1. The estimate running time (or memory) is a function of input size *N*. Explain as to why the results are the same for the following three examples.

⅙ *N* 3 + 20 *N* + 16 ~ ⅙ *N* 3

⅙ *N* 3 + 100 *N* 4/3  + 56 ~ ⅙ *N* 3

⅙ *N* 3 - ½ *N* 2+ ⅓ *N* ~ ⅙ *N* 3

Ans: We discard lower order terms because when N is large, terms are negligible.

2. Write the Java code samples for the running times: constant 1, logN, N, NlogN, N^2, N^3, 2^N. Mathematically, how do you describe each one of these examples in the form of following equation?

Ans: constant 1:

Since there is no iteration, so the Time Complexity is O(1).

logN Complexity:

i= 1, 2, 3, .. n

= 2^0, 2^1, 2^2… 2^k

=>2^k = n

=> k = logN

NLogN Complexity:

i = 1, 2, 3… n

For i = 1, j = 1 to n = n, For j = 2, j = 1 to n = n/2, For i = n, j = 1 to n i.e., n/n loops

So,

n + n/2 + n/3 + n/4 + .. + 1

n(1+ 1/2 + 1/3 + … 1/n)

n(Log n)

N^2 Complexity:

For i= 1, 2,3.. n;

For j = 1, 2, 3… n;

Since there are two for loops that runs for n times each. So, time complexity is : O(N^2)

N^3 Complexity:

For i= 1, 2,3.. n;

For j = 1, 2, 3… n;

For k = 1, 2, 3.. n;

Since there are three for loops that runs for n times each. So, time complexity is : O(N^3)

2^N Complexity:

For i = 2^0, 2^1, … 2^n;

Since the loop will run for 2^n times. So, the time complexity is O(2^N)

3. Write the code that results to following running time. The 3-Sum Triple loop has the following running time estimate. Do Not prove the math. Just want explaining the math, what it represents and why the result is 1/6 N^3

Ans: The equation describes a triple loop program. Each for loop will run n times. But, variables j and k are dependent on i and j respectively. So, the time complexity is 1/6N^3, ignoring lower order terms.

4. Human use Infix expression and computers use Postfix expression. You are to write a simple Calculator. There are three steps: a) Read Infix expression, b) Convert Infix expression to Postfix by hand, and c) Evaluate Postfix expression, d) Write Java code using the referenced c-program example, compile and run with Infix expression examples.

Infix: (A + B) \* C + D / (E + F \* G) - H

Postfix: A B + C \* D E F G \* + / + H -

|  |  |
| --- | --- |
| Stack | Expression |
| ( | A |
| + | AB |
| ) | AB+ |
| \* | AB+C |
| + | AB+C\* |
|  | AB+C\*D |
| / |  |
| ( | AB+C\*DE |
| + | AB+C\*DEF |
| \* | AB+C\*DEFG |
| ) | AB+C\*DEFG\*+ |
| - | AB+C\*DEFG\*+/+H- |

Infix: (300 + 23) \* (43 - 21) / (84 + 7)

Postfix: 300 23 + 43 21 - \* 84 7 + /

|  |  |
| --- | --- |
| Stack | Expression |
| ( | 300 |
| + | 300 23 |
| ) | 300 23 + |
| \* | 300 23 + 43 |
| ( | 300 23 + 43 21 |
| - | 300 23 + 43 21- |
| ) | 300 23 + 43 21 - \* |
| / |  |
| ( | 300 23 + 43 21 - \* 84 |
| + | 300 23 + 43 21 - \* 84 7 |
| ) | 300 23 + 43 21 - \* 84 7 + / |

Infix: (4 + 8) \* (6 - 5)/((3 - 2) \* (2 + 2))

Postfix: 4 8 + 6 5 - \* 3 2 - 2 2 + \* /

|  |  |
| --- | --- |
| Stack | Expression |
| ( | 4 |
| + | 4 8 |
| )\* | 4 8 + |
| ( | 4 8 + 6 |
| - | 4 8 + 6 5 |
| ) | 4 8 + 6 5 - |
| / | 4 8 + 6 5 - \* |
| ( |  |
| ( | 4 8 + 6 5 - \* 3 |
| - | 4 8 + 6 5 - \* 3 2 |
| )\* | 4 8 + 6 5 - \* 3 2 - |
| ( | 4 8 + 6 5 - \* 3 2 - 2 |
| + | 4 8 + 6 5 - \* 3 2 - 2 2 |
| ) | 4 8 + 6 5 - \* 3 2 - 2 2 + |
| ) | 4 8 + 6 5 - \* 3 2 - 2 2 + \* / |

5. Consider the following Algorithm to convert Infix expression to Postfix.

a) Infix expression example: A \* B / C + (D + E - (F \* (G / H)))

b) Apply Algorithm to Infix example, show step-by-step

Ans: Postfix : A B \* C / D E + F G H / \* - +

|  |  |
| --- | --- |
| Stack | Expression |
| \* | A B |
| / | A B \* C |
| + | A B \* C / |
| ( | A B \* C / D |
| + | A B \* C / D E |
| - | A B \* C / D E + |
| ( | A B \* C / D E + F |
| \*( | A B \* C / D E + F G |
| / | A B \* C / D E + F G H |
| ))) | A B \* C / D E + F G H / \* - + |