**O(n log n) (Quicksort):**

public class nlogn {

  public static void swap(int[] arr, int i, int j) {

    //Function to make swapping two items easier and more succint.

    int temp = arr[i];

    arr[i] = arr[j];

    arr[j] = temp;

  }

  public static int partition(int[] arr, int low, int high) {

    //Will pick the highest index element as a pivot, and sort all smaller to left of pivot

    //and all greater to the right of pivot.

    int pivot = arr[high];

    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {

      if (arr[j] < pivot) {

        i++;

        swap(arr, i, j);

      }

    }

    swap(arr, i + 1, high);

    return (i + 1);

  }

  //partitionIndex will be placed at its correct spot, and then

  //the function will recursively do this for each element lower and higher than it.

  public static void quickSort(int[] arr, int low, int high) {

    if (low < high) {

      int partitionIndex = partition(arr, low, high);

      quickSort(arr, low, partitionIndex - 1);

      quickSort(arr, partitionIndex + 1, high);

    }

  }

  public static void main(String[] args) {

    //Quicksort Algorithm

    int[] data = {15, 14, 12, 21, 69, 42, 99};

    System.out.println("Array Before Quicksort:");

    for (int i = 0; i < data.length; i++) {

      System.out.print(data[i] + " ");

    }

    quickSort(data, 0, data.length - 1);

    System.out.println("Array After Quicksort:");

    for (int i = 0; i < data.length; i++) {

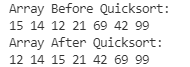
      System.out.print(data[i] + " ");

    }

  }

}

Output:



**O(6):**

import java.lang.Math;

public class n4 {

  public static void main(String[] args) {

    double n = 1000000;

    int count = 0;

    System.out.println("calculating...");

    while (n >= 1 && count <= 50) {

      n = Math.pow(n, 1.0/6.0);

      System.out.println(n);

      count += 1;

    }

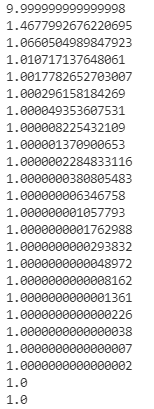
    System.out.println("Times run: " + count);

  }

}

It will eventually round to 1.0, where operations are miniscule and undoable on a double.

****

****

**O(4n):**

import java.lang.Math;

public class nroot {

  public static void main(String[] args) {

    double n = 1000000;

    int count = 0;

    System.out.println("calculating...");

    while (n >= 1 && count <= 35) {

      n = Math.pow(n, 1.0/4.0);

      System.out.println(n);

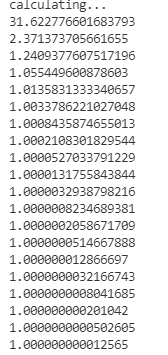
      count += 1;

    }

    System.out.println("Times run: " + count);

  }

}

****

This also eventually truncates to 1.

**O(n4):**

public class n4 {

  public static void main(String[] args) {

    int[] arr = {3, 4, 5, 7};

    int count = 0;

    for (int i = 0; i < arr.length; i++) {

      for (int j = 0; j < arr.length; j++) {

        for (int k = 0; k < arr.length; k++) {

          for (int l = 0; l < arr.length; l++) {

            System.out.println(count);

            count += 1;

          }

        }

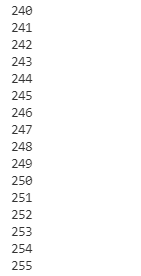
      }

    }

  }

}

This should run a total of 256 times because 44 is 256.



Since we started at 0, this is the correct solution.

