Name: Archit Agrawal
Student ID: 202052307

Introduction to Data Structures (CS102)

Assignment 1

1.1 Define Algorithm and describe the criteria for all algorithm.

An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. It is a well-defined computational procedure that takes some value, or set of values as input and produces some value, or set of values, as output. It is a step-by-step procedure to solve a problem in finite number of steps.

An algorithm must satisfy the following criteria:

(i) Input: An algorithm should have zero or more but should be a finite number of inputs.

(ii) Output: An algorithm must give at least one output. These output values are known as solution to the problem.

(iii) The steps in an algorithm must be clear, that is, there must not be any ambiguity in the steps.

(iv) Algorithm must terminate in a finite time.
(v) There should be no infinite sequence of steps.
(vi) It should be feasible with any specified computational device.

1.2 Describe the definition, objective and functions of ADT Natural Number.

An abstract data type is a data type that is organised in such a way that the specifications of the objects and the specification of the operations on the objects is seperated from the representation of the objects and implementation of the operations.

ADT Natural Number has:

objects: an ordered subrange of integers starting at 0 and ending at a maximum integer (INT_MAX) on the computer

functions:
for all x, y ∈ Natural Number
TRUE, FALSE ∈ boolean and where, +, −, <
and == are usual operators.

(i) Constructor Function
Natural Number Zero( ) ::= 0

(ii) Boolean iszero(x) ::= if(x) return FALSE
else return TRUE.

(iii) Boolean equal(x,y) ::= if (x==y) return TRUE
else return FALSE

(iv) Natural Number successor(x) ::= if (x==INT_MAX)
return x
else return x+1

(v) Natural Number add(x,y) ::=
if ((x+y) <= INT_MAX) return x+y
else return INT_MAX

(vi) Natural Number subtract(x,y) ::=

if (x<y) return 0
else return x-y

end Natural Number

1.3 What is the difference between performance analysis and performance of a program?

Performance Analysis focuses on obtaining estimates of time and space that are machine independent.

Performance Management focuses on estimating time and space that are machine dependent and at running time.

Name: Archit Agrawal
Student ID: 202052307

1.4 How do we measure the space and time complexity?

Space Complexity :- The space complexity of a program is the amount of memory that it needs to run to completion.

Time Complexity :- The time complexity of a program is the amount of computer time that it needs to run to completion.

For measuring the space complexity, we must know the size (memory used) of (by) different types of variables present in the program. The memory used by different types of variables might depend on the machine but the basic procedure remains same.

- Now as we know the memory used by each variable, we calculate how many variables of each type are there in the program.
- Then we multiply the frequency of each type of variable with the memory occupied by it and add them to get the space complexity.

Example :-

```
int arraySum (int a[n], int n){
    int sum = 0;
    for(int i = 0; i < n; i++){
        sum += a[i];
    }
    return sum;
}
```

In the above program, we can see only data type present is (int). Let us say size of int is 4 bytes

Now, there is an array of n integers, and there are integers n, sum, i and one int is stored as return type.

Hence, we have a total of n+4 integers

∴ Space complexity is (n+4) * 4, which is equal to 4n+16.

Time Complexity :- It measures that how many times the statements in a program have been executed and not the time in seconds.

Ex:

```
for (int i=0; i<n; i++){
    statement 1;
}
```

If we analyze the above piece of code we will find that,

- statement 1 will be executed n times
- inside the for loop, the loop variable will be initiated 1 time and it will be updated n times. So, the total time becomes n+1.

Now, if we compute the total time, it will come out to be 2n+1. But generally, we want to figure out the growth of time complexity relative to the input, and we can get that by the degree of the time complexity polynomial. This notation is known as the Big-Oh notation. Hence, the time complexity of above piece of code will be O(n).

Ex.

```
int add (int a, int b, int c){
    int x = a+b;
    int y = x+c;
    return y;
}
```

In this, you can see that the three statements will execute once. Hence, the time complexity polynomial is 3. Since, it is zero degree polynomial, the time complexity of this function is O(1).

1.5 Define three asymptotic notations. What are its significances? Why are these notations used in Data Structures concept? Compare these three notations.

Asymptotic notations enable us to make meaningful but inexact statements about the time and space complexities of a program. As the word asymptotic suggests, these notations helps us in approaching a value or curve arbitrarily close.

There are three different Asymptotic notations:-

1. Big-Oh Notation (O)
2. Big-Omega Notation (Ω)
3. Big-Theta Notation (Θ)

1. Big-Oh Notation :- This notation is known as the upper bound of the algorithm, or a worst case of an algorithm. It tells us that a certain function will never exceed a specified time for any value of input.

for example, consider the case of finding an element in an array, then the Big-Oh notation of time complexity will be $O(n)$, which means that the time complexity will never exceed n, defining the upper bound, hence saying that it can be lesser than or equal to n.

2. Big Omega Notation (Ω) :- Big Omega notation is used to define the lower bound of any algorithm, or we can say the best case of any algorithm. Simply, when we indicate time complexity in the form of big-Ω, we mean that the algorithm will take atleast this much time to complete its execution. for the example of linear search, the big-Ω notation will be $\Omega(1)$ because the best case is when we find the element at index 0 itself.

3 Big-Theta Notation (Θ) :- This notation is known as the tight bound of an algorithm, i.e. the complexity represented by Big Θ notation is like the average value or range within which the actual time of execution will lie.

For example, if for some algorithm, time complexity is represented by $3n^2 + 5n$, and we want to represent this as Big-Θ, then this complexity will be $\Theta(n^2)$. It means that the average time for any input n will remain in between $k_1 * n^2$ and $k_2 * n^2$, where

K_1 and K_2 are constants and hence tightly binding the expression representing the growth of algorithm.

Significance of these Notations -:

(i) Big Oh -: When we want to know that the running time grows at most this much, but it could grow more slowly, we use Big-Oh notation.

(ii) Big-Ω -: When we want to that the running time takes at least ~~this~~ a certain amount of time, we use Big-Ω notation.

(iii) Big-Θ :- When we want to talk about both lower and upper ~~bounds~~ asymptotic bounds, we use ~~Big-Oh~~ Big-Θ notation.

These notations are used in data structure concepts as they describe the time and space complexities of algorithms, and the change in time and space if the input is varied.

1.6. Show that the following statements are correct

(a) $5n^2 - 6n = \Theta(n^2)$

Name: Archit Agrawal
Student ID: 202052307

Date:
Page: 10

$f(n) = 5n^2 - 6n$, $g(n) = n^2$

for θ notation,

the equation $0 \leq c_1 n^2 \leq 5n^2 - 6n \leq c_2 n^2 \quad \forall n \geq n_0$

If $n_0 = 0$, $c_1 = 5$ and $c_2 = 6$ which holds true for all $n \geq 0$.

Since, there exists positive constants c_1, c_2, n_0

$\therefore \quad 5n^2 - 6n = \theta(n)$ is true.

(b) $\quad n! = O(n^n)$

Here, $f(n) = n! = n(n-1)(n-2) \cdots 1$

and, $g(n) = n^n = n \cdot n \cdot n \cdots n$

the equation: $0 \leq f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$

let $n_0 = 1$ and $c = 1$

since, $n > n-1$, $n > n-2$, $\cdots$, $n \geq 1$ and so on.

So,

$\underbrace{n \times n \times n \times \cdots \times n}_{n \text{ times}} \geq n(n-1)(n-2) \cdots 1 \quad \forall n \geq 1$

$\therefore \quad 0 \leq n! \leq n^n$

Since there exists positive coefficients $n_0 = 1$ and $c = 1$, the equation

$n! = O(n^n)$ is true.

(c) $2n^2 + n \log n = \Theta(n^2)$

$f(n) = 2n^2 + n \log n$

$g(n) = n^2$

as $\log n < n^2 \quad \forall n \geq 1$

$\therefore 2n^2 + \log n < 3n^2 \quad \forall n \geq 1$

and, for $\forall n \geq 1$

$n^2 \leq 2n^2$

$\therefore n^2 < 2n^2 + \log n$

$\therefore$ we have

$0 \leq n^2 \leq 2n^2 + \log n \leq 3n^2 \quad \forall n \geq 1$

Since, we have three positive values $n_0 = 1$, and $c_0 = 1$ and $c_1 = 3$, we can say that

$2n^2 + n \log n = \Theta(n^2)$ is true

Name: Archit Agrawal
Student ID: 2010522307

(d) $\sum_{i=0}^{n} i^2 = \Theta(n^3)$

$$f(n) = \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$g(n) = n^3$$

we have to verify

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$$

$$\frac{n(n+1)(2n+1)}{6} \leq c_2 n^3$$

$$2n^3 + 3n^2 + n \leq 6c_2 n^3$$

$$n^3(6c_2 - 2) - 3n^2 - n \geq 0$$

For this to hold, $6c_2 - 2 = 10$ is a possible solution,

hence, $c_2 = 2$ is valid solution.

and, $$\frac{n(n+1)(2n+1)}{6} \geq c_1 n^3$$

$$2n^3 + 3n^2 + n \geq 6c_1 n^3$$

$$n^3(6c_1 - 2) - 3n^2 - n \leq 0$$

$6c_1 - 2 = \frac{1}{8}$ is a solution for all $n \geq 2$

So, $c_1 = \left(\frac{1}{8} + 2\right)/6 \Rightarrow c_1 = 0.354$

Since, equation a has valid solution for positive constants $n_0 = 2$, $c_1 = 0.354$ and $c_2 = 2$.

$\therefore \sum_{i=0}^{n} i^2 = \theta(n^3)$ is true.

(e) $\sum_{i=0}^{n} i^3 = \theta(n^4)$

$f(n) = \sum_{i=0}^{n} i^3 = \frac{n^2 (n+1)^2}{4}$

$g(n) = n^4$

Clearly,

$(n(n+1))^2 \geq (n(n))^2 \quad \forall\ n \geq 1$

$\therefore \left(\frac{n(n+1)}{2}\right)^2 \geq c_1 (n^4) \quad \forall\ n \geq 1$

where $c_1 = \frac{1}{4}$ is a possible solution.

Now, $c_2 n^4 \geq \frac{1}{4} (n(n+1))^2 \quad \forall\ n \geq 1$

$4c_2 n^4 \geq (n(n+1))^2$

$4c_2 n^4 \geq n^4 + 2n^3 + n^2$

Putting $n = 1$

$4c_2 \geq 4$

$c_2 \geq 1$

Hence $c_2 = 1$ is valid

Name: Archit Agrawal
Student ID: 2020512307

we see that

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ is true

for $n_0 = 1$, $c_1 = \frac{1}{4}$ and $c_2 = 1$

Hence $\sum_{i=0}^{n} i^3 = \Theta(n^4)$ is true

(f) $n^{2^n} + 6.2^n = \Theta(n^{2^n})$

$f(n) = n^{2^n} + 6.2^n$

$g(n) = n^{2^n}$

$\therefore \quad n^{2^n} + 6.2^n \geq n^{2^n} \quad \forall n \geq 1$

$\therefore \quad c_1 = 1$

and, $n^{2^n} + 6.2^n \leq c_2 n^{2^n} \quad \forall n \geq 1$

holds true for $c_2 = 3$

$\therefore \quad 0 \leq c_1 (n^{2^n}) \leq f(n) \leq c_2 (n^{2^n}) \quad \forall n \geq (n_0 = 1)$

has a valid solution iff $n_0 = 1$, $c_1 = 1$, and $c_2 = 3$.

$\therefore \quad n^{2^n} + 6.2^n = \Theta(n^{2^n})$ is true.

(g) $n^3 + 10^6 n^2 = \Theta(n^3)$

$f(n) = n^3 + 10^6 n^2$

$g(n) = n^3$

Clearly $f(n) \geq g(n)$ for $\forall n \geq 1$

$\therefore c_1 = 1$

Now,

$n^3 + 10^6 n^2 \leq c_2 n^3$

$n^2 (n + 10^6) \leq c_2 n^3$

$n^2 [n(c_2 - 1) - 10^6] \geq 0$

which holds when $c_2 = 10^6 + 1 \quad \forall n \geq 1$

Since we have $n_0 = 1$, $c_1 = 1$ and $c_2 = 10^6 + 1$.

$\therefore n^3 + 10^6 n^2 = \Theta(n^3)$ is true

(h) $\dfrac{6n^3}{(\log n + 1)} = O(n^3)$

$f(n) = 6n^3/(\log n + 1)$, $g(n) = n^3$

$\log n \geq 0$ for $\forall n \geq 1$

$\therefore \dfrac{6n^3}{(\log n + 1)} \leq 6n^3 \quad \forall n \geq 1$

$\therefore \quad c = 6$

i.e. $\quad 0 \leq \dfrac{6n^3}{(\log n + 1)} \leq 6n^3 \quad \forall\, n \geq 1$

or $\quad 0 \leq f(n) \leq c\, g(n)$ holds true.

Hence, $\dfrac{6n^3}{(\log n + 1)} = O(n^3)$ is true

(i) $n^{1.001} + n \log n = \theta(n^{1.001})$

n^k dominates $\log(cn)$ for $k, c > 0$

But for n sufficiently large and $k > 0$
n^k dominates $\log n$.

$$\lim_{n \to \infty} \frac{\log n}{n^k} = \frac{1/n}{k\, n^{k-1}} = \frac{1}{k n^k} = 0$$

So, for large n, kn^k is dominating and tending to infinity.

$$n^{1.001} + n \log n = O(n^{1.001})$$

$$n^{1.001} + n \log n \geq n^{1.001}$$

Hence, $n^{1.001} + n \log n = \Omega(n^{1.001})$

$\therefore \quad n^{1.001} + n \log n = \theta(n^{1.001})$ is true

Name: Archit Agrawal
Student ID: 2020052307

Date:
Page: 17

(j) $n^k + n + n^k \log(n) = \theta(n^k \log n)$ for all $k \geq 1$

$f(n) = n^k + n + n^k \log(n)$

$g(n) = n^k \log n$

$n^k \geq n$ for all $k \geq 1$

$n^k \log n \geq n^k$ for all $k \geq 1$ and $\forall n \geq 10$

$\therefore n^k + n + n^k \log n = O(n^k \log n) \quad \forall n \geq 10$

$n^k + n + n^k \log n \leq n^k \log n \quad \forall n \geq 10$

$\therefore n^k + n + n^k \log n = \Omega(n^k \log n)$

$\therefore n^k + n + n^k \log n = \theta(n^k \log n)$ is true

(K) $10n^3 + 15n^4 + 100n^2 \cdot 2^n = O(n^2 2^n)$

since $n^4 \geq n^3 \quad \forall n \geq 1$

and, $2^n \geq n^2 \quad \forall n \geq 1$

$\therefore 100 n^2 2^n \geq n^4 \times 100 \quad \forall n \geq 1$

$\therefore 10n^3 + 15n^4 + \overset{100x}{n^2 2^n} \geq 117n^4 \quad \forall n \geq 1$

Hence, $0 \leq f(n) \leq 17n^4$ holds true for $n=1$ and $c=117$

$\therefore 10n^3 + 15n^4 + 100n^2 \cdot 2^n = O(n^2 2^n)$ is true

Name: Archit Agrawal
Student ID: 20205 2307

1-7 Show that the following statements are incorrect

(a) $10n^2 + 9 = O(n)$

We can see that

$10n^2 + 9 \geq c(n) \quad \forall n$ and $\forall c$

$\therefore 0 \leq 10n^2 + 9 \leq cn$ doesn't hold true.

Hence, $10n^2 + 9 = O(n)$ is incorrect.

(b) $n^2 \log n = \Theta(n^2)$

$f(n) = n^2 \log n$

$g(n) = n^2$

$\because \quad n^2 \log n \geq n^2 \quad \forall n \geq 10$

$\therefore$ there exists no c_2 for which

$n^2 \log n \leq cn^2 \quad \forall n \geq 10$

Hence, $n^2 \log n = \Theta(n^2)$ is incorrect.

Name: Archit Agrawal
Student ID: 201052307

(c) $\frac{n^2}{\log n} = \Theta(n^2)$

as $\log n \geq 1 \quad \forall n \geq 10$

$\therefore \frac{n^2}{\log n} \leq n^2 \quad \forall n \geq 10$

Hence, there is no positive value of c_2 for which

$\frac{n^2}{\log n} \geq n^2$ holds true

Hence, $\frac{n^2}{\log n} \geq \Theta(n^2)$ is incorrect.

(d) $n^3 2^n + 6n^2 \cdot 3^n = O(n^2 \cdot 2^n)$

$\therefore n^2 \cdot 3^n \geq n^2 \cdot 2^n \quad \forall n \geq 0$

$\therefore n^3 \cdot 2^n + 6n^2 \cdot 3^n \geq n^2 \cdot 2^n \quad \forall n \geq 0$

Hence, big-Oh condition is not satisfied.

$\therefore n^3 \cdot 2^n + 6n^2 \cdot 3^n = O(n^2 \cdot 2^n)$ is incorrect.

Name: Archit Agrawal
Student ID: 2020052367

Date:
Page: 20

(e) $3^n = O(2^n)$

Clearly, $3^n \geq 2^n \quad \forall n \geq 0$

Hence, Big-Oh condition is not satisfied.

$\therefore$ $3^n = O(2^n)$ is incorrect.