Purpose of Program Planning

- To write a correct program, a programmer must write each and every instruction in the correct sequence
- Logic (instruction sequence) of a program can be very complex
- Hence, programs must be planned before they are written to ensure program instructions are:
  - Appropriate for the problem
  - In the correct sequence

Algorithm

- Refers to the logic of a program and a step-by-step description of how to arrive at the solution of a given problem
- In order to qualify as an algorithm, a sequence of instructions must have following characteristics:
  - Each and every instruction should be precise and unambiguous
  - Each instruction should be such that it can be performed in a finite time
  - One or more instructions should not be repeated infinitely. This ensures that the algorithm will ultimately terminate
  - After performing the instructions, that is after the algorithm terminates, the desired results must be obtained

Sample Algorithm (Example 1)

There are 50 students in a class who appeared in their final examination. Their mark sheets have been given to you.

The division column of the mark sheet contains the division (FIRST, SECOND, THIRD or FAIL) obtained by the student.

Write an algorithm to calculate and print the total number of students who passed in FIRST division.

Sample Algorithm (Example 1)

(contd...)

Step 1: Initialize Total_First_Division and Total_Marksheets_Checked to zero.

Step 2: Take the mark sheet of the next student.

Step 3: Check the division column of the mark sheet to see if it is FIRST, if no, go to Step 5.

Step 4: Add 1 to Total_First_Division.

Step 5: Add 1 to Total_Marksheets_Checked.

Step 6: Is Total_Marksheets_Checked = 50, if no, go to Step 2.

Step 7: Print Total_First_Division.

Step 8: Stop.

Sample Algorithm (Example 2)

There are 100 employees in an organization. The organization wants to distribute annual bonus to the employees based on their performance. The performance of the employees is recorded in their annual appraisal forms.

Every employee's appraisal form contains his/her basic salary and the grade for his/her performance during the year. The grade is of three categories – 'A' for outstanding performance, 'B' for good performance, and 'C' for average performance.

It has been decided that the bonus of an employee will be 100% of the basic salary for outstanding performance, 70% of the basic salary for good performance, 40% of the basic salary for average performance, and zero for all other cases.

Write an algorithm to calculate and print the total bonus amount to be distributed by the organization.

Sample Algorithm (Example 2)

(contd..)

Step 1: Initialize Total_Bonus and Total_Employees_Checked to zero.

Step 2: Initialize Bonus and Basic_Salary to zero.

Step 3: Take the appraisal form of the next employee.

Step 4: Read the employee's Basic_Salary and Grade.

Step 5: If Grade = A, then Bonus = Basic_Salary. Go to Step 8.

Step 6: If Grade = B, then Bonus = Basic_Salary x 0.7. Go to Step 8.

Step 7: If Grade = C, then Bonus = Basic_Salary x 0.4.

Step 8: Add Bonus to Total_Bonus.

Step 9: Add 1 to Total_Employees_Checked.

Step 10: If Total_Employees_Checked < 100, then go to Step 2.

Step 11: Print Total_Bonus.

Step 12: Stop.

Representation of Algorithms

- As programs
- As flowcharts
- As pseudocodes

When an algorithm is represented in the form of a programming language, it becomes a program

Thus, any program is an algorithm, although the reverse is not true

Flowchart

- *Flowchart* is a pictorial representation of an algorithm
- Uses symbols (boxes of different shapes) that have standardized meanings to denote different types of instructions
- Actual instructions are written within the boxes
- Boxes are connected by solid lines having arrow marks to indicate the exact sequence in which the instructions are to be executed
- Process of drawing a flowchart for an algorithm is called *flowcharting*

Pseudocode

- A program planning tool where program logic is written in an ordinary natural language using a structure that resembles computer instructions
- " Pseudo" means imitation or false and " Code" refers to the instructions written in a programming language. Hence, pseudocode is an imitation of actual computer instructions
- Because it emphasizes the design of the program, pseudocode is also called ***Program Design Language (PDL)***

Complexity

Complexity is the function that gives the running time and/or space for some input.

Time Space Trade-off

Time space tradeoff is the choice to reduce the space by increasing the time or vice versa.

cases

Best case: The minimum value of f(n) for any input
Worst case: The maximum value of f(n) for any input
Average case: The expected value of f(n)

টাইম কমপ্লেক্সিটি (TIME COMPLEXITY)

- অ্যাসাইনমেন্ট অপারেশন
- কম্পারিজন অপারেশন
- গাণিতিক অপারেশন
- ফাংশন কল করা
- ফাংশনের ভেতরের কাজ

টাইম কমপ্লেক্সিটি (TIME COMPLEXITY)

```c
#include <stdio.h>

int main()
{
    int n1, n2, result;

    n1 = 10;
    n2 = 20;
    result = n1 + n2;

    return 0;
}
```

3
1

= 4

O(1)

O(4)?

টাইম কমপ্লেক্সিটি (TIME COMPLEXITY)

```c
#include <stdio.h>

int main()
{
    int n, result;

    scanf("%d", &n);

    result = n * (n + 1) / 2;

    printf("result = %d\n", result);

    return 0;
}
```

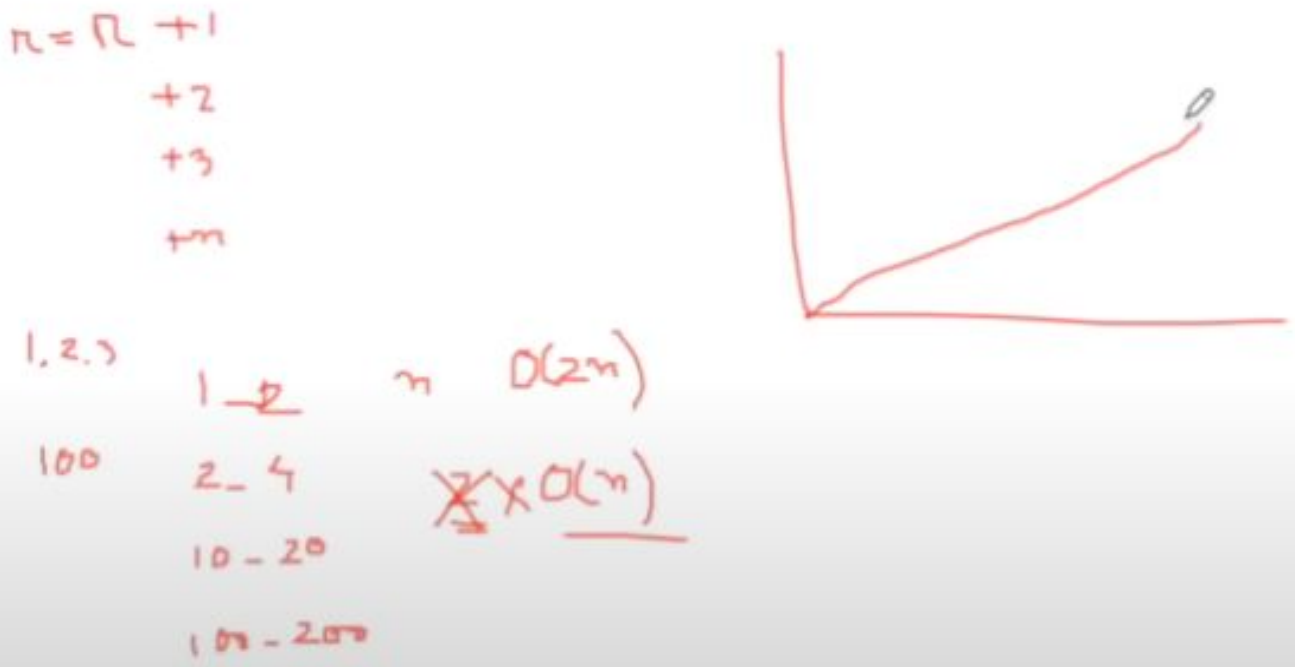
1+2+3+ +n

3

O(1)

টাইম কমপ্লেক্সিটি (TIME COMPLEXITY)

```c
#include <stdio.h>

int main()
{
    int i, n, result;

    scanf("%d", &n);

    result = 0;

    for (i = 1; i <= n; i++) {
        result = result + i;
    }

    printf("result = %d\n", result);

    return 0;
}
```



টাইম কমপ্লেক্সিটি (TIME COMPLEXITY)

```c
#include <stdio.h>

int main()
{
    int i, j, n, count;

    scanf("%d", &n);

    count = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            count = count + 1;
        }
    }

    printf("n = %d, count = %d\n", n, count);

    return 0;
}
```

$c = n^2$

20x20

400

$O(n^2)$

n	count
1	1
2	4
3	9
10	100
100	10000

টাইম কমপ্লেক্সিটি (TIME COMPLEXITY)

```c
#include <stdio.h>

int main()
{
    int i, j, k, n, count;

    scanf("%d", &n);

    count = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            for (k = 0; k < n; k++) {
                count = count + 1;
            }
        }
    }

    printf("n = %d, count = %d\n", n, count);

    return 0;
}
```

$O(n^3)$