Experiment No. 1

Name of the Experiment: Complexity of Algorithms

Duration: 1 cycle

Background Study: Chapter 1-2 (Theory and Problems of Data Structures Written by Seymour Lipschutz)

Definition of Big-O Notation: f(n) is O(g(n)) if and only if there exist two constants c and n_0 such that $|f(n)| < c|g(n)|$ for all $n > n_0$.

f(n) will normally represent the computing time of some algorithm. When we say that the computing time of an algorithm is O(g(n)) we mean that its execution takes no more than a constant times *g(n)*. *n* is a parameter which characterizes the inputs and/or outputs. For example *n* might be the number of inputs or the number of outputs or their sum or the magnitude of one of them.

Example

Let f(n) = 3n+2
Question: Find the Big-O notation of f(n).
We have, f(n)=3n+2
We can write, $f(n) \le 3n + n$ if $n \ge 2$
$f(n) \le 4n$,
From definition, $n_0 = 2$ and c = 4, g(n) = n
$f(n) \le cg(n)$
f(n) is O(g(n)) or f(n) is O(n)

Problem I: Find the Complexity of a Loop.

Algorithm1.1:

1. Repeat for K = 1 to n by 1
2. Write: K

   [End of Step 1 loop]

3. Exit

Complexity: the computing time of the loop, f(n) = n, So the complexity of the above algorithm is O(n). Here $f(n) \le cg(n)$, c =2, n_0 = 0, g(n) = n.

Complexity Table:

n	f(n) [from Program, Count Statement]	cg(n) [Theoretical]
10		
20		

Graph: Draw a Graph.

Problem II: Find the Complexity of the following Program.

Algorithm1.2:

1. Repeat for K = 1 to n by 1
2. Repeat for L = 1 to n by 1
3. Write: L

   [End of Step 2 loop]

4. Write: K

   [End of Step 1 loop]

5. Exit

Complexity: the computing time of the loop, $f(n) = n^2$, So the complexity of the above algorithm is $O(n^2)$. (Let c= 2)

Complexity Table:

n	f(n) [from Program, Count Statement]	cg(n) [Theoretical]
10		
20		

Graph: Draw a Graph.

Problem III: Find the Complexity of the elementary Sort algorithm.

Algorithm1.3: (Given a nonempty array A with n numerical values. This algorithm sorts the values)

1. Repeat for i = 2 to n by 1
2. Repeat for k = i to 1 by -1
3. If A[k]<A[k-1] then:
   Swap (A[k], A[k-1])
   [End of If Structure]
   [End of Step 2 loop]
   [End of Step 1 loop]
4. Exit

Complexity: The complexity of the above algorithm is $O(n^2)$. (Let c= 2)

Complexity Table:

n	f(n) [from Program, Count Statement]	cg(n) [Theoretical]
10		
20		

Graph: Draw a Graph.

Problem IV: Find the largest element in Array.

Algorithm1.4: (Given a nonempty array A with n numerical values. This algorithm finds the location LOC and the value MAX of the largest element of A)

1. Set K:=1, LOC:=1 and MAX:=A[1]
2. Repeat steps 3 and 4 while K≤n
3. IF MAX < A[K] then:
   Set LOC:=K and MAX:=A[K].
   [End of If structure]
4. K:=K+1.
   [End of step 2 loop]
5. Write: LOC, MAX.
6. Exit

Complexity: Not Required

Problem V: Linear Search.

Algorithm1.5: (Given a nonempty array A with n numerical values and a specific x of information is given. This algorithm finds the location LOC of x in the array A or Sets LOC=-1)

1. Set K:=1, LOC:=-1
2. Repeat steps 3 and 4 while LOC = -1 and K≤n
3. IF x = A[K] then:
   Set LOC:=K.
   [End of If structure]
4. K:=K+1.
   [End of step 2 loop]
5. If LOC = -1 then: Write: x is not in the array A.
   Else: Write: LOC is the location of x
   [End of If structure]
6. Exit

Complexity: The worst case complexity is C(n) =n and the average case complexity is C(n) = (n+1)/2.

MORE PROBLEMS

1. Programming Problems of Chapter 1 and 2 of "Data Structures" by Seymour Lipschutz.

LAB REPORT: You have to submit all assigned problems in next lab.