• First, there may exist some algorithms for the same problem, which are supposed to be correct.
• Then we compare these algorithms.
• The first question is, Which one is more efficient? (Why?)
• We focus on the growth rate of the running time or space requirement as a function of the input size $n$, denoted by $f(n)$. 
\textbf{O-notation}^{1}

\begin{itemize}
  \item In math, \emph{O-notation} describes the \textit{limiting behavior} of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions.
  \item \( f(n) \in O(g(n)) \) as \( n \to \infty \) if and only if there is a constant \( c > 0 \) and a real number \( n_0 \) such that
    \[
    |f(n)| \leq c|g(n)| \quad \forall n \geq n_0. \tag{1}
    \]
  \item Note that \( O(g(n)) \) is a set featured by some simple function \( g(n) \).
  \item Hence \( f(n) \in O(g(n)) \) is equivalent to say that \( f(n) \) is one instance of \( O(g(n)) \).
\end{itemize}

\textsuperscript{1}See any textbook for data structures and algorithms or https://en.wikipedia.org/wiki/Big_O_notation.
• For example, $8n^2 - 3n + 4 \in O(n^2)$.
• We could say that $8n^2 - 3n + 4 \in O(n^3)$ and $8n^2 - 3n + 4 \notin O(n)$. 
Common Fundamental Functions

<table>
<thead>
<tr>
<th>constant</th>
<th>logarithm</th>
<th>linear</th>
<th>n-log-n</th>
<th>quadratic</th>
<th>cubic</th>
<th>exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>log$n$</td>
<td>$n$</td>
<td>$n \log n$</td>
<td>$n^2$</td>
<td>$n^3$</td>
<td>$a^n$</td>
</tr>
</tbody>
</table>

See Table 4.1 and Figure 4.2 in Goodrich and etc, p. 161.
• We use $O$-notation to describe the asymptotic\textsuperscript{3} upper bound of complexity of the algorithm.

• So $O$-notation is widely used to classify algorithms by how they respond to changes in its input size.\textsuperscript{4}
  
  • Time complexity
  • Space complexity

• Note that we often make a trade-off between time and space.
  • Unlike time, we can reuse memory.

\textsuperscript{3}The asymptotic sense is that the input size $n$ grows toward infinity.

\textsuperscript{4}Actually, there are $\Theta$, $\theta$, $o$, $\Omega$, and $\omega$ which are used to classify algorithms.
References

- https://en.wikipedia.org/wiki/Game_complexity
class Lecture5 {
    "Arrays"
}
Arrays

An array stores a large collection of data which is of the same type.

```java
... // assume the size variable exists above
T[] A = new T[size];
// this creates an array of T type, referenced by A
...
```

- **T** can be any data type.
- This statement comprises two parts:
  - Declaring a reference
  - Creating an array
Variable Declaration for Arrays

- In the left-hand side, it is a declaration for an array variable, which does not allocate real space for the array.
- In reality, this variable occupies only a certain space for the reference to an array.\(^5\)
- If a reference variable does not refer to an array, the value of the variable is null.\(^6\)
- In this case, you cannot assign elements to this array variable unless the array object has already been created.

\(^5\)Recall the stack and the heap in the memory layout.
\(^6\)Moreover, this holds for any reference variable. For example, the **Scanner** type.
Creating A Real Array

- All arrays of Java are objects.
- As seen before, the `new` operator returns the memory address of that object.
  - Recall that the type of reference variables must be compatible to that of the array object.
- The variable `size` must be a positive integer for the number of elements.
- Note that the size of an array cannot be changed after the array is created.\(^7\)

\(^7\)Alternatively, you may try the class `ArrayList`, which is more useful in practice.
The array is allocated **contiguously** in the memory.

All arrays are **zero-based indexing**. (Why?)

So we have A[0], A[1], and A[2].

---

8 Same in C, C++, python, Javascript, and more.
Array Initializer

The elements of arrays are initialized once created.

- By default, every element is assigned as follows:
  - 0 for all numeric primitive data types
  - \u0000 for char type
  - false for boolean type

- An array can also be initialized by enumerating all the elements without using the new operator.

- For example,

```java
int[] A = {1, 2, 3};
```
Processing Arrays

When processing array elements, we often use *for* loops.

- Recall that arrays are objects.
- They have an attribute called *length* which records the size of the arrays.
  - For example, use `A.length` to get the size of `A`.
- Since the size of the array is known, it is natural to use a *for* loop to manipulate with the array.
Many Examples

Initialization of arrays by a Scanner object

```java
... // let x be an integer array with a certain size
for (int i = 0; i < A.length; ++i) {
    A[i] = input.nextInt();
}
...
```

Initialization of arrays by random numbers

```java
... for (int i = 0; i < A.length; ++i) {
    A[i] = (int) (Math.random() * 10);
}
...```
Display of array elements

```java
... for (int i = 0; i < A.length; ++i) {
    System.out.printf("%3d", A[i]);
}
...```

Sum of array elements

```java
... int sum = 0;
for (int i = 0; i < A.length; ++i) {
    sum += A[i];
}
...```
Extreme value problems of array elements

```java
... int max = A[0]; int min = A[0]; for (int i = 1; i < A.length; ++i) {
    if (max < A[i]) max = A[i];
    if (min > A[i]) min = A[i];
} ...
```

• How about the location of the extreme values?
Shuffling over array elements

... 

```java
for (int i = 0; i < A.length; ++i) {
    // choose j randomly
    int j = (int) (Math.random() * A.length);
    // swap
    int tmp = A[i];
    A[i] = A[j];
    A[j] = tmp;
}
...
```

- How to **swap** values of two variables without `tmp`?
- However, this naive algorithm is biased.\(^9\)

Exercise

Deck of Cards

Write a program which picks first 5 cards at random from a deck of 52 cards.

- 4 suits: Spade, Heart, Diamond, Club
- 13 ranks: 3, . . . , 10, J, Q, K, A, 2
- Label 52 cards by 0, 1, · · · , 51
- Shuffle the numbers
- Deal the first 5 cards
String[] suits = {"Spade", "Heart", "Diamond", "Club"};

String[] ranks = {

int size = 52;
int[] deck = new int[size];
for (int i = 0; i < deck.length; i++)
    deck[i] = i;

// shuffle over deck; correct version
for (int i = 0; i < size - 1; i++) {
    int j = (int) (Math.random() * (size - i)) + i;
    int z = deck[i];
    deck[i] = deck[j];
    deck[j] = z;
}

for (int i = 0; i < 5; i++) {
    String suit = suits[deck[i] / 13];
    String rank = ranks[deck[i] % 13];
    System.out.printf("%8s%3s\n", suit, rank);
}

...