**BIG O**

**What is the runtime of the below code?**

* **1 void foo(int[] array) {**
* **2 int sum = 0;**
* **3 int product = 1;**
* **4 for (inti= 0; i < array.length; i++) {**
* **5 sum += array[i);**
* **6 }**
* **7 for (int i= 0; i < array.length; i++) {**
* **8 product\*= array[i];**
* **9 }**
* **10 System.out.println(sum + ", " + product);**
* **11 }**

🡺**O(n)**

**==========================================================**

* **1 void printPairs(int[] array) {**
* **2 for (int i= 0; i < array.length; i++) {**
* **3 for (int j = 0; j < array.length; j++) {**
* **4 System.out.println(array[i] + "," + array[j]);**
* **5 }**
* **6 }**
* **7 }**

**🡺O(N\*N)**

**=============================================================**

* **1 void printUnorderedPairs(int[] array) {**
* **2 for (int i= 0; i < array.length; i++) {**
* **3 for (int j = i + 1; j < array.length; j++) {**
* **4 System.out.println(array[i] + "," + array[j]);**
* **5 }**
* **6 }**
* **7 }**

**🡺O(N\*N)**

**=============================================================**

* **1 void printUnorderedPairs(int[] arrayA, int[] arrayB) {**
* **2 for (inti= 0; i < arrayA.length; i++) {**
* **3 for (int j = 0; j < arrayB.length; j++) {**
* **4 if (arrayA[i] < arrayB[j]) {**
* **5 System.out.println(arrayA[i] + "," + arrayB[j]);**
* **6 }**
* **7 }**
* **8 }**
* **9 }**

**🡺O(a\*b)**

**===============================================================**

* **1 void printUnorderedPairs(int[] arrayA, int[] arrayB) {**
* **2 for (int i= 0; i < arrayA.length; i++) {**
* **3 for (int j = 0; j < arrayB.length; j++) {**
* **4 for (int k = 0; k < 100000; k++) {**
* **5 System.out.println(arrayA[i] + "," + arrayB[j]);**
* **6 }**
* **7 }**
* **8 }**
* **9 }**

**🡺O(a\*b),** **Nothing has really changed here. 100,000 units of work is still constant, so the runtime is 0( ab).**

**=============================================================**

**Which of the following are equivalent to O(N)? Why?**

* **O(N + P), where P < X**
* **0(2N)**
* **O(N + log N)**
* **O(N + M)**

**Let's go through these.**

**=======================🡺**

* **If P < X, then we know that N is the dominant term so we can drop the 0( P).**
* **0(2N) isO(N) since we drop constants.**
* **O(N) dominatesO(log N),so we can drop theO(log N).**

**There is no established relationship between N and M, so we have to keep both variables in there.**

**Therefore, all but the last one are equivalent to O(N).**

**=============================================================**

* **1 boolean isPrime(int n) {**
* **2 for (int x = 2; x \* x <= n; x++) {**
* **3 if (n % X == 0) {**
* **4 return false;**
* **5 }**
* **6 }**
* **7 return true;**

**🡺As I starts from 2 and go to n-1.Means it will end upto i\*i=(n-1)**

**i=underroot(n) I will go for 2,3,5,…..tillroot(n)**

**hence,O(root(n))**

**========================================================**

* **1 int factorial(int n) {**
* **2 if (n < 0) {**
* **3 return -1;**
* **4 } else if (n**
* **5 return 1;**
* **6 } else {**
* **7 return n \***
* **8 }**
* **9 }**
* **{**
* **factorial(n - 1);**

**🡺This is just a straight recursion from n to n -1 to n - 2 down to 1. It will take O ( n) time.**

**=============================================================**