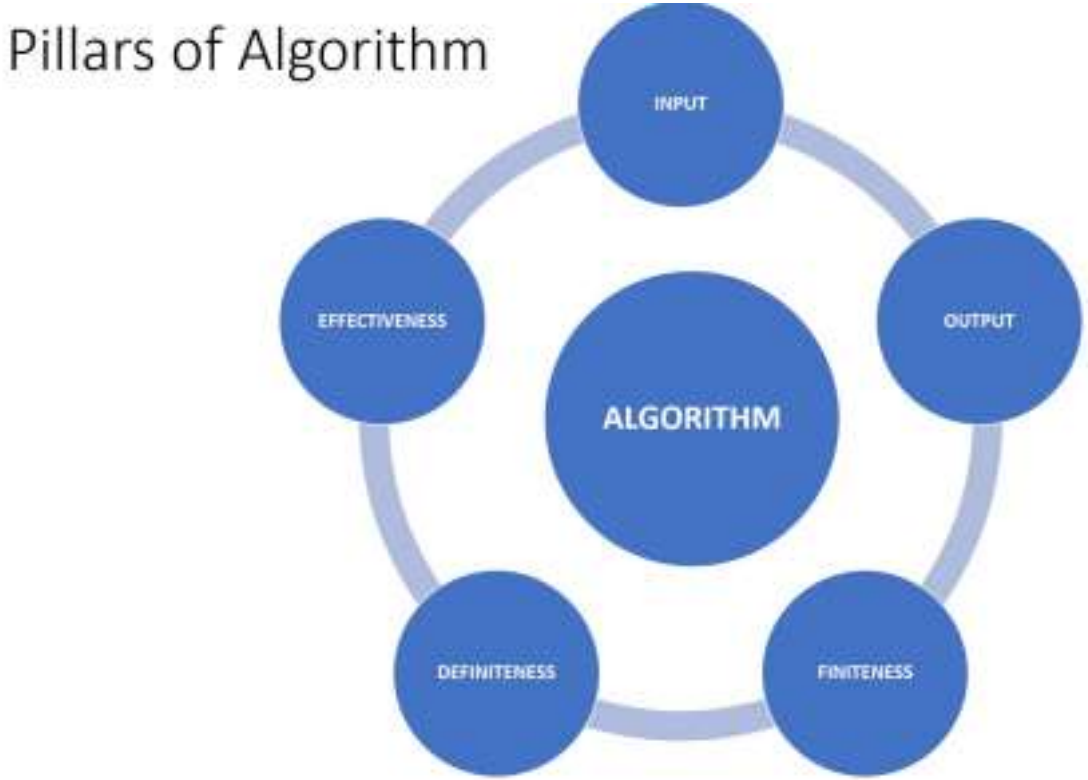
What is Algorithm ?

- Set of finite steps if followed will solve given problem in finite steps and finite time.



each step should be clearly defined

Analysis ?

- **Why ?**
  - **So one can predict time and space complexity and then select tl one.**

Analysis ?

- When ?
- Priori
  - Done before implementation
  - Using pen paper
- Posteriori
  - Done after implementation
  - Using software by professionals

Analysis ?

- **What we get ?**
  - **Time Complexity**
  - **Space Complexity**

Analysis ?

- **On What?**
  - **Best Case**
  - **Worst Case**
  - **Average Case**

Time Complexity

- **Comments –0**
- **Operation----1**
- **Loop----------n+k**

Time Complexity

- **Comments –0**
- **Operation----1**
- **Loop----------n+k**

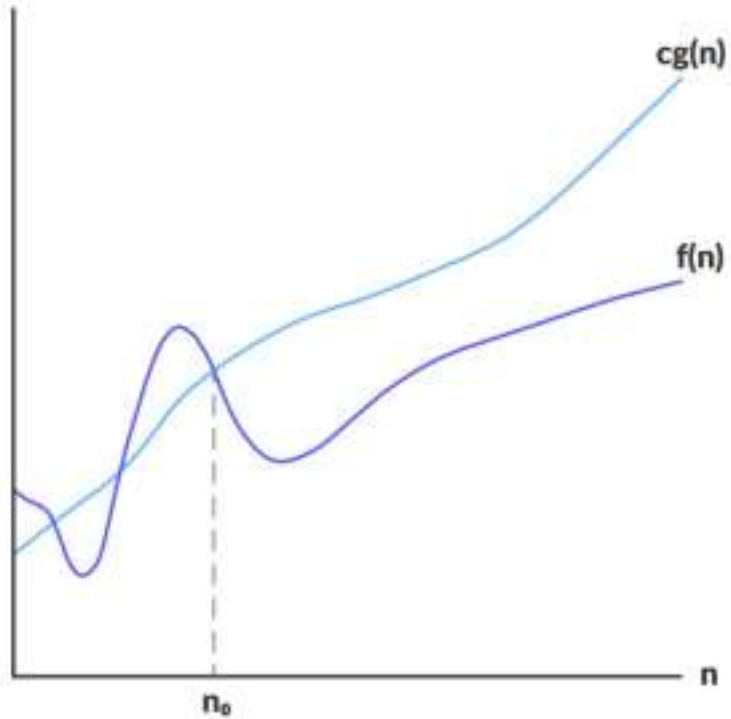
Problem

- **An algorithm takes 0.5milli seconds for 100 inputs calculate time if input is increased to 500 and if complexities are**
- **Linear**
- **Quadratic**
- **Cubic**
- **Log based**

Asymptotic Notation

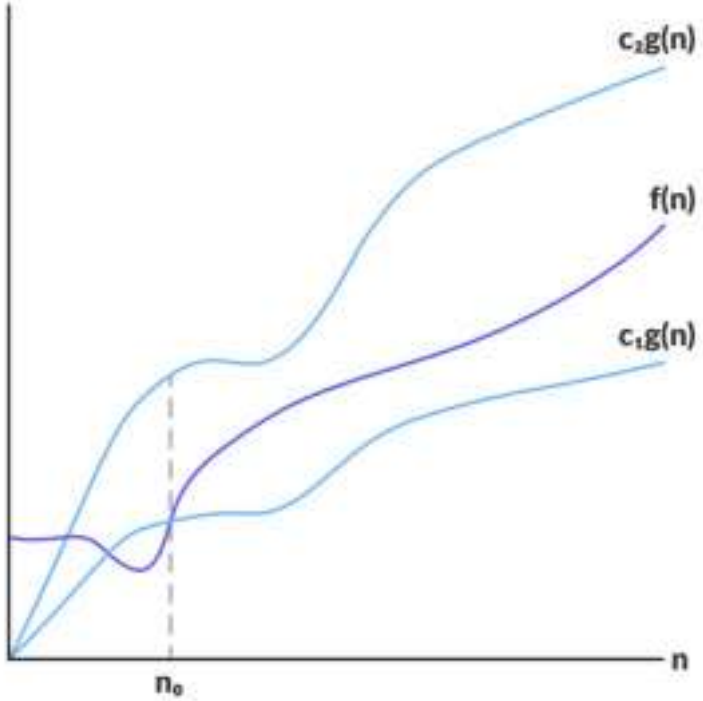
- **Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a given inp**
- **Algorithmic Common Runtimes**
- fastest to slowest are:
- constant: Θ(1)
- logarithmic: Θ(log N)
- linear: Θ(N)
- polynomial: Θ(N^2)
- exponential: Θ(2^N)
- factorial: Θ(N!)

- **Big-O Notation (O-notation)**
- Big-O notation represents the upper bound of the running time of an algorithı Thus, it gives the worst-case complexity of an algorithm.
-



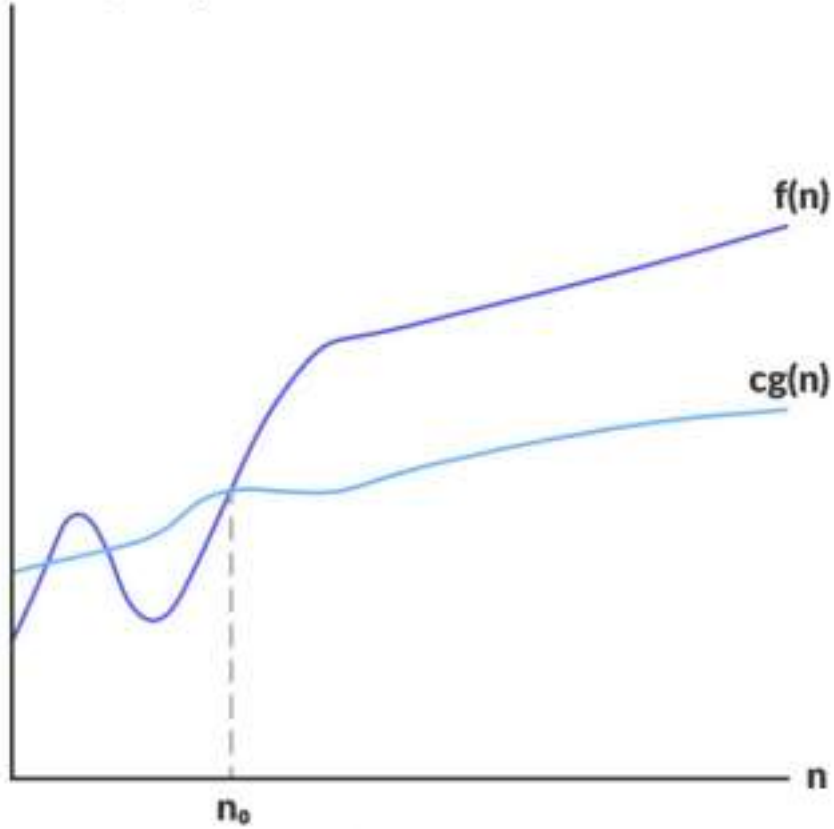
$f(n) = O(g(n))$

- **Big-O Notation (O-notation)**
  - **Rules**
    - **Constant has no value**
    - **In addition use max(worst only)**
    - **In nesting multiply and then use addition rule**

- **Theta Notation (Θ-notation)**
- Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.
- 

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

- **Omega Notation (Ω-notation)**
- Omega notation represents the lower bound of the running time of an algo
Thus, it provides the best case complexity of an algorithm.



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

DataStructures - NetBeans IDE 8.2

```
void bubblesort()
{
    int i,j,t;
    for(i=0;i<a.length-1;i++)
    {
        for(j=0;j<a.length-1;j++)
        {
            if(a[j]>a[j+1])
            {
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
            }
        }
```

```
 void selectionsort()
{
    int i,j,min,pos;
    for(i=0;i<a.length-1;i++)
    {
       min=a[i];
       pos=i;
        for(j=i+1;j<a.length;j++)
        {
            if(a[j]<min)
            {
                min=a[j];
                pos=j;
            }
        }
        a[pos]=a[i];
        a[i]=min;
    }
}
```

```
 void selectionsort()
{
   int i,j,min,pos;
   for(i=0;i<a.length-1;i++) {
      min=a[i];
      pos=i;
       for(j=i+1;j<a.length;j++) {
           if(a[j]<min)
           {
               min=a[j];
               pos=j;
           }
       }
       a[pos]=a[i];
       a[i]=min;
   }
}
```

```
void insertionsort()
 { int i,j,newelement;
    for(i=0;i<a.length-1;i++)
    {
       newelement=a[i+1];
      j=i+1;
       while (j>0 && a[j-1]>newelement)
       {
           a[j]=a[j-1];
          j--;
       }
       a[j]=newelement;
    }

 }
```

```
void insertionsort()
 { int i,j,newelement;
    for(i=0;i<a.length-1;i++)
    {
       newelement=a[i+1];
      j=i+1;
       while (j>0 && a[j-1]>newelement)
       {
           a[j]=a[j-1];
          j--;
       }
       a[j]=newelement;
    }

 }
void quicksort(int start,int end)
```

```
{
    int i,j,pivot;
    i=start; j=end;pivot=start;
    while(i<j)
    {
        while(a[i]<a[pivot])
            i++;
        while(a[j]>a[pivot])
            j--;
        if(i<j)
        {
            int t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
    }
    if(i<end)
        quicksort(i+1,end);
    if(start<j)
        quicksort(start,j-1);
}
  void quicksort(int start,int end)
  {
      int i,j,pivot;
      i=start; j=end;pivot=start;
      while(i<j)
```

```
        {
            while(a[i]<a[pivot])//i will work if pivot is at end
                i++;
            while(a[j]>a[pivot])//j will work if pivot is at 1st
                j--;
            if(i<j)
            {
                int t=a[i];
                a[i]=a[j];
                a[j]=t;
            }
        }
        if(i<end)
            quicksort(i+1,end);//for pivot at start
        if(start<j)
            quicksort(start,j-1);//for pivot at end
    }
void mergesort(int start,int end)
{
    if(start<end)
    {
        int mid=(start+end)/2;
        mergesort(start,mid);
```

```
        mergesort(mid+1,end);
        merger(start,mid,end);
    }
}
    void merger(int start,int mid,int
    end) {
        int temp[]=new
        int[a.length]; int i,j,index;
        i=index=start;
       j=mid+1;
       while(i<=mid && j<=end)
       {
           if(a[i]<a[j])
               temp[index++]=a[i++];
           else
               temp[index++]=a[j++];
       }
       while(i<=mid)
           temp[index++]=a[i++];
       while(j<=end)
           temp[index++]=a[j++];
```

```
    for (i=start;i<=end;i++)
        a[i]=temp[i];

}
```

data=index->a[data%10]