Unit

DATE: / 20
PAGE No

27/02/19

Analysis of Algorithm

- Efficiency of Algorithm checked by:
- Correctness
- Implementation
- Simplicity
- Execution time and memory requirements

– Three types of analysis:
- Worst Case
- Best case
- Average case

x Worst case:
- Provide an upper bound on running time.
- An absolute guarantee that the algorithm would not run longer, no matter what the inputs are

x Best case:
- Provides a lower bound on running time.
- Input is the one for which the algorithm runs the fastest

x average case:
- Provides a prediction about the running time

DATE : / 20
PAGE No

- Assumes that the input is random

Lower Bound ≤ Running time ≤ upper Bound

* Complexity of Algorithm:
  - Running time of an algorithm as a function of input size n is called Complexity.
  - Expressed using only the highest order term instead of exact running time.

* Types:
  - Space Complexity: Amount of memory required to run an algorithm
  - Time Complexity: Amount of time required to run an algorithm

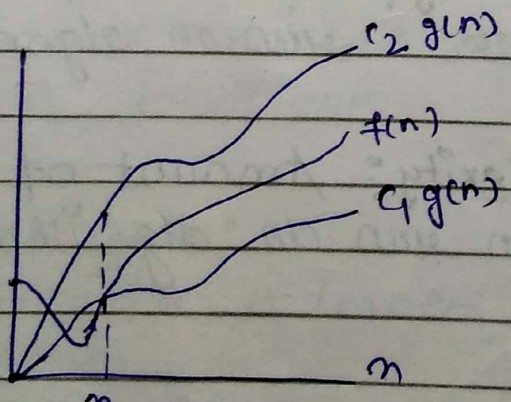
DATE: / 20
PAGE No

* Asymptotic Notation :-

- O (Big Oh) notation :
  asymptotic "less than"
  $(f(n) = O(g(n)))$ implies : $f(n)$ "$\leq$" $g(n)$

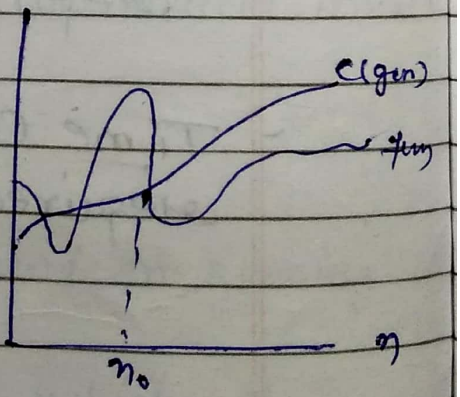
- Ω (Big omega) notation :
  asymptotic "greater than"
  $f(n) = \Omega(g(n))$ implies : $f(n)$ "$\geq$" $g(n)$

- Θ (Theta) notation : asymptotic "equality"
  $f(n) = \Theta(g(n))$
  implies : $f(n)$ "$=$" $g(n)$



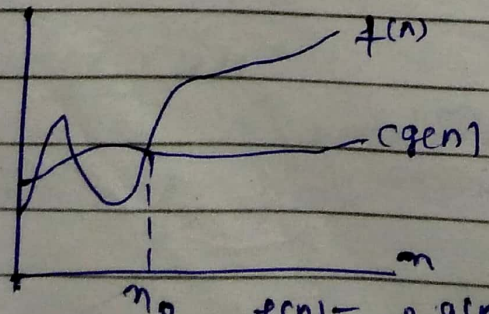
$f(n) = \Theta g(n)$

(a)



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

(b)



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

(c)

DATE : / 20
PAGE No

* Searching Algorithm:
(i) Linear (ii) Binary.

- Linear:
- step-1 : Input array 'A' of 'n' element, 'data' to be searched.
step-2 : Initialize i=0
step 3 : Repeat step 4 & 5 while i<n
step 4 : if (data == A[i])
go to step 6.
step 5 : i = i+1
step 6 : if (i<n)
element found at i+1
else
element not found
step 7 : exit.

- Binary:
- element must be sorted (~~Arranged~~).

DATE : / 20
PAGE No

* Complexity of Algorithms:

Algorithm	Best Case	Avg. case	worst case
Linear Search	O(1)	O(n)	O(n)
Binary Seach	O(1)	O(logn)	O(logn)
Bubble Sort	O(n)	O(n^2)	O(n^2)
insertion Sort	O(n)	O(n^2)	O(n^2)
selection Sort	O(n^2)	O(n^2)	O(n^2)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)

* Binary Search: (Apply when Algorithm is sorted)

Step.1 Input array 'A' of 'n' elements, 'Data' to be Searched

Step.2 Set LB = 0, UB = n-1, MID = int((LB+UB)/2)

4

1st Sessional

DATE : / 20
PAGE No

step-3 Repeat step 4 & 5 while LB <= UB and A[MID] != Data

step-4 if (Data < A[MID])

UB = MID - 1

else

LB = MID + 1

step 5. MID = int((LB + UB)/2)

step 6: If (A[MID] == Data)

Element found at MID+1

else

Element not found

Step 7: Exit

* Bubble sort :

① (40) (10) 70 30 50

10 (40) (70) 30 50

10 40 (70) (30) 50

10 40 30 (70) (50)

10 40 30 50 70

DATE : / 20
PAGE No

(II) 10 40 30 50 70

10 40 30 50 70

10 30 40 50 70

10 30 40 50 70

(III) 10 30 40 50 70

10 30 40 50 70

10 30 40 50 70

(iv) 10 30 40 50 70

10 30 40 50 70

Step-1 Input array 'A' of 'n' elements

Step-2 initialize $i=0$ and repeat through step 6 while $i<n-1$

Step-3 Initialize $j=0$ and repeat step 4 & 5 while $j<n-i-1$

DATE : / 20
PAGE No

Step-4 if (A[j] > A[j+1])
(a) temp = A[j]
(b) A[j] = A[j+1]
(c) A[j+1] = temp.

Step.5 j = j+1
step.6 i = i+1
step.7 Print the sorted array 'A'
stp-8 exit.

* Insertion Sort:

Step.1 Input array 'A' of 'n' elements
step.2 Initilize i=1 and repeat through step 8 while i<n
step.3 Initialize key = A[i] and j = i-1
step.4 Repeat step 5 & 6 while j>=0 and A[j] > key
Step.5 A[j+1] = A[j]
step.6 j = j-1
step7 A[j+1] = key
Step8 i = i+1
step 9 Print the sorted array 'A'
step 10 Exit

DATE : / 20
PAGE No

Eg:

70 30 60 10 40 20 50

30 70 60 10 40 20 50

30 60 70 10 40 20 50

10 30 60 70 40 20 50

10 30 40 60 70 20 50

10 20 30 40 60 70 50

10 20 30 40 50 60 70

* Selection sort:

20 60 10 30 50 70 40

10 60 20 30 50 70 40

10 20 60 30 50 70 40

10 20 30 60 50 70 40

10 20 30 40 50 70 60

10 20 30 40 50 70 60

10 20 30 40 50 60 70