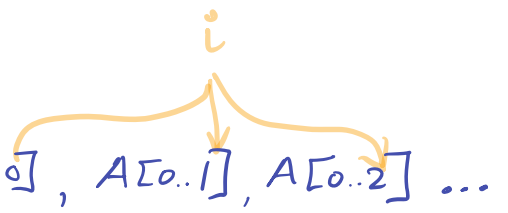
E.g. Problem "Sorting"

Input: An array A of orderable items, like int
Output: array A is now in non-decreasing order



Algorithmic Approach #1 "More of the input"

Insertion Sort



1. `i = 0`
   // we will progressively ensure that A[0..0], A[0..1], A[0..2] ... are sorted.
   // Note that A[0..i] is currently sorted.
2. Loop
   // we know that A[0..i] is already sorted
3. `i++`
4. `if i ≥ n exit loop`
5. `j = i`
6. `while (A[j] < A[j-1])`
7. `Swap(A[j], A[j-1])`
8. `j--`
9. end loop

// Now A[0..n-1] has same elements, in sorted order

```
swap(a, b)
    temp = a
    a = b
    b = temp
```

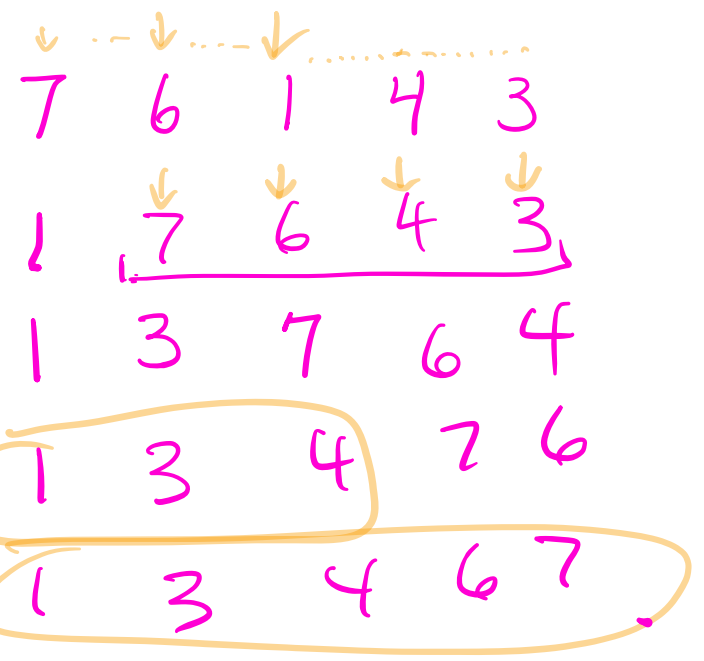
A: 7 6 1 4 3

6 7 1 4 3

1 6 7 4 3

1 4 6 7 3

1 3 4 6 7



Algorithmic Approach #2 "More of the Output"

Selection Sort

// We will construct our "output" array A[0..n-1] (sorted) by

```
1     i = 0 ;   K = 0
2     Loop:
          //  A[0..i-1] contains the i smallest elements of A
          //    in sorted order
2.5       if   i = n     exit
3         for (j = i ; j < n ; j++)
4             if  A[j] < A[k]
5                 K = j
6         Swap (A[i], A[K])
7         i++
8     end Loop
```

scan from position i to n-1 to find smallest element.

Analyze Insertion Sort - Worst Case Analysis

1. $i = 0$ — $O(1)$
2. Loop
3. i++
4. if $i \geq n$ exit loop
5. $j = i$
6. while $(A[j] < A[j-1])$
7. swap $(A[j], A[j-1])$
8. j--

. end loop

$= \max j$

$\left.\begin{array}{l} \{ O(1) \\ \} \end{array}\right\} O(n^2)$.

$1+2+\cdots+n-1$

$= \frac{n(n-1)}{2}$

A: 7 6 1 4 3

6 7 1 4 3

1 6 7 4 3

1 4 6 7 3

1 3 4 6 7

```
swap (a, b)
    temp = a
    a = b
    b = temp
```

$O(1)$

Analyze Selection Sort – Worst case

```
1       i = 0 ;   K = 0                                   } O(1)
2       Loop:
            exit when . . .
3           for (j = i; j < n; j++)                       } n-1 + n-2 + ... 1 = n(n-1)/2
4               if A[j] < A[k]                            }   = O(n^2).
5                   K = j
6           swap (A[i], A[K])
7           i++
8       end Loop
```

$O(n^2)$.

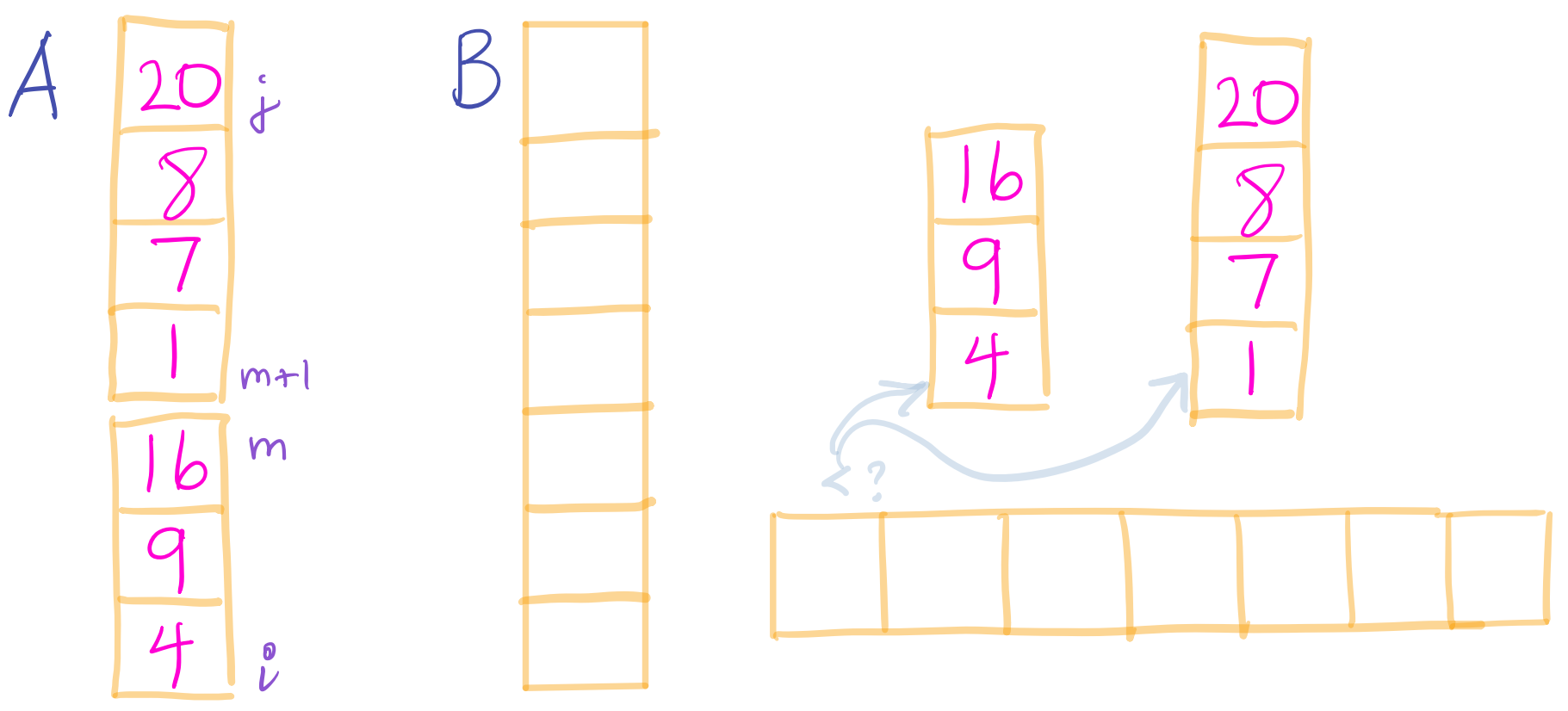
Algorithmic Approach: Divide & Conquer

```
MergeSort(A, i, j)
// A[0..n-1] is an array, 0 ≤ i < n, 0 ≤ j < n
if j > i
    mid = ⌊(i + j)/2⌋
    MergeSort(A, i, mid)
    MergeSort(A, mid+1, j)
    Merge(A, i, mid, j)
// end.
// Now A is sorted
```

end of first range

Merge (A, i, m, j)

```
k = 1 ; x = i ; y = m+1 ;
while x ≤ m and y ≤ j
    if A[x] ≤ A[y]
        B[k++] = A[x++]
    else B[k++] = A[y++]
if x ≤ m copy A[x..m] into B[k..j]
else if y ≤ m copy A[y..j] into B[k..j]
Copy B[1..j-i+1] into A[i..j]
```



- Running time is proportional to number of assignment operation. =
- Each element value is assigned to an array position .... ___ times.. How comparisons?

Analyze MergeSort

1. Running time of Merge is

2. Let time for MergeSort on n elements be $T(n)$

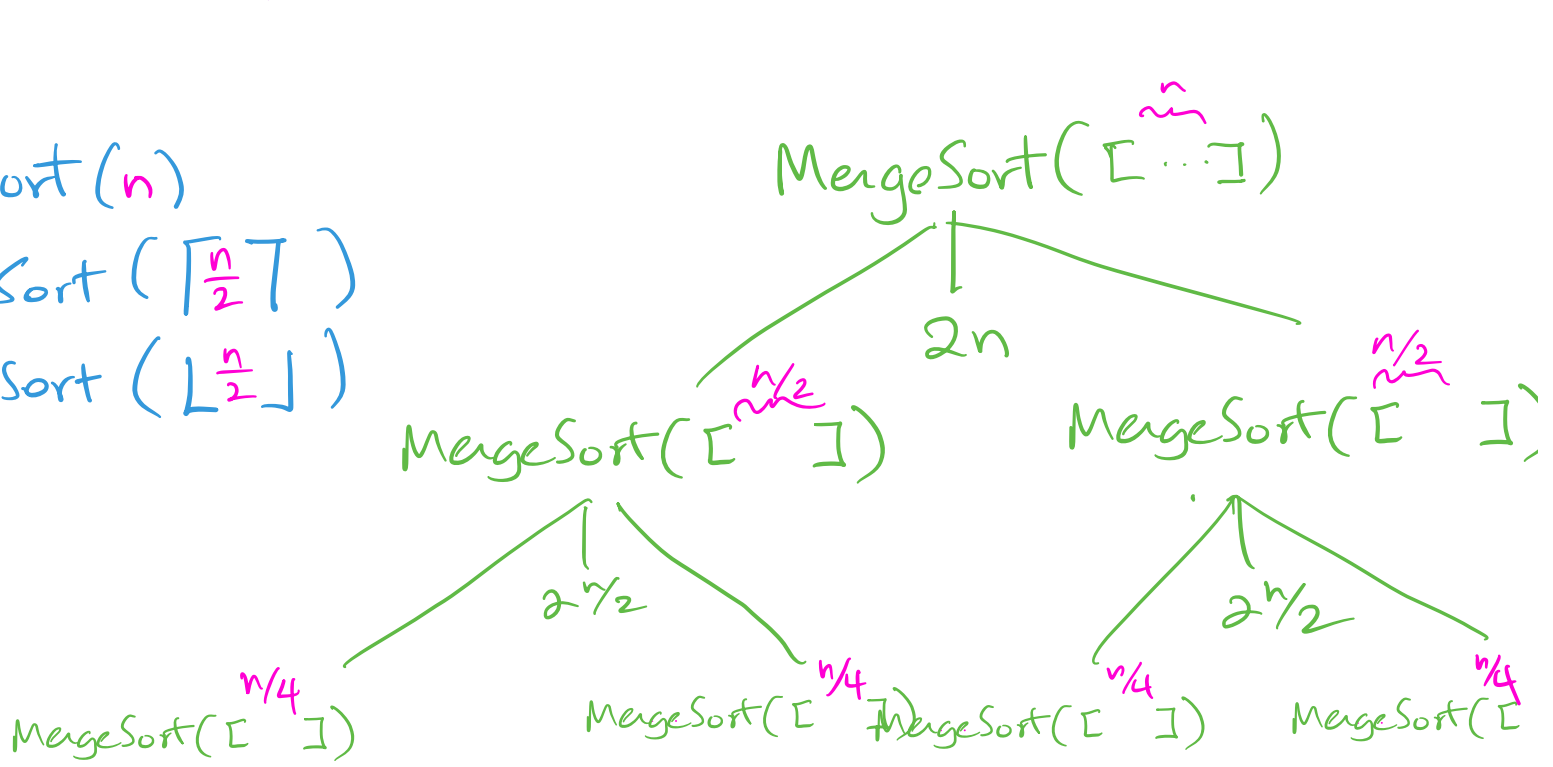
Then $T(n) =$

$T_{MergeSort}(n)$

$= T_{MergeSort}(\lceil \frac{n}{2} \rceil)$

$+ T_{MergeSort}(\lfloor \frac{n}{2} \rfloor)$

$+ 2n$



The Master Theorem

For a function $T(n)$ defined on positive integers, where $T(n) = aT(\frac{n}{b}) + f(n)$ and $f(n)$ is pos^ve valued function, and $a \geq 1$ and $b > 1$, then:

1. $f(n) \in O(n^{\log_b a - \varepsilon})$ for some const $\varepsilon > 0$

   $\Rightarrow T(n) \in \Theta(n^{\log_b a})$

2. $f(n) \in \Theta(n^{\log_b a})$

   $\Rightarrow T(n) \in \Theta(n^{\log_b a} \lg n)$

3. $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ for some const $\varepsilon > 0$,

   regularity { AND $\exists$ constant c s.t. $0 < c < 1$ and $a f(n/b) < c f(n)$ when n sufficiently large,

   $\Rightarrow T(n) \in \Theta(f(n))$.

Eg: $T(n) = 2T(\frac{n}{2}) + n^2$

$n^2 \in O(n^{\lg 2 - \varepsilon})$?

$\in \Theta(n)$?

$\in \Omega(n^{1+\varepsilon})$

Hence $T(n) \in O(n^2)$

$\exists ? 0 < c < 1$

where

$2\left(\left(\frac{n}{2}\right)^2\right) < c\, n^2$.

$\frac{n^2}{2} < c\, n^2$

$c = \frac{1}{2}$.

$T(n) = 2T(\frac{n}{2}) + n$ ← $f(n)$.

$n \in O(n^{1-\varepsilon})$

$\Theta(n)$

$\Omega(n^{1+\varepsilon})$

$\therefore T(n) \in \Theta(n \lg n)$, by MT part 2.

$T(n) = 2T(\frac{n}{2}) + \lg n$ ← !

$\lg n \stackrel{?}{\in} O(n^{1-\varepsilon})$

$\Theta(n)$

$\Omega(n^{1+\varepsilon})$

$\lg n \in O(n^{1-\varepsilon})$, for $\varepsilon = 0.1$

$\therefore$ By MT, case 1, $T(n) \in \Theta(n)$