Linear Search

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);
        break;
    }
}
...

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

- Time complexity: $O(\log n)$
- Overall time complexity (sorting + searching): still $O(\log n)$?
... 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);
...
Beyond 1-Dimensional Arrays

- 2D or high-dimensional arrays are common, say an image which is represented by 2D arrays.
- For example, we can create a 2D T-type array with 4 rows and 3 columns as follows:

```java
... int rowSize = 4; // row size
int colSize = 3; // column size
T[][] x = new T[rowSize][colSize];
...```

Case (c) shows that we can create a 2D array by enumeration.
```
int[][] triangleArray = {
    {1, 2, 3, 4, 5},
    {2, 3, 4, 5},
    {3, 4, 5},
    {4, 5},
    {5}
};
```
Example

```java
... int[][] A = {{1, 2, 3}, {4, 5}, {6}};

    // traditional for loop
    for (int i = 0; i < A.length; i++) {
        for (int j = 0; j < A[i].length; j++)
            System.out.printf("%2d", A[i][j]);
        System.out.println();
    }

    // for-each loop
    for (int[] B: A) {
        for (int x: B)
            System.out.printf("%2d", x);
        System.out.println();
    }

...
Write a program which determines $C = A \times B$ for the input matrices $A_{m \times n}$ and $B_{n \times q}$ for $m, n, q \in \mathbb{N}$.

- You may use the formula

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

where $a_{ik}, i = 1, 2, \ldots, m$ is a shorthand for $A$ and $b_{kj}, j = 1, 2, \ldots, q$ for $B$.

- Time complexity: $O(n^3)$ (Why?)
class Lecture6 {

    "Methods"

}

// keywords:
return
Methods

• Methods can be used to define reusable code, and organize and simplify code.
• The idea of function originates from math, that is,

\[ y = f(x), \]

where \( x \) is the input parameter\(^2\) and \( y \) is the function value.
• In computer science, each input parameter should be declared with a specific type, and a function should be assigned with a return type.

---

\(^2\)Recall the multivariate functions. The input can be a vector, say the position vector \((x, y, z)\).

\(^3\)Aka procedures and functions.
Example: max

Define a method

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
```

Invoke a method

```
int z = max(x, y);
```
The modifier could be static and public (for now).
The returnType could be primitive types and reference types.
  • If the method does not return any value, then the return type is void.
The listOfParameters is the input of the method, separated by commas if there are multiple items.
  • Note that a method could have no input.\(^4\)
The method name and the parameter list together are called the method signature.\(^5\)

\(^4\)For example, Math.random().
\(^5\)Method overloading depends this. We will see it soon.
More Observations

- There are alternatives to the method `max()`:

```java
public static int max(int x, int y) {
    if (x > y) {
        return x;
    } else {
        return y;
    }
}
```

```java
public static int max(int x, int y) {
    return x > y ? x : y;
}
```
“All roads lead to Rome.”
– Anonymous

“但如你根本並無招式，敵人如何來破你的招式？”
– 風清揚，笑傲江湖。第十回。傳劍
The return Statement

- The return statement is the end point of the method.
- A callee is a method invoked by a caller.
- The callee returns to the caller if the callee
  - completes all the statements (w/o a return statement, say main());
  - reaches a return statement;
  - throws an exception (introduced later).
- As you can see, the return statement is not necessarily at the bottom of the method.\(^6\)
- Once one defines the return type (except void), the method should guarantee to return a value or an object of that type.

---

\(^6\)Thanks to a lively discussion on November 22, 2015.

Zheng-Liang Lu

AP Computer Science A: Java Programming
Bad Examples

```java
... static int fun1() {
    while (true);
    return 0; // unreachable code
}

static int fun2(int x) {
    if (x > 0) {
        return x;
    }
    // what if x < 0?
}
...`
```
Method Invocation

- Note that the input parameters are sort of variables declared within the method as placeholders.
- When calling the method, one needs to provide arguments, which must match the parameters in order, number, and compatible type, as defined in the method signature.
• In Java, method invocation uses pass-by-value.
• When the callee is invoked, the program control is transferred from the caller to the callee.
• For each invocation of methods, OS creates an frame which stores necessary information, and the frame is pushed in the call stack.
• The callee transfers the program control back to the caller once the callee finishes its job.
(a) The main method is invoked.

(b) The max method is invoked.

(c) The max method is being executed.

(d) The max method is finished and the return value is sent to k.

(e) The main method is finished.
Variable Scope

- The variable scope is the **region** where the variable can be referenced in the program.

- Variables can be declared in **class level**, **method level**, and **loop level**.

- In general, a balanced curly brackets defines a particular scope.

- One **cannot** declare the variables with the same name in the same scope.
Example

```java
public class ScopeDemo {

    static int i = 1; // class level

    public static void main(String[] args) {
        System.out.printf("%d\n", i); // output 1
        int i = 2; // method level; local
        i++;
        System.out.printf("%d\n", i); // output 3
        p();
        System.out.printf("%d\n", i); // output ?
    }

    static void p() {
        i = i + 1;
        System.out.printf("%d\n", i); // output 7
    }
}
```

- What if p(int i) in Line 14?\(^7\)

\(^7\)Thanks to a lively discussion on January 20, 2017.
A Math Toolbox: **Math** Class

- The **Math** class provides basic mathematical functions and 2 global constants `Math.PI`\(^8\) and `Math.E`\(^9\).
- All methods are **public** and **static**.
  - For example, `max`, `min`, `round`, `ceil`, `floor`, `abs`, `pow`, `exp`, `sqrt`, `cbrt`, `log`, `log10`, `sin`, `cos`, `asin`, `acos`, and `random`.
- Full document for **Math** class can be found [here](#).
- You are expected to read the document!

---

\(^8\)The constant \(\pi\) is a mathematical constant, the ratio of a circle’s circumference to its diameter, commonly approximated as 3.141593.

\(^9\)The constant \(e\) is the base of the natural logarithm. It is approximately equal to 2.71828.
Method Overloading

- Methods with the same name can coexist and be identified by the method signatures.

```java
... 
static int max(int x, int y) { ... }  
// different numbers of inputs
static int max(int x, int y, int z) { ... }
// different types
static double max(double x, double y) { ... }
... 
```
Recursion is the process of defining something in terms of itself.

- A method that calls itself is said to be recursive.
- Recursion is an alternative form of program control.
- It is repetition without any loop.
• Try Fractal.
Example

The **factorial** of a non-negative integer $n$, denoted by $n!$, is the product of all positive integers less than and equal to $n$.

- Note that $0! = 1$.
- For example,

$$4! = 4 \times 3 \times 2 \times 1$$

$$= 4 \times 3!$$

$$= 24.$$

- Can you find the pattern?
  - $n! = n \times (n - 1)!$
  - In general, $f(n) = n \times f(n - 1)$. 
Write a program which determines $n!$.

```java
... static int factorial(int n) {
    if (n > 0)
        return n * factorial(n - 1);
    else
        return 1; // base case
}
...```

- Note that there must be a base case in recursion.
- Time complexity: $O(n)$
- Can you implement the same method by using a loop?
• Time complexity: $O(n)$
• One intriguing question is, Can we always turn a recursive method into a loop version of that?
• Yes, theoretically.\footnote{The Church-Turing Thesis proves it if the memory serves.}
Remarks

- Recursion bears substantial overhead.
- So the recursive algorithm may execute a bit more slowly than the iterative equivalent.
- Additionally, a deeply recursive method depletes the call stack, which is limited, and causes stack overflow soon.
Memory Layout

Memory

$2^{32} - 1$

Stack

Heap

BSS (uninitialized)

Data (initialized)

Text (Code)