Algorithm Analysis

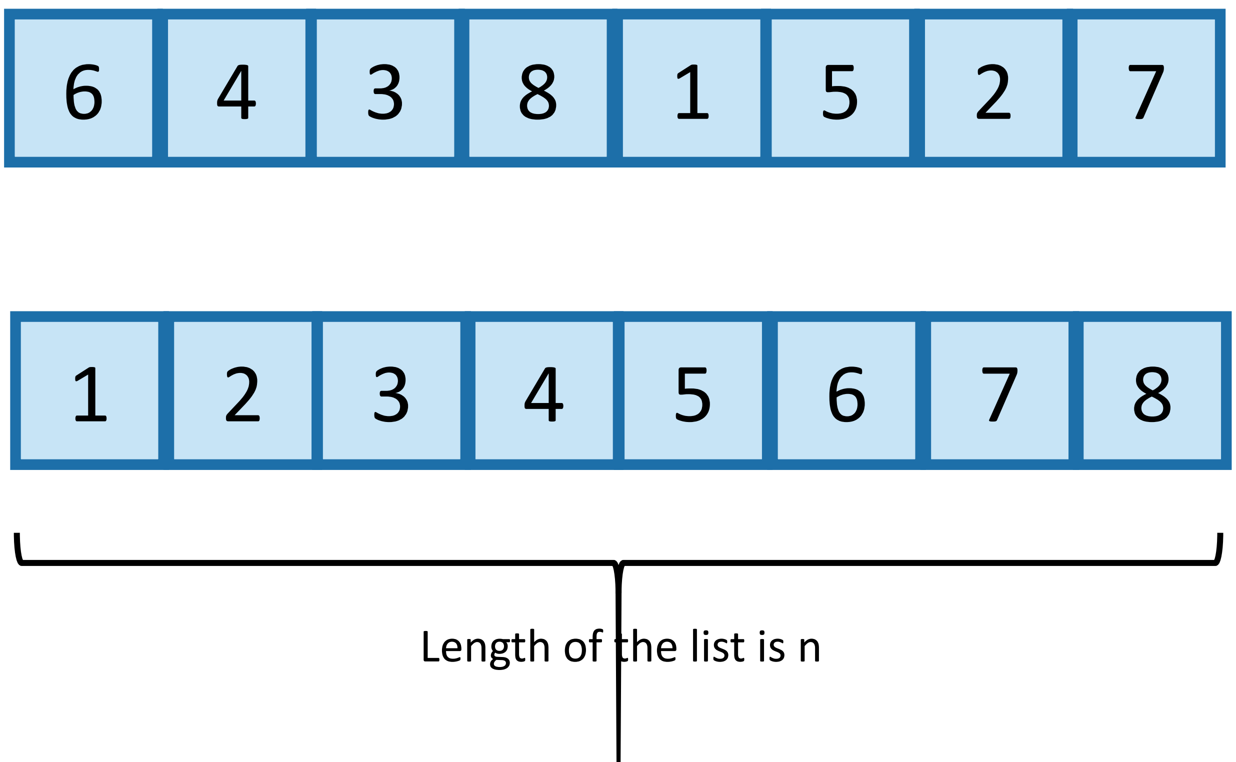
Insertion Sort

Sorting

- Arrange an unordered list of elements in some order.
- Some common algorithms
  - Bubble Sort
  - Insertion Sort
  - Merge Sort
  - Quick Sort

Sorting

- Important primitive
- For today, we'll pretend all elements are distinct.

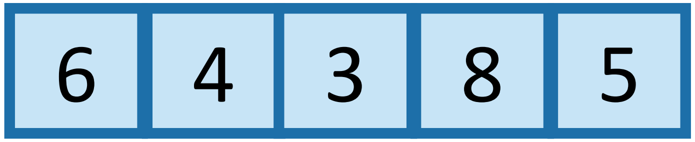


Insertion Sort

```
INSERTION-SORT(A, n)
for i = 2 to n
    key = A[i]
    // Insert A[i] into the sorted subarray A[1 : i − 1].
    j = i − 1
    while j > 0 and A[j] > key
        A[j + 1] = A[j]
        j = j − 1
    A[j + 1] = key
```

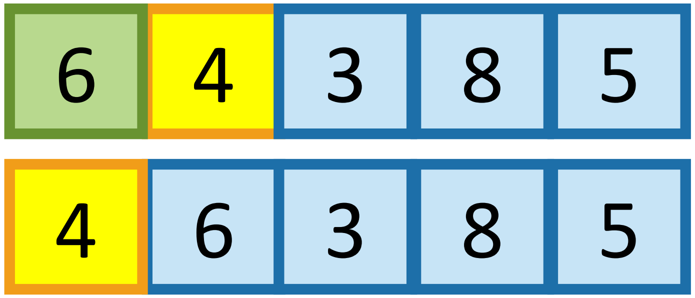
Insertion Sort

example




```
Insertion-Sort(A, n)
    for i = 1 to n - 1
    key = A[i]
    j = i - 1
    while j >= 0 and A[j] > key
        A[j + 1] = A[j]
        j = j - 1
    A[j + 1] = key
```

Start by moving A[1] toward the beginning of the list until you find something smaller (or can't go any further):



Insertion Sort

example

6	4	3	8	5

```
Insertion-Sort(A, n)
     for i = 1 to n - 1
     key = A[i]
     j = i - 1
     while j >= 0 and A[j] > key
          A[j + 1] = A[j]
          j = j - 1
     A[j + 1] = key
```

Then move A[2]:

key = 3

4	6	3	8	5
4	6	6	8	5
4	4	6	8	5
3	4	6	8	5

Insertion Sort

example

6	4	3	8	5

```
Insertion-Sort(A, n)
    for i = 1 to n - 1
    key = A[i]
    j = i - 1
    while j >= 0 and A[j] > key
        A[j + 1] = A[j]
        j = j - 1
    A[j + 1] = key
```

Then move A[3]:

key = 8

3	4	6	8	5

3	4	6	8	5

Insertion Sort

example

6	4	3	8	5

```
Insertion-Sort(A, n)
     for i = 1 to n - 1
     key = A[i]
     j = i - 1
     while j >= 0 and A[j] > key
          A[j + 1] = A[j]
          j = j - 1
     A[j + 1] = key
```

Then move A[4]:

key = 5

3	4	6	8	5
3	4	6	8	8
3	4	6	6	8
3	4	5	6	8

Insertion Sort

example

6	4	3	8	5

Start by moving A[1] toward the beginning of the list until you find something smaller (or can't go any further):

6	**4**	3	8	5
4	6	3	8	5

Then move A[2]:

4	6	**3**	8	5
3	4	6	8	5

Then move A[3]:

3	4	6	**8**	5
3	4	6	**8**	5

Then move A[4]:

3	4	6	8	**5**
3	4	**5**	6	8

Then we are done!

Why does this work?

- Say you have a sorted list, $\boxed{3}\boxed{4}\boxed{6}\boxed{8}$, and another element $\boxed{5}$.
- Insert $\boxed{5}$ right after the largest thing that's still smaller than $\boxed{5}$. (Aka, right after $\boxed{4}$).
- Then you get a sorted list: $\boxed{3}\boxed{4}\boxed{5}\boxed{6}\boxed{8}$

This sounds like a job for…

Proof By Induction!

Outline of a proof by induction

Let `A` be a list of length `n`

- Base case:
  - `A[:1]` is sorted at the end of the 0'th iteration. ✓
- Inductive Hypothesis:
  - `A[:i+1]` is sorted at the end of the i^{th} iteration (of the outer loop).
- Inductive step:
  - For any `0 < k < n`, if the inductive hypothesis holds for `i=k-1`, then it holds for `i=k`.
  - Aka, if `A[:k]` is sorted at step `k-1`, then `A[:k+1]` is sorted at step k (previous slide)
- Conclusion:
  - The inductive hypothesis holds for i = 0, 1, …, n-1.
  - In particular, it holds for i=n-1.
  - At the end of the n-1'st iteration (aka, at the end of the algorithm), `A[:n] = A` is sorted.
  - That's what we wanted! ✓

Algorithm Analysis

- Estimate the resources required by an algorithm
  - Memory
  - Communication Bandwidth
  - Energy Consumption
  - Computational Time

- Helps identify the most efficient one

Algorithm Analysis (Insertion Sort)

- Timing the run of insertion sort on our computer
- We will get runtime estimates for,
  - A particular computer
  - A particular input
  - A particular implementation
  - A particular compiler/interpreter
  - Particular libraries and background tasks
- What about others?

Algorithm Analysis

- Assumptions
  - One-processor
  - Random-access machine (RAM) model of computation

Random-access Machine

- Random-access machine (RAM)
  - Instructions execute one after another
  - No concurrent operations
  - Each instructions takes the same amount of time
  - Each data access takes the same amount of time
  - Contains commonly found instructions
    - Arithmetic (add subtract, multiply, divide, remainder, floor, ceiling)
    - Data movement (load, store, copy) and
    - Control (branching, call and return)
  - Includes common data types
  - Doesn’t model memory hierarchy

Algorithm Analysis (Insertion Sort)

- Runtime depends on inputs
  - Sort an array of 10000 items vs Sort and array of 3 items

- Input Size
  - Problem specific
  - For sorting, the number of items in the input
  - For multiplication, the total number of bits needed for representation
  - For graph, the number of nodes and edges

Algorithm Analysis (Insertion Sort)

- Running time of an algorithm
  - For a given input
  - The number of instructions and data access executed

- Assumption
  - Constant time taken by each line of the pseudocode

Algorithm Analysis (Insertion Sort)

INSERTION-SORT(A, n)	*cost*	*times*
1 **for** $i = 2$ **to** n	c_1	n
2 $\quad$ $key = A[i]$	c_2	$n-1$
3 $\quad$ // Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
4 $\quad$ $j = i - 1$	c_4	$n-1$
5 $\quad$ **while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6 $\quad\quad$ $A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7 $\quad\quad$ $j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8 $\quad$ $A[j+1] = key$	c_8	$n-1$

t_i denotes the number of times the **while** loop test in line 5 is executed for that value of i

Algorithm Analysis (Insertion Sort)

- T(n) denote the running time of an algorithm on an input of size of n

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n-1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$j = i - 1$	c_4	$n-1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n}(t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n}(t_i - 1)$
$A[j+1] = key$	c_8	$n-1$

Algorithm Analysis (Insertion Sort)

$$
\begin{aligned}
T(n) = {} & c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) \\
& + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) .
\end{aligned}
$$

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$\quad key = A[i]$	c_2	$n-1$
$\quad$ // Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$\quad j = i - 1$	c_4	$n-1$
$\quad$ **while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$\qquad A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$\qquad j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$\quad A[j+1] = key$	c_8	$n-1$

Algorithm Analysis (Insertion Sort)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) .$$

- Best Case
  - The array is sorted
  - $t_i = 1$ for all $i = 2, 3, \dots, n$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

Algorithm Analysis (Insertion Sort)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) .$$

- Best Case
  - The array is sorted
  - $t_i = 1$ for all $i = 2, 3, \dots, n$

A linear function of n

$$T(n) = an + b$$

where $a = c_1 + c_2 + c_4 + c_5 + c_8$

and $b = -(c_2 + c_4 + c_5 + c_8)$

Algorithm Analysis (Insertion Sort)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) .$$

- Worst Case
  - The array is sorted in reverse order
  - $t_i = i$ for all $i = 2, 3, \ldots, n$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1 \right) \\ &\quad + c_6 \left(\frac{n(n-1)}{2} \right) + c_7 \left(\frac{n(n-1)}{2} \right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n \\ &\quad - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

Algorithm Analysis (Insertion Sort)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1).$$

- Best Case
  - The array is sorted
  - $t_i = i$ for all $i = 2, 3, \dots, n$

A quadratic function of n

$$T(n) = an^2 + bn + c$$

where a, b and c are constants

Algorithm Analysis (Insertion Sort)

- Instead of exact function, we estimate the rate of growth or the order of growth

- Best Case
  - Most significant term an
  - Linear

- Worst Case
  - Most significant term an^2
  - Quadratic

Reference

- CLRS Chapter 2
  - Sections 2.1, 2.2