C Programming

Lesson 9: Algorithms

Table Of Contents

Lesson Objectives

In this lesson, you will learn about:

- This course introduces the analysis and design of computer algorithms
- Various notations best, worst, average cases. Big Oh, small oh and theta notations
- Comparing all sorting and searching algorithms

Concept of Algorithms

- An Algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.
- An Algorithm is thus a sequence of computational steps that transform the input into the output. It is a tool for solving a well - specified computational problem.
- The analysis of algorithms is the determination of the amount of resources (such as time and storage) necessary to execute them.
- Time Complexity vs Space Complexity

Concept of Algorithm analysis Contd…

- These estimates provide an insight into reasonable directions of search for efficient algorithms.

- In theoretical analysis of algorithms it is common to estimate their complexity in the asymptotic sense, i.e., to estimate the complexity function for arbitrarily large input.

- Big O notation, Big-omega notation and Big-theta notation are used to the theoretical analysis .

- For instance, binary search is said to run in a number of steps proportional to the logarithm of the length of the list being searched, or in O(log(n)), colloquially "in logarithmic time".

Cost model

- Time efficiency estimates depend on what one define to be a step.
- For the analysis to correspond usefully to the actual execution time, the time required to perform a step must be guaranteed to be bounded above by a constant.
- One must be careful here; for instance, some analyses count an addition of two numbers as one step.
- For example, if the numbers involved in a computation may be arbitrarily large, the time required by a single addition can no longer be assumed to be constant.
- Two cost models are
  - The uniform cost model
  - The logarithmic cost model

Best, worst and average case

- Best, worst and average cases of a given algorithm express what the resource usage is at least, at most and on average, respectively.

- Usually the resource being considered is running time, i.e. time complexity, but it could also be memory or other resources.

- Average performance and worst-case performance are the most used in algorithm analysis. Less widely found is best-case performance, but it does have uses.

- For example, where the best cases of individual tasks are known, they can be used to improve the accuracy of an overall worst-case analysis.

Best, worst and average case (cont..)

Worst-Case, Best-Case, and Average-Case

algorithm SequentialSearch(A[0..n − 1], K)

// Searches for a value in an array

// Input: An array A and a search key K

// Output: The index where K is found or −1

```
for i ← 0 to n − 1 do
        if A[i] = K then return i
    return −1
```

- Basic Operation: The comparison in the loop
- Worst-Case: n comparisons
- Best-Case: 1 comparison
- Average-Case: (n+1)/2 comparisons assuming each element equally likely to be searched.

Comparison of all sorting Algorithms

Algorithm	Data Structure	Time Complexity:Best	Time Complexity:Average	Time Complexity:Worst	Space Complexity:Worst
Quick Sort	Array	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Merge sort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Heap sort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Insertion sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Selection sort	Array	O(n^2)	O(n^2)	O(n^2)	O(1)

Comparison of all Data Structures

Data structure	Time Complex: Avg: Search	Time Complex: Avg: Insertion o(n2)	Time Complex: Avg: Deletion	Time Complex: Worst: Search	Time Complex: Worst: Insertion	Time Complex: Worst: Deletion	Space Complex: Worst
Basic Array	O(n)	-	-	O(n)	-	-	O(n)
Dynamic array	O(n)	O(n)	-	O(n)	O(n)	-	O(n)
Singly linked list	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)	O(n)
Doubly linked list	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)	O(n)
Binary Search Tree	O((log n))	O((log n))	O((log n))	O(n)	O(n)	O(n)	O(n)

Big O, Omega and Theta

- Big O notation is used to classify algorithms by how they respond (e.g., in their processing time or working space requirements) to changes in input size

- The difference between Big O notation and Big Omega notation is that Big O is used to describe the worst case running time for an algorithm.

- Big Omega is used to represent the lower bound, which is also the “best case” for that algorithm

- It is also possible to consider the "greater than or equal to" relation and "equal to" relation in a similar way. Big-Omega is for the former and big-theta is for the latter.

Big O, Omega and Theta With an example C Code

- For Example in an array of n elements, If we wanted to access the first element of the array this would be O(1) since it doesn't matter how big the array is, it always takes the same constant time to get the first item.

```
int array[n];
x = array[0];
        for(int i = 0; i < n; i++)
        {       if(array[i] == numToFind)
                        { return i; }
         }   // to find a number in the list:
```

- This would be O(n) since at most we would have to look through the entire list to find our number.

- The Big-O is still O(n) even though we might find our number the first try and run through the loop once because Big-O describes the upper bound for an algorithm (omega is for lower bound and theta is for tight bound).

Time vs Space Complexity

- The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input.
- The time complexity of an algorithm is commonly expressed using big O notation, which excludes coefficients and lower order terms.
- Space Complexity, It represents the total amount of memory space that a "normal" physical computer would need to solve a given computational problem with a given algorithm.
- Space Complexity corresponds to the amount of physical computer memory needed to run a given program.

IGATE
Speed.Agility.Imagination

Example C Program to calculate Time Complexity

```
#define LISTSIZE 100000    //Number of integers to be generated
 #include <stdio.h>
 #include <math.h>
 #include <time.h>
 #include <conio.h>

int main()
{
   clock_t start;
   double d;
   int primesFound;
   long int list[LISTSIZE],i,j;
   int listMax = (int)sqrt(LISTSIZE), primeEstimate = (int)(LISTSIZE/log(LISTSIZE));

   for(i=0; i < LISTSIZE; i++)
              list[i] = i+2;
   start=clock();
```

Example C Program (Contd…)

```
for(i=0; i < listMax; i++)
  {
          //If the entry has been set to 0 ('removed'), skip it
          if(list[i] > 0)
          {
                  //Remove all multiples of this prime
                  //Starting from the next entry in the list
                  //And going up in steps of size i
                  for(j = i+1; j < LISTSIZE; j++)
                  {
                          if((list[j] % list[i]) == 0)
                                  list[j] = 0;
                  }
          }
  }
```

Example C Program (Contd…)

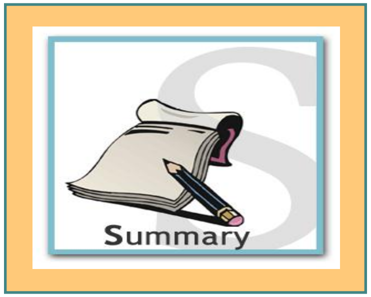
```
d=(clock()-start)/(double)CLOCKS_PER_SEC;

   //Output the primes
   primesFound = 0;
   for(i=0; i < LISTSIZE; i++)
   {
           if(list[i] > 0)
           {
                   primesFound++;

                   printf("%ld\n", list[i]);
           }
   }
   printf("\n%f",d);
   getch();
   return 0;
}
```

Summary

- **In this lesson, you have learnt:**
  - The Definition of Algorithm Analysis
  - The various cost analysis models
  - Best, worst and average case
  - Comparison of all sorting Algorithms
  - Comparison of all Data Structures
  - The definitions of Big O, Omega and Theta
  - Example for Big O, Omega and Theta understandings
  - Time VS Space Complexity



Review Questions

1. Which of the following case does not exist in complexity theory
   a. Best case
   b. Worst case
   c. Average case
   d. Null case
2. The Worst case occur in linear search algorithm when
   a. Item is somewhere in the middle of the array
   b. Item is not in the array at all
   c. Item is the last element in the array
   d. Item is the last element in the array or is not there at all
3. Two main measures for the efficiency of an algorithm area.
   Processor and memory
   b. Complexity and capacity
   c. Time and space
   d. Data and space