SORTING

Input: a list of numbers $8, \sqrt{2}, 5, 21, 1/3$

Output: a permutation that is in increasing order $1/3, \sqrt{2}, 5, 8, 21$

SORTING

Input: a list of numbers $8, \sqrt{2}, 5, 21, 1/3$

Output: a permutation that is in increasing order $1/3, \sqrt{2}, 5, 8, 21$

An algorithm should do the following:

- produce correct output for all possible input

SORTING

Input: a list of numbers $8, \sqrt{2}, 5, 21, \frac{1}{3}$

Output: a permutation that is in increasing order $\frac{1}{3}, \sqrt{2}, 5, 8, 21$

An algorithm should do the following:

1) produce correct output for all possible input
2) terminate quickly
3) not use a lot of space

SORTING

Input: a list of numbers $8, \sqrt{2}, 5, 21, \frac{1}{3}$

Output: a permutation that is in increasing order $\frac{1}{3}, \sqrt{2}, 5, 8, 21$

An algorithm should do the following:

1) produce correct output for all possible input
2) terminate quickly
3) not use a lot of space
4) be described clearly

etc

Issues that affect algorithmic design:

- input type: e.g., are numbers distinct? integer/real/irrational/etc?
  Do they have bounded size?

Issues that affect algorithmic design:

- input type: e.g., are numbers distinct? integer/real/irrational/etc?
  Do they have bounded size?
- allowed operations: compare, add, truncate, etc

Issues that affect algorithmic design:

- input type: e.g., are numbers distinct? integer/real/irrational/etc?
  Do they have bounded size?
- allowed operations: compare, add, truncate, etc
- data structure: array? linked list? tree? etc

Issues that affect algorithmic design:

- input type: e.g., are numbers distinct? integer/real/irrational/etc?
  Do they have bounded size?
- allowed operations: compare, add, truncate, etc
- data structure: array? linked list? tree? etc
- model of computation: time/space complexity of operations

Issues that affect algorithmic design:

- input type: e.g., are numbers distinct? integer/real/irrational/etc?
  Do they have bounded size?
- allowed operations: compare, add, truncate, etc
- data structure: array? linked list? tree? etc
- model of computation: time/space complexity of operations

We focus on:
- comparison-based algorithms
- constant time for basic ops (more later)

Insertion sort

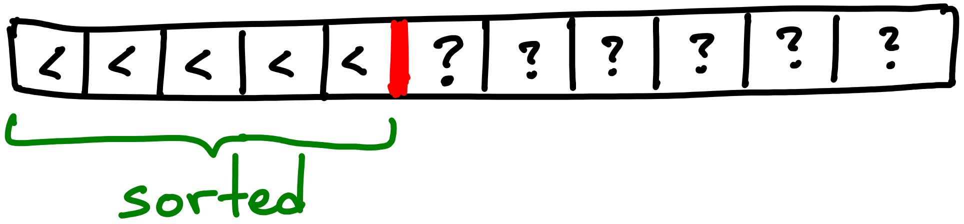
- Start with a sorted prefix of size 1.
- Extend size of sorted prefix by 1
- Repeat

Insertion sort

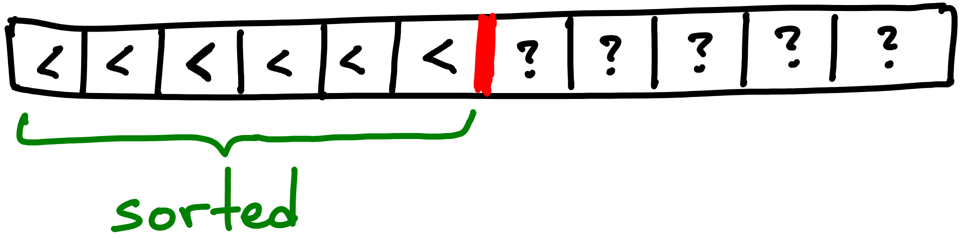
- Start with a sorted prefix of size 1.
- Extend size of sorted prefix by 1
- Repeat

In general:

Before



After



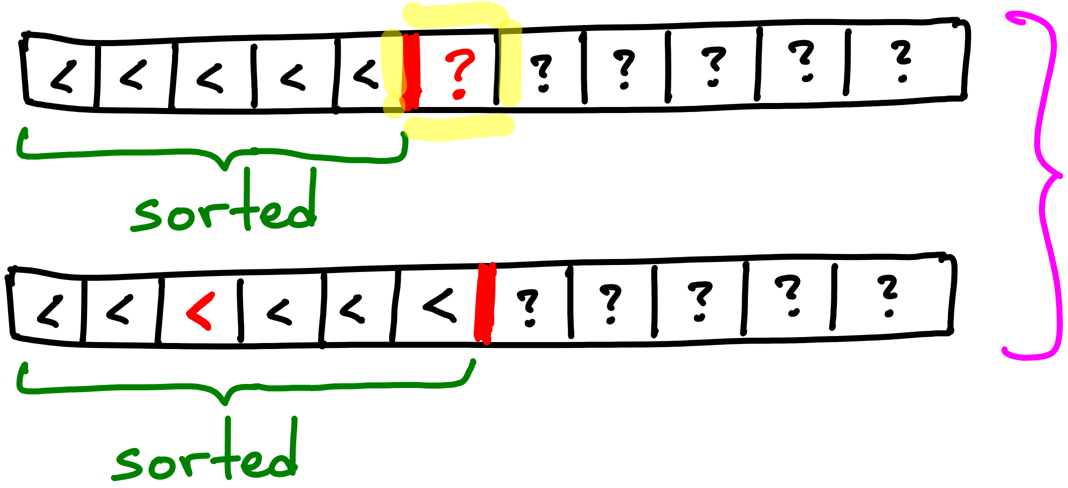
Insertion sort

- Start with a sorted prefix of size 1.
- Extend size of sorted prefix by 1
- Repeat

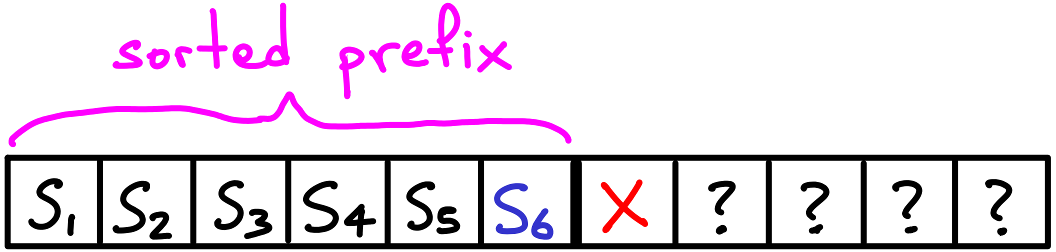
In general:

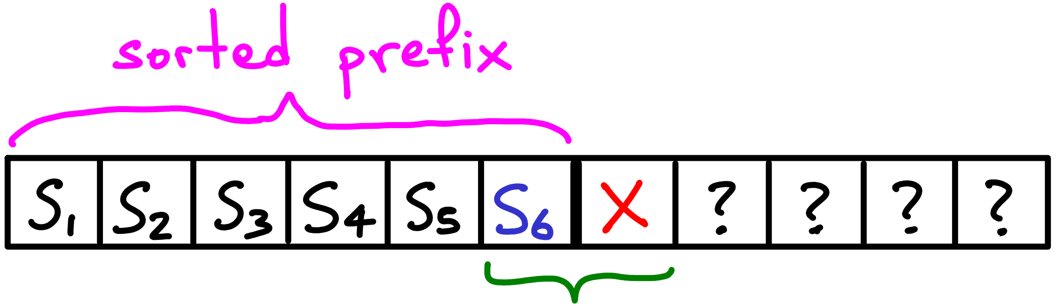
Before

After



Use the element next to the prefix to extend the prefix...


sorted prefix
S1 S2 S3 S4 S5 S6 X ? ? ? ?



if $S_6 \leq x$ $\rightarrow$ trivial extension

$$\overbrace{\boxed{S_1}\boxed{S_2}\boxed{S_3}\boxed{S_4}\boxed{S_5}\boxed{S_6}}^{\text{sorted prefix}}\boxed{x}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$$

if $S_6 \leq x$ $\rightarrow$ trivial extension

else $S_6 > x$

sorted prefix

S_1	S_2	S_3	S_4	S_5	S_6	X	?	?	?	?

if $S_6 \leqslant X$ $\rightarrow$ trivial extension

else $S_6 > X$ $\rightarrow$ swap

S_1	S_2	S_3	S_4	S_5	X	S_6	?	?	?	?

sorted prefix

S_1	S_2	S_3	S_4	S_5	S_6	X	?	?	?	?

if $S_6 \leq X$ → trivial extension

else $S_6 > X$ → swap & compare X to S_5

S_1	S_2	S_3	S_4	S_5	X	S_6	?	?	?	?

?

sorted prefix

S_1	S_2	S_3	S_4	S_5	S_6	X	?	?	?	?

if $S_6 \leqslant X$ → trivial extension

else $S_6 > X$ → swap & compare X to S_5

S_1	S_2	S_3	S_4	S_5	X	S_6	?	?	?	?

?

etc

If prefix size = j then we can insert X after at most j comparisons

$\leq j$ comparisons $\rightarrow$ increase the sorted prefix size from j to $j+1$

Terminate when prefix size $= n$ (entire array)

$\leq j$ comparisons $\rightarrow$ increase the sorted prefix size from j to $j+1$

Terminate when prefix size = n (entire array)

comparisons $\leq \sum_{j=1}^{n-1} j = 1 + 2 + 3 + \dots + (n-1)$

$\leq j$ comparisons $\rightarrow$ increase the sorted prefix size from j to $j+1$

Terminate when prefix size $= n$ (entire array)

$$\text{comparisons} \leq \sum_{j=1}^{n-1} j = 1 + 2 + 3 + \cdots + (n-1) = \frac{n(n-1)}{2}$$

$$= \frac{1}{2}n^2 - \frac{1}{2}n$$ = worst case #comparisons

$\le j$ comparisons → increase the sorted prefix size from j to $j+1$

Terminate when prefix size = n (entire array)

$$\text{comparisons} \le \sum_{j=1}^{n-1} j = 1+2+3+\cdots+(n-1) = \frac{n(n-1)}{2}$$

$$= \frac{1}{2}n^2 - \frac{1}{2}n$$ = worst case #comparisons

---

To actually implement, we need extra time & space

but just a constant amount per comparison e.g., $5\cdot\left(\frac{1}{2}n^2-\frac{1}{2}n\right)$

At an introductory level:

- we don't focus on whether it takes 1 or 2 or 5 operations to compare, swap & iterate: $5 \cdot \left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$

At an introductory level:

- we don't focus on whether it takes 1 or 2 or 5 operations to compare, swap & iterate: $5 \cdot \left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$
- we don't really care about non-leading terms: $5 \cdot \left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$

At an introductory level:

- we don't focus on whether it takes 1 or 2 or 5 operations to compare, swap & iterate: $5 \cdot \left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$

- we don't really care about non-leading terms: $5 \cdot \left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$

When n is HUGE, both of the above are unimportant.

At an introductory level:

- we don't focus on whether it takes 1 or 2 or 5 operations to compare, swap & iterate: $5 \cdot \left(\frac{1}{2} n^2 - \frac{1}{2} n \right)$

- we don't really care about non-leading terms: $5 \cdot \left(\frac{1}{2} n^2 - \frac{1}{2} n \right)$

When n is HUGE, both of the above are unimportant.

$$\begin{bmatrix} \text{see} & 5 \cdot \left(\frac{1}{2} n^2 - \frac{1}{2} n \right) \\ \text{interpret} & n^2 \end{bmatrix}$$

At an introductory level:

- we don't focus on whether it takes 1 or 2 or 5 operations

  to compare, swap & iterate: $5 \cdot \left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$

- we don't really care about non-leading terms: $5 \cdot \left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$

When n is HUGE, both of the above are unimportant.

$$\left[\begin{array}{ll} \text{see} & 5 \cdot \left(\frac{1}{2}n^2 - \frac{1}{2}n\right) \\ \text{interpret} & n^2 \end{array}\right]$$

This leads to Θ-notation

aka big-O notation