Outline

1. Syllabus

2. Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3. Asymptotic Notations

4. Common Running times
Sorting Problem

**Input:** sequence of \( n \) numbers \((a_1, a_2, \cdots, a_n)\)

**Output:** a permutation \((a'_1, a'_2, \cdots, a'_n)\) of the input sequence such that \(a'_1 \leq a'_2 \leq \cdots \leq a'_n\)

Example:

- **Input:** 53, 12, 35, 21, 59, 15
- **Output:** 12, 15, 21, 35, 53, 59
At the end of $j$-th iteration, the first $j$ numbers are sorted.

- iteration 1: $53, 12, 35, 21, 59, 15$
- iteration 2: $12, 53, 35, 21, 59, 15$
- iteration 3: $12, 35, 53, 21, 59, 15$
- iteration 4: $12, 21, 35, 53, 59, 15$
- iteration 5: $12, 21, 35, 53, 59, 15$
- iteration 6: $12, 15, 21, 35, 53, 59$
Example:

- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

**insertion-sort(A, n)**

1: for $j \leftarrow 2$ to $n$ do
2:   $key \leftarrow A[j]$
3:   $i \leftarrow j - 1$
4:    while $i > 0$ and $A[i] > key$ do
5:       $A[i + 1] \leftarrow A[i]$
6:       $i \leftarrow i - 1$
7:    $A[i + 1] \leftarrow key$
Example:
- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

**insertion-sort(A, n)**

1: **for** $j \leftarrow 2$ to $n$ **do**
2: \hspace{1em} $key \leftarrow A[j]$
3: \hspace{1em} $i \leftarrow j - 1$
4: \hspace{1em} **while** $i > 0$ and $A[i] > key$ **do**
5: \hspace{2em} $A[i + 1] \leftarrow A[i]$
6: \hspace{2em} $i \leftarrow i - 1$
7: \hspace{1em} $A[i + 1] \leftarrow key$

- $j = 6$
- $key = 15$

12 21 35 53 59 15

↑

↓

$i$
Example:
- **Input:** 53, 12, 35, 21, 59, 15
- **Output:** 12, 15, 21, 35, 53, 59

**insertion-sort**($A, n$)

1: \textbf{for} $j \leftarrow 2$ \textbf{to} $n$ \textbf{do}

2: \hspace{1em} key $\leftarrow A[j]$

3: \hspace{1em} $i \leftarrow j - 1$

4: \hspace{1em} while $i > 0$ and $A[i] >$ key \textbf{do}

5: \hspace{2em} $A[i + 1] \leftarrow A[i]$

6: \hspace{2em} $i \leftarrow i - 1$

7: \hspace{1em} $A[i + 1] \leftarrow$ key

- $j = 6$
- key $= 15$

12 21 35 53 59 59
Example:

- **Input**: 53, 12, 35, 21, 59, 15
- **Output**: 12, 15, 21, 35, 53, 59

**insertion-sort**\( (A, n) \)

1: \textbf{for} \( j \leftarrow 2 \) \textbf{to} \( n \) \textbf{do}
2: \hspace{1em} \textit{key} \leftarrow A[j]
3: \hspace{1em} i \leftarrow j - 1
4: \hspace{1em} \textbf{while} \( i > 0 \) \textbf{and} \( A[i] > \textit{key} \) \textbf{do}
5: \hspace{2em} A[i + 1] \leftarrow A[i]
6: \hspace{1em} i \leftarrow i - 1
7: \hspace{1em} A[i + 1] \leftarrow \textit{key}

- \( j = 6 \)
- \( \textit{key} = 15 \)

<table>
<thead>
<tr>
<th>12</th>
<th>21</th>
<th>35</th>
<th>53</th>
<th>59</th>
<th>59</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\uparrow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\hspace{1em} ( i )</td>
</tr>
</tbody>
</table>
Example:

- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

**insertion-sort**($A, n$)

1: for $j \leftarrow 2$ to $n$ do
2:   $key \leftarrow A[j]$
3:   $i \leftarrow j - 1$
4:   while $i > 0$ and $A[i] > key$ do
5:     $A[i + 1] \leftarrow A[i]$
6:     $i \leftarrow i - 1$
7:   $A[i + 1] \leftarrow key$

- $j = 6$
- $key = 15$

12 21 35 53 53 59
Example:

- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

**insertion-sort**(A, n)

1: \textbf{for} \ j \leftarrow 2 \textbf{ to } n \ \textbf{do}
2: \quad \textit{key} \leftarrow A[j]
3: \quad i \leftarrow j - 1
4: \quad \textbf{while} \ i > 0 \ \textbf{and} \ A[i] > key \ \textbf{do}
5: \quad \quad A[i + 1] \leftarrow A[i]
6: \quad \quad i \leftarrow i - 1
7: \quad A[i + 1] \leftarrow \textit{key}
Example:
- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

**insertion-sort**(\(A, n\))

1: for \(j \leftarrow 2\) to \(n\) do
2:   \(key \leftarrow A[j]\)
3:   \(i \leftarrow j - 1\)
4:   while \(i > 0\) and \(A[i] > key\) do
5:     \(A[i + 1] \leftarrow A[i]\)
6:   \(i \leftarrow i - 1\)
7:   \(A[i + 1] \leftarrow key\)

- \(j = 6\)
- \(key = 15\)

12  21  35  35  53  59

↑

\(i\)
Example:

- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

insertion-sort\( (A, n) \)

1: \textbf{for } j \leftarrow 2 \textbf{ to } n \textbf{ do} \\
2: \quad \textit{key} \leftarrow A[j] \\
3: \quad i \leftarrow j - 1 \\
4: \quad \textbf{while } i > 0 \textbf{ and } A[i] > \textit{key} \textbf{ do} \\
5: \quad \quad A[i + 1] \leftarrow A[i] \\
6: \quad \quad i \leftarrow i - 1 \\
7: \quad A[i + 1] \leftarrow \textit{key} \\

- \textit{j} = 6 \\
- \textit{key} = 15 \\

12 \quad 21 \quad 35 \quad 35 \quad 53 \quad 59

\uparrow

i
Example:
- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

**insertion-sort**($A, n$)

1: for $j \leftarrow 2$ to $n$ do
2: \hspace{1em} key $\leftarrow A[j]$
3: \hspace{1em} $i \leftarrow j - 1$
4: \hspace{1em} while $i > 0$ and $A[i] > key$ do
5: \hspace{2em} $A[i + 1] \leftarrow A[i]$
6: \hspace{2em} $i \leftarrow i - 1$
7: \hspace{1em} $A[i + 1] \leftarrow key$

- $j = 6$
- $key = 15$

12 \hspace{1em} 21 \hspace{1em} 21 \hspace{1em} 35 \hspace{1em} 53 \hspace{1em} 59

$\uparrow$

$i$
Example:

- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

**insertion-sort**\((A, n)\)

1. **for** \(j \leftarrow 2\) **to** \(n\) **do**
2. \(key \leftarrow A[j]\)
3. \(i \leftarrow j - 1\)
4. **while** \(i > 0\) and \(A[i] > key\) **do**
5. \(A[i + 1] \leftarrow A[i]\)
6. \(i \leftarrow i - 1\)
7. \(A[i + 1] \leftarrow key\)

- \(j = 6\)
- \(key = 15\)

\[
\begin{array}{ccccccc}
12 & 21 & 21 & 35 & 53 & 59 \\
\uparrow & & & & & \\
& i & & & & \\
\end{array}
\]
Example:

- Input: 53, 12, 35, 21, 59, 15
- Output: 12, 15, 21, 35, 53, 59

insertion-sort(A, n)

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

- \( j = 6 \)
- \( \text{key} = 15 \)

12 15 21 35 53 59
Outline

1. Syllabus

2. Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3. Asymptotic Notations

4. Common Running times
Analysis of Insertion Sort

- Correctness
- Running time
Invariant: after iteration $j$ of outer loop, $A[1..j]$ is the sorted array for the original $A[1..j]$.

- after $j = 1: 53, 12, 35, 21, 59, 15$
- after $j = 2: 12, 53, 35, 21, 59, 15$
- after $j = 3: 12, 35, 53, 21, 59, 15$
- after $j = 4: 12, 21, 35, 53, 59, 15$
- after $j = 5: 12, 21, 35, 53, 59, 15$
- after $j = 6: 12, 15, 21, 35, 53, 59$
Analyzing Running Time of Insertion Sort

- Q1: what is the size of input?
Q1: what is the size of input?
A1: Running time as the function of size
Analyzing Running Time of Insertion Sort

Q1: what is the size of input?
A1: Running time as the function of size

possible definition of size:
- Sorting problem: \# integers,
- Greatest common divisor: total length of two integers
- Shortest path in a graph: \# edges in graph
Q1: what is the size of input?
A1: Running time as the function of size
possible definition of size:
- Sorting problem: \# integers,
- Greatest common divisor: total length of two integers
- Shortest path in a graph: \# edges in graph

Q2: Which input?
For the insertion sort algorithm: if input array is already sorted in ascending order, then algorithm runs much faster than when it is sorted in descending order.
Q1: what is the size of input?

A1: Running time as the function of size

possible definition of size:
- Sorting problem: # integers,
- Greatest common divisor: total length of two integers
- Shortest path in a graph: # edges in graph

Q2: Which input?

For the insertion sort algorithm: if input array is already sorted in ascending order, then algorithm runs much faster than when it is sorted in descending order.

A2: Worst-case analysis:
- Running time for size \( n \) = worst running time over all possible arrays of length \( n \)
Analyzing Running Time of Insertion Sort

- Q3: How fast is the computer?
- Q4: Programming language?
Q3: How fast is the computer?
Q4: Programming language?
A: They do not matter!
Q3: How fast is the computer?
Q4: Programming language?
A: They do not matter!

Important idea: asymptotic analysis

Focus on growth of running-time as a function, not any particular value.
Asymptotic Analysis: $O$-notation

Informal way to define $O$-notation:
- Ignoring lower order terms
- Ignoring leading constant
Asymptotic Analysis: $O$-notation

Informal way to define $O$-notation:

- Ignoring lower order terms
- Ignoring leading constant

$3n^3 + 2n^2 - 18n + 1028 \Rightarrow 3n^3 \Rightarrow n^3$
Asymptotic Analysis: $O$-notation

Informal way to define $O$-notation:

- Ignoring lower order terms
- Ignoring leading constant

- $3n^3 + 2n^2 - 18n + 1028 \Rightarrow 3n^3 \Rightarrow n^3$
- $3n^3 + 2n^2 - 18n + 1028 = O(n^3)$
Asymptotic Analysis: $O$-notation

Informal way to define $O$-notation:

- Ignoring lower order terms
- Ignoring leading constant

- $3n^3 + 2n^2 - 18n + 1028 \Rightarrow 3n^3 \Rightarrow n^3$
- $3n^3 + 2n^2 - 18n + 1028 = O(n^3)$
- $n^2/100 - 3n + 10 \Rightarrow n^2/100 \Rightarrow n^2$
Asymptotic Analysis: $O$-notation

Informal way to define $O$-notation:

- Ignoring lower order terms
- Ignoring leading constant

\[3n^3 + 2n^2 - 18n + 1028 \Rightarrow 3n^3 \Rightarrow n^3\]
\[3n^3 + 2n^2 - 18n + 1028 = O(n^3)\]
\[n^2/100 - 3n + 10 \Rightarrow n^2/100 \Rightarrow n^2\]
\[n^2/100 - 3n + 10 = O(n^2)\]
Asymptotic Analysis: $O$-notation

- $3n^3 + 2n^2 - 18n + 1028 = O(n^3)$
- $n^2/100 - 3n^2 + 10 = O(n^2)$
Asymptotic Analysis: $O$-notation

- $3n^3 + 2n^2 - 18n + 1028 = O(n^3)$
- $n^2/100 - 3n^2 + 10 = O(n^2)$

$O$-notation allows us to ignore
- architecture of computer
- programming language
- how we measure the running time: seconds or \# instructions?
Asymptotic Analysis: $O$-notation

- $3n^3 + 2n^2 - 18n + 1028 = O(n^3)$
- $n^2/100 - 3n^2 + 10 = O(n^2)$

$O$-notation allows us to ignore

- architecture of computer
- programming language
- how we measure the running time: seconds or # instructions?

To execute $a \leftarrow b + c$:

- program 1 requires 10 instructions, or $10^{-8}$ seconds
- program 2 requires 2 instructions, or $10^{-9}$ seconds
Asymptotic Analysis: $O$-notation

- $3n^3 + 2n^2 - 18n + 1028 = O(n^3)$
- $n^2/100 - 3n^2 + 10 = O(n^2)$

$O$-notation allows us to ignore
- architecture of computer
- programming language
- how we measure the running time: seconds or # instructions?

- to execute $a \leftarrow b + c$:
  - program 1 requires 10 instructions, or $10^{-8}$ seconds
  - program 2 requires 2 instructions, or $10^{-9}$ seconds
- they only change by a constant in the running time, which will be hidden by the $O(\cdot)$ notation
Algorithm 1 runs in time $O(n^2)$
Algorithm 2 runs in time $O(n)$
Algorithm 1 runs in time $O(n^2)$
Algorithm 2 runs in time $O(n)$
Does not tell which algorithm is faster for a specific $n$!
Asymptotic Analysis: $O$-notation

- Algorithm 1 runs in time $O(n^2)$
- Algorithm 2 runs in time $O(n)$
- Does not tell which algorithm is faster for a specific $n$!
- Algorithm 2 will eventually beat algorithm 1 as $n$ increases.
Algorithm 1 runs in time $O(n^2)$
Algorithm 2 runs in time $O(n)$

Does not tell which algorithm is faster for a specific $n$!
Algorithm 2 will eventually beat algorithm 1 as $n$ increases.

For Algorithm 1: if we increase $n$ by a factor of 2, running time increases by a factor of 4
Algorithm 1 runs in time $O(n^2)$
Algorithm 2 runs in time $O(n)$

Does not tell which algorithm is faster for a specific $n$!
Algorithm 2 will eventually beat algorithm 1 as $n$ increases.

For Algorithm 1: if we increase $n$ by a factor of 2, running time increases by a factor of 4
For Algorithm 2: if we increase $n$ by a factor of 2, running time increases by a factor of 2
Asymptotic Analysis of Insertion Sort

```latex
\begin{algorithm}
\textbf{insertion-sort}(A, n)
\begin{algorithmic}
\State 1: \textbf{for} $j \leftarrow 2$ to $n$ \textbf{do}
\State 2: \hspace{1em} \textit{key} $\leftarrow A[j]$
\State 3: \hspace{1em} $i \leftarrow j - 1$
\State 4: \hspace{1em} \textbf{while} $i > 0$ and $A[i] > \textit{key}$ \textbf{do}
\State 5: \hspace{2em} $A[i + 1] \leftarrow A[i]$
\State 6: \hspace{2em} $i \leftarrow i - 1$
\State 7: \hspace{1em} $A[i + 1] \leftarrow \textit{key}$
\end{algorithmic}
\end{algorithm}
```

Worst-case running time for iteration $j$ of the outer loop?

Answer: $O(j)$

Total running time = $\sum_{j=2}^{n} O(j) = O(n^2)$
Asymptotic Analysis of Insertion Sort

insertion-sort(A, n)

1: for j ← 2 to n do
2:    key ← A[j]
3:    i ← j − 1
4:    while i > 0 and A[i] > key do
6:        i ← i − 1
7:    A[i + 1] ← key

- Worst-case running time for iteration $j$ of the outer loop?
Asymptotic Analysis of Insertion Sort

**insertion-sort**\((A, n)\)

1. **for** \(j \leftarrow 2\) **to** \(n\) **do**
2. \(key \leftarrow A[j]\)
3. \(i \leftarrow j - 1\)
4. **while** \(i > 0\) **and** \(A[i] > key\) **do**
5. \(A[i + 1] \leftarrow A[i]\)
6. \(i \leftarrow i - 1\)
7. \(A[i + 1] \leftarrow key\)

- Worst-case running time for iteration \(j\) of the outer loop?
  Answer: \(O(j)\)
Asymptotic Analysis of Insertion Sort

**insertion-sort**($A, n$)

1: for $j \leftarrow 2$ to $n$ do
2: \hspace{1em} key $\leftarrow A[j]$
3: \hspace{1em} $i \leftarrow j - 1$
4: while $i > 0$ and $A[i] > key$ do
5: \hspace{2em} $A[i + 1] \leftarrow A[i]$
6: \hspace{2em} $i \leftarrow i - 1$
7: \hspace{1em} $A[i + 1] \leftarrow key$

- Worst-case running time for iteration $j$ of the outer loop?
  Answer: $O(j)$

- Total running time $= \sum_{j=2}^{n} O(j) = O(\sum_{j=2}^{n} j)$
  \[ = O\left(\frac{n(n+1)}{2} - 1\right) = O(n^2) \]
Computation Model

Random-Access Machine (RAM) model

A \[ j \] takes \( O(1) \) time.

Basic operations such as addition, subtraction and multiplication take \( O(1) \) time.

Each integer (word) has \( c \log n \) bits, \( c \geq 1 \) large enough. 

Reason: often we need to read the integer \( n \) and handle integers within range \([n^c, n^c]\), it is convenient to assume this takes \( O(1) \) time.

What is the precision of real numbers?

Most of the time, we only consider integers.

Can we do better than insertion sort asymptotically?

Yes: merge sort, quicksort and heap sort take \( O(n \log n) \) time.
Computation Model

- Random-Access Machine (RAM) model
- reading and writing $A[j]$ takes $O(1)$ time

Basic operations such as addition, subtraction and multiplication take $O(1)$ time. Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough.

Reason: often we need to read the integer $n$ and handle integers within range $[n^c, n^c]$, it is convenient to assume this takes $O(1)$ time.

What is the precision of real numbers? Most of the time, we only consider integers.

Can we do better than insertion sort asymptotically? Yes: merge sort, quicksort and heap sort take $O(n \log n)$ time.
Computation Model

- Random-Access Machine (RAM) model
  - reading and writing $A[j]$ takes $O(1)$ time
- Basic operations such as addition, subtraction and multiplication take $O(1)$ time
Computation Model

- Random-Access Machine (RAM) model
  - reading and writing $A[j]$ takes $O(1)$ time
- Basic operations such as addition, subtraction and multiplication take $O(1)$ time
- Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough
  - Reason: often we need to read the integer $n$ and handle integers within range $[-n^c, n^c]$, it is convenient to assume this takes $O(1)$ time.
Computation Model

- Random-Access Machine (RAM) model
  - reading and writing $A[j]$ takes $O(1)$ time
- Basic operations such as addition, subtraction and multiplication take $O(1)$ time
- Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough
  - Reason: often we need to read the integer $n$ and handle integers within range $[-n^c, n^c]$, it is convenient to assume this takes $O(1)$ time.
- What is the precision of real numbers?
Computation Model

- Random-Access Machine (RAM) model
- reading and writing $A[j]$ takes $O(1)$ time
- Basic operations such as addition, subtraction and multiplication take $O(1)$ time
- Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough
  - Reason: often we need to read the integer $n$ and handle integers within range $[-n^c, n^c]$, it is convenient to assume this takes $O(1)$ time.
- What is the precision of real numbers?
  - Most of the time, we only consider integers.
Computation Model

- Random-Access Machine (RAM) model
  - reading and writing $A[j]$ takes $O(1)$ time
- Basic operations such as addition, subtraction and multiplication take $O(1)$ time
- Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough
  - Reason: often we need to read the integer $n$ and handle integers within range $[-n^c, n^c]$, it is convenient to assume this takes $O(1)$ time.
- What is the precision of real numbers?
  - Most of the time, we only consider integers.
- Can we do better than insertion sort asymptotically?
Computation Model

- Random-Access Machine (RAM) model
  - reading and writing $A[j]$ takes $O(1)$ time
- Basic operations such as addition, subtraction and multiplication take $O(1)$ time
- Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough
  - Reason: often we need to read the integer $n$ and handle integers within range $[-n^c, n^c]$, it is convenient to assume this takes $O(1)$ time.
- What is the precision of real numbers?
  - Most of the time, we only consider integers.
- Can we do better than insertion sort asymptotically?
- Yes: merge sort, quicksort and heap sort take $O(n \log n)$ time
Outline

1. Syllabus

2. Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3. Asymptotic Notations

4. Common Running times
Asymptotically Positive Functions

Def. $f : \mathbb{N} \rightarrow \mathbb{R}$ is an asymptotically positive function if:

- $\exists n_0 > 0$ such that $\forall n > n_0$ we have $f(n) > 0$
Asymptotically Positive Functions

Def. $f : \mathbb{N} \rightarrow \mathbb{R}$ is an asymptotically positive function if:

- $\exists n_0 > 0$ such that $\forall n > n_0$ we have $f(n) > 0$

- In other words, $f(n)$ is positive for large enough $n$. 

We only consider asymptotically positive functions.
Def. $f : \mathbb{N} \rightarrow \mathbb{R}$ is an asymptotically positive function if:

- $\exists n_0 > 0$ such that $\forall n > n_0$ we have $f(n) > 0$

- In other words, $f(n)$ is positive for large enough $n$.

- $n^2 - n - 30$
Asymptotically Positive Functions

**Def.** \( f : \mathbb{N} \rightarrow \mathbb{R} \) is an asymptotically positive function if:

- \( \exists n_0 > 0 \) such that \( \forall n > n_0 \) we have \( f(n) > 0 \)

- In other words, \( f(n) \) is positive for large enough \( n \).

- \( n^2 - n - 30 \) Yes
Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \rightarrow \mathbb{R}$ is an asymptotically positive function if:
- $\exists n_0 > 0$ such that $\forall n > n_0$ we have $f(n) > 0$

- In other words, $f(n)$ is positive for large enough $n$.
- $n^2 - n - 30$ Yes
- $2^n - n^{20}$ No

We only consider asymptotically positive functions.
Asymptotically Positive Functions

**Def.** \( f : \mathbb{N} \rightarrow \mathbb{R} \) is an asymptotically positive function if:

- \( \exists n_0 > 0 \) such that \( \forall n > n_0 \) we have \( f(n) > 0 \)

- In other words, \( f(n) \) is positive for large enough \( n \).
- \( n^2 - n - 30 \quad \text{Yes} \)
- \( 2^n - n^{20} \quad \text{Yes} \)
Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \rightarrow \mathbb{R}$ is an *asymptotically positive function* if:

- $\exists n_0 > 0$ such that $\forall n > n_0$ we have $f(n) > 0$

- In other words, $f(n)$ is positive for large enough $n$.
- $n^2 - n - 30$ \hspace{1cm} Yes
- $2^n - n^{20}$ \hspace{1cm} Yes
- $100n - n^2/10 + 50$?
Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \rightarrow \mathbb{R}$ is an asymptotically positive function if:

- $\exists n_0 > 0$ such that $\forall n > n_0$ we have $f(n) > 0$

- In other words, $f(n)$ is positive for large enough $n$.

- $n^2 - n - 30$ \quad Yes

- $2^n - n^{20}$ \quad Yes

- $100n - n^2/10 + 50$? \quad No
Def. \( f : \mathbb{N} \rightarrow \mathbb{R} \) is an asymptotically positive function if:

- \( \exists n_0 > 0 \) such that \( \forall n > n_0 \) we have \( f(n) > 0 \)

- In other words, \( f(n) \) is positive for large enough \( n \).
- \( n^2 - n - 30 \) \quad Yes
- \( 2^n - n^{20} \) \quad Yes
- \( 100n - n^2/10 + 50? \) \quad No

We only consider asymptotically positive functions.
**O-Notation: Asymptotic Upper Bound**

**O-Notation** For a function $g(n)$,

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.$$
O-Notation: Asymptotic Upper Bound

**O-Notation** For a function $g(n)$,

$$O(g(n)) = \{\text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0\}.$$

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c > 0$ and every large enough $n$. 
**$O$-Notation: Asymptotic Upper Bound**

**$O$-Notation** For a function $g(n)$,

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.$$ 

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c > 0$ and every large enough $n$. 

![Graph showing $f(n) = O(g(n))$ and $cg(n)$](chart.png)
**O-Notation: Asymptotic Upper Bound**

**O-Notation**  For a function \( g(n) \),

\[
O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}. 
\]

- In other words, \( f(n) \in O(g(n)) \) if \( f(n) \leq cg(n) \) for some \( c > 0 \) and every large enough \( n \).
- \( 3n^2 + 2n \in O(n^2 - 10n) \)
**O-Notation: Asymptotic Upper Bound**

**O-Notation**  For a function \( g(n) \),

\[
O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } \quad f(n) \leq cg(n), \forall n \geq n_0 \}. 
\]

- In other words, \( f(n) \in O(g(n)) \) if \( f(n) \leq cg(n) \) for some \( c > 0 \) and every large enough \( n \).
- \( 3n^2 + 2n \in O(n^2 - 10n) \)

**Proof.**

Let \( c = 4 \) and \( n_0 = 50 \), for every \( n > n_0 = 50 \), we have,

\[
3n^2 + 2n - c(n^2 - 10n) = 3n^2 + 2n - 4(n^2 - 10n) 
\]

\[
= -n^2 + 42n \leq 0.
\]

\[
3n^2 + 2n \leq c(n^2 - 10n)
\]
**O-Notation**  For a function $g(n)$, 

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}. $$

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.
- $3n^2 + 2n \in O(n^2 - 10n)$
**O-Notation**  For a function $g(n)$,

$$O(g(n)) = \{\text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0\}.$$ 

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.
- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
**O-Notation** For a function $g(n)$,

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.$$  

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.
- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
- $n^{100} \in O(2^n)$
**O-Notation**  For a function $g(n)$,

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.$$ 

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.
- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
- $n^{100} \in O(2^n)$
- $n^3 \notin O(10n^2)$
**O-Notation**  For a function \( g(n) \),
\[
O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.
\]

- In other words, \( f(n) \in O(g(n)) \) if \( f(n) \leq cg(n) \) for some \( c \) and large enough \( n \).
- \( 3n^2 + 2n \in O(n^2 - 10n) \)
- \( 3n^2 + 2n \in O(n^3 - 5n^2) \)
- \( n^{100} \in O(2^n) \)
- \( n^3 \notin O(10n^2) \)

<table>
<thead>
<tr>
<th>Asymptotic Notations</th>
<th>( O )</th>
<th>( \Omega )</th>
<th>( \Theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>( \leq )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We use “$f(n) = O(g(n))$” to denote “$f(n) \in O(g(n))$”
Conventions

- We use \( f(n) = O(g(n)) \) to denote \( f(n) \in O(g(n)) \)
- \( 3n^2 + 2n = O(n^3 - 10n) \)
- \( 3n^2 + 2n = O(n^2 + 5n) \)
- \( 3n^2 + 2n = O(n^2) \)
- \( n^2 \) is asymmetric! Following equalities are wrong:
  \( O(n^3 - 10n) = 3n^2 + 2n \)
  \( O(n^2 + 5n) = 3n^2 + 2n \)
  \( O(n^2) = 3n^2 + 2n \)
Conventions

- We use \( f(n) = O(g(n)) \) to denote \( f(n) \in O(g(n)) \)
- \( 3n^2 + 2n = O(n^3 - 10n) \)
- \( 3n^2 + 2n = O(n^2 + 5n) \)
- \( 3n^2 + 2n = O(n^2) \)

“=” is asymmetric! Following equalities are wrong:
- \( O(n^3 - 10n) = 3n^2 + 2n \)
- \( O(n^2 + 5n) = 3n^2 + 2n \)
- \( O(n^2) = 3n^2 + 2n \)
Conventions

- We use \( f(n) = O(g(n)) \) to denote \( f(n) \in O(g(n)) \)
- \( 3n^2 + 2n = O(n^3 - 10n) \)
- \( 3n^2 + 2n = O(n^2 + 5n) \)
- \( 3n^2 + 2n = O(n^2) \)

“\( = \)” is asymmetric! Following equalities are wrong:
- \( O(n^3 - 10n) = 3n^2 + 2n \)
- \( O(n^2 + 5n) = 3n^2 + 2n \)
- \( O(n^2) = 3n^2 + 2n \)

Analogy: Mike is a student. A student is Mike.
\textbf{\textit{O}}-Notation  For a function $g(n)$,
\[ O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}. \]

\textbf{\textit{\Omega}}-Notation  For a function $g(n)$,
\[ \Omega(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0 \}. \]
**Ω-Notation: Asymptotic Lower Bound**

**O-Notation** For a function \(g(n)\),

\[
O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.
\]

**Ω-Notation** For a function \(g(n)\),

\[
Ω(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0 \}.
\]

- In other words, \(f(n) \in Ω(g(n))\) if \(f(n) \geq cg(n)\) for some \(c\) and large enough \(n\).
\(\Omega\)-Notation: Asymptotic Lower Bound

\[\Omega(g(n)) = \{\text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0\}.\]
Again, we use “=” instead of $\in$.

- $4n^2 = \Omega(n - 10)$
- $3n^2 - n + 10 = \Omega(n^2 - 20)$
Again, we use “=” instead of $\in$.

- $4n^2 = \Omega(n - 10)$
- $3n^2 - n + 10 = \Omega(n^2 - 20)$

<table>
<thead>
<tr>
<th>Asymptotic Notations</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td>$\geq$</td>
<td></td>
</tr>
</tbody>
</table>

$\Omega$-Notation: Asymptotic Lower Bound
- Notation: Asymptotic Lower Bound

Again, we use “≡” instead of $\in$.

- $4n^2 = \Omega(n - 10)$
- $3n^2 - n + 10 = \Omega(n^2 - 20)$

| Asymptotic Notations | $O$ | $\Omega$ | $\Theta$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td>$\geq$</td>
<td></td>
</tr>
</tbody>
</table>

**Theorem** $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$. 
\(\Theta\)-Notation: Asymptotic Tight Bound

\(\Theta\)-Notation  For a function \(g(n)\),

\[ \Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}. \]
**Θ-Notation: Asymptotic Tight Bound**

**Θ-Notation**  For a function \( g(n) \),

\[
\Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}. 
\]

- \( f(n) = \Theta(g(n)) \), then for large enough \( n \), we have \( f(n) \approx g(n) \).
**Θ-Notation: Asymptotic Tight Bound**

**Θ-Notation**  For a function $g(n)$,

$$\Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}.$$ 

- $f(n) = \Theta(g(n))$, then for large enough $n$, we have “$f(n) \approx g(n)$”. 

![Diagram showing the relationship between $f(n)$, $c_1 g(n)$, and $c_2 g(n)$, with $n_0$ marking the point where $f(n) = \Theta(g(n))$.](image-url)
**Θ-Notation: Asymptotic Tight Bound**

**Θ-Notation**  For a function \( g(n) \),

\[
\Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}.
\]

- \( 3n^2 + 2n = \Theta(n^2 - 20n) \)
\( \Theta \)-Notation: Asymptotic Tight Bound

**\( \Theta \)-Notation**  For a function \( g(n) \),

\[
\Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}.
\]

- \( 3n^2 + 2n = \Theta(n^2 - 20n) \)
- \( 2^{n/3} + 100 = \Theta(2^{n/3}) \)
Θ-Notation: Asymptotic Tight Bound

**Θ-Notation** For a function \( g(n) \),

\[
\Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}. 
\]

- \( 3n^2 + 2n = \Theta(n^2 - 20n) \)
- \( 2^{n/3+100} = \Theta(2^{n/3}) \)

<table>
<thead>
<tr>
<th>Asymptotic Notations</th>
<th>( O )</th>
<th>( \Omega )</th>
<th>( \Theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>( \leq )</td>
<td>( \geq )</td>
<td>( = )</td>
</tr>
</tbody>
</table>
**Θ-Notation: Asymptotic Tight Bound**

**Θ-Notation**  For a function $g(n)$,

$$\Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}.$$  

- $3n^2 + 2n = \Theta(n^2 - 20n)$
- $2^{n/3 + 100} = \Theta(2^{n/3})$

<table>
<thead>
<tr>
<th>Asymptotic Notations</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td>$\geq$</td>
<td>=</td>
</tr>
</tbody>
</table>

**Theorem**  $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. 