CSE 1201

Data Structure

Week-1

Lecture 02

Instructors:
Md. Shahid Uz Zaman &
Md. Nazrul Islam Mondal
Dept of CSE, RUET

Mathematical Notation

Floor and Ceiling Functions

Let x be any real number. Then x lies between two integers called the floor and the ceiling of x. Specifically,

$\lfloor x \rfloor$, called the *floor* of x, denotes the greatest integer that does not exceed x.

$\lceil x \rceil$, called the *ceiling* of x, denotes the least integer that is not less than x.

If x is itself an integer, then $\lfloor x \rfloor = \lceil x \rceil$; otherwise $\lfloor x \rfloor + 1 = \lceil x \rceil$.

Example 2.1

$$\lfloor 3.14 \rfloor = 3, \quad \lfloor \sqrt{5} \rfloor = 2, \quad \lfloor -8.5 \rfloor = -9, \quad \lfloor 7 \rfloor = 7$$

$$\lceil 3.14 \rceil = 4, \quad \lceil \sqrt{5} \rceil = 3, \quad \lceil -8.5 \rceil = -8, \quad \lceil 7 \rceil = 7$$

Remainder Function; Modular Arithmetic

Let k be any integer and let M be a positive integer. Then

$$k \pmod{M}$$

(read k *modulo* M) will denote the integer remainder when k is divided by M. More exactly, $k \pmod{M}$ is the unique integer r such that

$$k = Mq + r \quad \text{where} \quad 0 \le r < M$$

When k is positive, simply divide k by M to obtain the remainder r. Thus

$$25 \pmod 7 = 4, \quad 25 \pmod 5 = 0, \quad 35 \pmod{11} = 2, \quad 3 \pmod 8 = 3$$

Mathematical Notation

Integer and Absolute Value Functions

Let x be any real number. The *integer value* of x, written INT(x), converts x into an integer by deleting (truncating) the fractional part of the number. Thus

$$\text{INT}(3.14) = 3, \quad \text{INT}(\sqrt{5}) = 2, \quad \text{INT}(-8.5) = -8, \quad \text{INT}(7) = 7$$

Observe that $\text{INT}(x) = \lfloor x \rfloor$ or $\text{INT}(x) = \lceil x \rceil$ according to whether x is positive or negative.

The *absolute value* of the real number x, written ABS(x) or $|x|$, is defined as the greater of x or $-x$. Hence $\text{ABS}(0) = 0$, and, for $x \neq 0$, $\text{ABS}(x) = x$ or $\text{ABS}(x) = -x$, depending on whether x is positive or negative. Thus

$$|-15| = 15, \quad |7| = 7, \quad |-3.33| = 3.33, \quad |4.44| = 4.44, \quad |-0.075| = 0.075$$

We note that $|x| = |-x|$ and, for $x \neq 0$, $|x|$ is positive.

Summation Symbol; Sums

Here we introduce the summation symbol Σ (the Greek letter sigma). Consider a sequence $a_1, a_2, a_3, \ldots$. Then the sums

$$a_1 + a_2 + \cdots + a_n \quad \text{and} \quad a_m + a_{m+1} + \cdots + a_n$$

will be denoted, respectively, by

$$\sum_{j=1}^{n} a_j \quad \text{and} \quad \sum_{j=m}^{n} a_j$$

Mathematical Notation

Permutations

A *permutation* of a set of n elements is an arrangement of the elements in a given order. For example, the permutations of the set consisting of the elements a, b, c are as follows:

$$abc, \quad acb, \quad bac, \quad bca, \quad cab, \quad cba$$

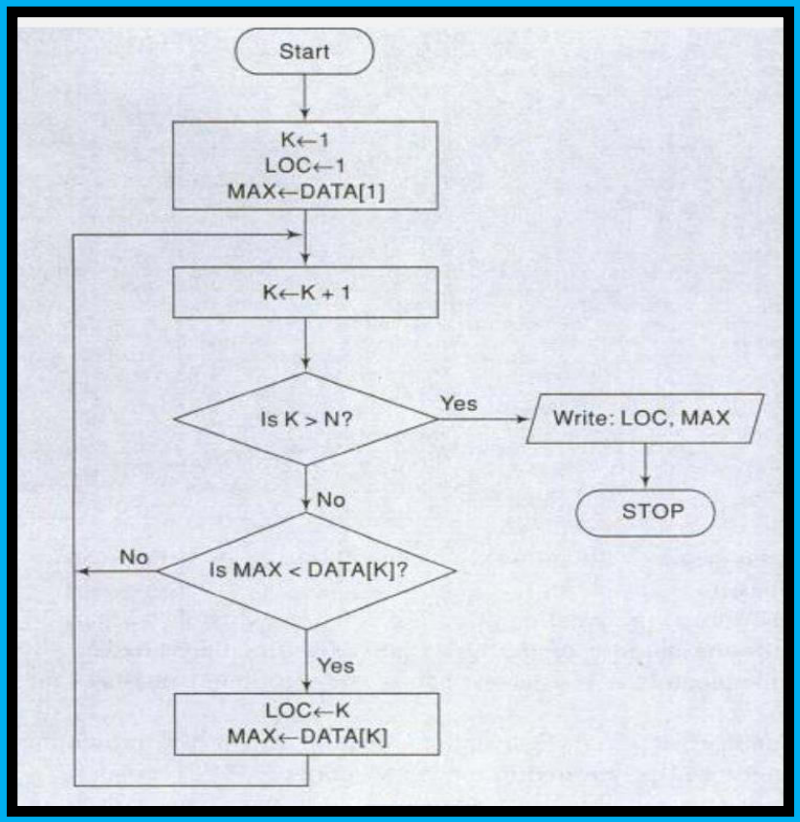
One can prove: *There are $n!$ permutations of a set of n elements.* Accordingly, there are $4! = 24$ permutations of a set with 4 elements, $5! = 120$ permutations of a set with 5 elements, and so on.

Algorithm Notation

An array DATA of numerical values is in memory. We want to find the location LOC and the value MAX of the largest element of DATA. Given no other information about DATA, one way to solve the problem is as follows:

Initially begin with LOC = 1 and MAX = DATA[1]. Then compare MAX with each successive element DATA[K] of DATA. If DATA[K] exceeds MAX, then update LOC and MAX so that LOC = K and MAX = DATA[K]. The final values appearing in LOC and MAX give the location and value of the largest element of DATA.

Step 1. [Initialize.] Set K := 1, LOC := 1 and MAX := DATA[1].
Step 2. [Increment counter.] Set K := K + 1.
Step 3. [Test counter.] If K > N, then:
Write: LOC, MAX, and Exit.
Step 4. [Compare and update.] If MAX < DATA[K], then:
Set LOC := K and MAX := DATA[K].
Step 5. [Repeat loop.] Go to Step 2.



Algorithm & Flowchart

Algorithm Notation

Comments

Each step may contain a comment in brackets which indicates the main purpose of the step. The comment will usually appear at the beginning or the end of the step.

Variable Names

Variable names will use capital letters, as in MAX and DATA. Single-letter names of variables used as counters or subscripts will also be capitalized in the algorithms (K and N, for example), even though lowercase may be used for these same variables (k and n) in the accompanying mathematical description and analysis. (Recall the discussion of italic and lowercase symbols in Sec. 1.3 of Chapter 1, under "Arrays.")

Assignment Statement

Our assignment statements will use the dots-equal notation := that is used in Pascal. For example,

Max := DATA[1]

assigns the value in DATA[1] to MAX. In C language, we use the equal sign = for this operation.

Input and Output

Data may be input and assigned to variables by means of a Read stateme.. with the following form:

Read: Variables names.

Similarly, messages, placed in quotation marks, and data in variables may be output by means of a Write or Print statement with the following form:

Write: Messages and/or variable names.

Algorithm Complexities

Types of Complexities

1. **Best Case**: the minimum value of c(n) for any possible input.
   For linear search, best case c(n)=1
2. **Worse Case**: the maximum value of c(n) for any possible input.
   For linear search, worse case c(n)=n
3. **Average Case**: the average value of c(n) for any possible input.
   For linear search, average case c(n)=(n+1)/2

Rate of Growth; Big *O* Notation

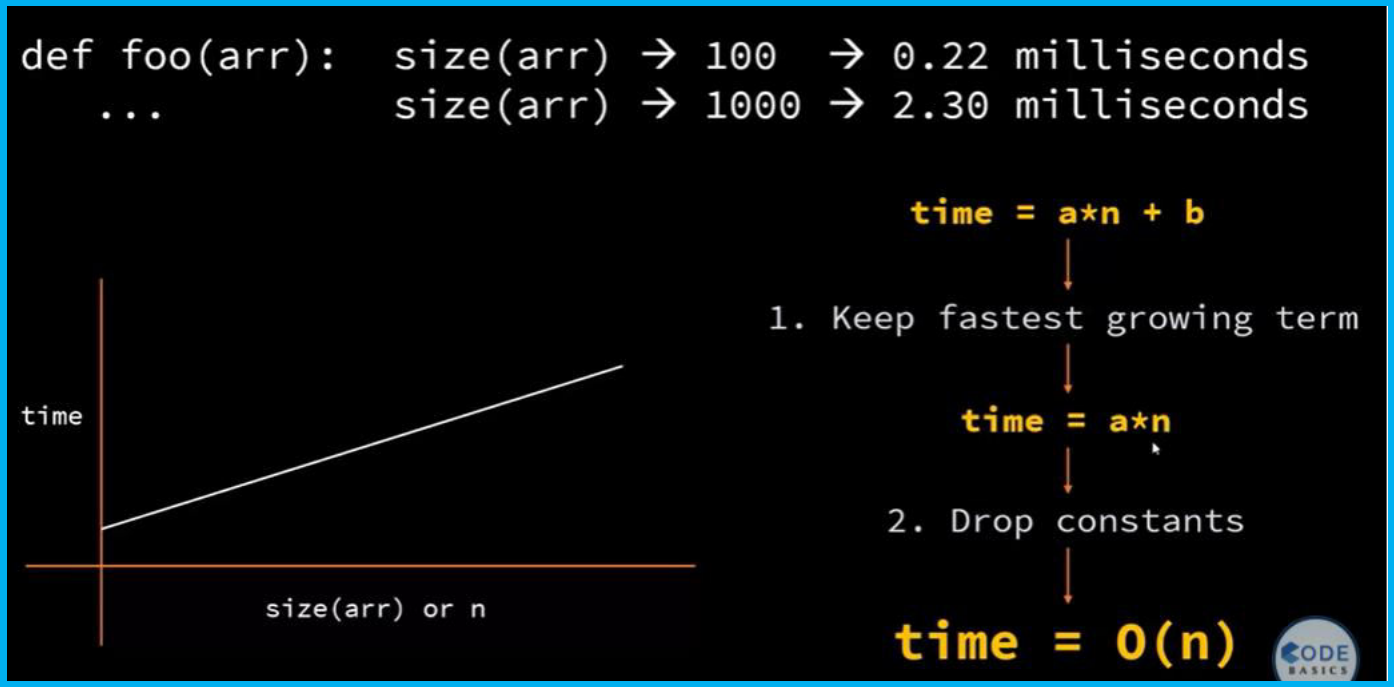
Suppose M is an algorithm, and suppose n is the size of the input data. Clearly the complexity $f(n)$ of M increases as n increases. It is usually the rate of increase of $f(n)$ that we want to examine. This is usually done by comparing $f(n)$ with some standard function, such as

$$\log_2 n, \quad n \log_2 n, \quad n^2, \quad n^3, \quad 2^n$$

n \ $g(n)$	$\log n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

Algorithm Complexities

Big O notation



Time taken for a particular program is different in different computers. So time complexity should be defined mathematically which is known as Big O().

Algorithm Complexities

Big O notation…. contd

Big O notation (O) $1 < \log(n) < \sqrt{n} < n < n \log(n) < n^2 < n^3 < \ldots < n^n$

>> Big O notation specifically describes worst case scenario.
>> It represents the upper bound running time complexity of an algorithm.

Mathematically -

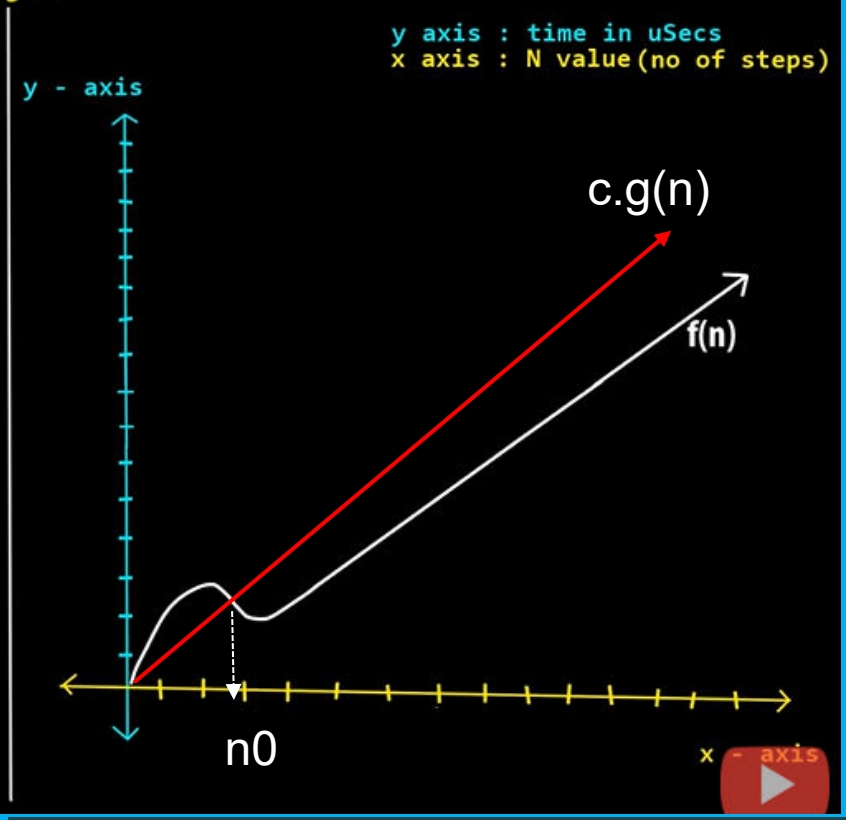
Let f and g be functions of n -> where n is natural no denoting size or steps of the algorithm then -

$f(n) = O(g(n))$

IFF

$f(n) <= c.g(n)$ where $n>=n0$, $c>0$, $n0>=1$

let
f(n)=2n+3 and g(n)=n
Now 2n+3<=c.n, it is true for
c=5 and n=1
So upper limit of f(n)=O(n)



Example
Let $f(n)=2n^2+5$
Then Time complexity=$O(n^2)$

Example
Let $f(n)=5n^5+3n^2+9$
Then Time complexity=$O(n^5)$

Algorithm Complexities

Big Ω notation

Big Omega (Ω)

$1 < \log(n) < \sqrt{n} < n < n \log(n) < n^2 < n^3 < .... < n^n$

>> Big Omega notation specifically describes best case scenario.
>> It represents the lower bound running time complexity of an algorithm.
>> Basically it tells you what is the fastest time/behavior in which the algorithm can run.

Mathematically -

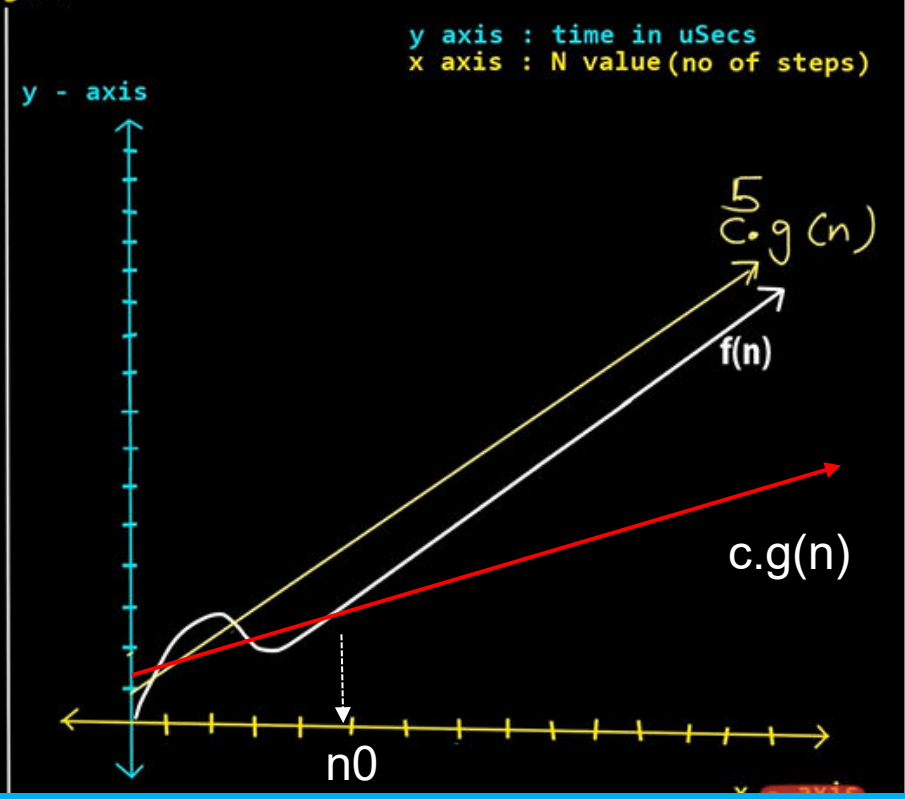
Let f and g be functions of n -> where n is natural no denoting size or steps of the algorithm then -

$f(n) = \Omega(g(n))$

IFF

$f(n) >= c.g(n)$ where $n>=n0$, $c>0$, $n0>=1$

let
f(n)=2n+3 and g(n)=n
Now 2n+3>=c.n, it is true for c=1 and n=1
So lower limit of f(n)=Ω(n)



Example
Let $f(n)=90n^2+5n+6$
Here $f(n)>80n^2$ so, $f(n)= \Omega(n^2)$

Algorithm Complexities

Big θ notation…. contd

Big Theta (θ)

$1 < \log(n) < \sqrt{n} < n < n \log(n) < n^2 < n^3 < .... < n^n$

>> Big Omega notation specifically describes average case scenario.
>> It represents the most realistic time complexity of an algorithm.

Mathematically -

Let f and g be functions of n -> where n is natural no denoting size or steps of the algorithm then -

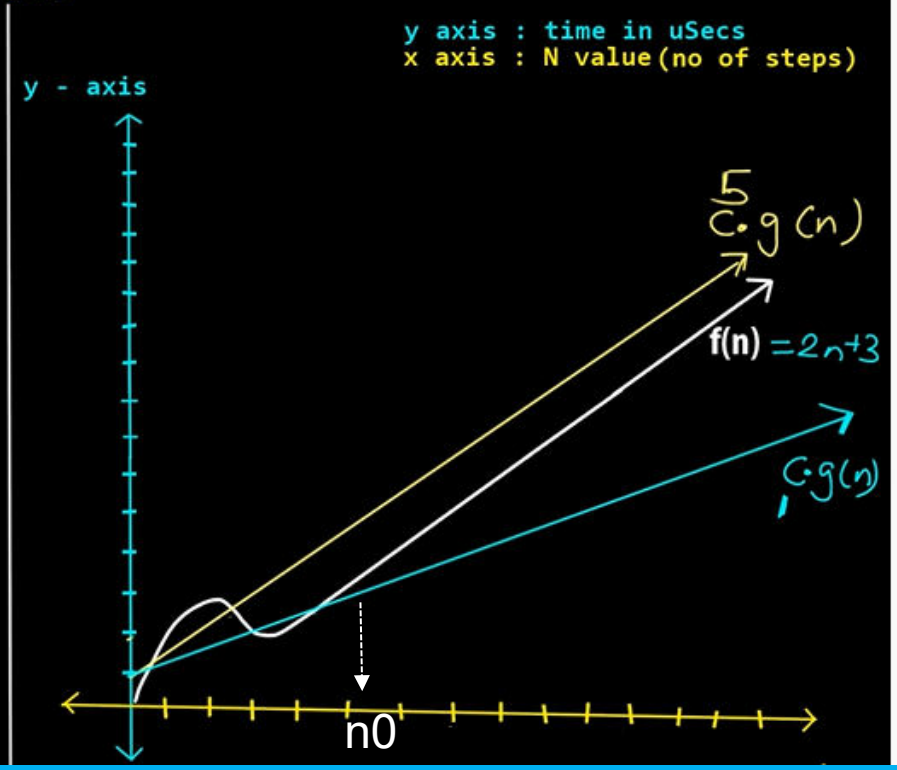
$f(n) = \theta(g(n))$

IFF

$c1.g(n) <= f(n) <= c2.g(n)$

where $n>=n0$, $c1, c2>0$, $n>=n0$, $n0>=1$

let
f(n)=2n+3 and g(n)=n
Now 1.n<=2n+3<=5.n, it is true for c1=1, c2=5 and n=1
So average case of f(n)=θ(n)



Algorithm Complexities

Example: Linear Search

$$1 < \log(n) < \sqrt{n} < n < n \log(n) < n^2 < n^3 < \ldots < n^n$$

Why 3 different analysis ?

Big Ω - Best Case

Big O - Worst Case

Big θ - Average Case

Example -

Find best case, average case & worst case of Linear Search Algorithm.

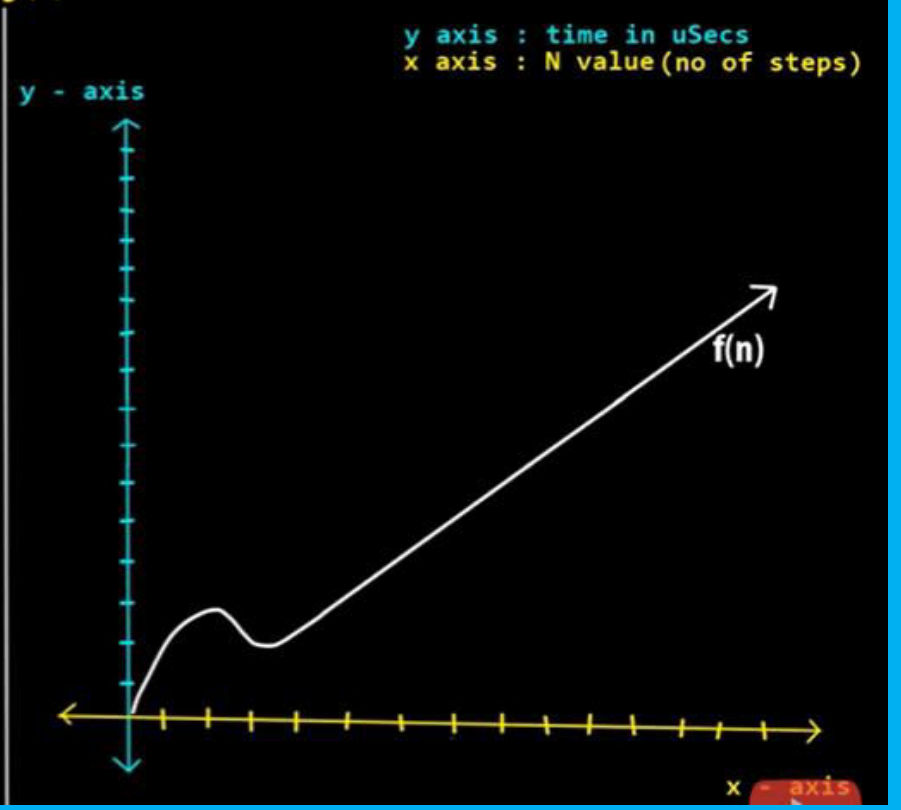
Integer array on N numbers

Here

Time Complexity = Ω (1) [best case]

= O(n) [worst case]

= θ(n+1/2) =θ(n) [average case



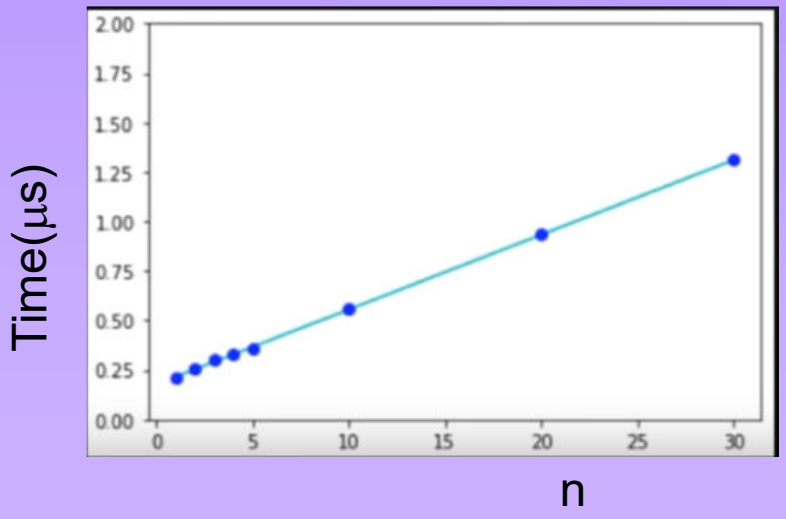
Analyzing Algorithms

We often ask for a algorithm:

- How much time does it take to finish (running time) → Time complexity
- How much space does it take (memory use) → Space complexity

```
int a[10]={1,2,3,……,10};

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



Analyzing Algorithms

Ex 0: Find the complexity of the following function?

```
int a[10][10];

int sum(int a[])
{
  int sum=0;
  printf("sum=%d",sum)
}
```

O(1)

O(1)

T=O(1)+O(1)=c1+c2
=c3=1xc3
Complexity= O(1)

Analyzing Algorithms

Ex 1: Find the complexity of the following function?

```
int a[10];

int sum(int a[],n)
{
  int sum=0;
  for(i=0;i<n;i++)
    sum=sum+a[i];
  printf("sum=%d",sum)
}
```

O(1)

O(1)

O(1)

$T=O(1)+n\times O(1)+O(1)$
$=c1+n\times c2+c3$
$=c4+n\times c2$
$=n\times c2$
Complexity= $O(n)$

Analyzing Algorithms

Ex 2. Find the complexity of the following function?

```
void A(n){
 int i=0, s=0;
 while(s<=n){
    i=i+1;
    s=s+i;
  printf("%d\n",s);
 }
}
```

s = 1 3 6 10 ….
i = 1 2 3 4 …
Suppose i=k when s>n
➔ $k(k+1)/2 > n$
➔ $k^2+k > 2n$
➔ $k=O(\sqrt{n})$
Complexity= $O(\sqrt{n})$

Analyzing Algorithms

Ex 3 Find the complexity of the following function?

```
void A(n){
 int i,j,k;
 for(i=1;i<=n;i++)
  for(j=1;j<=i;j++)
   for(k=1;k<=100;k++)
     printf("CSE\n");
 }
```

Analyzing Algorithms

Ex 4. Find the complexity of the following function?

```
void A(n){
 int i,j,k;
 for(i=1;i<=n;i++)
  for(j=1;j<=i;j++)
   for(k=1;k<=100;k++)
     printf("CSE\n");
 }
```

When i=1
Then j executes 1 time
And k executes 100 times

When i=2
Then j executes 2 times
And k executes 2*100 times

When i=3
Then j executes 3 times
And k executes 3*100 times

When i=n
Then j executes n times
And k executes n*100 times

Total=100+2*100+3*100+…+n*100
=100 (1+2+3+……..+n)
=100 [n(n+1)/2]
=$O(n^2)$

Analyzing Algorithms

Ex 5. Find the complexity of the following function?

```
void A(n){
 int i,j,k;
 for(i=1;i<=n;i++)
  for(j=1;j<=i²;j++)
   for(k=1;k<=n/2;k++)
     printf("CSE\n");
 }
```

Analyzing Algorithms

Ex 5. Find the complexity of the following function?

```
void A(n){
 int i,j,k;
 for(i=1;i<=n;i++)
  for(j=1;j<=i²;j++)
   for(k=1;k<=n/2;k++)
     printf("CSE\n");
 }
```

When i=1
Then j executes 1 time
And k executes n/2*1 times

When i=2
Then j executes 4 times
And k executes n/2*4 times

When i=3
Then j executes 9 times
And k executes n/2*9 times

When i=n
Then j executes n^2 times
And k executes n/2*n^2 times

Total = n/2 (1+4+9+ …..+ n^2)
=n/2 [n(n+1)(2n+1)/6]
=O(n^4)

Analyzing Algorithms

Ex 6. Find the complexity of the following function?

```
void A(n){
 int i;
 for(i=1;i<n;i=i*2)
   printf("CSE\n");
 }
```

Analyzing Algorithms

Ex 6. Find the complexity of the following function?

```
void A(n){
 int i;
 for(i=1;i<n;i=i*2)
   printf("CSE\n");
 }
```

i = 1 2 4 8 ….. N
= 2^0 2^1 2^3 2^4 …. 2^k

➔ $2^k = n$

➔ $k = \log_2(n)$

➔ Complexity = $O(\log_2 n)$

Analyzing Algorithms

Ex 7. Find the complexity of the following function?

```
void A(n){
 int i,j,k;
 for(i=n/2;i<=n;i++)
  for(j=1;j<=n/2;j++)
   for(k=1;k<=n;k=k*2)
     printf("CSE\n");
 }
```

Analyzing Algorithms

Ex 7. Find the complexity of the following function?

```
void A(n){
 int i,j,k;
 for(i=n/2;i<=n;i++)
  for(j=1;j<=n/2;j++)
   for(k=1;k<=n;k=k*2)
     printf("CSE\n");
 }
```

i loops executes n/2 times
j loop executes n/2 times
k loop executes $\log_2 n$ times

Total = $n/2 * n/2 * \log_2 n$
= $O(n^2 \log_2 n)$

Algorithm Complexities

Subalgorithm

A *subalgorithm* is a complete and independently defined algorithmic module which is used (or *invoked* or *called*) by some main algorithm or by some other subalgorithm. A subalgorithm receives values, called *arguments*, from an originating (calling) algorithm; performs computations; and then sends back the result to the calling algorithm. The subalgorithm is defined independently so that it may be called by many different algorithms or called at different times in the same algorithm. The relationship between an algorithm and a subalgorithm is similar to the relationship between a main program and a subprogram in a programming language.

The main difference between the format of a subalgorithm and that of an algorithm is that the subalgorithm will usually have a heading of the form

$$\text{NAME}(\text{PAR}_1, \text{PAR}_2, \ldots, \text{PAR}_K)$$

Here NAME refers to the name of the subalgorithm which is used when the subalgorithm is called, and $\text{PAR}_1, \text{PAR}_2, \ldots, \text{PAR}_K$ refer to parameters which are used to transmit data between the sub algorithm and the calling algorithm.

Quiz Time

Let's have some fun!

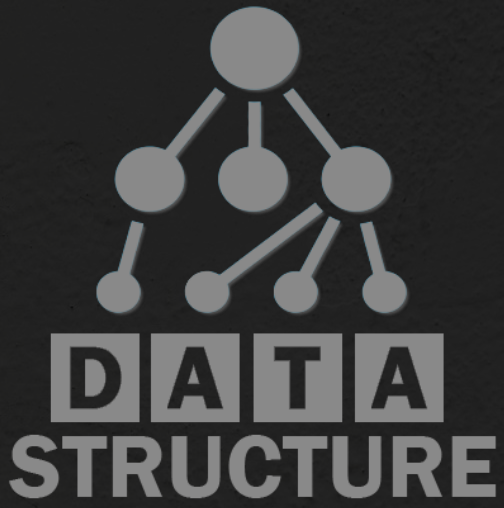
Q1: A(n) _____ has different properties

A. Field

B. Entity

C. File

D. Record



30

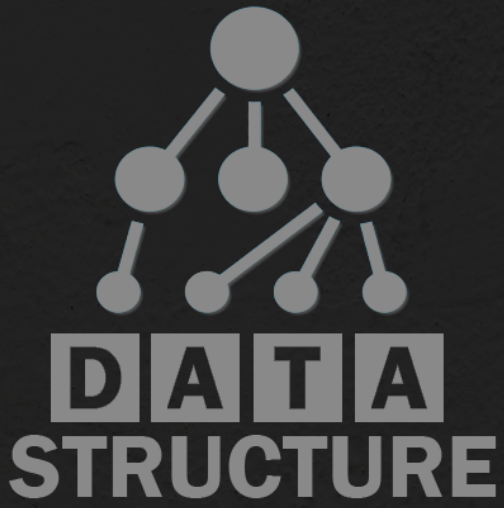
Q2: Which in NOT a primitive data structure?

A Boolean

B Arrays

C Integer

D Character



30

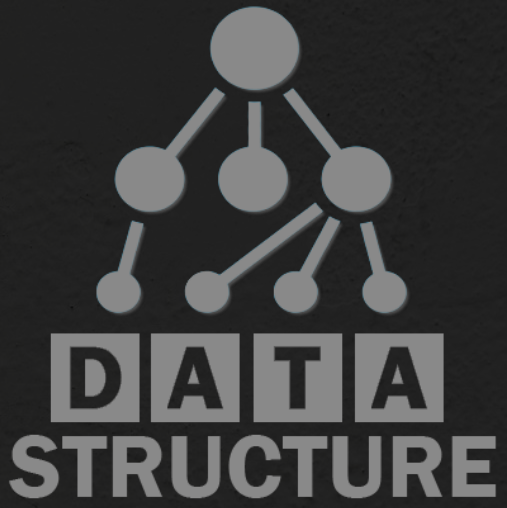
Q2: Which is a non-linear data structure?

A Arrays

B List

C Stack

D Graph



30

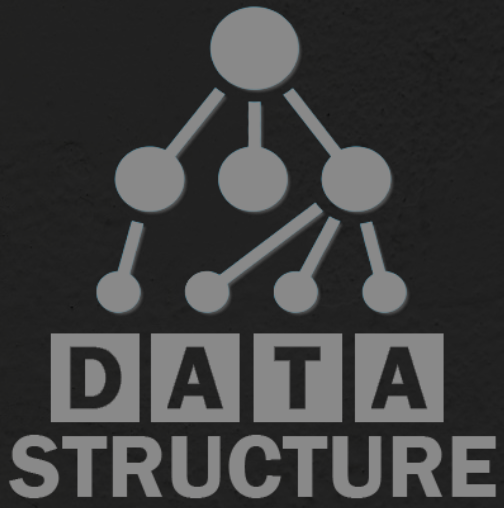
Q4: Which operation accesses each record exactly once so that contain item may be processed?

A Inserting

B Deleting

C Searching

D Traversing



30

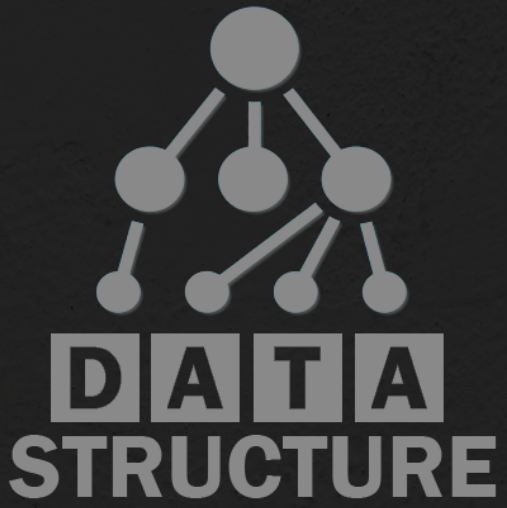
Q5: _____ involves arranging the records in a logical order

A Merging

B Sorting

C Searching

D Traversing



30

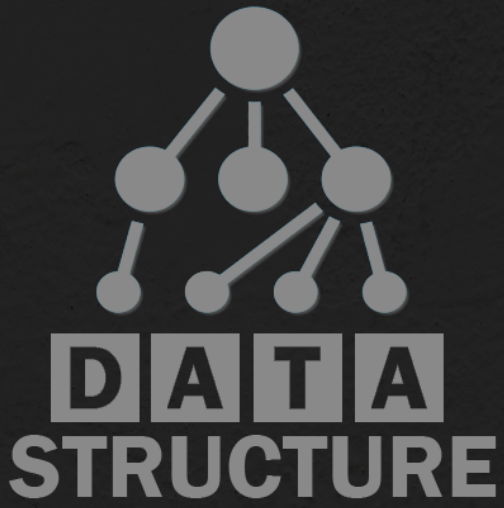
Q6: Complexity is a function of _____

A. Input data size

B. Output data size

C. Amount of time taken

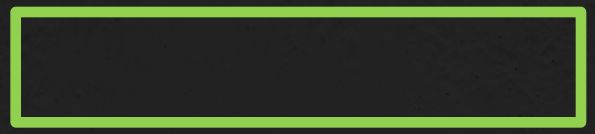
D. Total space taken



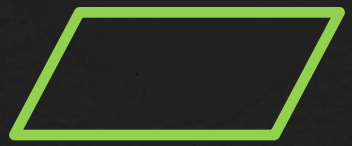
30

Q7: Which symbol is used for decision making?

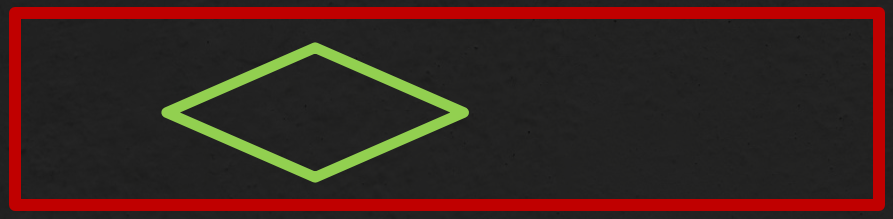
A



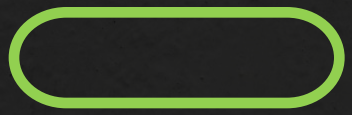
B

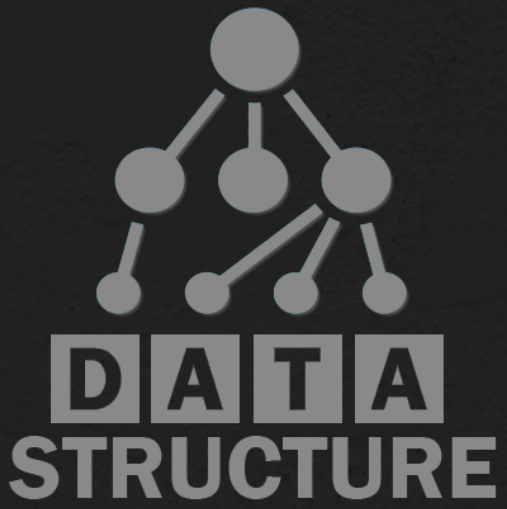


C



D





30

Q8: Identify the device?

A HDD

B SDD

C SSD

D RAM

HDD	SSD
✗ More Physical Damage	✓ Less Physical Damage
✗ Noisy, Vibrate, Hot	✓ No Noise/Vibration or Heat
✗ Consumes More Power	✓ Consumes Less Power
✗ Bootup 30-40 Sec	✓ Bootup 10-13 Sec
✗ Write = 50-120MBps	✓ Write = 200-500MBps
✗ File Opens 30% Slower	✓ 30% Faster than HDD
✓ 1TB HDD = 5-7k BDT	✗ 1TB SSD = 30-32k BDT

30