# Big-O Run Time Analysis

By: Matthew S Montoya

# 1 Introduction

In this lab, a program will calculate the time complexity, data input size, and/or the big-O runtime of a user's input. Looping through a "main menu" of options, the program will continuously ask the user what they would like to calculate (time complexity, data input size, or big-O runtime), until the user specifies they wish to exit the program. All the input data will be given by the user and the program will determine the output accordingly, by analyzing the algorithm(s) and mathematically computing the appropriate output.

# 2 Proposed Solution & Design Implementation

To achieve the end goal, the program will consist of seven different methods. Each method will be used to solve one part of the problem. For example, the main method will call a separate method, in which the user will determine which of the five different scenarios they wish to peruse, and one additional method for four out of the five scenarios. This is done because the fifth option is to terminate the program, which doesn't require an extra method. For the sake of simplicity, many of these methods will use *switches* to determine the appropriate action to take, based on the user's input. When asking for user input, the program will use a scanner, and to prevent any errors, will flush itself prior to taking input of a different variable kind (eg. Integers, then doubles).

## 2.1— The Main Method

The main method's only line of code will call a second method to display a "main menu" output for the user, asking them to choose which of the five options they would like to pursue. This is done to prevent unnecessary problem-solving solving within the main method.

## 2.2— The *programMessage* Method

The *programMessage* method will display the "main menu" of options for the user, and ask them to select which of the five options they would like the program to run. Because the program must return to this main menu after running four out of the five specified options (e.g. after a user has the program calculate the big-O runtime, it must return to this main menu), the method will be on a loop, triggered by a Boolean, set *true* by default. Until the Boolean is triggered *false* by a user's appropriate input (selection #5), the program will continuously return to this method, prompting the user for further instruction. This method will then call the *selectComputation* method, prompting the user for further input.

## 2.3— The *selectComputation* Method

The *selectComputation* method utilizes a switch to determine what the next course of action should be for the program. This will also prompt the user to provide the

appropriate input data for the program to run its calculations. It will take an integer as its parameter (given by the user in the *programMessage* method). For example, if the user chose option #1, the input would be an integer with the value 1, and the method would locate case 1, and prompt the use for further input. Each respective case has its own method call to calculate the output of each case, with the given parameters.

### 2.4— The *computeCase1* Method

The *computeCase1* method determines T(n1), the time it takes to run a data size of n1, with the given big-O running time. Using a switch, method takes in the parameters of n0 and n1 (the size of the data), t(n0) (the time it takes to run the data size of n0), and f(n) (the runtime of the data). F(n) will be in the form of a char variable, and used as part of a switch to properly determine T(n1). For example, char *a* would represent f(n) = "log n." Using the switch, the program will look for *case 'a'* and execute the appropriate algebraic steps to find T(n1) (in seconds); in this case, applying (*log n*) to the data sets n0 and n1, dividing n1 by n0, then multiplying that outcome by T(n0). Each one of the six cases contain unique algebraic steps, respective to the F(n) input. If the program can calculate T(n1), it will return a Boolean *true* to the *selectComputation* method, which will allow the program to continue looping.

### 2.5— The *computeCase2* Method

The *computeCase2* method will calculate n1, the data size of the second set of data. Using a switch, this method takes in the parameters n0 (the size of a different set of data), t(n0) (the time it takes to run the data size of n0), and f(n) (the runtime of the data). F(n) will be in the form of a char variable, and used as part of a switch to properly determine n0. Like *computeCase1*, a char variable would represent f(n). Using the switch, the program will look for that specific case and execute the appropriate algebraic steps to find n1. Using the specified runtime, the cases within the method will divide T(n1) by T(n0) and multiply that product by n0. If the program can calculate n1, it will return a Boolean *true* to the *selectComputation* method, which will allow the program to continue looping.

### 2.6— The *computeCase3* Method

The *computeCase3* method will calculate f(n), the big-O runtime of the input data. Using if-else statements, this method takes in the parameters n0 and n1(the size of different sets of data), t(n0) and t(n1) (the time it takes to run the data size of n0 and n1, respectively). Testing the different big-O runtimes (log n, n, n log n, n^2, n^3, 2^n) with the if statements, this method will print out the runtime of the input data. The runtime will be tested by setting T(n1)/T(n0) equal to n1, n0, applying the big-) runtimes to n0 and n1, and determining if they are equal. Once a match is made, the information is then printed on the screen and the method returns a Boolean *true* to the *selectComputation* method, which will allow the program to continue looping.

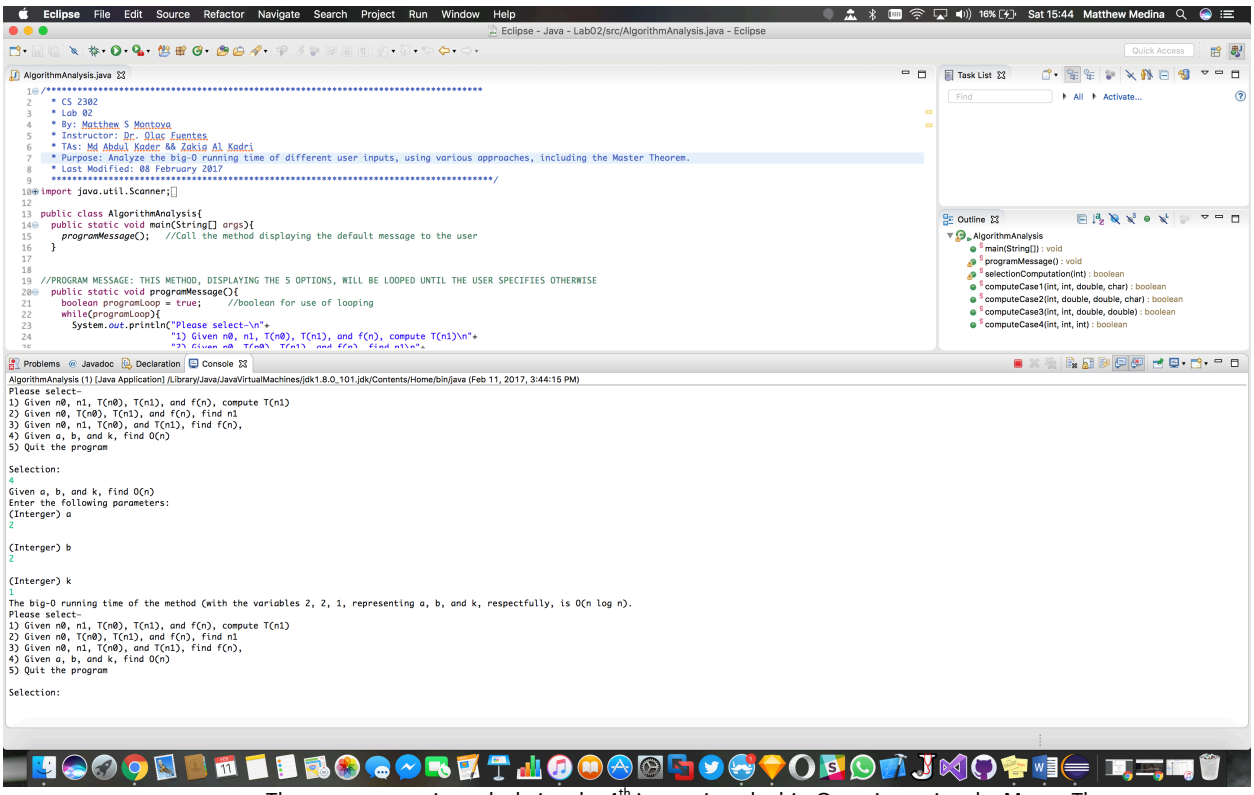
### 2.5— The *computeCase4* Method

The *computeCase4* method will calculate f(n), the big-O runtime of the input data, using the *Master Theorem*, a more mathematical approach to determining the big-O runtimes. Using if-else statements, this method takes in the parameters *a*, *b*, and *k* and determine the run time by analyzing *a*'s relationship to $b^k$. Once the big-O notation is calculated, this method will print out the runtime of the data, and return a Boolean *true* to the *selectComputation* method, allowing the entire program to continuously loop until otherwise specified by the user.

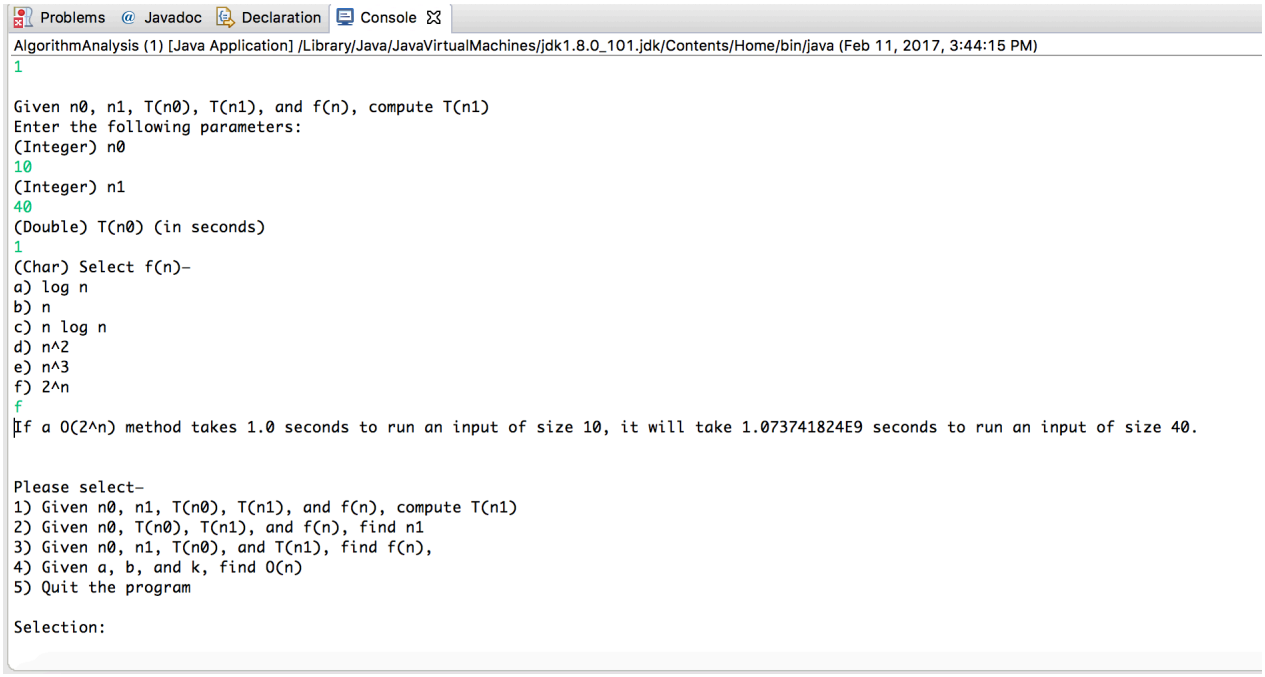
## 3 Experimental Results

### 3.1— Program Output

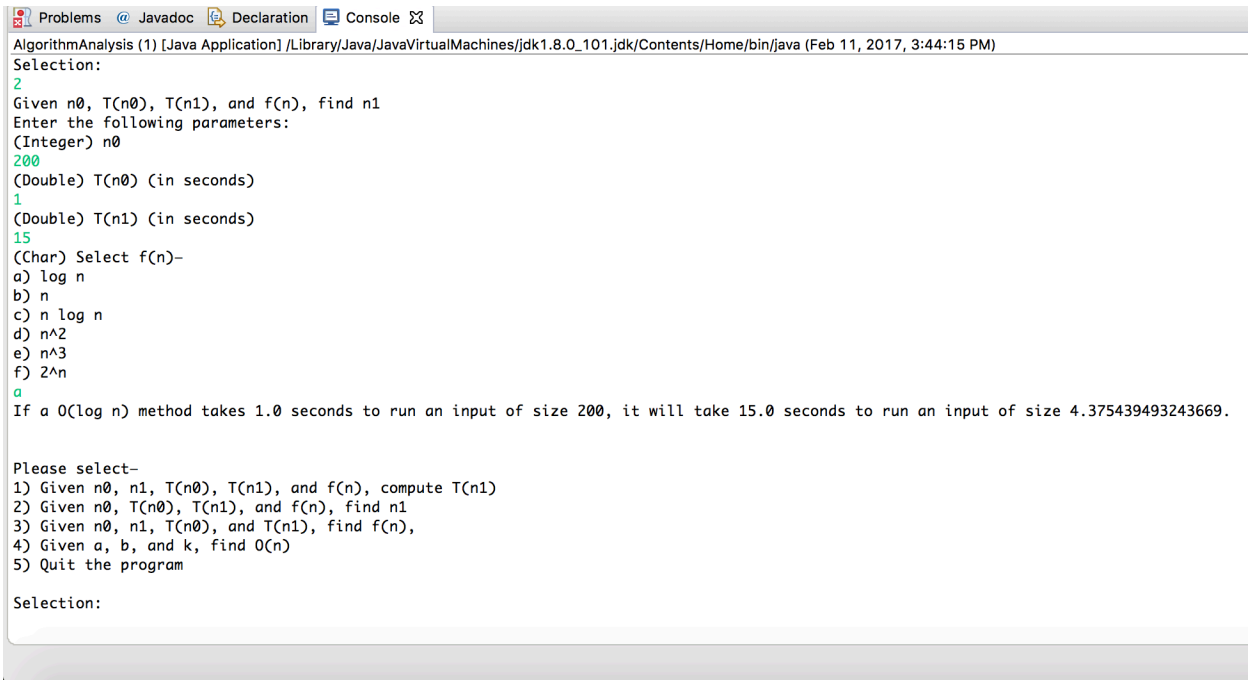
The output of the code (found in the appendix) is shown below. The two images are from the same test. Note: The first image includes the beginning of the code (which can also be found in the *Appendix* section), as proof the executed code is not the work of others. The following images is an enhanced version of the program's output during various executions.



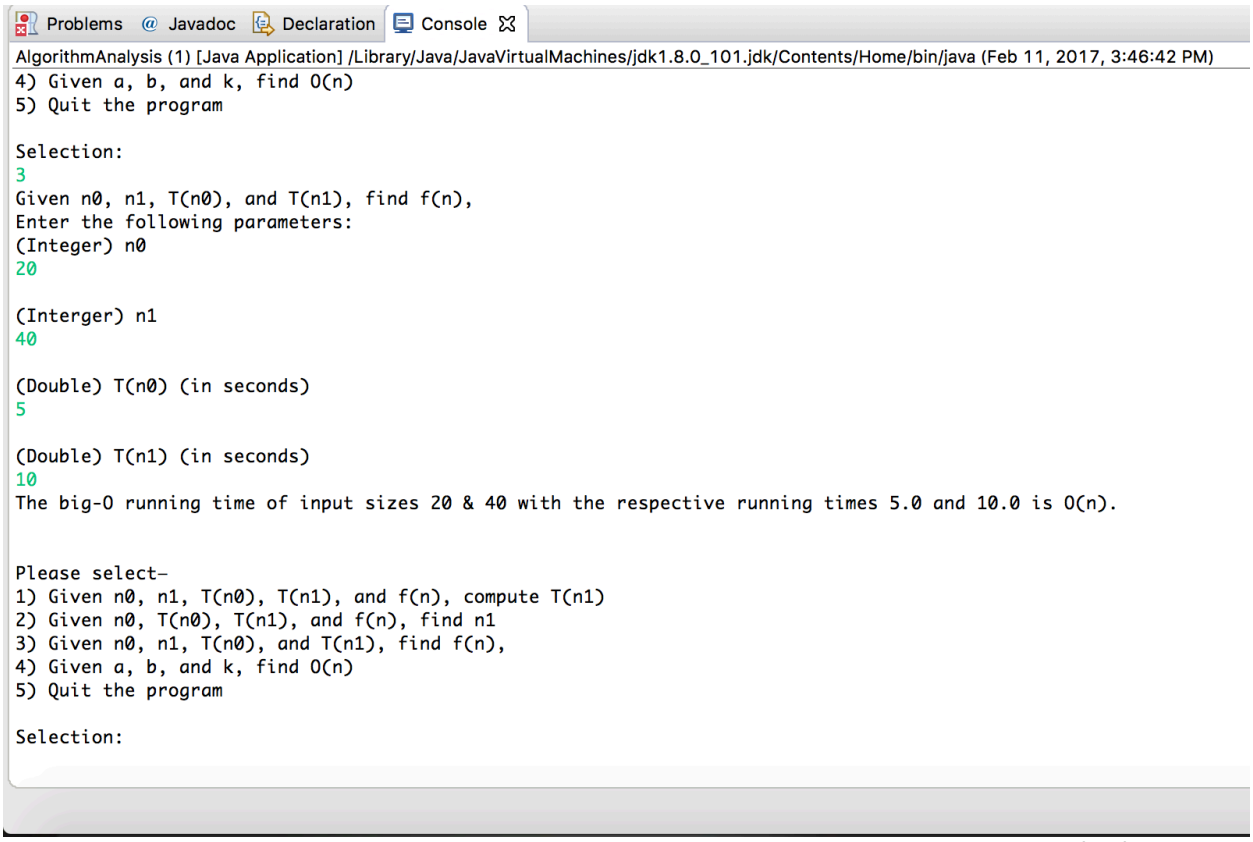

The program running calculating the 4th instruction: the big-O runtime using the Master Theorem




The program calculating the first instruction: finding T(n1) using n0, n1, T(n0) and f(n), compute T(n1)




The program running the second instruction: given n0, T(n0), T(n1), and f(n), find n1.




The program running the third instruction: given n0, n1, T(n0), and T(n1), find f(n).

# 4 Conclusions

In this lab, I learned how easy it is to calculate the big-O runtime of data, using the Java programming language. Moreover, I learned how to mathematically calculate the big-O runtime of data using the *Master Theorem*, and learned that (when calculating this information by hand), this saves time in identifying the runtime, when compared to other styles of calculation. Furthermore, with the increased usage of code, adapting an alternative approach to method documentation will be necessary.

# 5 Appendix

## 5.1— Documentation & Disclosure

- Methods are documented using the following format—

```
//METHOD NUMBER: BRIEF MENTION OF METHOD'S PURPOSE
public static void methodName(parameters){
}
```

- Non-obvious code is documented using the following format—

```
for(int i=0; i<x.length-1; i++){          //summarizes the function of the loop
        x[i] = i;          //may include brief logical reasoning
}
```

- Disclosure: Though keeping the original digital format of the code (indention lengths, spaces, etc.) the use of Microsoft Word has reformatted the code in

the document to be of different space and tab lengths. The font size has been reduced to help alleviate some visual misrepresentations.

## 5.2— Student Code

```
/*********************************************************************************
  * By: Matthew S Montoya
  * Purpose: Analyze the big-O running time of different user inputs, using various approaches, including
the Master Theorem.
  * Last Modified: 09 January 2018

*********************************************************************************/
import java.util.Scanner;
import java.lang.Math;

public class AlgorithmAnalysis{
  public static void main(String[] args){
    programMessage();            //Call the method displaying the default message to the user
  }


//PROGRAM MESSAGE: THIS METHOD, DISPLAYING THE 5 OPTIONS, WILL BE LOOPED
UNTIL THE USER SPECIFIES OTHERWISE
  public static void programMessage(){
    boolean programLoop = true;                    //boolean for use of looping
    while(programLoop){
      System.out.println("Please select—\n"+
                    "1) Given n0, n1, T(n0), T(n1), and f(n), compute T(n1)\n"+
                    "2) Given n0, T(n0), T(n1), and f(n), find n1\n"+
                    "3) Given n0, n1, T(n0), and T(n1), find f(n),\n"+
                    "4) Given a, b, and k, find O(n)\n"+
                    "5) Quit the program\n\n"+

                    "Selection: ");
      Scanner input = new Scanner(System.in);
      int userSelection = input.nextInt();

      if(userSelection < 1 || userSelection> 5)                //handle input errors gracefully
        System.out.println("Invalid selction. Please selection an integer between 1-5\nProgram
resetting...\n");

      programLoop = selectionComputation(userSelection);//Further investigate the user's selection in
another method
    }
    System.exit(0);
  }


//USER WILL SELECT THEIR METHOD (CASE) OF COMPUTATION//////////////////
  public static boolean selectionComputation(int userSelection){
    boolean continueLooping = true;
```

```
   Scanner input = new Scanner(System.in);
   switch (userSelection){                  //case statements: one for each possible selection the user can make
     case 1:   //find T(n1)
       System.out.println("\nGiven n0, n1, T(n0), T(n1), and f(n), compute T(n1)\nEnter the following
parameters:\n(Integer) n0");
       int n0 = input.nextInt();
       System.out.println("(Integer) n1");
       int n1 = input.nextInt();
       System.out.println("(Double) T(n0) (in seconds)");
       input.nextLine();                    //flush the scanner
       double t_n0 = input.nextDouble();
       input.nextLine();                    //flush the scanner again (because now we're taking in chars)
       System.out.println("(Char) Select f(n)—\n"+
                          "a) log n\n"+
                          "b) n\n"+
                          "c) n log n\n"+
                          "d) n^2\n"+
                          "e) n^3\n"+
                          "f) 2^n");
       char f_n = input.nextLine().toLowerCase().charAt(0);
       boolean computation = computeCase1(n0, n1, t_n0, f_n);          //if the case can be computed, it
will return true
       continueLooping = computation;                                                              //then
call back the user selection option
       break;
     case 2:   //find n1
       System.out.println("Given n0, T(n0), T(n1), and f(n), find n1\nEnter the following
parameters:\n(Integer) n0");
       n0 = input.nextInt();
       System.out.println("(Double) T(n0) (in seconds)");
       input.nextLine();                    //flush the scanner
       t_n0 = input.nextDouble();
       System.out.println("(Double) T(n1) (in seconds)");
       double t_n1 = input.nextDouble();
       input.nextLine();                    //flush the scanner again
       System.out.println("(Char) Select f(n)—\n"+
                          "a) log n\n"+
                          "b) n\n"+
                          "c) n log n\n"+
                          "d) n^2\n"+
                          "e) n^3\n"+
                          "f) 2^n");
       f_n = input.nextLine().toLowerCase().charAt(0);          //Handles if the user types a capital letter
       computation = computeCase2(n0, t_n0, t_n1, f_n);//if the case can be computed, it will return true
       continueLooping = computation;
       break;
     case 3:   //find f(n)
       System.out.println("Given n0, n1, T(n0), and T(n1), find f(n),\nEnter the following
parameters:\n(Integer) n0");
       n0 = input.nextInt();
       System.out.println("\n(Interger) n1");
       n1 = input.nextInt();
       input.nextLine();                    //flush the scanner, because now we're switching to doubles
```

```
        System.out.println("\n(Double) T(n0) (in seconds)");
        t_n0 = input.nextDouble();
        System.out.println("\n(Double) T(n1) (in seconds)");
        t_n1 = input.nextDouble();
        computation = computeCase3(n0, n1, t_n0, t_n1);
        continueLooping = true;//returns the method's default true value || continues looping reguardless of
user input
        break;
      case 4:    //find the big-O notation via Master Theorem
        System.out.println("Given a, b, and k, find O(n)\nEnter the following parameters:\n(Interger) a");
        int a = input.nextInt();
        System.out.println("\n(Interger) b");
        int b = input.nextInt();
        System.out.println("\n(Interger) k");
        int k = input.nextInt();
        computation = computeCase4(a, b, k);
        continueLooping = true;//returns the method's default true value || continues looping reguardless of
user input
        break;
      case 5:    //program termination
        System.out.println("Program terminated!\nResetting instructions...");
        continueLooping = false;
        return false;
    }
    return continueLooping;
  }


//COMPUTE CASE 1////////////////////////////////////////////////////////////
  public static boolean computeCase1(int n0, int n1, double t_n0, char f_n){
    boolean canCompute = false;
    double t_n1;
    String method;      //the big-O notation (minus the "O")

    switch (f_n){       //The selection the user made
      case 'a':             //log n
        method = ("log n");
        t_n1 = (((Math.log(n1))/(Math.log(n0)))*t_n0);              //identify t_n1 using log n for (n0/n1)
        canCompute = true;
        break;

      case 'b':             //n
        method = ("n");
        t_n1 = (((n1)/(n0))*t_n0);                                                  //identify t_n1
using n for (n0/n1)
        canCompute = true;
        break;

      case 'c':             //n log n
        method = ("n log n");
        t_n1 = (((n1*(Math.log(n1)))/(n0*Math.log(n0)))*t_n0);    //identify t_n1 using n log n for (n0/n1)
        canCompute = true;
        break;
```

```
    case 'd':               //n^2
      method = ("n^2");
      t_n1 = (((Math.pow(n1,2))/(Math.pow(n0,2)))*t_n0);                    //identify t_n1 using n^2 for
(n0/n1)
      canCompute = true;
      break;

    case 'e':               //n^3
      method = ("n^3");
      t_n1 = (((Math.pow(n1,3))/(Math.pow(n0,3)))*t_n0);                    //identify t_n1 using n^3 for
(n0/n1)
      canCompute = true;
      break;

    case 'f':               //2^n
      method = ("2^n");
      t_n1 = (((Math.pow(2,n1))/(Math.pow(2,n0)))*t_n0);                    //identify t_n1 using 2^n for
(n0/n1)
      canCompute = true;
      break;

    default:
      System.out.println("Invalid answer. Please selection a letter between a-f\nProgram resetting...\n");
      return false;
  }
  System.out.println("If a O("+method+") method takes "+t_n0+" seconds to run an input of size "+n0+
                     ", it will take "+t_n1+" seconds to run an input of size "+n1+".\n\n");//print the computation
  return canCompute;                        //Was the program able to compute using the given inputs?
 }


//COMPUTE CASE 2//////////////////////////////////////////////////////////////////
 public static boolean computeCase2(int n0, double t_n0, double t_n1, char f_n){
  boolean canCompute = false;
  double n1;
  String method;      //the big-O notation (minus the "O")

  switch (f_n){
    case 'a':               //log n
      method = ("log n");
      n1 = (t_n1/t_n0)*(Math.log(n0));                          //n1 using log n*(t_n0/t_n1)
      n1 = (Math.log(n1));
      canCompute = true;
      break;

    case 'b':               //n
      method = ("n");
      n1 = ((t_n1/t_n0)*(n0));                                  //n1 using n*(t_n0/t_n1)
      canCompute = true;
      break;

    case 'c':               //n log n
```

```
        method = ("n log n");
        n1 = ((t_n1/t_n0)*((n0)*(Math.log(n0))));//n1 using n log n*(t_n0/t_n1)
        n1 = ((n1)*(Math.log(n1)));
        canCompute = true;
        break;

      case 'd':               //n^2
        method = ("n^2");
        n1 = ((t_n1/t_n0)*(Math.pow(n0,2)));                  //n1 using n^2*(t_n0/t_n1)
        n1 = Math.pow(n1,2);
        canCompute = true;
        break;

      case 'e':               //n^3
        method = ("n^3");
        n1 = ((t_n1/t_n0)*(Math.pow(n0,3)));                  //n1 using n^3*(t_n0/t_n1)
        n1 = Math.pow(n1,3);
        canCompute = true;
        break;

      case 'f':               //2^n
        method = ("2^n");
        n1 = ((t_n1/t_n0)*(Math.pow(2, n0)));                 //n1 using 2^n*(t_n0/t_n1)
        n1 = Math.pow(2, n1);
        canCompute = true;
        break;

      default:
        System.out.println("Invalid answer. Please selection a letter between a-f\nProgram resetting...\n");
        return false;
    }
    System.out.println("If a O("+method+") method takes "+t_n0+" seconds to run an input of size "+n0+
                    ", it will take "+t_n1+" seconds to run an input of size "+n1+".\n\n");//print the computation
    return canCompute;
  }


//COMPUTE CASE 3////////////////////////////////////////////////////////////////
  //LIST THE OUTPUT WITHIN THE METHOD; EASIER FOR A CASE-BY-CASE BASIS//////////
  public static boolean computeCase3(int n0, int n1, double t_n0, double t_n1){
    String f_n = null;
    //identify what f(n) is, by applying algebric methods to each side of the given equation values
    if((t_n1/t_n0) == ((Math.log(n1))/Math.log(n0))){
      f_n = "log n";
      System.out.println("The big-O running time of input sizes "+n0+" & "+n1+" with the respective
running times "+t_n0+" and "+t_n1+" is O("+f_n+").\n\n");
    }
    else if((t_n1/t_n0) == (n1/n0)){
      f_n = "n";
      System.out.println("The big-O running time of input sizes "+n0+" & "+n1+" with the respective
running times "+t_n0+" and "+t_n1+" is O("+f_n+").\n\n");
    }
    else if((t_n1/t_n0) == ((n1*(Math.log(n1)))/n0*(Math.log(n0)))){
```

```
      f_n = "n log n";
      System.out.println("The big-O running time of input sizes "+n0+" & "+n1+" with the respective
running times "+t_n0+" and "+t_n1+" is O("+f_n+").\n\n");
    }
    else if((t_n1/t_n0) == ((Math.pow(n1,2))/(Math.pow(n0,2)))){
      f_n = "n^2";
      System.out.println("The big-O running time of input sizes "+n0+" & "+n1+" with the respective
running times "+t_n0+" and "+t_n1+" is O("+f_n+").\n\n");
    }
    else if((t_n1/t_n0) == ((Math.pow(n1,3))/(Math.pow(n0,3)))){
      f_n = "n^3";
      System.out.println("The big-O running time of input sizes "+n0+" & "+n1+" with the respective
running times "+t_n0+" and "+t_n1+" is O("+f_n+").\n\n");
    }
    else if((t_n1/t_n0) == ((Math.pow(2,n1))/(Math.pow(2,n0)))){
      f_n = "2^n";
      System.out.println("The big-O running time of input sizes "+n0+" & "+n1+" with the respective
running times "+t_n0+" and "+t_n1+" is O("+f_n+").\n\n");
    }
    else
      System.out.println("Unable to compute the running time based on your input.");

    return true;//allows the program to continue cycling until the user specifies otherwise
  }


//COMPUTE CASE 4/////////////////////////////////////////////////////////////////
  //USING THE MASTER THEOREM/////////////////////////////////////////////////////
  public static boolean computeCase4(int a, int b, int k){
    String big_O = null;
    if(a<(Math.pow(b,k))){      //Case 1: a<b^k
      if(k != 1)
        big_O = "n^"+k;
      else
        big_O = "n";
    }

    else if(a == Math.pow(b,k)){          //Case 2: a=b^k
      if(k != 1)
        big_O = "n^"+k+" log n";
      else       //if k=1
        big_O = "n log n";
    }

    else{       // assuming the last possibility is case 3, where a>b^k
        double logNumber = (Math.log(a)/Math.log(b));    //because if log is not base-2, rather base-4 or base-
X
        int parsedLog = (int)logNumber;                                    //as it's a double, we need to parse
it into an integer
        big_O = "n^"+parsedLog;
    }
    System.out.println("The big-O running time of the method (with the variables "+a+", "+b+", "+k+
      //print the result of big-O
```

```
                    ", representing a, b, and k, respectfully, is O("+big_O+").");
    return true;            //to keep the program looping
  }
}
```