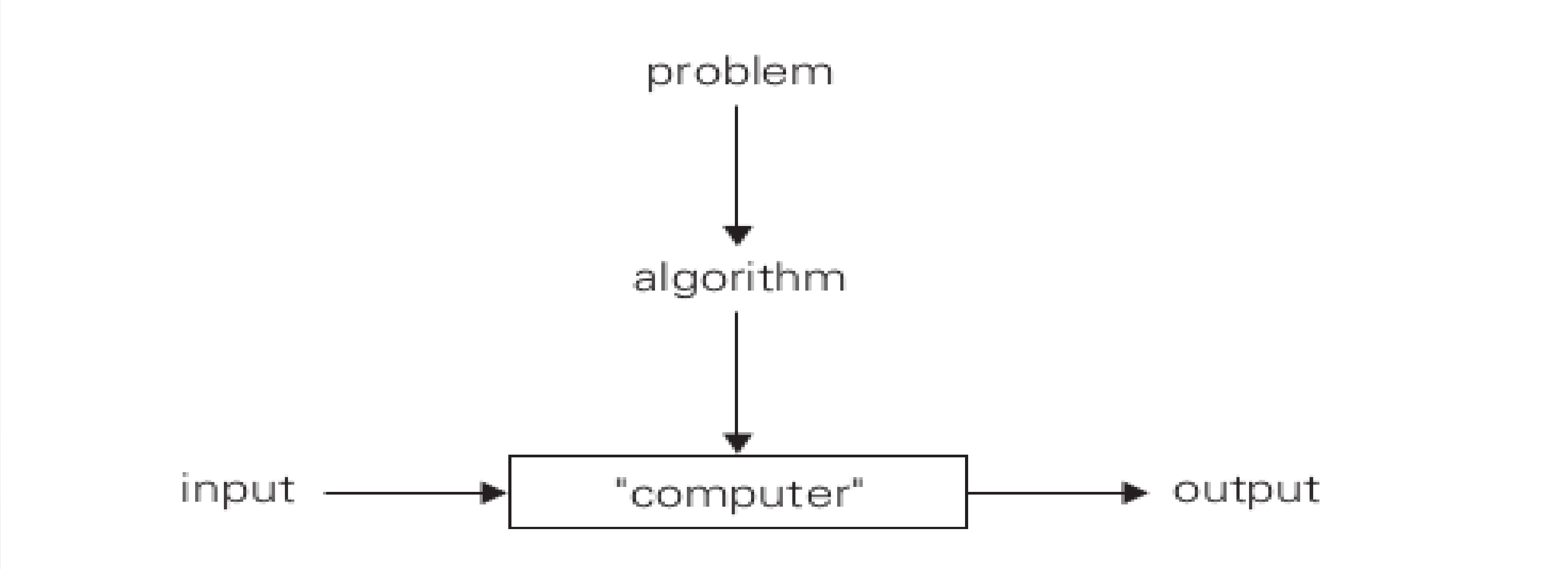
# What's Algorithm?

- An algorithm is a sequence of unambiguous instructions for solving a problem.



# Properties of Algorithm

- Finiteness
- Definiteness
- Correctness
- Generality
- Sequence

# Data Types?

- Data type of a variable is the set of values that the variable may assume.
- Some examples in c++
  - Int
  - String
  - Double
  - char

# Abstract Data Type(ADT)

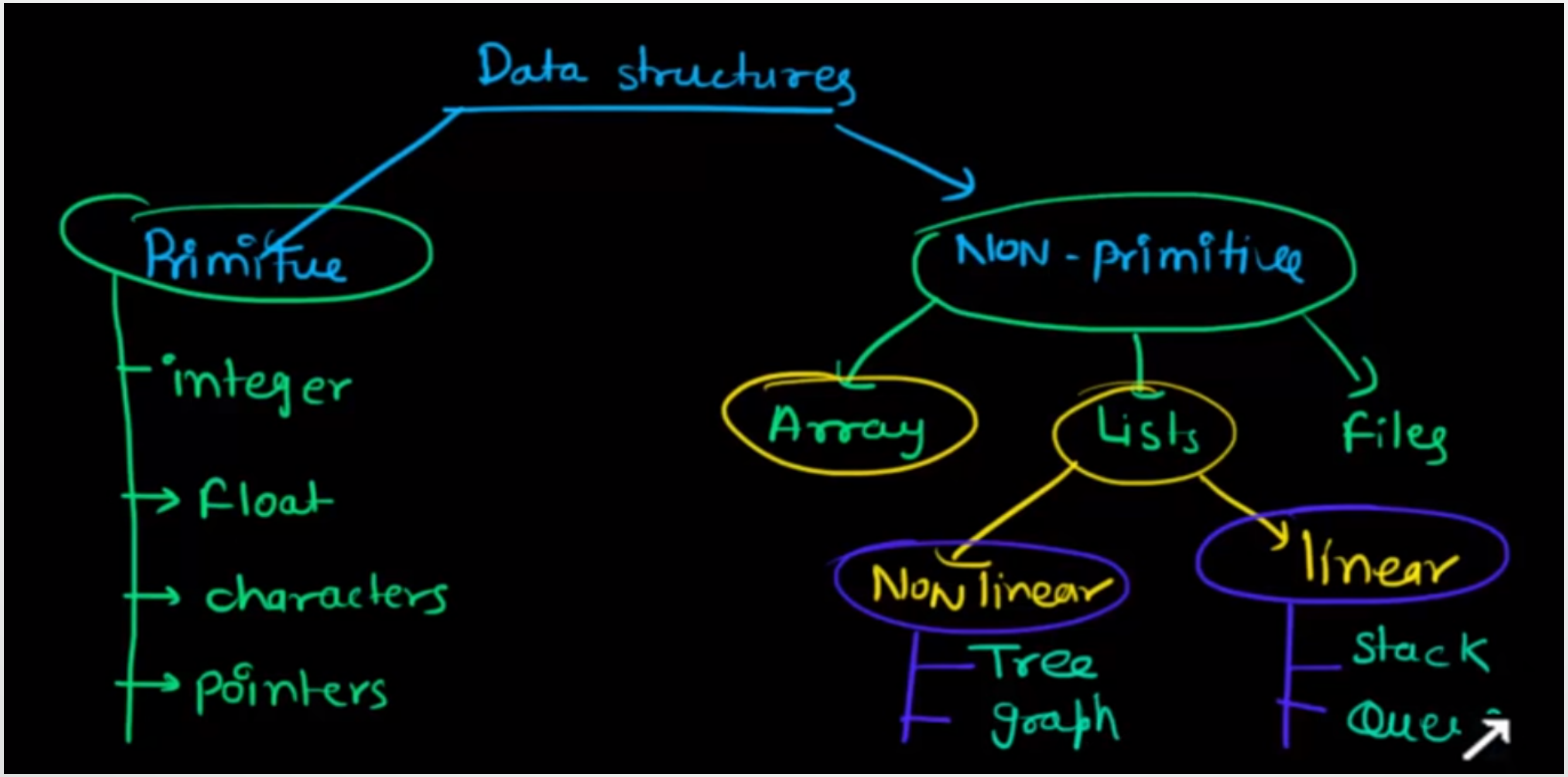
- An ADT is a set of elements with a collection of well defined operations.
- The logiacal picture of the data and the operation to manipulate it element.
- Logical level
- The ADT specifies:
  - 1.What can be stored in the Abstract Data Type
  - 2.What operations can be done on/by the ADT.

# Data Structures

- Data structure is the actual representation of the data and the algorithm to manipulate it’s element.
- Or simply the implementation level
- Some standard c++ ADT examples
  - Stack
  - Queue
  - List etc...

# Type of Data structures

1) Primitive
2) Non-premitive
    1. Linear
    2. Non-linear.



# Analysis of algorithms

- The process of determining the amount of computing time and storage space required by different algorithms.
- Why because resources are limited
    - Running time(most important)
    - Memory usage
    - Communication Bandwidth

# Complexity Analysis

- The systematic study of the cost of computation
  - Time complexity
  - Space complexity

# Space Complexity

- The amount of memory required for the algorithm to finish execution.
  - Fixed: variables and constants
  - Dynamic: dynamic data structres and recursion call

# Time Complexity

- The amount of time it takes to run an algorithm
- Estimated by counting number of elementary operation performed by the algorithm.
- Elementary operations are:
  - • Assignment Operation
  - • Single Input/Output Operation
  - • Single Boolean Operations
  - • Single Arithmetic Operations
  - • Function Return

# Example 1

| Code | Cost<br>Time require for line ( Units ) | Repeatation<br>No. of Times Executed | Total<br>Total Time required in worst case |
|---|---|---|---|
| `int sumOfList( int A[ ], int n)` | | | |
| `{` | | | |
| `int sum = 0, i;` | 1 | 1 | 1 |
| `for(i = 0; i < n; i++)` | 1 + 1 + 1 | 1 + (n+1) + n | 2n + 2 |
| `sum = sum + A[i];` | 2 | n | 2n |
| `return sum;` | 1 | 1 | 1 |
| `}` | | | |
| | | | **4n + 4**<br>Total Time required |

# Example 2

```
void func(){
 int x=0, i=0, j=1;
 cout<< “Enter an Integer value”;
 cin>>n;
 while (i<n){
  x++;
  i++;
 }
 while (j<n){
  j++;
 }
}
```

$T(n) = 1 + 1 + 1 + 1 + 1 + (n+1) + n + (n-1) = 5n + 5$

# Formal Approach

- Analysis can be simplified by using some formal approach in which case we can ignore initializations, loop control, and book keeping.
- For loops

```
for (int i = 1; i <= N; i++) {
     sum = sum+i;
}
```

$$\sum_{i=1}^{N} 1 = N$$

# Formal Approach

- For nested loops

```
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= M; j++) {
        sum = sum+i+j;
    }
}
```

$$\sum_{i=1}^{N}\sum_{j=1}^{M} 2 = \sum_{i=1}^{N} 2M = 2MN$$

# Formal Approach

- For consecutive statements

```
for (int i = 1; i <= N; i++) {
     sum = sum+i;
}
for (int i = 1; i <= N; i++) {
     for (int j = 1; j <= N; j++) {
          sum = sum+i+j;
     }
}
```

$$\left[\sum_{i=1}^{N} 1\right] + \left[\sum_{i=1}^{N}\sum_{j=1}^{N} 2\right] = N + 2N^2$$

# Example

```
int k=0;
for (int i=1; i<n; i*=2)
    for(int j=1; j<n; j++)
        k++;
```

- T(n) = $f(x) = \sum_{i=1}^{\log(n)} \sum_{j=1}^{n} 1$

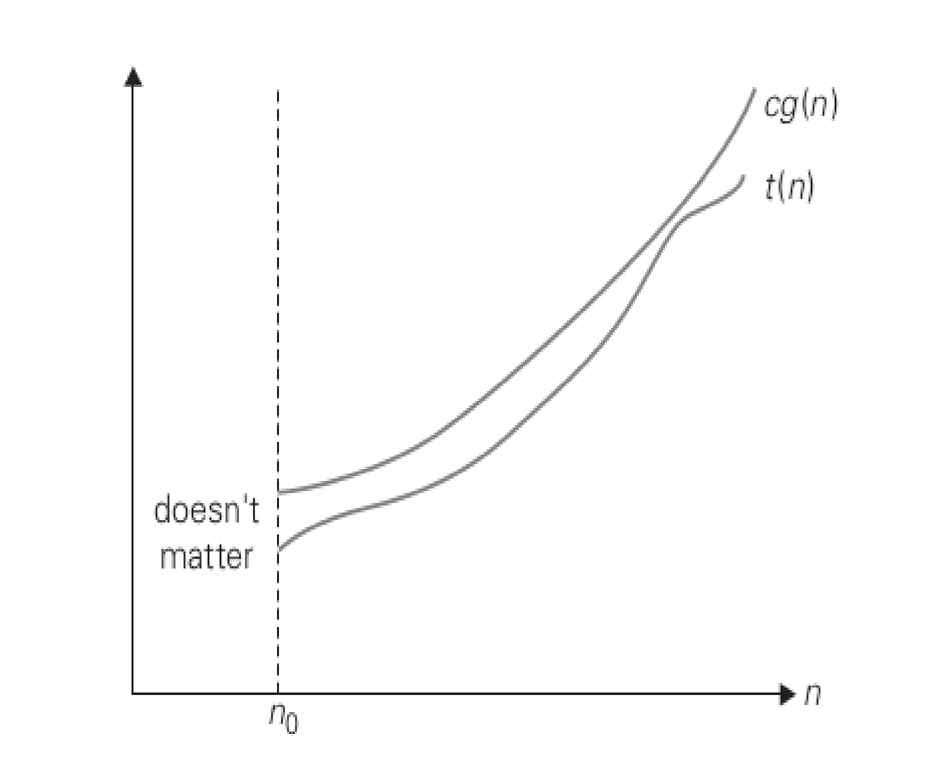
# Asymptotic Analysis

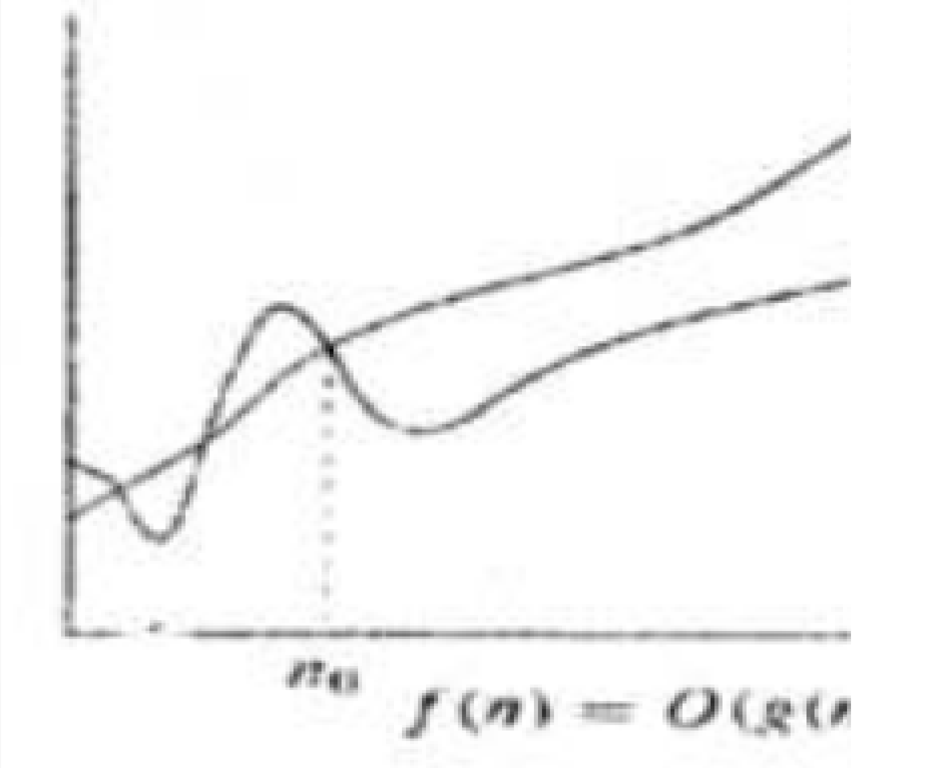
- Mathematical represenation of algorithm’s complexity.
- Concerned with how the running time of an algorithm increases with the size of the input.
  - • Big-Oh Notation (O)
  - • Big-Theta Notation (Θ)
  - • Big-Omega Notation (Ω)

# Big-Oh Notation (O)

- Simply upper bound
- A function t (n) is said to be in O(g(n)), denoted t (n) ∈ O(g(n)), if t (n) is bounded above by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some nonnegative integer $n_0$ such that
  - $t(n) \le c*g(n)$ for all $n \ge n_0$

# Examples



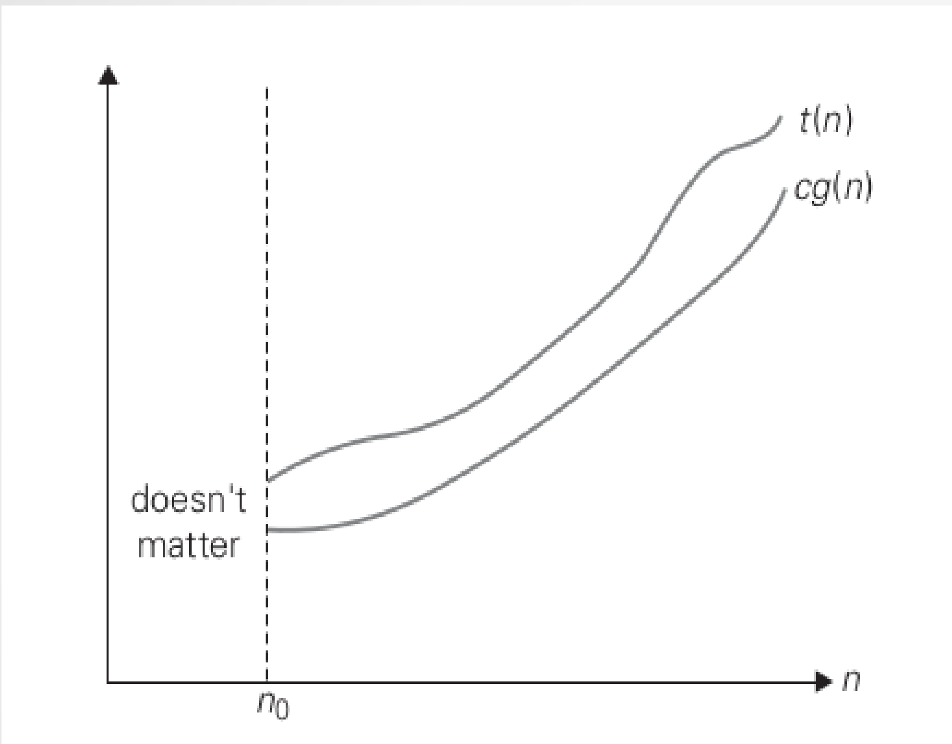



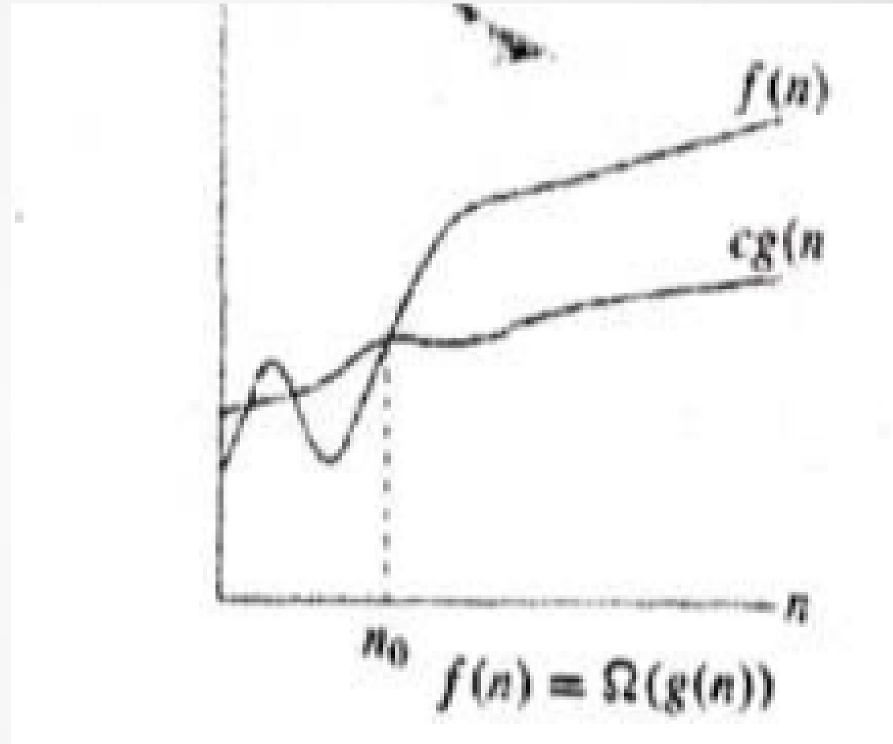

- $100n + 5 \in O(n^2)$
- Because $100n + 5 \leq 100n + n$ (for all $n \geq 5$) $= 101n \leq 101n^2$

# Big-Omega Notation (Ω)

- Simply lower bound
- A function t (n) is said to be in (g(n)), denoted t (n) ∈ (g(n)), if t (n) is bounded below by some positive constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some nonnegative integer $n_0$ such that
  - $t(n) \geq cg(n)$ for all $n \geq n_0$.

# Example




- $n^3 \in (n^2)$
- $n^3 \geq n^2$ i.e. we can select c = 1 and n 0 = 0. for all n ≥ 0

# Big-Theta Notation (Θ)

- A function t (n) is said to be in (g(n)), denoted t (n) ∈ (g(n)), if t (n) is bounded below by some positive constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some nonnegative integer n 0 such that
  - t (n) ≥ cg(n) for all n ≥ n 0 .

# Example