Analysis of Algorithms

- First, there may exist some algorithms for the same problem.
- Then we compare these algorithms: Which one is more efficient?
- We focus on time complexity and space complexity.
- To do so, we need to estimate the growth rate of running time or space usage as a function of the input size $n$. 
Big-O

• In math, the notation Big-O describes the limiting behavior of a function when the argument approaches infinity, usually in terms of simple functions.

• Now we define \( f(n) \in O(g(n)) \) as \( n \to \infty \) if and only if there is a constant \( c > 0 \) and a real number \( n_0 \) such that

\[
|f(n)| \leq c|g(n)| \quad \forall n \geq n_0. \tag{1}
\]

• \( O(g(n)) \) is a set featured by \( g(n) \).

• Hence \( f(n) \in O(g(n)) \) is equivalent to say that \( f(n) \) is one instance of \( O(g(n)) \).

---

1See https://en.wikipedia.org/wiki/Big_O_notation.
• This is used for the asymptotic upper bound of complexity of the algorithm.
• In layman’s term, Big-O describes the worst case of this algorithm.
For example, $8n^2 - 3n + 4 \in O(n^2)$.

- For large $n$, you could ignore the last two terms.
- It is easy to find a constant $c > 0$ so that $cn > 8n^2$, say $c = 9$.

Also, $8n^2 - 3n + 4 \in O(n^3)$ but we seldom say this. (Why?)

However, $8n^2 - 3n + 4 \not\in O(n)$. (Why?)

What is this analysis related to the program?

Any insight?
Common Simple 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.
Some Interesting Facts

• We often make a trade-off between time and space.
  • Unlike time, we can reuse memory.
  • Users are sensitive to time.

• Playing game well is hard.³

• Earn money? Try to solve one of Millennium Prize Problems.⁴

class Lecture5 {

    "Arrays"

}
Arrays

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

```java
... // Assume that the size is known.
T[] A = new T[size];
// This creates an T-type array of size, referenced by A.
...
```

- **T** can be any data type.
- The variable size must be a nonnegative integer for the capacity of arrays.
- Now we proceed to study Line 3, by treating it in two stages.
Stage 1: Array Creation

- First we focus on the right-hand side of Line 3.
- In Java, all arrays are objects.
- Recall the stack and the heap in the memory layout.
- So all elements stored in arrays are located at the heap.
- As seen before, the new operator returns the memory address of the array.
- Note that the size of an array cannot be changed after the array is created.\(^5\)

\(^5\)Alternatively, you may try the class **ArrayList**, which is more useful in practice.
In the left-hand side, we declare a reference to the array, for example, the variable A in Line 3.

Note that A is not the array.

You can read from and write to the array via this reference.

All arrays are zero-based indexing.\(^6\) (Why?)

\(^6\)Same in C, C++, C#, Python, JavaScript, and more.
Array in Memory

- The array is allocated **contiguously** in the memory.

```
int[] A = new int[3];
```
Array Initialization

- 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 array.
  - For example, A.length.
- Since the size is known, it is natural to use a for loop to manipulate the array.
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);
}
...```

Zheng-Liang Lu Java Programming 166 / 171
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 values in the array

```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?
- Can you find the 2nd max of A?
- Can you keep the first $k$ max of A?
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.\(^7\)

\(^7\)See https://blog.codinghorror.com/the-danger-of-naivety/
Exercise

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("%-3s%8s\n", rank, suit);
}

...