# **Algorithms for problems**

# Square root of an integer algorithm

1. Create a variable (counter) ***i*** and take care of some base cases, (i.e when the given number is 0 or 1, return itself).
2. Run a loop until ***i\*i <= n***, where n is the given number. Increment i by 1.
3. The floor of the square root of the number is ***i – 1***

# Next perfect square number algorithm

1. Find the square root of the number (use the above algorithm if built-in methods are not allowed)
2. Calculate the floor value of the square root
3. Then add 1 to it.
4. Print square of that number

# Check prime number algorithm

1. Read the number N
2. start the loop from i= 2 to i=N-1 (OR the condition can be made i=2 upto i\*i<=N)
3. If N is divided by **i** in the loop, then it is not prime - break the loop
4. If(i==N), the loop has executed till the end of range, so N is not divided by any number So it is a prime number
5. Else it is not a prime number

# Find **all** prime factors of a number

1. The shortest way is loop the following, let there is a function that checks if a number is prime.

|  |
| --- |
| for (int i = 2; i<=n; )  {  if(isPrime(i) && n%i == 0){  System.out.print(i + " ");  n = n/I;  }  else  i++;  } |

OR use the following algorithm

1. While n is divisible by 2, print 2 and **divide** **n by 2**.
2. After counting all “2” factors in step 1, n must be odd and factors of an odd number are all odd numbers. Now start a loop from i = 3 to the square root of n. **While i divides n**, print i, and divide n by i. After i fails to divide n, **increment i by 2** and continue because odd numbers are found at a difference of 2.
3. If final remaining value of n after all the above divisions is a prime number and is greater than 2, b/c n will not become 1 by the above two steps. So print n if it is greater than 2.

|  |
| --- |
| // Print the number of 2s that divide n  while (n%2==0)  {  System.out.print(2 + " ");  n /= 2;  }  // n must be odd at this point. So we can skip one element (Note i = i +2)  for (int i = 3; i <= Math.sqrt(n); i+= 2) //i\*i <= n  {  // While i divides n, print i and divide n  while (n%i == 0)  {  System.out.print(i + " ");  n /= i;  }  }  // This condition is to handle the case when the last value of n, after all above division,  // is a prime number greater than 2  if (n > 2)  System.out.print(n); |

# Extract digits of a number

1. If number not equal to 0, repeat the following
2. digit = number % 10
3. number = number / 10
4. print digit

# Reverse digits of a number

1. Initialize rev\_num = 0
2. Loop while num > 0
   1. Multiply rev\_num by 10 and add remainder of num divide by 10 to rev\_num i.e. rev\_num = rev\_num\*10 + num%10;
   2. Divide num by 10
3. Return rev\_num

# Convert number to binary algorithm without using array or string

Binary of a number **n** using module(%) operator, without using array,string, &built in methods:

1. initialize bn=0, pc=0; (binary number bn and power counter pc)
2. loop while n > 0
   1. int reminder = n%2
   2. p10 = 1; (initialize power of 10)
   3. calculate 10 to the power of pc
      1. loop for i=0; i< pc; i++
         * if(i==0) p10 = p10 \* 1
         * else p10 = p10 \* 10;
   4. bn = bn + reminder \* p10
   5. pc++
   6. n = n/2
3. return bn ­­

# Fibonacci number algorithm

1. nth Fibonacci number using the recurrence relation:

* if *n* = 0, then return 0.
* if n = 1, then it should return 1.
* for n > 1, it should return Fn-1 + Fn-2

1. If (n <= 1) return n;
2. Else return fib(n - 1) + fib(n - 2);