Algorithm Analysis Practice

## Big-O versus Actual Running Time

For each of the following program fragments, do the following

(a) Give a Big-O analysis of the running time based on static analysis of the code fragment

(b) Implement the code and run with several values of n. For example, n = 5, n = 50, n = 1,000

(c) Compare your analysis with the actual running time by completing the table.

// Fragment 1

for (int i = 0; i < n; i++)

sum++;

// Fragment 2

for (int i = 0; i < n; i += 2)

sum++;

// Fragment 3

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

sum++;

// Fragment 4

for (int i = 0; i < n; i++)

sum++;

for (int j = 0; j < n; j++)

sum++;

// Fragment 5

for (int i = 0; i < n; i++)

for (int j = 0; j < n \* n; j++)

sum++;

// Fragment 6

for (int i = 0; i < n; i++)

for (int j = 0; j < i; j++)

sum++;

// Fragment 7

for (int i = 1; i < n; i = i \* 2)

sum++;

// Fragment 8

for (int i = 0; i < n; i++) n

for (int j = 0; j < n \* n; j++) n \* n

for (int k = 0; k < j; k++) n \* n

sum++; n^5

To find the running time, the following code can be used:

long start = System.nanoTime();

//The code being timed goes here

long end = System.nanoTime();

long diff = end - start;

System.out.println(“Time to compute Fragment “ + fragNum + “ was “ + diff + “ milliseconds.”);

Complete the table below (indicate the actual value of n on the line and enter the time for each code fragment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code**  **Fragment** | **BigO**  **Analysis** | **Small value of n**  **­­­­\_\_\_\_\_1\_\_\_\_** | **Larger value of n**  **\_\_\_\_10\_\_\_\_\_** | **Largest n**  **\_\_\_1000\_\_\_\_\_\_** |
| #1 | O(n) | 1 | 10 | 1000 |
| #2 | O(n) | 1 | 5 | 500 |
| #3 | O(n^2) | 1 | 100 | 1000000 |
| #4 | O(n) | 2 | 200 | 2000 |
| #5 | O(n^3) | 1 | 1000 | 10000000 |
| #6 | O(n^2) |  |  |  |
| #7 | O( log2(n) ) |  |  |  |
| #8 | O(n^5) |  |  |  |