**CTA 3: Big Oh**

Jordon Paynter

Department of Computer Science, Colorado State University Global

CSC 400: Data Structures and Algorithms

Professor Santosh Kumar Gottipamula

October 01, 2023

**CTA 3: Big Oh**

**Exercise # 10**

What is the big Oh of the following computation:

int sum = 0;

for (int counter = n; counter > 0; counter = counter -2)

sum = sum + counter;

The given code consists of a for loop that iterates from n down to 1, decreasing the counter by 2 in each iteration. This results in roughly n/2 iterations. Since the number of iterations is directly proportional to n, the time complexity of this code is O(n).

**Exercise # 12**

Suppose that your implementation of a particular algorithm appears in Java as follows:

for (int pass = 1; pass <= n; pass++) O(n) based on its iteration affected by the amount of n

{

for ( int index = 0; index < n; index++){ O(n) Same here as outer loop

for ( int count = 1; count < 10; count ++){// inner loop is O(1) due to it being constant regardless of n.

some code execution...

}

}

The code is O(n^2). We get this by multiplying each part together. The first loop is O n, the second loop is O n, and the third loop is constant, so it is a O 1. n\*n\*1 = n^2.

**Exercise # 16**

Consider two programs, A and B. Program A requires 1000 x n^2 operations, and Program B requires 2^n operations. For which values of n will program A execute faster than Program B?

package CTA3;

public class TwoProgramsComparisons {

public static void main(String[] args) {

for (int nIteration = 1; nIteration <= 100; nIteration++ ) {

double aProgram = 1000 \* Math.pow(nIteration, 2); // program a alogrithm

double bProgram = Math.pow(2, nIteration); // program b alogrithm

if (aProgram < bProgram) {

System.out.println(

"N at " + nIteration + "; A = " + (int) aProgram + " which is less than b: " + (int) bProgram

);

break; // break out of the loop to show when a is finally less than b

}

}

}

}  
**Figure 1.**

What iteration Program A became faster than Program B.



Program A is faster than program B at integers 19 and up.

**Exercise # 24a**

Consider an Array of length n containing unique integers in random order and range 1 to n + 1. For example, an array length of 5 contains 5 unique integers selected randomly from the integers 1 through 6. Thus, the array might contain 3,5,4,6,1 of the integers 1 through 6; notice that the two were not chosen or in the array. Write a Java code that finds the integer that does not appear in such an array. The solution should use

1. O(n^2) operation

*package* ***CTA****3*;

*public* *class* MissingInteger {

*public* *static* *void* *main*(*String*[] args) {

*int*[] *arr* *=* { 4, 2, 9, 8, 3, 6, 7, 1, 5 };

*int* *missing* *=* *findMissingInteger*(arr);

*System*.*out*.*println*(

      missing *>* *-*1 *?*  "Missing integer: " *+* missing *:* "No missing integer found" *// Prints out the correct message using a ternary operator*

    );

  }

*public* *static* *int* *findMissingInteger*(*int*[] arr) {

*for* (*int* *outerIndex* *=* 1; *outerIndex* *<* *arr*.*length* *+* 2; *outerIndex++* ) { *// O n*

*boolean* *found* *=* false;

*for* (*int* *innerIndex* *=* 0; *innerIndex* *<* *arr*.*length* ; *innerIndex++* ) { *// O n*

*if* (arr[*innerIndex*] *==* *outerIndex*) { *// O 1 because it is a constant time operation*

          found *=* true;

*break*;

        }

      }

*if* (*!*found) {

*return* *outerIndex*;

      }

    } *// n \* n = n^2*

*return* *-*1;

  }

}

**Figure 2.**

Result of Missing Integer.

A screen shot of a computer

Description automatically generated