• We use $O$-notation to describe the asymptotic\(^1\) 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.\(^2\)
  - Time complexity
  - Space complexity

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

---

\(^1\)The asymptotic sense is that the input size $n$ grows toward infinity.

\(^2\)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.

```
...  
// 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.\(^3\)
- If a reference variable does not refer to an array, the value of the variable is **null**.\(^4\)
- In this case, you cannot assign elements to this array variable unless the array object has already been created.

---

\(^3\) Recall the **stack** and the **heap** in the memory layout.

\(^4\) 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.\(^5\)

\(^5\)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?)


---

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
  - \000 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.
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();
}
...`
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-naivete/](https://blog.codinghorror.com/the-danger-of-naivete/)
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 = {
    "3", "4", "5", "6", "7",
    "8", "9", "10", "J", "Q", "K",
    "A", "2"
};

int size = 52;
ing[] 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);
}

...
Cloning Arrays

• In practice, one might duplicate an array for some reason.
• One could attempt to use the assignment statement (=), for example,

```
... T[] A = {...}; // assume A is an array
T[] B = A;  // shallow copy; you don't have a new array
...
```

• However, this is impossible to make two distinct arrays.
• Recall that the array variables are simply references to the arrays in the heap.
Moreover, all the reference variables share this property!

For example,
Use a loop to copy individual elements one by one.

```java
int[] A = {2, 1, 3, 5, 10};

// deep copy
int[] B = new int[A.length];
for (int i = 0; i < A.length; ++i) {
    B[i] = A[i];
}
```

Alternatively, you may use the `arraycopy` method in the `System` class.

```java
System.arraycopy(A, 0, B, 0, A.length);
```
for-each Loops

• A for-each loop is designed to iterate over a collection of objects, such as arrays and other data structures, in strictly sequential fashion, from start to finish.

• For example,

```java
... T[] A = {...}; // assume some T-type array
for (T element: A) {
    // body
}
...
```

• Note that the type \( T \) should be compatible to the element type of \( A \).

---

\(^8\)Beginning with JDK5.

Zheng-Liang Lu

Java Programming
Example

```java
... int[] A = {1, 2, 3}; int sum = 0; for (int i = 0; i < A.length; ++i) sum += A[i]; ...
```

- Not only is the syntax streamlined, but it also prevents boundary errors.

```java
... int[] A = {1, 2, 3}; int sum = 0; for (int item: A) sum += item; ...
```
A data structure is a particular way of organizing data in a program so that it can be used efficiently.

Data structures can implement one or more particular abstract data types (ADT), which specify the operations that can be performed on a data structure and the computational complexity of those operations.

In comparison, a data structure is a concrete implementation of the specification provided by some ADT.

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks.\(^9\)

\(^9\)See http://bigocheatsheet.com/.
Common Operations on Data

- A specific data structure is chosen in one problem.
- Then the operations are implemented accordingly.
- The **Arrays** class contains useful methods for common array operations such as **sorting** and **searching**.
- For example,

```java
import java.util.Arrays;

...  
    int[] A = {5, 2, 8};
    Arrays.sort(A); // sort the whole array

    char[] B = {'A', 'r', 't', 'h', 'u', 'r'};
    Arrays.sort(B, 1, 4); // sort the array partially

...  
```
Selection Sort

// selection sort: \(O(n^2)\) time

```java
for (int i = 0; i < A.length; i++) {
    int k = i; // the position of min starting from i
    for (int j = i + 1; j < A.length; j++) {
        if (A[k] > A[j])
            k = j;
    }
    // swap(A[i], A[k])
    int tmp = A[k];
    A[k] = A[i];
    A[i] = tmp;
}
```

- Time complexity: \(O(n^2)\)
- You can find more sorting algorithms.\(^{10}\)

\(^{10}\)See http://visualgo.net/.
Write a program which searches for the index associated with the key.

- For convenience, assume that there is no duplicate key.
- The linear search approach compares the key with each element in the array sequentially.
// assume A is an array
// linear search
for (int i = 0; i < A.length; i++) {
    if (A[i] == key) {
        System.out.printf("%3d", i);
    }
}

• Time complexity: $O(n)$
Alternative: Binary Search

- Time complexity: $O(\log n)$
- Overall time complexity (sorting + searching): still $O(\log n)$?

---

**List:**

- **key is 11:**
  - $\text{low: } [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$  
  - $\text{mid: } [11]$  
  - $\text{high: } [12]$  

- **key < 50:**
  - $\text{list: } [2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79]$  
  - $\text{low: } [0]$  
  - $\text{mid: } [1]$  
  - $\text{high: } [2]$  
  - $\text{low: } [0]$  
  - $\text{mid: } [1]$  
  - $\text{high: } [2]$  

- **key > 7:**
  - $\text{list: } [2, 4, 7, 10, 11, 45]$  
  - $\text{low: } [0]$  
  - $\text{mid: } [1]$  
  - $\text{high: } [2]$  
  - $\text{low: } [0]$  
  - $\text{mid: } [1]$  
  - $\text{high: } [2]$  

- **key == 11:**
  - $\text{list: } [10, 11, 45]$  
  - $\text{low: } [10]$  
  - $\text{mid: } [11]$  
  - $\text{high: } [45]$
... int index = -1; // why?
int high = A.length - 1, low = 0, mid;
while (high > low) {
    mid = (high + low) / 2;
    if (A[mid] == key) {
        index = mid;
        break;
    } else if (A[mid] > key)
        high = mid - 1;
    else
        low = mid + 1;
}
if (index > -1)
    System.out.printf("%d: %d\n", key, index);
else
    System.out.printf("%d: does not exist\n", key);
...