Algorithm Analysis 1: Step Counting

- Objectives: Students should be able to...
  - identify constant-time operations from simple pseudocode (using standard assumptions),
  - count the number of time steps for simple looping procedures, such as sorting algorithms, under best-case and worst-case assumptions,
  - know the limitations to the usefulness of step counting.

- Reading: CLRS pp. 16-29 (today's example).
More about step counting

- Example algorithm: insertion sort
- Input: an array of keys with a total order. Keys in the array may be jumbled.
- Output: keys are *permuted* within the array so they are in nondecreasing order.
  - permuted: none added or lost, just rearranged.
- Strategy: keep the left side of the array sorted, and “grow” it until all keys are sorted
- This is called *insertion sort*. 
Insertion sort

InsertionSort(A[1..n])

\[\text{for } j \leftarrow 2 \text{ to } \text{length}(A)\]

\[\text{key } \leftarrow A[j]\]

\[i \leftarrow j - 1\]

\[\text{while } i > 0 \text{ and } A[i] > \text{key} \]

\[A[i+1] \leftarrow A[i]\]

\[i \leftarrow i - 1\]

\[A[i+1] \leftarrow \text{key}\]

Pseudocode conventions: Function name, and input array passed by ref., with how it is indexed
Insertion sort

InsertionSort(A[1..n])

1. for j ← 2 to length(A)
2. key ← A[j]
3. i ← j - 1
4. while i > 0 and A[i] > key
6. i ← i - 1
7. A[i+1] ← key

Pseudocode conventions:
left arrow denotes assignment.
Insertion sort

InsertionSort(A[1..n])

c_1 for j ← 2 to length(A)

c_2 key ← A[j]

c_3 i ← j - 1

c_4 while i > 0 and A[i] > key

c_5 A[i+1] ← A[i]

c_6 i ← i - 1

c_7 A[i+1] ← key

Pseudocode conventions:
for loop tests its condition n total times.
(n-1) times: success
one time: fail when j=n+1
Insertion sort

\[ \text{InsertionSort}(A[1..n]) \]

1. \( \text{for } j \leftarrow 2 \text{ to } \text{length}(A) \)

2. \( \text{key} \leftarrow A[j] \)

3. \( i \leftarrow j - 1 \)

4. \( \text{while } i > 0 \text{ and } A[i] > \text{key} \)

5. \( A[i+1] \leftarrow A[i] \)

6. \( i \leftarrow i - 1 \)

7. \( A[i+1] \leftarrow \text{key} \)

**Constant-time steps:** each line here, in isolation, takes approximately constant time. (all primitive ops)
**Insertion sort: step counting**

\[
\text{InsertionSort}(A[1..n])
\]

\[
c_1 \quad \text{for } j \leftarrow 2 \text{ to length}(A)
\]

\[
c_2 \quad \text{key} \leftarrow A[j]
\]

\[
c_3 \quad i \leftarrow j - 1
\]

\[
c_4 \quad \text{while } i > 0 \text{ and } A[i] > \text{key}
\]

\[
c_5 \quad A[i+1] \leftarrow A[i]
\]

\[
c_6 \quad i \leftarrow i - 1
\]

\[
c_7 \quad A[i+1] \leftarrow \text{key}
\]

\[
\text{Time Cost of Line 1:}
\]

We could just write as \((c_1 n)\).

More abstractly, the cost is \(c_1\) for each value that \(j\) takes:

\[
\sum_{j=2}^{n+1} c_1
\]
Insertion sort: step counting

InsertionSort(A[1..n])

\( c_1 \) for \( j \leftarrow 2 \) to length(A)

\( c_2 \) \( \text{key} \leftarrow A[j] \)

\( c_3 \) \( i \leftarrow j - 1 \)

\( c_4 \) while \( i > 0 \) and \( A[i] > \text{key} \)

\( c_5 \) \( A[i+1] \leftarrow A[i] \)

\( c_6 \) \( i \leftarrow i - 1 \)

\( c_7 \) \( A[i+1] \leftarrow \text{key} \)

Time Cost of Line 2:

Cost is \( c_2 \) for each value that \( j \) takes there:

\[ \sum_{j=2}^{n+1} c_1 \]

\[ \sum_{j=2}^{n} c_2 \]
Insertion sort: step counting

InsertionSort(A[1..n])

\[ \sum_{j=2}^{n+1} c_1 \]
\[ \sum_{j=2}^{n} c_2 \]

1. for \( j \leftarrow 2 \) to \( \text{length}(A) \)
2. \( \text{key} \leftarrow A[j] \)
3. \( i \leftarrow j - 1 \)
4. while \( i > 0 \) and \( A[i] > \text{key} \)
5. \( A[i+1] \leftarrow A[i] \)
6. \( i \leftarrow i - 1 \)
7. \( A[i+1] \leftarrow \text{key} \)

Time Cost of Line 3:

Cost is \( c_3 \) for each value that \( j \) takes there:
\[ \sum_{j=2}^{n} c_3 \]
Insertion sort

\[ \text{InsertionSort}(A[1..n]) \]

1. for \( j \leftarrow 2 \) to length(A)
2. \( \text{key} \leftarrow A[j] \)
3. \( i \leftarrow j - 1 \)
4. while \( i > 0 \) and \( A[i] > \text{key} \)
5. \( A[i+1] \leftarrow A[i] \)
6. \( i \leftarrow i - 1 \)
7. \( A[i+1] \leftarrow \text{key} \)

\[
\sum_{j=2}^{n+1} c_1 \\
\sum_{j=2}^{n} c_2 \\
\sum_{j=2}^{n} c_3
\]

**Time Cost of Line 4:**
Cost is \( c_4 \) for each value that \( i, j \) take there:

\[
\sum_{j=2}^{n} \sum_{i=?}^{??} c_4
\]
Insertion sort: pause the analysis! What's that inner loop doing?

\begin{align*}
\text{InsertionSort}(A[1..n]) \\
&c_1 \quad \text{for } j \leftarrow 2 \text{ to } \text{length}(A) \\
&c_2 \quad \text{key } \leftarrow A[j] \\
&c_3 \quad i \leftarrow j - 1 \\
&c_4 \quad \text{while } i > 0 \text{ and } A[i] > \text{key} \\
&\quad A[i+1] \leftarrow A[i] \\
&c_5 \quad i \leftarrow i - 1 \\
&c_7 \quad A[i+1] \leftarrow \text{key}
\end{align*}

\[
\sum_{j=2}^{n+1} c_1 \\
\sum_{j=2}^{n} c_2 \\
\sum_{j=2}^{n} c_3
\]
Insertion sort (worst case)

\[
\text{InsertionSort}(A[1..n])
\]

1. \( \text{for } j \leftarrow 2 \text{ to } \text{length}(A) \)
2. \( \text{key} \leftarrow A[j] \)
3. \( i \leftarrow j - 1 \)
4. while \( i > 0 \) and \( A[i] > \text{key} \)
5. \( A[i+1] \leftarrow A[i] \)
6. \( i \leftarrow i - 1 \)
7. \( A[i+1] \leftarrow \text{key} \)

Time Cost of Line 4, worst case:
\[
\sum_{j=2}^{n+1} c_1 \\
\sum_{j=2}^{n} c_2 \\
\sum_{j=2}^{n} c_3 \\
\sum_{j=2}^{n} \sum_{i=???}^{???} c_4
\]
Insertion sort (worst case)

InsertionSort(A[1..n])

\[ \begin{align*}
  & c_1 \quad \text{for } j \leftarrow 2 \text{ to } \text{length}(A) \\
  & c_2 \quad \text{key} \leftarrow A[j] \\
  & c_3 \quad i \leftarrow j - 1 \\
  & c_4 \quad \text{while } i > 0 \text{ and } A[i] > \text{key} \\
  & c_5 \quad A[i+1] \leftarrow A[i] \\
  & c_6 \quad i \leftarrow i - 1 \\
  & c_7 \quad A[i+1] \leftarrow \text{key}
\end{align*} \]

Time Cost of Line 5, worst case:

\[ \sum_{j=2}^{n+1} c_1 \]

\[ \sum_{j=2}^{n} c_2 \]

\[ \sum_{j=2}^{n} c_3 \]

\[ \sum_{j=2}^{n} \sum_{i=0}^{j-1} c_4 \]

\[ \sum_{j=2}^{n} \sum_{i=1}^{j-1} c_5 \]
Insertion sort (worst case)

\begin{align*}
\text{InsertionSort}(A[1..n]) \\
\text{c}_1 & \quad \text{for } j \leftarrow 2 \text{ to length}(A) \\
\text{c}_2 & \quad \text{key } \leftarrow A[j] \\
\text{c}_3 & \quad i \leftarrow j - 1 \\
\text{c}_4 & \quad \text{while } i > 0 \text{ and } A[i] > \text{key} \\
\text{c}_5 & \quad A[i+1] \leftarrow A[i] \\
\text{c}_6 & \quad i \leftarrow i - 1 \\
\text{c}_7 & \quad A[i+1] \leftarrow \text{key}
\end{align*}

\begin{align*}
\sum_{j=2}^{n+1} c_1 \\
\sum_{j=2}^{n} c_2 \\
\sum_{j=2}^{n} c_3 \\
\sum_{j=2}^{n} \sum_{i=0}^{j-1} c_4 \\
\sum_{j=2}^{n} \sum_{i=1}^{j-1} c_5 \\
\sum_{j=2}^{n} \sum_{i=1}^{j-1} c_6
\end{align*}

Line 6, worst case
Insertion sort (worst case)

\[
\text{InsertionSort}(A[1..n])
\]

\begin{align*}
&\text{for } j \leftarrow 2 \text{ to } \text{length}(A)
\end{align*}

\begin{align*}
&\text{key } \leftarrow A[j] \\
&i \leftarrow j - 1 \\
&\text{while } i > 0 \text{ and } A[i] > \text{key}
\end{align*}

\begin{align*}
&A[i+1] \leftarrow A[i] \\
&i \leftarrow i - 1 \\
&A[i+1] \leftarrow \text{key}
\end{align*}

\[
\sum_{j=2}^{n+1} c_1 \\
\sum_{j=2}^{n} c_2 \\
\sum_{j=2}^{n} c_3 \\
\sum_{j=2}^{n} \sum_{i=0}^{j-1} c_4 \\
\sum_{j=2}^{n} \sum_{i=1}^{j-1} c_5 \\
\sum_{j=2}^{n} \sum_{i=1}^{j-1} c_6 \\
\sum_{j=2}^{n} c_7
\]

\text{Line 7}
Insertion sort (worst case)

- Total worst-case time is the sum of all those sums, which we can simplify:

\[ T_w(n) = \sum_{j=2}^{n+1} c_1 + \sum_{j=2}^{n} (c_2 + c_3 + c_7) + \sum_{j=2}^{n} \left( \sum_{i=0}^{j-1} c_4 + \sum_{i=1}^{j-1} (c_5 + c_6) \right) \]
Insertion sort (worst case)

- Total worst-case time is the sum of all those sums, which we can simplify:

\[ T_w(n) = \sum_{j=2}^{n+1} c_1 + \sum_{j=2}^{n} (c_2 + c_3 + c_7) + \sum_{j=2}^{n} (\sum_{i=0}^{j-1} c_4 + \sum_{i=1}^{j-1} (c_5 + c_6)) \]

\[ = nc_1 + (n-1)(c_2 + c_3 + c_7) + \sum_{j=2}^{n} c_4 + (j-1)(c_4 + c_5 + c_6) \]
Insertion sort (worst case)

- Total worst-case time is the sum of all those sums, which we can simplify:

\[ T_w(n) = \sum_{j=2}^{n+1} c_1 + \sum_{j=2}^{n} (c_2 + c_3 + c_7) + \sum_{j=2}^{n} \left( \sum_{i=0}^{j-1} c_4 + \sum_{i=1}^{j-1} (c_5 + c_6) \right) \]

\[ = nc_1 + (n-1)(c_2 + c_3 + c_7) + \sum_{j=2}^{n} c_4 + (j-1)(c_4 + c_5 + c_6) \]

\[ = n(c_1 + c_2 + c_3 + c_4 + c_7) - c_2 - c_3 - c_4 - c_7 + (c_4 + c_5 + c_6) \sum_{j=2}^{n} (j-1) \]
Insertion sort (worst case)

- Total worst-case time is the sum of all those sums, which we can simplify:

\[ T_w(n) = \sum_{j=2}^{n+1} c_1 + \sum_{j=2}^{n} (c_2+c_3+c_7) + \sum_{j=2}^{n} \left( \sum_{i=0}^{j-1} c_4 + \sum_{i=1}^{j-1} (c_5+c_6) \right) \]

\[ = nc_1 + (n-1)(c_2+c_3+c_7) + \sum_{j=2}^{n} c_4 + (j-1)(c_4+c_5+c_6) \]

\[ = n\left(c_1+c_2+c_3+c_4+c_7\right) - c_2 - c_3 - c_4 - c_7 + (c_4+c_5+c_6)\sum_{j=2}^{n} (j-1) \]

\[ = n\left(c_1+c_2+c_3+c_4+c_7\right) - c_2 - c_3 - c_4 - c_7 + (c_4+c_5+c_6)n(n-1)/2 \]
Insertion sort (worst case)

- Total worst-case time is the sum of all those sums, which we can simplify:

\[
T_w(n) = \sum_{j=2}^{n+1} c_1 + \sum_{j=2}^{n} (c_2 + c_3 + c_7) + \sum_{j=2}^{n} \left( \sum_{i=0}^{j-1} c_4 + \sum_{i=1}^{j-1} (c_5 + c_6) \right)
\]

\[
= n c_1 + (n-1)(c_2 + c_3 + c_7) + \sum_{j=2}^{n} c_4 + (j-1)(c_4 + c_5 + c_6)
\]

\[
= n(c_1 + c_2 + c_3 + c_4 + c_7) - c_2 - c_3 - c_4 - c_7 + (c_4 + c_5 + c_6) \sum_{j=2}^{n} (j-1)
\]

\[
= n(c_1 + c_2 + c_3 + c_4 + c_7) - c_2 - c_3 - c_4 - c_7 + (c_4 + c_5 + c_6)n(n-1)/2
\]

\[
= \frac{c_4 + c_5 + c_6}{2} n^2 + \left( c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7 \right) n - c_2 - c_3 - c_4 - c_7.
\]
Insertion sort (worst case)

- Total worst-case time is the sum of all those sums, which we can simplify:

\[
T_w(n) = \sum_{j=2}^{n+1} c_1 + \sum_{j=2}^{n} (c_2 + c_3 + c_7) + \sum_{j=2}^{n} (\sum_{i=0}^{j-1} c_4 + \sum_{i=1}^{j-1} (c_5 + c_6))
\]

\[
= nc_1 + (n-1)(c_2 + c_3 + c_7) + \sum_{j=2}^{n} c_4 + (j-1)(c_4 + c_5 + c_6)
\]

\[
= n(c_1 + c_2 + c_3 + c_4 + c_7) - c_2 - c_3 - c_4 - c_7 + (c_4 + c_5 + c_6) \sum_{j=2}^{n} (j-1)
\]

\[
= n(c_1 + c_2 + c_3 + c_4 + c_7) - c_2 - c_3 - c_4 - c_7 + (c_4 + c_5 + c_6) n(n-1)/2
\]

\[
= \frac{c_4 + c_5 + c_6}{2} n^2 + (c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7) n - c_2 - c_3 - c_4 - c_7.
\]

Quadratic
Insertion sort (best case)

- Best case occurs when the input is already sorted (line 4 test fails immediately).

\[ T_b(n) = \sum_{j=2}^{n+1} c_1 + \sum_{j=2}^{n} (c_2 + c_3 + c_7) + \sum_{j=2}^{n} c_4 \]

\[ = (c_1 + c_2 + c_3 + c_4 + c_7) n - c_2 - c_3 - c_4 - c_7. \]

- Getting the exact polynomial is fascinating and fun, but also slow, error-prone, and it gives us more detail than we need.

- Usually we care more about the polynomial degree than the exact coefficients.
Better: Asymptotic notation

• Intuitive idea: retain just the dominant term, disregard constants and slower-growing terms.

• Theoretical approach: define sets (“equivalence classes”) of functions.
  – Example: quadratic-growth functions, $\Theta(n^2)$.
  – Example: linear-growth functions, $\Theta(n)$.
  – Example: log-growth functions, $\Theta(\log n)$.
  – and many others.

• Usually easy to determine the eqv. class for a function, without resorting to step-counting.