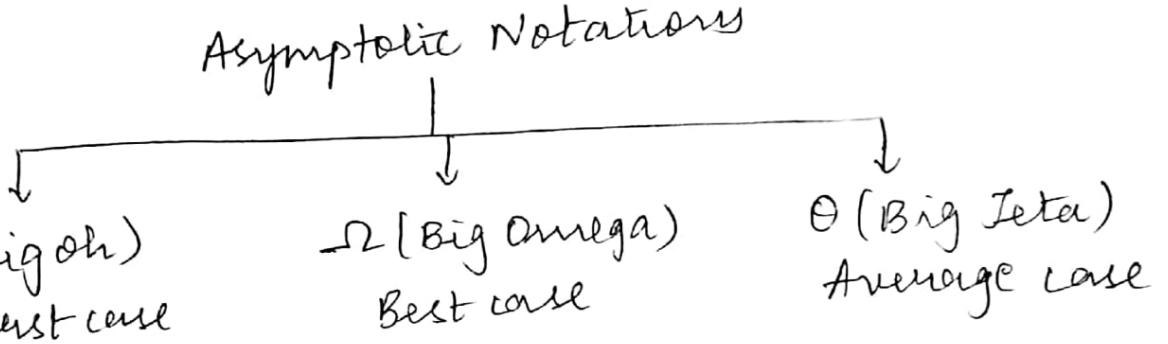
DAA ASSIGNMENT-2

Name: Sanjanya Laxmi.B
Class: IV 'B'
USN: 1CR16IS108
Date: 15/3/18.

① Explain the asymptotic notations for analysis of algorithms. Support your answer with proper graphs and examples.



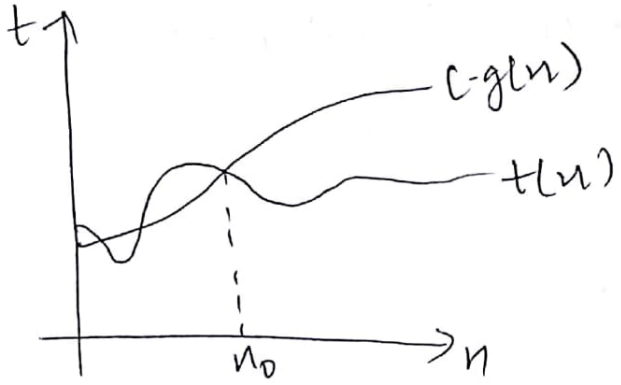
(i) O (Big oh) :-

→ A function $t(n) \in O\,g(n)$ if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large values of n i.e, if there exists a positive constant 'c' & some non-negative integer n_0 such that,

$$t(n) \leq c \cdot g(n) \quad \forall n \geq n_0$$

→ $O(g(n))$ – It stands for the set of all functions with a smaller or same order of growth as $g(n)$. constant as $n \to \infty$.

$n \in O(n^2)$; $100n + 5 \in O(n^2)$; $n^2 - 1 \in O(n^2)$;
$n^3 \notin O(n^2)$; $n^4 + n + 1 \notin O(n^2)$



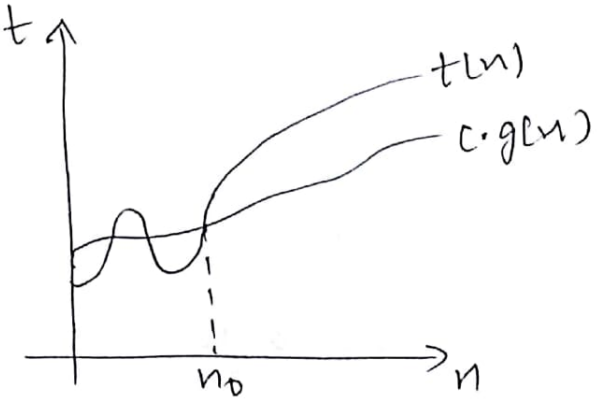
(ii) Ω (Big Omega) :-

→ A function $t(n) \in \Omega(g(n))$ if $t(n)$ is bounded below by some positive constant c & some non-negative integer n_0 such that,

$$t(n) \geq c \cdot g(n) \quad \forall n \geq n_0$$

→ $\Omega(g(n))$ – It stands for a set of all functions with a larger or same order of growth as $g(n)$ constant when $n \to \infty$.

$n^3 \in \Omega(n^2)$; $\frac{1}{2}n(n-1) \in \Omega(n^2)$; $10n+5 \notin \Omega(n^2)$.

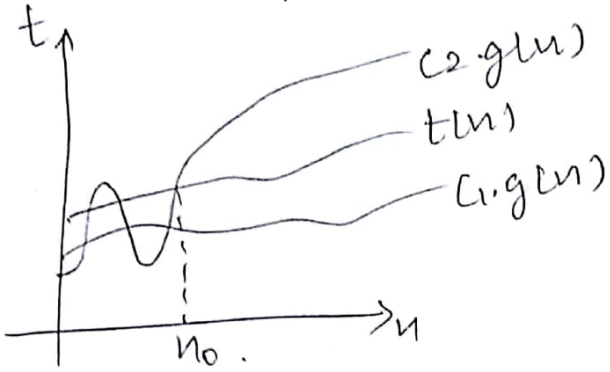


(iii) Θ (Big Teta) :-

→ A function $t(n) \in \Theta(g(n))$ if $t(n)$ is bounded both above & below by some positive constant c_1 & c_2 for all value of n greater than or equal to n_0.

$\therefore \boxed{c_1 \cdot g(n) \leq t(n) < c_2 \cdot g(n) \quad \forall n \geq n_0}$

→ $\Theta(g(n))$ - Set of all functions that have the same order of growth as $g(n)$. constant as $n \to \infty$.
$an^2+bn+c \in \Theta(n^2)$ provided $a>0$.



(2a) Write the control abstraction for the Divide & conquer technique, thus explaining the strategy.

→ Algorithm D&C(P)
{
  if Small(P) then return S(P);
  else
  {
    divide P into smaller instances $P_1, P_2, \ldots, P_k; k \geq 1;$
    Apply D&C to each of the subproblems;
    return combine(D&C(P_1), D&C(P_2), . . .
                   . . , D&C(P_k));
  }
}

→ control abstraction means a procedure whose flow of control is clear but primary operations are specified by other procedures whose precise meanings are left undefined.

→ Small(P) is a boolean value function that determines whether the i/p size is small enough, that the answer can be computed without splitting. If this is so, the function S is invoked.

→ Combine is a function that determines the solution to P using the solutions to the k subproblems.

(2b) Solve the following recurrence using substitution method for the case when the constants have values $a=2$, $b=2$, $f(n)=n$.

$T(n) = aT(n/b) + f(n)$ when $n>1$

$T(n) = 1$ when $n=1$.

→ $T(n) = 2T(n/2) + n$.

$T(n) = 2\left[2T(n/2^2) + n/2\right] + n$

$T(n) = 2^2 T(n/2^2) + 2n$.

$T(n) = 2^2\left[2T(n/2^3) + n/2^2\right] + 2n$

$T(n) = 2^3 T(n/2^3) + 3n$.

$\vdots$

$T(n) = 2^i T(n/2^i) + in$.

$T(n/2) = 2T(n/4) + n/2$

$= 2T(n/2^2) + n/2$

$T(n/2^2) = 2T(n/2^3) + n/2^2$

given base condition $T(1) = 1$,

$\therefore n/2^i = 1 \Rightarrow 2^i = n \Rightarrow i = \log_2 n$.

$\therefore T(n) = 2^{\log_2 n} T(1) + \log_2(n)$.

$$\boxed{T(n) = n + n\log n}$$

(4) The recurrence relation for Strassen's algo for Matrix multiplication is

$T(n) = 7T(n/2) + 18n^2$; if $n > 2$

$T(n) = 1$ if $n <= 2$.

Solve the recurrence and determine its complexity.

→ $T(n) = 7T(n/2) + 18n^2$

put $T(n/2) = 7T(n/2^2) + 18(n/2)^2$

$\therefore T(n) = 7\left[7T(n/2^2) + 18(n/2)^2\right] + 18n^2$

$T(n) = 7^2 \cdot T(n/2^2) + 7 \cdot (18/2)^2 18 + 1 \cdot 18n^2$.

put $T(n/2^2) = 7T(n/2^3) + 18(n/2^2)^2$.

$\therefore T(n) = 7^2\left[7T(n/2^3) + 18(n/2^2)^2\right] + 18 \cdot 7(n/2)^2 + 1 \cdot 18n^2$

$T(n) = 7^2\left[7 \cdot T(n/2^3) + 18(n/4)^2\right] + (7/4)^1 \cdot 18n^2 + (7/4)^0 \cdot 18n^2$.

⋮

$T(n) = 7^i \cdot T(n/2^i) + \left[(7/4)^{i-1} + (7/4)^{i-2} + \cdots + (7/4)^1 + (7/4)^0\right] 18n$

Base condition $T(1) = 1$,

$\therefore n/2^i = 1 \Rightarrow i = \log_2 n$

$\therefore T(n) = 7^{\log_2 n} T(1) + [\text{sol}^n \text{ of GP}] \cdot 18n^2$

$= n^{\log_2 7}(1) + \cdots 18n^2 = n^{2.81} + \cdots 18n^2$

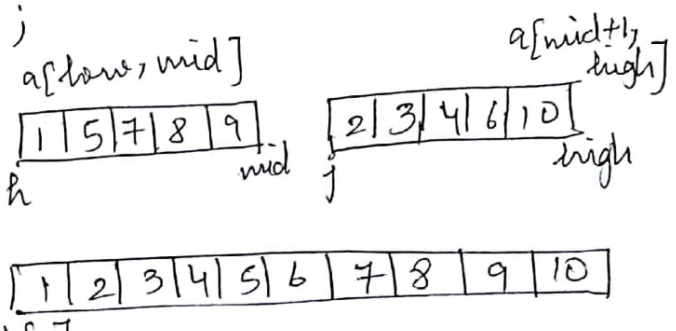
$\approx \Theta(n^{2.81})$ //.

OR (4) Write the algo for Merge Sort and solve its recurrence relation to determine its complexity.

→ Algorithm MergeSort (low, high)
//a[low:high] is a global array to be sorted;
Small(P) is true if there is only one element in the array. In this case your list is already sorted//.
{
if (low < high) then // there are more than one elements //
{ // Divide P into subproblems. Find where to split the set //.
mid = $\lfloor (low+high)/2 \rfloor$;
// solve the subproblems //.
MergeSort (low, mid);
MergeSort (mid+1, high);
// combine the sol^n to the subproblems //.
Merge (low, mid, high);
}
}

Algorithm Merge (low, ~~low~~ mid, high)
//a[low:high] is a global array containing two sorted subsets; a[low, mid]; a[mid+1, high].
The goal is to merge these two sets into a single set residing in a[low:high] using array b[] as auxillary//.

```
{
  h=low; i=low; j=mid+1;
  while (h ≤ mid) and (j ≤ high) do
  {
    if (a[h] ≤ a[j]) then
    {
      b[i] = a[h]; h=h+1;
    }
    else
    {
      b[i] = a[j]; j=j+1;
    }
    i=i+1;
  }
  if (h > mid) then
  for k=j to high do
  {
    b[i] = a[k]; i=i+1;
  }
  else
  for k=h to mid do
  {
    b[i] = a[k]; i=i+1;
  }
  for k = low to high
  do a[k] = b[k];
}
```



Recurrence relation

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn, & \text{if } n > 1 \end{cases}$$

For worst case, $C(n) = n - 1$.

$\therefore 2T(n/2) \cdot T(n) = n \log_2 n - n + 1$.

$O(n \log n)$

(5) What strategy does the Binary Search algorithm use? Write the recursive algorithm for Binary Search and state its complexity.

→ Binary Search is a remarkably efficient algo for searching in a sorted array.
It works by comparing a search key k with the arrays middle element A[m].
If they match, the algo stops otherwise the same operation is repeated recursively for the first half of the array if $k < A[m]$ & second half of array if $k > A[m]$.

$A[0], A[1], \ldots, A[m-1]$, $A[m]$, $A[m+1], \ldots, A[n-1]$

k points to $A[m]$

$k < A[m]$ $\qquad$ $k > A[m]$

Algorithm BinSrch(a, i, l, x)
// a[i:l] gives an array of elements in non-decreasing order $1 \le i \le l$.
Determine where x is present & if so return j such that $x = a[j]$; else return 0 //.

```
{
  if (l == i) then // Small(P).
  {
    if (x == a[i]) then return i;
    else return 0;
  }
  else
  // Reduce P into a smaller subproblems //.
  mid = ⌊(i+l)/2⌋;
  if (x == a[mid]) then return mid
  else if (x < a[mid] then
    return BinSrch(a, i, mid-1, x);
  else return
    BinSrch(a, mid+1, l, x);
}.
```

$T(1) = 1$ // Only one element in array

$T(n) = T(n/2) + c$

$T(n) = T(n/2^2) + c$

$T(n) = T(n/2^3) + c$

$\vdots$

$T(n) = T(n/2^i) + c$

$n/2^i = 1 \quad (\because T(1) = 1)$

$\Rightarrow 2^i = n \Rightarrow i = \log_2 n.$

$\therefore T(n) = \log_2 n$.

Best case $= \Omega(1)$

Worst case $= \Theta(\log_2 n)$

OR

(5) For the following code fragment, compute the worst case asymptotic complexity (as a fn of n), where the loop body is a constant number of lines. You may assume the loop body to be a constant = 1.

```
for (i=0; i<=(n-1); i++) {
    for (j=(i+1); j<=(n-1); j++) {
        // loop body
    }
}
```

→ Worst complexity, $O(T(n)) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1$

$$T(n) = \sum_{i=0}^{n-1} (n-1-i+1) = \sum_{i=0}^{n-1} (n-i)$$

$$= (n-1) \sum_{i=0}^{n-1} 1 - \sum_{i=0}^{n-1} i$$

$$\sum_{i=a}^{b} = b-a+1$$

$$= (n-1)(n-0+1) - \frac{n(n-1)}{2}$$

$$= (n-1)(n) - \frac{n(n-1)}{2}$$

$$= \frac{n(n-1)}{2}$$

$$\approx \frac{1}{2} n^2$$

$\therefore \frac{1}{2} n^2 \in O(n^2)$ //

)
```
Algorithm Mystery(n)
// I/p: a non negative int n.
    S ← 0
    for i ← 1 to n do
    S ← S + i*i;
    return S;
```

(a) What does this algorithm compute?

→ Sum of squares of all numbers upto n.

(b) What is its basic operation?

→ Addition & multiplication [S ← S + i*i].

(c) How many times is the basic operation executed?

→ Since the for loop runs from 1 to n, it executes n times

(d) What ~~to~~ is the efficiency class of this algorithm?

→ The efficiency class of this algorithm is linear function or is O(n).

OR (3) You are given set of n elements. Your task is to design an algo to find the max & min element in the set. Use Divide & Conquer approach. The algo should satisfy all the criteria of a good algorithm.

→ Algorithm MaxMin (i, j, max, min)
// a[1:n] is your global array, parameter i & j are integers $1 \le i \le j \le n$.
To set max & min to the largest & smallest values in the array A[i:j] respectively //.

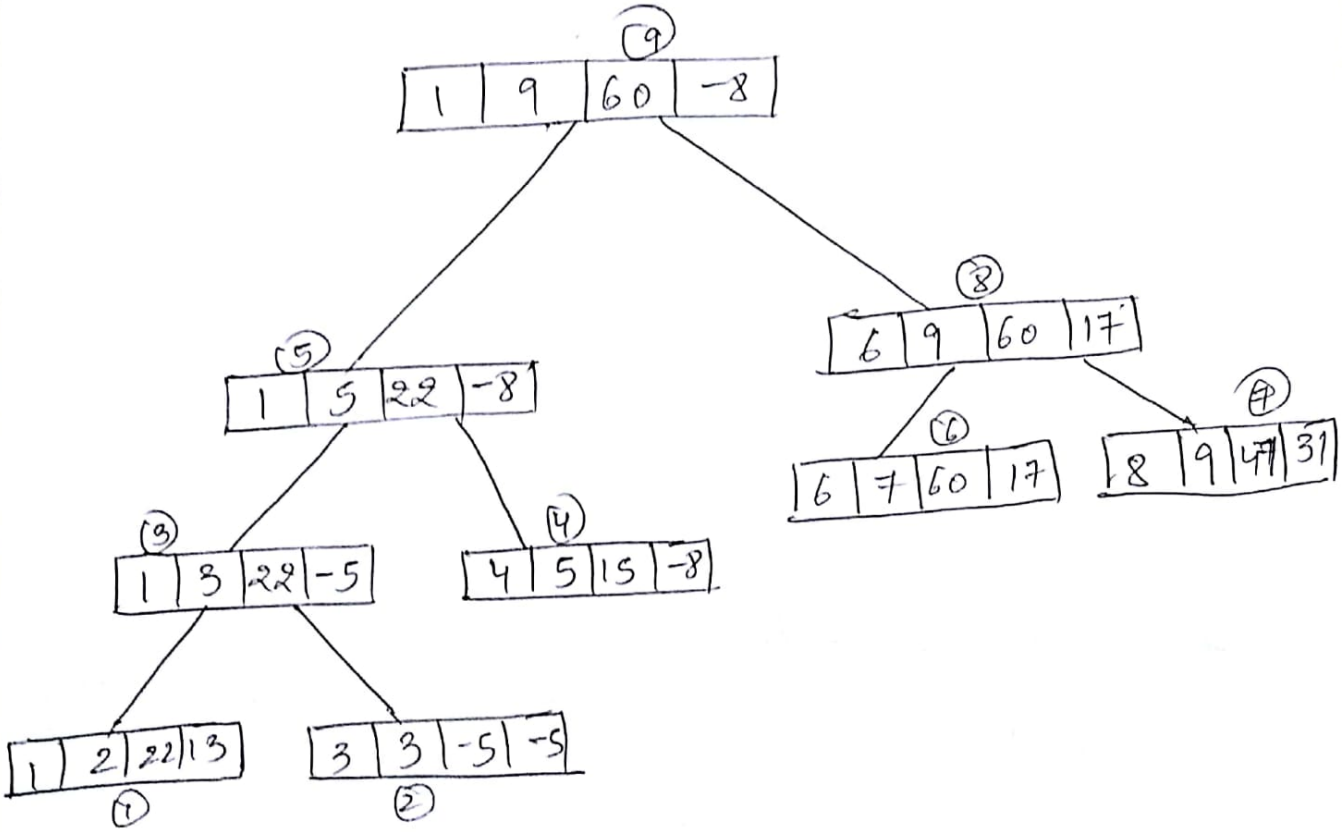
```
{ if (i == j) then max := min := a[i]; //Small(P)
  else if (i == j-1) then // Small(P) with 2 ele  (with 1 ele)
  {
    if (a[i] < a[j]) then
    { max := a[j]; min := a[i];
    }
    else
    { max := a[i]; min := a[j];
    }
  }
  else
  { // Divide P into subproblems //.
    mid = ⌊(i+j)/2⌋;
    // solve subproblems //
    MaxMin (i, mid, max, min);
    MaxMin (mid+1, j, max1, min1);
  // combine the solutions //.
  if (max < max1) then ; max = max1;
```

```
    if (min > min1) then, min = min1;
  }
}.
```

The procedure is initially invoked by the call MaxMin(1, n, x, y).

	1	2	3	4	5	6	7	8	9
a[ ]	22	13	-5	-8	15	60	17	31	47

max = 60
min = -8



(6) What is an algorithm? Give 2 real world examples of algos which are used in daily life. Give the criteria that a good algorithm should satisfy.

→ Algorithm is finite set of instructions, that if followed accomplishes a particular task.

An algorithm must satisfy the following criteria.

(i) Input - Zero or more quantities are externally supplied.

(ii) Output - Atleast one quantity is produced.

(iii) Definiteness - Each instruction is clear and unambiguous.

(iv) Finiteness - If we trace out the instructions for an algo, then for all cases, the algo should terminate after a finite number of steps.

(v) Effectiveness - Every instruction must be very basic so that it can be carried out in principle by a person using only pencil & paper.

The 2 real world examples of algos which are used in daily life are making coffee and cooking maggi.

Algo 1 - Making coffee.

Inputs - cup, coffee powder, boiled milk, sugar

Output - cup of coffee.

Steps:

1) If above ingredients
2) If any one of the above ingredients are not available, go to step 6.

2) Take a cup and add coffee powder in it.
3) Now add required amount of sugar.
4) Pour the milk (boiled milk) into the cup.
5) Cup of coffee is ready.
6) Stop.

Algo - 2 - cooking maggi

Inputs - Maggi (noodles), Bowl, water, spoon, Maggi masala, stove.
Output - Cooked Maggi.

Steps:

1) If any one of the above ingredients are not available, go to step 6.
2) First place the Bowl on the stove, add 2 cups of water and leave it to boil.
3) When the water is boiling, add maggi noodles in it and leave it for a minute.
4) Now add maggi masala and stir it with the spoon and leave it on the stove for a minute
5) Now your cooked maggi is ready.
6) Stop.