Session 2

Table of Contents

Revision

Practice: https://www.hackerrank.com/challenges/grading

Arrays

Arrays are contiguous blocks of memory.

Declaration, Intialization and Use

Arrays can be declared using the following syntax

```
// Both declarations are perfectly valid
int a[];
long[] a;
```

Arrays can be initialized using the following syntax

```
int a[];
a = new int[10];    // 10 here is the size of the array

// You can do this in one line, which is usually preferable
int b[] = new int[10];

// Arrays can be initialized with a dynamic size
int n = sc.nextInt();
long[] c = new long[n];

// However, the size HAS to be an int
long wontWork = sc.nextLong();
long[] d = new long[wontWork];  // Won't compile

// Another convenient way of initializing *constant* arrays is the following:
int[] noOfDaysInMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Arrays give you instant random access to any element, by means of index.

```
// Note that arrays are 0-indexed
// Indices               0   1   2   3   4   5   6   7   8   9   10  11
int[] noOfDaysInMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

// Prints 28
System.out.println("No of days in February are: " + noOfDaysInMonth[1]);

// An interesting property of an array is length
// Note that it is NOT a function call
System.out.println("No of months in a year are: " + noOfDaysInMonth.length)
```

Iteration

You can iterate through arrays in 2 ways:

1. for

```
for (int i = 0; i < a.length; i++) {
    System.out.println("The salary of employee " + i + " is " + salary[i]);

    // You can change array content using the index
    salary[i] = salary[i] * 2;          // Bonus for everyone!
}
```

1. for-each style

```
// k here is NOT the index, but the element itself
for (int days : noOfDaysInMonth) {
    System.out.println(days);
}

// Clearly, since you don't have access to the index
// You cannot modify the array elements
// This is an excellent way of iterating when you know you don't have to modify the array

for (long salary : employeeSalaries) {
    // This (sadly) has no effect on the underlying array
    salary = salary * 2;
}
```

Array Practice

1. https://www.hackerrank.com/challenges/service-lane

Functions

Basic Syntax

A function is used to factor out a common piece of functionality. You have already seen a few functions till now. (main, println, nextInt)

In Java, functions HAVE to belong to a class. They can be either static or non-static. We will see non-static functions later.

An example of a static function is:

```
class FunctionExample {
    public static int addOne(int a) {
        return a + 1;
    }

    public static boolean isOdd(long n) {
        return n % 2 == 0;
    }
}
```

Using Math class for common math operations. (abs, max, min, pow, floor, ceil, log, sqrt, PI, etc.)

1. Write a function to print number of perfect squares less than n
2. https://www.hackerrank.com/challenges/sherlock-and-squares
3. https://www.hackerrank.com/challenges/flatland-space-stations

Recursion

1. Terrible application of recursion: Factorial, Fibonacci
   Understand why it is terrible.
2. Write non-recursive implementations of both.

Example of recursion for factorial

```
// Remember that this is only good as an example
// You never want to actually do this when doing competetive programming
public static int factorial(int n) {
  if (n == 0 || n == 1) {
    return 1;
  }
  return n * factorial(n - 1);
}
```

Recursion is typically used when the mathematical or logical representation of a problem itself is recursive. Some examples are:
1. Greatest Common Divisor
2. Factorial
3. Fibonacci
4. Pascal's Triangle
5. Exponentiation (think of two ways it can be written recursively)

Memoization

Recursive functions can end up repeating a lot of work, and so are unlikely to pass test cases on HackerRank/CodeChef.
Unfortunately, non-recursive functions will also not work when you have repeated test cases.

For example, consider the following question:

> The first line of input is the number of test cases, T. T lines follow. Each line has a single integer n less than 1000. Print the nth Fibonacci number

> modulo $10^9 + 7$.

If you were to write a function to calculate the n^{th} Fibonacci number each time, you would often end up re-calculating a lot of stuff.

Instead, we can use arrays for memoization.

From Wikipedia:

> In computing, memoization or memoisation is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

1. Write factorial of a number using memoization
2. Write nth Fibonacci number using memoization

Varargs to a function

Varargs, or variable arguments, is a convenient way of writing a function that takes multiple arguments of the same type.
Let's say you want to write a function that returns the sum of all its arguments. However, you want to write it in a flexible manner. Varargs lets you do this by:

```
class VarArgsExample {
    public static int sum(int... nums) {
        int sum = 0;
        for (int k : nums) {
            sum += k;
        }
        return sum;
    }

    public static void main(String[] args) {
        // Following are all valid invocations of the above function
        int shouldBeFive = sum(1, 2, 2);
        int shouldBeHundred = sum(5, 10, 15, 20, 50);
    }
}
```

What actually happens is that the varargs gets converted into an array. You can treat the variable `nums` just like you would a regular 1-D array!

2D arrays

2-D arrays can be visualized as matrices.

```
// Code snippet for populating a 2-D array with the mutiplication table

n = sc.nextInt();
long[][] mat = new long[n][n];

for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    mat[i][j] = i * j;          // If you were to take input, you would say sc.nextLong() here instead.
  }
}
```

Practice:
1. Accept n. Accept a 2D array of dimensions n x n 2. Print the sum of each row
3. Print the sum of each column
4. Print the sum of each diagonal
5. Print the max such sum that occurs

Assignments

HackerRank

1. https://www.hackerrank.com/challenges/diagonal-difference
2. https://www.hackerrank.com/challenges/2d-array
3. https://www.hackerrank.com/challenges/restaurant
4. https://www.hackerrank.com/challenges/sherlock-and-gcd
5. https://www.hackerrank.com/challenges/apple-and-orange
6. https://www.hackerrank.com/challenges/divisible-sum-pairs
7. https://www.hackerrank.com/challenges/bon-appetit
8. https://www.hackerrank.com/challenges/java-negative-subarray
9. https://www.hackerrank.com/challenges/jumping-on-the-clouds

Miscellaneous

1. Write a recursive and non-recursive implementation for getting arbitrary Pascal triangle numbers.
   ${}^{n}C_{r} = {}^{n-1}C_{r} + {}^{n-1}C_{r-1}$
   ${}^{n}C_{r} = n! / (r! * (n – r)!)$
2. In both implementations, there are calculations that are being repeated. Can you avoid repeated calculations?
3. Write a function that calculates the Greatest Common Divisor of two integers. (https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations)
4. Write a function that checks if a number is prime.
5. Consider problem 8 in the HackerRank section. Clearly, when calculating sums of sub-arrays, you end up solving the same problem over and over again. For instance, consider,
   {1, 4, 3, 6}.
   Sub-arrays of length 2 are {1, 4}, {4, 3}, {3, 6}, and their sums are 5, 7 and 9.
   Sub-arrays of length 3 are {1, 4, 3}, {4, 3, 6}.
   In this case, you end up having to recalculate the sum of {1, 4} and {4, 3}, which you already calculated earlier. Is there a way to memoize the results?