create the sorted sequence in an incremental way
start with a sorted sequence of length 1 and insert
one more element in each iteration

INSERTION-SORT (A, n)
for j ←2 to n do
    key ←A[j]
    i ←j −1
    while i > 0 and A[i] > key do
        A[i+1] ←A[i]
        i ←i −1
    A[i+1] = key
    i
Which of the following claims about insertion sort are true?

**Claim 1:** On input 1,2,3,4,…,n the algorithm will finish after $O(n)$ steps

**Claim 2:** On input n,n-1,…,1 the algorithm will finish after $O(n^2)$ steps

**Claim 3:** On input 1,…,n/2, n,n-1,…n/2+1, the algorithm will finish after $O(n\sqrt{n})$ steps

A. All of the above. B. None of the above. C. Claim 1 only. D. Claim 3 only. E. Claims 1 and 2 only.
What is the running time of Insertion Sort?

Number of times the while-loop is executed depends on the input

- increasingly sorted input is fast; decreasing is slow.
- Worst case? \( \sum_{j=2}^{n} j < n^2 \)
- Average case?

What all do we count/have to count when analyzing time?

- In (internal) sorting algorithm we generally count the number of comparison
Pseudo code has two nested loops

- while loop moves left from j to 1
- total time won’t be more than quadratic.

Note: A doubly nested loop does not necessarily result in quadratic time

Worst case: \( T(n) = O(n^2) \)

- Work is bounded by summing the first \( n-1 \) integers which is equal to \( n(n-1)/2 \)
- Time is proportional to \( n^2 \)
- Also, \( T(n) = \Theta (n^2) \)
INSERTION-SORT (A, n)

for j ←2 to n do


key ← A[j]

i ← j − 1

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

A[i+1] ← A[i]

i ← i − 1

A[i+1] = key

INSERTION-SORT \((A, n)\)

for \(j \leftarrow 2\) to \(n\) do

- **Pre-Condition:** \(A[1] \leq A[2] \ldots \leq A[j - 1]\)
- \(key \leftarrow A[j]\)
- \(i \leftarrow j - 1\)

while \(i > 0\) and \(A[i] > key\) do

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

\(A[i+1] = key\)


Post-Condition when \(j=n\) \(\Rightarrow\) entire array \(A\) is sorted.

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

Define \(A_{\text{orig}}[1, \ldots, j-1] := A[1, \ldots, j-1]\)

while \(i > 0\) and \(A[i] > key\) do

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

Invariant: \(A[1, \ldots, i-1] = A_{\text{orig}}[1, \ldots, i-1]\) (untouched)
\(A[i+2, \ldots, j] = A_{\text{orig}}[i+1, \ldots, j-1]\) (shift once*)
\(key < A[i+2]\)

\(A[i+1] = key\)

Let $T(n)$ be a claim in which $n$ is a positive integer. Prove claim $T(n)$ correct.

**Common proof methods**
- Direct proof
- Indirect proof
  - By contraposition, by contradiction
- Mathematical induction
  - weak and strong induction
- Invariants
What do we need to prove in 381?

- Claim of a run time
  Could be a recurrence for a recursive solution, a sum for an iterative solution, an amortized analysis, etc.

- Correctness of your algorithm approach
  Often an inductive argument (even for iterative solutions) or a proof by contradiction

- NP-completeness of a problem

- Lower bound of a problem
Weak Induction

- Basis: $T(1)$ holds (basis is often 1 or 0, or a larger value)
- Induction hypothesis: for every $n>1$, assume $T(n-1)$ holds
- Using the induction hypothesis, show that $T(n)$ holds.
- It follows that $T(n)$ holds for all $n$ and the claim is proven.
Claim: $\sum_{i=1}^{n} i2^i = (n - 1)2^{n+1} + 2$

Basis for $n=1$: $2=0+2$ true

Assume claim holds for $n-1$: $\sum_{i=1}^{n-1} i2^i = (n - 2)2^n + 2$

Prove the claim for $n$:

\[
\sum_{i=1}^{n} i2^i = n2^n + \sum_{i=1}^{n-1} i2^i \quad \text{(Split the sum)}
\]
\[
= n2^n + (n - 2)2^n + 2 \quad \text{(Inductive Hyp)}
\]
\[
= (2n - 2)2^n + 2 \quad \text{(algebra)}
\]
\[
= 2(n - 1)2^n + 2 \quad \text{(algebra)}
\]
\[
= (n - 1)2^{n+1} + 2 \quad \text{(algebra)}
\]

QED
Strong Induction

- Basis: T(1) holds
- Induction hypothesis: For every \( n > 1 \), assume the claim holds for \( k = 1, 2, 3, \ldots, n-1 \)
- Using the induction hypothesis, show that T(n) holds.
- It follows that T(n) holds for all n and the claim is proven.
Claim: Prove that every binary tree with n nodes has n-1 edges

Let T(n) be the statement that any binary tree with n nodes has n-1 edges

Base Case: n=1  (check)

Inductive Hypothesis: For all j < n the statement T(j) holds

Inductive Step: Prove that T(n) holds
Let T be a tree with \( n > 1 \) nodes and let \( u \) be the root of the tree.

**Case 1:** \( u \) has 1 child \( v \)
- Let \( T_v \) be tree rooted at \( v \).
- Since, \( T_v \) has \( n-1 \) nodes, by IH \( T_v \) has \( n-2 \) edges.
- Total edges: \( 1 + n-2 = n-1 \)

**Case 2:** \( u \) has 2 children \( w \) and \( v \)
- Let \( T_w \) (resp. \( T_v \)) be the corresponding trees with \( n_w \) (resp. \( n_v \)) nodes with \( n_w + n_v = n - 1 \).
- By (Strong) IH \( T_w \) (resp. \( T_v \)) has \( n_w - 1 \) edges (resp. \( n_v - 1 \) edges).
- Total Edges: \( 2 + n_w - 1 + n_v - 1 = n_w + n_v = n - 1 \)

QED
Other Relevant/Common Sums

- $\sum_{i=1}^{n} i$
- $\sum_{i=1}^{n} i^2$
- $\sum_{i=1}^{n} i^k \; \text{k constant}$
- $\sum_{i=1}^{n} 2^i$
- $\sum_{k=1}^{n} x^k$

... 

Exact bounds and asymptotic bounds
CLRS, 182 Text, TCS Cheat Sheet, etc.
Base case $n=4$
- $4! = 24 > 2^4 = 16$

Induction hypothesis:
For any $n = k > 3$ it holds that $k! > 2^k$

Show the claim for $k+1$:
$(k + 1)! = (k + 1) \cdot k! > (k + 1) \cdot 2^k$ (by IH)
$> 2 \cdot 2^k$ (since $k > 3$)
$\geq 2^{k+1}$
Analysis of Algorithms

Worst case analysis
- in an asymptotic sense, the maximum time the algorithm takes on any input of size $n$.

Average case analysis
- expected time; often meaningful
- may need assumptions on the statistical distribution of input data

Best case analysis
- does not mean much; generally easy to determine

In some case, the three bounds are identical
- means performance does not depend on the value of the data
- For some algorithms, average case performance is only known experimentally.
What do we count?

- Time and space
  - time in terms of number of basic operations on basic data types

- Ignore machine dependent factors, but remain realistic

- Random Access Model (RAM)
  - no concurrency
  - count instructions (arithmetic operation, comparison, data movement)
  - each instruction takes constant time
  - realistic assumption on the size of the numbers (to represent n, it takes log n bits)
Asymptotic notation: Big-O

O(g(n)) = \{ f(n) | \text{ there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c g(n) \text{ or all } n \geq n_0 \}

We write \( f(n) = O(g(n)) \) if there exist constants \( c > 0, n_0 > 0 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \).

\[ 4n + 23\log n - 28 = O(n) \]
- Drops low-order terms
- Ignores leading constants
- May not hold for small values of \( n \)
$f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq c g(n)$ for all $n \geq n_0$.

$f(n) = 3n^2 - 4n + 512 \\ \leq 3n^2 + 512 \\ \leq 4n^2$ for $n \geq 23$

- $f(n) = O(n^2)$
- $f(n) = O(n^3)$ also holds
- $f(n) = O(n)$ is false

CLRS text Figure 3.1
Which statements are true?

\[ 3n^3 + 90n^2 - 5n = O(n^3) \]

\[ 3n^3 + 90n^2 - 5n = O(2^n) \]

\[ 3n^3 + 90n^2 - 5n = O(n^2) \]

\[ 5 \log n = O(n) \]

\[ \sqrt{n} = O(\log n^8) \]

\[ n \log n = O(n) \]

\[ 4n = O(n \log n) \]

\[ n / \log n = O(\sqrt{n}) \]
\[3n^3 + 90n^2 - 5n = O(n^3) \quad \text{true}\]
\[3n^3 + 90n^2 - 5n = O(2^n) \quad \text{true}\]
\[3n^3 + 90n^2 - 5n = O(n^2) \quad \text{false}\]
\[5 \log n = O(n) \quad \text{true}\]
\[\sqrt{n} = O(\log n^8) \quad \text{false}\]
\[n \log n = O(n) \quad \text{false}\]
\[4n = O(n \log n) \quad \text{true}\]
\[n / \log n = O(\sqrt{n}) \quad \text{false}\]
Consider two running times: $4n \log n$ and $8n^{n^{1/8}}$

Which relationships hold?

1. $4n \log n = O(8n^{n^{1/8}})$
2. $8n^{n^{1/8}} = O(4n \log n)$
3. $4n \log n = \Theta(8n^{n^{1/8}})$
4. $8n^{n^{1/8}} = \Theta(4n \log n)$

A. None  
B. 1  
C. 2  
D. 1 and 3  
E. 4
\[ O(g(n)) = \{ f(n) \mid \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \ g(n) \text{ for all } n \geq n_0 \} \]

\( O \) captures upper bounds

\[ \Theta(g(n)) = \{ f(n) \mid \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 \ g(n) \leq f(n) \leq c_2 \ g(n) \text{ for all } n \geq n_0 \} \]

\( \Theta \) captures upper and lower bounds
Examples

- $3n^3 + 90n^2 - 5n$ is $O(n^3)$ and $\Theta(n^3)$ is true
- $3n^3 + 90n^2 - 5n$ is $O(2^n)$ true, but $\Theta(2^n)$ false
- $5 \log n$ is $O(n)$ true, but $\Theta(n)$ false
- $4n = O(n \log n)$ is true, but $\Theta(n \log n)$ false
Asymptotic Bounds

\( \mathcal{O}(g(n)) = \{ f(n) \mid \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \, g(n) \text{ for all } n \geq n_0 \} \)

\( \mathcal{O} \) captures upper bounds

\( \Theta(g(n)) = \{ f(n) \mid \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 \, g(n) \leq f(n) \leq c_2 \, g(n) \text{ for all } n \geq n_0 \} \)

\( \Theta \) captures upper and lower bounds

\( \Omega(g(n)) = \{ f(n) \mid \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq c g(n) \leq f(n) \text{ for all } n \geq n_0 \} \)

\( \Omega \) captures lower bounds

\[ 4n \log n = \Omega(n) \]
We will generally assume that \( n \) is “nice”
- E.g., power of 2
- We are not implementing the algorithms and only need to consider crucial the boundary/special cases

When asked to design an efficient algorithm
- sometimes you will be given a target asymptotic bound
  - other times you need to find the “best” one

You can use known data structures
- State how they are implemented and give time bounds of operations
How many times is F called?

Assume \( n \) is a power of 4 (\( n = 4^k \))

```plaintext
while \( n > 1 \) do
  for \( i = 1 \) to \( n \) do
    F(i, n)
  \n  \( n = n/4 \)
```

\( \Theta(n \log n) \)

\( O(n \log n) \) \hspace{1cm} \( \Theta(n \log n) \)

\( O(n^2) \) \hspace{1cm} \( \Theta(n^2) \)

\( O(n) \) \hspace{1cm} \( \Theta(n) \)

\( O(\log n) \) \hspace{1cm} \( \Theta(\log n) \)
Assume $n$ is a power of 4 ($n=4^k$)

```
while $n > 1$ do
    for $i = 1$ to $n$ do
        F($i$, $n$)
    $n = n/4$
```

$O(n \log n)$  $\Theta(n \log n)$
$O(n^2)$  $\Theta(n^2)$
$O(n)$  $\Theta(n)$
$O(\log n)$  $\Theta(\log n)$

How many times is F called?
**Useful Fact: Geometric Series**

**Fact:** Suppose $0 < x < 1$ then $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$

**Proof:**

$$\sum_{i=0}^{\infty} x^i = \frac{1-x}{1-x} \sum_{i=0}^{\infty} x^i$$

$$= \frac{1}{1-x} \left( \sum_{i=0}^{\infty} x^i - \sum_{i=0}^{\infty} x^{i+1} \right)$$

$$= \frac{1}{1-x} \left( \sum_{i=0}^{\infty} x^i - \sum_{i=1}^{\infty} x^{i} \right) = \frac{1}{1-x}$$

**Example:** $x < \frac{1}{4}$ we have $\sum_{i=0}^{\infty} \left( \frac{1}{4} \right)^i = \frac{1}{1-\frac{1}{4}} = \frac{4}{3}$
Assume \( n \) is a power of 4 (\( n=4^k \))

\[
\text{while } n > 1 \text{ do} \\
\quad \text{for } i = 1 \text{ to } n \text{ do} \\
\quad \quad F(i, n) \\
\quad n = n/4
\]

- \( \Theta(n \log n) \)
- \( \Omega(n \log n) \)
- \( \Theta(n^2) \)
- \( \Omega(n^2) \)
- \( \Theta(n) \)
- \( \Omega(n) \)
- \( \Theta(\log n) \)
- \( \Omega(\log n) \)