CIS-277 Data Structures and Algorithms

Review: Analysis of Algorithms

1. One way to determine the execution speed of an algorithm is to implement it in a programming language, execute the program, and directly measure the actual time it takes to complete. Briefly explain the possible problem(s) with this approach.
2. For each of the following code segments:
   * identify the statements that are “mice”.
   * identify the Big-0 run-time (show how you arrived at the answer)
3. sum\_sqr = 0; **0(1)**

for( ct = 0; ct < size; ++ ct ) **0(N\*1)**

sum\_sqr += data[ct] \* data[ct]; mean\_sqr = sum\_sqr / size; **O(n) constant** loops

1. sum = 0

for( ct = 1; ct <= size; ++ct ) sum += ct; **O(n) 1 for loop**

product = 1;

for( ct = 1; ct <= size; ++ct ) product \*= ct;

difference = product – sum; **O(n) 1 for loop**

1. max = data[0, 0];

for( ctr = 0; ctr < n; ++ctr )

for( ctc = 0; ctc < n; ++ctc

if ( data[ctr] [ctc] > max ) max = data[ctr] [ctc]; **O(n^2) nested loops**

1. max = data[0, 0];

for( ctr =0; ctr < 5; ++ctr )

for( ctc = 0; ctc < 5; ++ctc )

if ( data[ctr] [ctc] > max ) max = data[ctr] [ctc]; **O(n^2) nested loops**

1. sum = 0;

for( ctr = 0; ctr < 3; ++ctr )

for( ctc = 0; ctc < n; ++ctc ) sum += data[ctr] [ctc]; **O(n^2) nested loops**

1. total =0; 0(1)

for( ctr = 0; ctr < n; ++ctr ) 0(n)

for( ctc = ctr + 1; ctc < n; ++ctc 0(n)

total += data[ctr] [ctc];

T(0(n\*n) + 1

**O(n^2) nested loops**

1. found = 0;

in.open( “a:client.dta:, ios::binary ); in.read((char\*)&client, sizeof(client)); while( !in.eof( ) && !found )

if ( client.id == id )

found = 1;

else in.close( ); **O(log n) Binary Seacrh**

in.read((char\*)&client, sizeof(client));

1. in >> size;

for( ctr = 0; ctr < size; ++ctr )

for( ctc = 0; ctc < size; ++ctc ) in >> score[ctr][ctc];

for( ctr = 0; ctr < ctr < size; ++ctr ) { total = 0;

for( ctc =0; ctc < size; ++ctc ) total += score[ctr] [ctc];

r\_mean[ctr] = total / column;

}

**O(n^2) nested loops**

1. cout << “Enter the number of data values; ”; cin >> size;

for ( ct = 0; ct < size; ++ct ) { cout << “Value:

cin >> data[ct];

}

sum = 0;

for ( ct = 0; ct < size; ++ct ) sum += data[ct];

mean = sum / size; **O(n) constant loops**

1. for ( ctr = 0; ctr < n; ++ ctr ) { sum[ctr] = 0;

for ( ctc = 0; ctc < n; ++ctc ) sum[ctr] += data[ctr] [ctc];

}

for ( ctr = 0; ctr < n; ++ctr )

cout << “Sum for row ”, ctr, “ is: ”, sum[ctr] << endl;

**O(n) constant loops**

Determine the Big-0 run-time of:

1. selection sort
2. insertion sort
3. binary search (worst case behavior )
4. Suppose that it takes 0.004 seconds to run a program on a test data set of size n = 200. Assume that the number of items in the actual data set is 4000. What would be the expected run time (show your work) if the program is applied to the actual data set and the underlying algorithm is:
5. Finding value of c in case of O(n)

* Execution time = c\*n
* c = Execution time / n
* c = 0.0004 / 200
* c = 2\*10-6

Execution time for n = 4000

* Execution time = c\*n
* Execution time = 2\*10-6\*4000 = 0.008 seconds

b)

Finding value of c in case of O(n2)

* Execution time = c\*n\*n
* c = Execution time / n\*n
* c = 0.0004 / 200\*200
* c = 1\*10-8

Execution time for n = 4000

* Execution time  = c\*n\*n
* Execution time  = 1\*10-8\*4000\*4000 = 0.16 seconds

1. Assume that it takes 0.0012 seconds to run a program on a test data set of size n = 50. Assume that the number of items in the actual data set is 80,000. What would be the expected run-time (show your work) if the program is applied to the actual data set and the underlying algorithm is:

a. 0(n) b. 0(n2)

1. Assume that an algorithm consists of two parts, one part is 0(n2) and the other in 0(n). What is the Big-0 run-time for the entire algorithm?
2. **The Paradox of Speed:**

**Suppose that computers become 16 times faster. If the same amount of computer time is to be used, how much can the data set size be increased if the underlying algorithm is:**

if algorithm is like O(n^2) + O(n) then we get O(n^2) as n^2 grows faster than n

and hence the algorithm is O(n^2)

(a) since O(n) is linear if computers are made 16 times faster, we get the new data as 16n

(b) here the computer is made 16 times faster

let us say for n we were taking T time

now for n we n we will take T/16 time

and for n=4\*n we will take 4n\*4n = 16 times the time taken earlier

hence if we were taking n data earlier now we can take 4\*n data

1. Determine the answer and explain why it is the answer: a. 0(n) + 0(n) = 0(2n) = **0(n)**

b. 0(n) + 0(n2) = 0(n + n^2) = 0(2n^2) = **0(n^2)**

c. 0(log n) + 0(n) = **0(n)**