# Big-O Exercises

**Question 1**

This code computes the product of two variables, what is the runtime of this code?

**int** **product**(**int** a, **int** b) {

**int** sum = 0;

**for** (**int** I = 0; I < b; I++) {

sum += a;

}

**return** sum;

}

O(b)

**Question 2**

This code computes A ^ B, what would be the runtime?

**static** **int** **power**(**int** a, **int** b) {

**if** (b < 0) **return** a;

**if** (b == 0) **return** 1;

**int** sum = a;

**for** (**int** I = 0; I < b - 1; I++) {

sum \*= a;

}

**return** sum;

}

O(b-1)

**Question 3**

This code computes A% B, what would be the runtime?

**int** **mod**(**int** a, **int** b) {

**if** (b <=a) **return** -1;

**int** div = a / b;

**return** a - div \* b;

}

**Question 4**

This code computes a division between whole integers (assuming both are positive), what would be the runtime?

**int** **div**(**int** a, **int** b) {

**int** count = a;

**int** sum = b;

**while** (sum <= a) {

sum += b;

count++;

}

**return** count;

}

O(a/b)

**Question 5**

The following code calculates the square root of an integer. If the number is not a perfect square (there is no whole square root), then return -1. If N is 100, first guess if N is 50. Too high? Try something lower, halfway between 1 and 50, etc. What is the big-o?

**int** **sqrt**(**int** n) {

**return** sqrt\_helper(n, 1, n);

}

**int** **sqrt\_helper**(**int** n, **int** min, **int** max) {

**if** (max < min) **return** -1;

**int** guess = (min + max) / 2;

**if** (guess \* guess == n) {

**return** guess;

} **else** **if** (guess \* guess < n) {

**return** sqrt\_helper(n, guess + 1, max);

} **else** {

**return** sqrt\_helper(n, min, guess - 1);

}

}

O(log(n))

**Question 6**

The following code calculates the square root of an integer. If the number is not a perfect square (there is no whole square root), then return -1. It does so by trying larger and larger numbers until it finds the correct value (or is too high). What is your runtime?

**int** **sqrt**(**int** n) {

**for** (**int** guess = 1; guess \* guess < n; guess++) {

**if** (guess \* guess == n) **return** guess;

}

**return** -1;

}

O(sqrt(n))

**Question 7**

The appendToNew method adds a value to an array by creating a new, longer array and returning this longer array. How long does it take to copy the array?

**int**[] copyArray(**int**[] array) {

**int**[] copy = **new** **int**[0];

**for** (**int** value : array) {

copy = appendToNew(copy, value);

}

**return** copy;

}

**int**[] appendToNew(**int**[] array, **int** value) {

**int**[] bigger = **new** **int**[array.length + 1];

**for** (**int** I = 0; I < array. length; I++) {

bigger[I] = array[I];

}

bigger[bigger.length - 1] = value;

**return** bigger;

}

**Question 8**

The following code adds the digits of a number. What is your runtime?

**int** **sumDigits**(**int** n) {

**int** sum = 0;

**while** (n > e) {

sum += n % 10;

n /= 10;

}

**return** sum;

}

**Question 9**

The following code calculates the intersection (the number of elements in common) of two Arrays. Assuming that no Array has duplicates. Calculate the big-O notation of the intersection function which is ordering Array B and then iterating through Array A - making a check (Binary Search) if each value is in B. What is its execution time?

**int** **intersection**(**int**[] a, **int**[] b) {

mergesort(b);

**int** intersect =0;

**for** (**int** x : a) {

**if** (binarySearch(b, x) >= 0) {

intersect++;

}

}

**return** intersect;

}