Assignment #2

Due Date: Monday, September 14 at 11:59pm

Submit: eLearning

Late Policy: -10 points per hour late

Instructions: This is an individual assignment. Answers should be your own work.

Chapter 2

6 pts

1. In the definition of Big-O, why is the "for N >= n0" needed?

Solutions:

When a graph is compared for two functions, initially one function g(n) may be greater than f(n) at lower values of N but at the later points after a fixed point the f(n) is always greater than g(n). This is used to explain the relative rate of growth of functions.

6 pts

2. If f1(N) = 2N and f2(N) = 3N, why are they both O(N), since 3N is larger than 2N for N>=1?

Solutions:

A function can have many upper bounds but O(N) represents tightest upper bound for a function, in both cases the upper bound of the function is order of N. In O(N) definition T(N) = O(f(N)) means that T(N) <= cf(N) for some constant c and for N >= n0. Since there is a constant c both functions f1 and f2 will have same O(N).

6 pts

3. a) For f1(N) = 2N and f2(N) = 3N:

calculate f1(5) and f2(5), then f1(10) and f2(10). When N was doubled in each case, what happened to the result?

Solutions:

f1(5)=2\*5=10 and f2(5)=3\*5=15

while,

f1(10)=2\*10=20 and f2(10)=3\*10=30

Although both were different functions when the input was doubled their function values also doubled which means that they are O(N) i.e, linear growth (bound).

b) For f1(N) = 2N\*N and f2(N) = 3N\*N:

calculate f1(5) and f2(5), then f1(10) and f2(10). When N was doubled in each case, what happened to the result?

Solutions:

f1(5)=2\*5\*5=50 and f2(5)=3\*5\*5=75

while,

f1(10)=2\*10\*10=200 and f2(10)=3\*10\*10=300

Although both were different functions when the input was doubled their function values became four times (since, (2\*N)^2 = 4\*(N^2)) which means that they are O(N2) .

6 pts

4. Since Big-O notation is a mathematical tool for functions like f(N) or g(N), how is it applicable to algorithm analysis?

Solutions:

The algorithms consists of various steps to solve the problem whose running time can be compared to various functions like f(N) and g(N) to which such notations can be applied.

6 pts

5. Which grows faster, 2^n or n! ? Why?

Solutions:

For greater values like n>4, n! is always greater than 2^n, since n! Function increases a variable n while 2^n increases by a constant 2 only. Thus n! grows faster than 2^n.

10 pts (2 each)

6. Give the Big-O notation for the following expressions:

a. 4n^5 + 3n^2 - 2

b. 5^n - n^2 + 19

c. (3/5)\*n

d. 3n \* log(n) + 11

e. [n(n+1)/2 + n] / 2

Solutions:

1. O(n^5)
2. O(5^n)
3. O(n)
4. O(nlog(n))
5. O(n^2)

Questions 7-12 are 10 points each.

Assume numItems has the role of N, which may vary from one run to the next.

7. What is the Big-O running time for this code? Explain your answer.

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

System.out.println(i+1);

Solutions:

for (int i=0; i<numItems; i++) // 1+(numItems+1)+(numItems)

System.out.println(i+1); // 2\*numItems

Totally: 1+ (numItems+1) + (numItems) + 2\*numItems= 4\*numItems+2.

Thus O(numItems)

8. What is the Big-O running time for this code? Explain your answer.

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

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

System.out.println( (i+1) \* (j+1) );

Solutions:

for (int i=0; i<numItems; i++) // 1+(numItems+1)+(numItems)

for (int j=0; j<numItems; j++) // numItems\*[ 1+(numItems+1)+(numItems)]

System.out.println( (i+1) \* (j+1) ); // 4\*{ numItems\*[ 1+(numItems+1)+(numItems)]}

Totally = 1+(numItems+1)+(numItems) + numItems\*[ 1+(numItems+1)+(numItems)] + 4\*{ numItems\*[ 1+(numItems+1)+(numItems)]}

= [2\* numItems + 2] + {numItems\*[2\* numItems + 2]} + {4\*[ numItems\*(2\* numItems + 2)]}

= [2\* numItems + 2] + {2\* numItems^2 + 2\* numItems} + {4\*[2\* numItems^2 + 2\* numItems]}

= [2\* numItems + 2] + {2\* numItems^2 + 2\* numItems} + {8\* numItems^2 + 8\* numItems}

= 10\* numItems^2 + 12\* numItems + 2

Thus O(numItems^2)

9. What is the Big-O running time for this code? Explain your answer.

for (int i=0; i<numItems+1; i++)

for (int j=0; j<2\*numItems; j++)

System.out.println( (i+1) \* (j+1) );

Solutions:

for (int i=0; i<numItems+1; i++) // 1+(numItems+2)+(numItems+1)

for (int j=0; j<2\*numItems; j++) // (numItems+1)\*[1+(2\* numItems+1)+(2\* numItems)]

System.out.println( (i+1) \* (j+1) ); // 4\*{(numItems+1)\*[1+(2\*numItems+1)+(2\*numItems)]}

Totally: {1+(numItems+2)+(numItems+1)} + {(numItems+1)\*[1+(2\* numItems+1)+(2\* numItems)]} +

{4\*{(numItems+1)\*[1+(2\*numItems+1)+(2\*numItems)]}}

= {2\*numItems + 4} + {(numItems+1)+[ 4\*numItems + 2]} + { 4\*{(numItems+1)+[ 4\*numItems + 2]}}

= {2\*numItems + 4} + {4\*numItems^2 + 6\* numItems + 2} + {4\*{4\*numItems^2 + 6\* numItems + 2}}

= {2\*numItems + 4} + {4\*numItems^2 + 6\* numItems + 2} + {16\*numItems^2 + 24\* numItems + 8}

= 20\* numItems^2 + 32\* numItems +14

Thus O(numItems^2)

10. What is the Big-O running time for this code? Explain your answer.

if ( num < numItems )

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

{

System.out.println(i);

}

else

System.out.println("too many");

Solutions:

if ( num < numItems ) // 1

for (int i=0; i<numItems; i++) // 1+(numItems+1)+(numItems)

{

System.out.println(i); // 1\*numItems

}

else

System.out.println("too many");

Totally: 1+1 + (numItems+1) + (numItems) + 1\* numItems

=3\*numItems + 3

Thus O(numItems)

11. What is the Big-O running time for this code? Explain your answer.

int i = numItems;

while (i > 0)

i = i / 2; // integer division will eventually reach zero

Solutions:

int i = numItems; // 1

while (i > 0) // log(numItems) + 1

i = i / 2; // log(numItems)

Totally: 1+ log(numItems) + 1 + log(numItems)

=2\* log(numItems) + 2

Thus O(log(numItems))

12. What is the Big-O running time for this code? Explain your answer.

(You do not need to work out a recurrence formula).

public static int div(int numItems)

{

if (numItems == 0) //1

return 0;

else

return numItems%2 + div(numItems/2); // 1+1+1+1

}

Solutions:

In case of numItems=0

Run-time is 2

In other cases

5 (other case happens log(numItems))

Total of other cases = 5\* log(numItems)

Totally: 5 log(numItems) + 2

Thus O(log(numItems))

Submit these files:

hw2.doc (.doc can be .txt, .jpg, etc.)