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:

```
Array Before Quicksort:
15 14 12 21 69 42 99
Array After Quicksort:
12 14 15 21 42 69 99
```

$O(\sqrt[6]{n})$:

```
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.

```
Times run: 51

9.999999999999998
1.4677992676220695
1.0660504989847923
1.010717137648061
1.0017782652703007
1.000296158184269
1.000049353607531
1.000008225432109
1.000001370900653
1.0000002284833116
1.0000000380805483
1.0000000063467 58
1.0000000010577 93
1.0000000001762988
1.0000000000293832
1.0000000000048972
1.0000000000008162
1.0000000000001361
1.0000000000000226
1.0000000000000038
1.0000000000000007
1.0000000000000002
1.0
1.0
```

$O(\sqrt[4]{n})$:

```
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);
  }
}
```

```
calculating...
31.622776601683793
2.371373705661655
1.2409377607517196
1.055449600878603
1.0135831333340657
1.0033786221027048
1.0008435874655013
1.0002108301829544
1.0000527033791229
1.0000131755843844
1.0000032938798216
1.0000008234689381
1.0000002058671709
1.0000000514667888
1.000000012866697
1.0000000032166743
1.0000000008041685
1.000000000201042
1.0000000000502605
1.000000000012565
```

This also eventually truncates to 1.

$O(n^4)$:

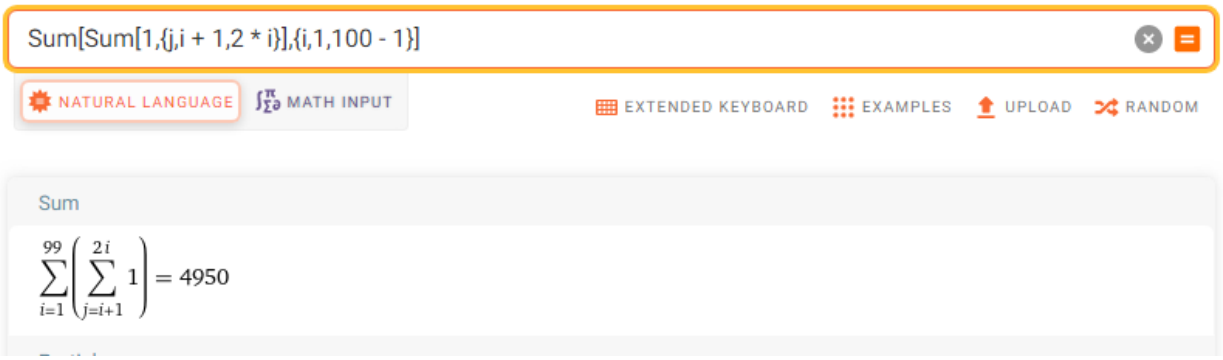
```
public class n4 {
  public static void main(String[] args) {
    int[] arr = {3, 4, 5, 7};
    int count = 0;
    for (int i = 0; i < arr.length; i++) {
      for (int j = 0; j < arr.length; j++) {
        for (int k = 0; k < arr.length; k++) {
          for (int l = 0; l < arr.length; l++) {
            System.out.println(count);
            count += 1;
          }
        }
      }
    }
  }
}
```

This should run a total of 256 times because 4^4 is 256.

```
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
```

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

```
1.  for (int i = 1; i < 100; i++)
2.      for (int j = i+1; j < 2*i; j++)
3.              something O(1)
```

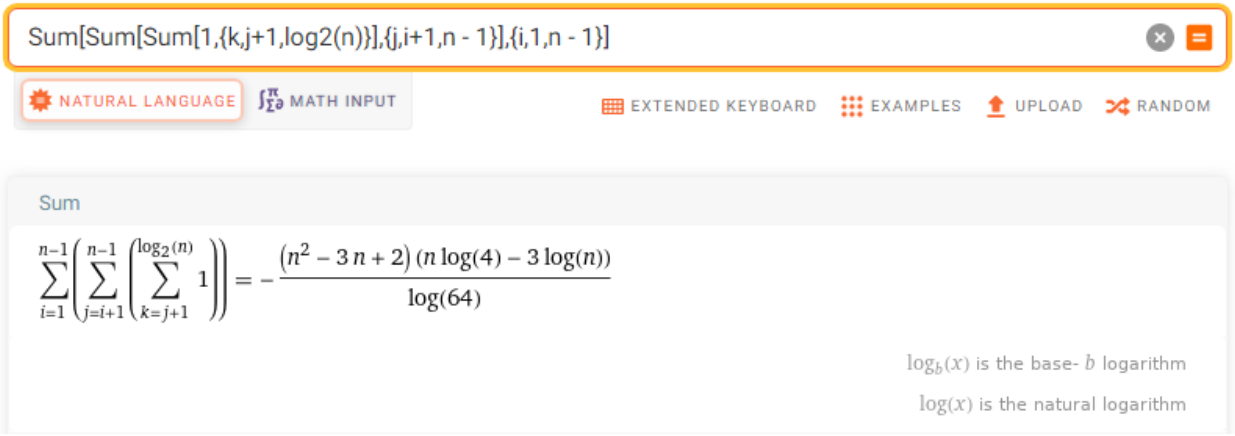


Choose the Alpha tool to conduct Big-O notations

```
1. for (int j = 1; j <= n; j++)
2.     for (int i = j+100; i<=n; i=i+1)
3.         something O(1)
4. for (int k = 0; k <= n; k=k*2)
5.     something O(1)
```

Choose the Alpha tool to conduct Big-O notations

```
1. for (int i = 1; i < n; i++)
2.     for (int j = i+1; j < n; j++)
3.         for (int k = j+1; k < n; k=k*2)
4.             something O(1)
```



Choose the Alpha tool to conduct Big-O notations

```
1. for (int i = 1; i < 100; i++)
2.     for (int j = i+1; j < 2*i; j++)
3.         for (int k = j+1; k < n; k=k*2)
4.             something O(1)
```