ADITYA PRATAPSINGH :-

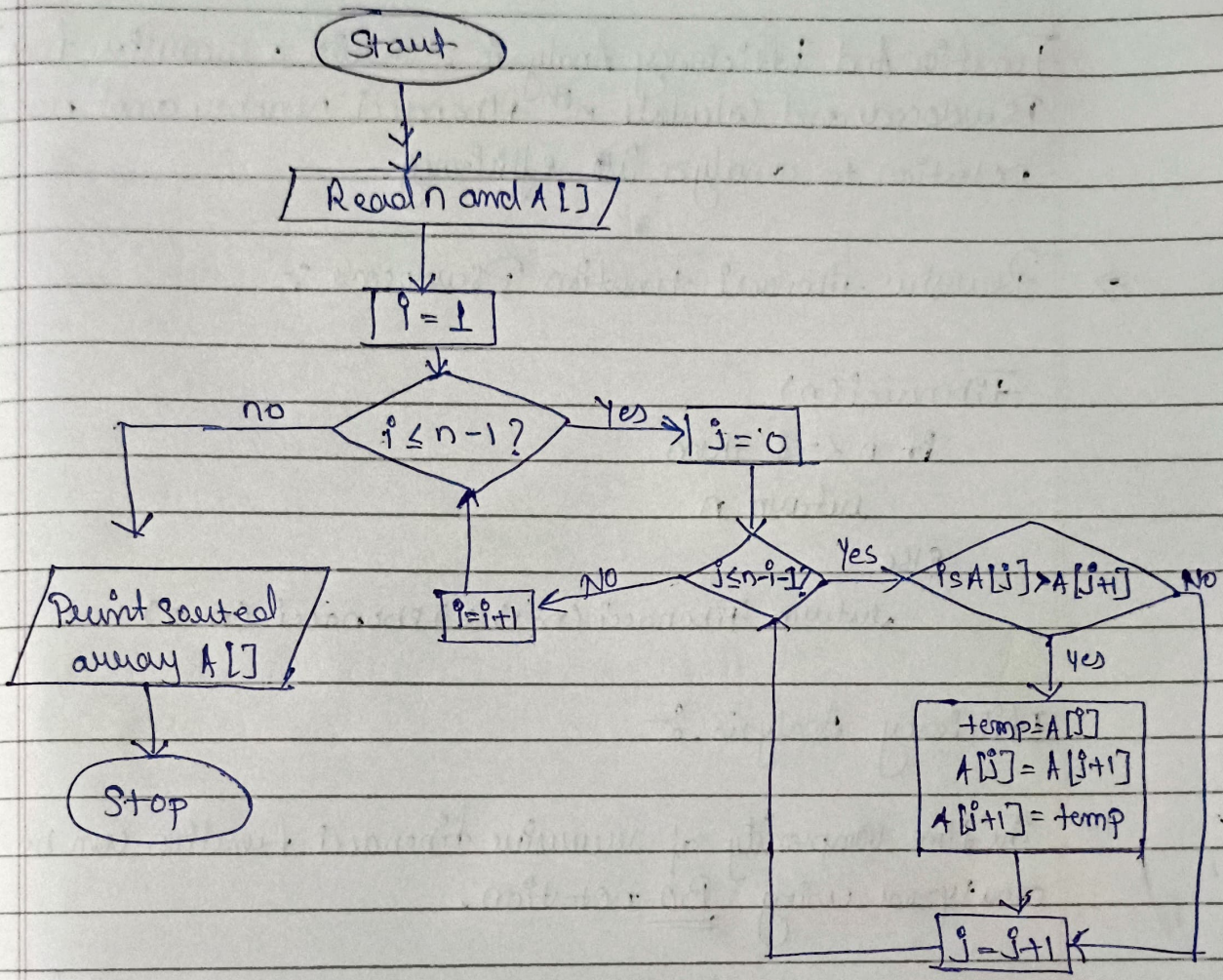
ASSIGNMENT-01

Pseudocode & Flowchart for Sorting Algorithm - Write Pseudo code & Create a flowchart for a bubble sort algo.

⇒ Pseudo code :-

```
Algorithm Bubblesort () {
  Read n, the array size.
  Read the elements of array A[]
  for i = 1 to n-1 do {
      for j = 0 to n-i-1 do {
          If A[j] > A[j+1] then {
              temp = A[j]
              A[j] = A[j+1]
              A[j+1] = temp.
          }
      }
  }
  Print the Sorted array A[]
}
```

FLOWCHART:-


Start
Read n and A[]
i = 1
i ≤ n-1 ?
no
Yes
j = 0
j ≤ n-i-1 ?
NO
Yes
A[j] > A[j+1]
NO
Yes
temp = A[j]
A[j] = A[j+1]
A[j+1] = temp
j = j+1
i = i+1
Print Sorted array A[]
Stop

ASSIGNMENT - 02.

Function And Efficiency Analysis :- Write a recursive fun" Pseudocode and calculate n^{th} fibonacci number and use Big O notation to analyze its efficiency.

⇒ Recursive fibonacci function Pseudocode :-

```
Fibonacci(n)
    if n <= 1 then
        return n
    Else
        return Fibonacci(n-1) + Fibonacci(n-2)
```

Efficiency Analysis :-

The time complexity of recursive fibonacci function can be analyzed using Big O notation.

Let's denote the time complexity of calculating the n^{th} fibonacci number using recursive approach as T(n).

In worst case scenario, each call to the fibonacci function leads to two additional recursive call. until it reaches the base case ($n <= 1$),

Therefore, the time complexity can be expressed by the recurrence relation :-

$$T(n) = T(n-1) + T(n-2) + O(1)$$

This recurrence relation is similar to fibonacci Sequence itself. it grows exponentially, resulting in a time complexity of approximately $O(2^n)$.