SOLVED QUESTION BANK

[Sequence given as per syllabus]

INTRODUCTION

Asymptotic notation is used to describe the running time of an algorithm. This shows the order of growth of function. Asymptotic notation describes the algorithm efficiency and performance in a meaningful way.Asymptotic notation also describes the behaviour of time or space complexity for large instance characteristics.

ASYMPTOTIC NOTATIONS OF ANALYSIS OF ALGORITHMS

Q.1. What are the different asymptotic notations? Explain each one in detail. *CT : S-09, W-09,12(7M), W-11(6M), CS : S-11(5M)*

OR Explain asymptotic notation of analysis of algorithm in detail.

CS : S-12(5M)

CT : S-13(6M)

OR Define asymptotic notations. *CS : W-10(1M), W-11(4M)*

OR Explain different asymptotic notations with their respective functions while doing the analysis of algorithms.

CT : W-06(6M), S-10, W-13(5M)

OR Write short note on asymptotic notations. *CT : S-07, W-07(6M)*

OR Explain asymptotic notation in detail. *CT : S-06(6M), S-08(7M)*

OR Define and explain Big O notation to express time complexity of algorithm. *CT :W-08(2M)*

OR State various asymptotic notation available for analysis of algorithms. *CT : S-14(4M)*

Ans. Asymptotic notations :

- Asymptotic notations are used to find out three phase complexity of a function.
- The asymptotic running time of an algorithm is defined in terms of functions.
- These functions are set of natural numbers and real numbers.
- A way of comparing functions that ignores constant factors and small input size is known as asymptotic notations.
- There are three types of asymptotic notations :

  (1) Theta notation (θ)

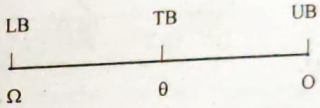
  (2) Omega notation (Ω)

  (3) Big Oh notation (O)
- The complexity of any algorithm is based upon the following three classes :

  (i) Best case

  (ii) Worst case

  (iii) Average case



(1) Theta notation (θ) :

- This notation is also called as tight bound.
- The function $f(n) = \theta(g(n))$ if there exists positive constants C_1, C_2 and n_0 such that

  $C_1 g(n) \le f(n) \le C_2 g(n) \forall n > n_0$
- A function f(n) is sandwitched between $C_1g(n)$ and $C_2g(n)$. Since $\theta(g(n))$ is a set and $f(n) \in \theta(g(n))$ we can write :

  $\boxed{f(n) = \theta(g(n))} \forall n > n_0$

Example :

To find θ notation for the function

$$f(n) = \frac{1}{3}n^3 + \frac{1}{2}n^3 + \frac{1}{6}n,$$

We follow the steps as :

Given that,

$$f(n) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

$$= \left(\frac{1}{3} + \frac{1}{2n} + \frac{1}{6n^2}\right)n^3$$

$$\le \left(\frac{1}{3} + \frac{1}{2} + \frac{1}{6}\right)n^3 \le C_1 (n^3)$$

$g(n) = n^3$

$f(n) \le C_1 g(n)$

$f(n) = \theta(n^3)$

(2) **Omega notation (Ω) :**

- This notation is called as lower bound.
- The function $f(n) = \Omega(g(n))$ is called as lower bound of 'g' if there exists a positive constant C and n_0 such that,

$f(n) \geq g(n) * C$ for $n \geq n_0$

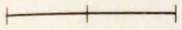
- To calculate lower bound the value closest to f(n) should be selected. This is possible by ignoring the time for singular instruction involved in f(n).

Example :

$f(n) = 3n + 2$

Here, $f(n) = 3n + 2$

$g(n) = 3n$



n 3n + 2 4n

(3) **Big Oh notation :**

The function $f(n) = O(g(n))$ is called as big Oh of g if there exists a positive constant 'C' and n_0 such that,

$f(n) <= g(n) * C$ for $n \geq n_0$

Example :

$f(n) = 3n + 2$

Find g(n)

Here, $f(n) = 3n + 2$

As Big Oh notation is called as upper bound.

$\therefore$ Upper bound of 3n is 4n

$\therefore$ $g(n) = 4$

$C = 2$.

Q.2. **Find upper bound, lower bound, tight bound for following :**

(i) $2n + 5$

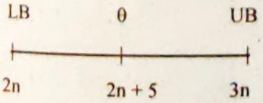
(ii) $20\,n^2 + 8n + 10$

(iii) $5^{2n} + n^2$ **CS : W-10(3M)**

Ans. (i) $2n + 5$:

Upper bound :

Here $f(n) = 2n + 5$



$g(n) = 3n$

constant $C = 5$

Lower bound :

$g(n) = 2n$

$C = 5$

- **Tight bound :**

$f(n) = 2n + 5$

2n 2n + 5 3n

n = 1 2 $\leq$ 7 $\leq$ 3 False

n = 2 4 $\leq$ 9 $\leq$ 6 False

n = 3 6 $\leq$ 11 $\leq$ 9 False

n = 2 8 $\leq$ 13 $\leq$ 12 False

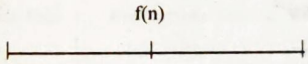
n = 5 10 $\leq$ 15 $\leq$ 15 True

At n = 5 it is valid

$\therefore$ $n_0 \geq 5$ for tight bound.

(ii) **$20\,n^2 + 8n + 10$:**

$f(n) = 20n^2 + 8n + 10$



Lower bound :

Here $g(n) = 20\ n^2$

$C = 10$

Upper bound :

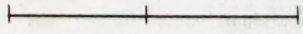
$f(n) = 20n^2 + 8n + 10$

$g(n) = 21n^2$

$C = 10$

Tight bound :

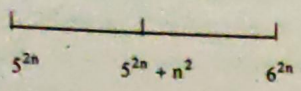
$f(n) = 2n^2 + 8n + 10$



		$20n^2$	$20n^2 + 8n + 10$	$21n^2$
for	n = 1	20	38	21
	n = 2	80	106	42
	n = 3	180	214	63
	n = 4	225	267	84
	n = 5	500	550	105
	n = 6	720	778	126
	n = 7	980	1046	147
	n = 8	1280	1354	168
	n = 9	1620	1720	189
	n = 10	2000	2090	210

(iii) $5^{2n} + n^2$:

$f(n) = 5^{2n} + n^2$



Lower bound :

$f(n) = 5^{2n} + n^2$

$C_1 = 5$

$g(n) = 5^{2n}$

$g(n) = 2^n$

Upper bound :

$g(n) = 6^{2n} \Rightarrow g(n) = 2n$

$C_2 = 6$

Tight bound :

A function must be upper and lower bounded as

$C_1\, g(n) \le f(n) \le C_2\, g(n)$

$5^{2n} \le 5^{2n} + n^2 \le 6^{2n}$

$\therefore \quad f(n) = \theta\, g(n)$

$f(n) = \theta\,(2n)$

Q.3. Find the values of constants using various approaches :

(i) 3n + 2

(ii) $10n^2 + 4n + 2$ CS : S-11, W-12(2M)

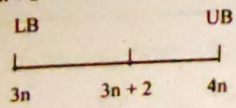
Ans.

(i) 3n + 2 :

Here $f(n) = 3n + 2$

Lower bound :

$f(n) = 3n + 2$



Here

$C_1 = 3$

$g(n) = 3n$

Upper bound :

$f(n) = 3n + 2$

$g(n) = 4n$

$C = 4$

Tight bound :

$f(n) = 3n + 2$

Lower bound for given $f(n)$

$3n \le 3n + 2 \quad \forall\, n > n_0$

Here, $C_1 = 3$

$g(n) = n$

Upper bound for given f(n)

$3n + 2 \le 4n \quad \forall\, n \ge n_0$

Here $C_2 = 4$

$g(n) = n$

$f(n) = \theta\, g(n)$

$f(n) = \theta(n)$

(ii) $10n^2 + 4n + 2$:

$f(n) = 10n^2 + 4n + 2$

Lower bound of function f(n) :

Here,

$g(n) = n^2$

$C = 10$

Upper bound of function f(n) :

Here,

$g(n) = 11n^2$

$C = 11$

Tight bound of function f(n) :

Here, $g(n) = n^2$

$C_1 = 10$ and $C_2 = 11$

$\therefore \quad f(n) = \theta\,(g(n))$

$f(n) = \theta(n^2)$

Q.4. Find upper bound, lower bound and tight bound range for following :

(i) 3n + 3

(ii) $6^{2n} + n^2$ CS : W-12(4M)

Ans. (i) 3n + 3 :

Here $f(n) = 3n + 3$

Upper bound :

$f(n) = 3n + 3$

$n_0 = 3$

$g(n) = 4n$

$g(n) = n$

$C = 4$

Lower bound :

$f(n) = 3n + 3$

$n_0 = 3$

$g(n) = 3n$

$g(n) = n$

$C = 3$

Tight bound :

$C_1\, g(n) \le f(n) \le C_2\, g(n)$

$3 \cdot 3n \le 3n + 3 \le 4.4(n)$

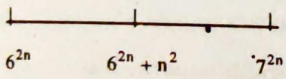
$f(n) = \theta g(n)$

$f(n) = \theta(n)$

(ii) $6^{2n} + n^2$:

Here, $f(n) = 6^{2n} + n^2$

$C_1 = 6$



Upper bound :

$f(n) = 6^{2n} + n^2$

$n_0 = 6$

$g(n) = 2n$

$C = 7$

Lower bound :

$f(n) = 6^{2n} + n^2$

$g(n) = 2n$

$C = 6$

i.e. $C_1 = 6, C_2 = 7$

Tight bound :

$C_1\, g(n) \le f(n) \le C_2\, g(n)$

$6 \times 2n \le 6^{2n} + n^2 \le 7 \times 2n$

$\therefore\ f(n) = \theta g(n)$

$\therefore\ \boxed{f(n) = \theta(2n)}$

ANALYZING CONTROL STRUCTURES

Q.5. Give the analysis of control structures available in algorithms.

Ans. Analyzing control structures :

- The analysis of an algorithms is calculated by considering its individual instructions.
- The individual instructions are calculated and then according to the control structures, we combine these times.

Some algorithm control structures are :

(1) Sequencing.

(2) If-then-else.

(3) "For" loop.

(4) "While" loop.

(5) Recursion.

(1) Sequencing :

- Suppose our algorithms consists of two parts A and B.
- A takes time t_A and B takes time t_B for computation. The total computation time "$t_A + t_B$" is calculated according to the sequencing rule. According to maximum rule this computation time is $(\max(t_A, t_B))$.

Example : Suppose $t_A = O(n)$ and $t_B = \theta(n^2)$. Calculate the total time computation as,

A $t_A = O(n)$

↓

B $t_B = \theta(n^2)$

Computation time = $t_A + t_B$

$= (\max(t_A, t_B))$

$= \max(O(n), \theta(n^2))$

$= \theta(n^2)$

(2) If-then-else :

- The total computation time is according to the conditional rule"if-then-else".
- According to maximum rule this computation time is $\max(t_A\ t_B)$.

Example :

Suppose $t_A = O(n^2)$ and $t_B = \theta(n^2)$ calculate the total

computation time for the following

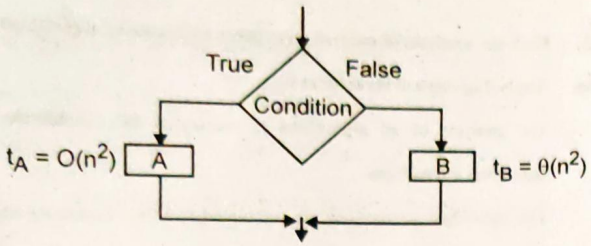


Fig.

Total computation = $\max(t_A, t_B)$

$= \max(O(n^2), \theta(n^2))$

$= \theta(n^2)$

(3) For loop :

- Consider the following loop

```
for i ← 1 to m
  {
    P(i)
  }
```

- If the computation time t_i for P(i) varies as a function of 'i' then total computation time for the loop is given not by the multiplication but by the sum i.e.

```
for i ← 1 to m
  {
    P(i)
  }
```

Take $\sum_{i=1}^{m} t_i$ time i.e. $\sum_{i=1}^{m} \theta(1) = \theta \sum_{i=1}^{m} 1 = \theta(m)$

- If algorithm consists of nested "for" loops then computational time is :

```
for i ← 1 to m
  {
    for j ← 1 to m   Σ_{i=1}^{m} Σ_{j=1}^{m} t_ij
      {
        P(ij)
      }
  }
```

(4) While loop :

- In this there is no obvious method which determines how may times we shall have to repeat the loop.
- The simple technique for analysing the loops is to first determine function of variable involved whose value decreases each time around.
- Secondly, for terminating the loop, it is necessary that this value must be a positive integer. By keeping track of how many times the value of function decreases, one can obtain the number of repetition of the loop.
- The other approach for analysing "while" loop is to treat them as a recursive algorithm.

Example :

- The running time of algorithm array max for computing the maximum element in an array of n integer is O(n).

So,

```
array max (A, N)
current max ← A[0]
for i ← 1 to n – 1
do if current max < A[i]
    then current max ← A[i]
    return current max.
```

- The number of primitive operations t(n) executed by this algorithm is atleast

$2 + 1 + n + 4(n-1) + 1 = 5n$ and atmost

$2 + 1 + n + 6(n-1) + 1 = 7n - 2$

- The best case $t(n) = 5n$ occurs when A[0] is the maximum element. The worst case $t(n) = 7n - 2$ occurs when the elements are sorted in increasing order.
- We may therefore apply the big-oh definition with $C = 7$ and $n_0 = 1$ and conclude the running time of this is O(n).

(5) Recursion :

- A procedure that calls itself, directly or indirectly, is said to be recursive.
- The use of recursion often permits more lucid and concise descriptions of algorithms than would be possible without recursion.
- It is easier to understand recursive programs.

Example :

Recursive procedure for in order

```
procedure INORDER(VERTEX):
begin
1.  if LEFTSON[VERTEX] ≠ 0 then
        INORDER(LEFTSON[VERTEX]);
2.  NUMBER[VERTEX] ← COUNT;
3.  COUNT ← COUNT + 1;
4.  if RIGHTSON[VERTEX] ≠ 0 then
        INORDER(RIGHTSON[VERTEX])
end
```

WORST CASE AND AVERAGE CASE ANALYSIS

Q.6. Write a detailed note on worst and average case analysis.

OR Explain worst case and average case analysis.

Ans. Worst case and average case analysis :

- Worst case performance analysis and average case performance analysis have similarities but usually require different tools and approaches in practice.
- Determining what average input means is difficult and often that average input has properties which makes it difficult to characterize mathematically.
- When a sensible description of a particular "average case" is possible they tend to result in more difficult to analyse equations.
- Worst case analysis has similar problems, typically it is impossible to determine the exact worst case scenario.
- Instead, a scenario is considered which is atleast as bad as the worst case.

Example : When analysing an algorithm it may be possible to find the longest possible path through the algorithm even if it is not possible to determine the exact input that could generate this indeed, such an input may not exist. This lead to a safe analysis, but which is pessimistic.

- When analysing algorithm which often take a small time to complete but periodically require a much larger time, amortized analysis can be used to determine the worst case running time over a series of operations.
- The other thing that has to be decided when masking these considerations is whether what is of interest is the average performance of a program or whether it is important to guarantee that even in the worst case, performance obeys certain rules.

AMORTIZED ANALYSIS

Q.7. What do you mean by amortized analysis of algorithms? Explain any one with suitable example.

CT : S-08(6M), W-10, S-13(7M)

OR What is amortized complexity? Explain amortized complexity for 4 bit binary number from 0 to 8. Write algorithm for binary instrumentation operation.

CS : W-10(6M), W-12,13(7M), S-14(4M)

OR Explain three methods to implement amortized complexity.

CT : W-13(6M)

CS : W-11, S-12(4M)

OR What is amortized complexity? Explain all methods of it.

CS : S-11(6M)

Ans. Amortized analysis :

- An amortized analysis is any strategy for analyzing a sequence of operations to show that average cost per operation is small, even though a single operation within the sequence might be expensive.
- It refers to find the average running time per operation over a worst case sequence of operations.
- In amortized analysis, the time required to perform a sequence of data structure operations is average over all operation performed.
- Amortized analysis assumes worst case input and typically does not allow random choices.
- The three method used in amortized analysis are as follows:

  (1) Aggregate method.

  (2) Accounting method.

  (3) Potential method.

(1) Aggregate method :

- Aggregate method determines the upper bound T(n) on the total cost of a sequence of n operations, then calculates the average cost to be T(n)/n.
- In this method initially total time requirement is calculated and then the time is divided by size of the input.

Example : Stack operation with MULTIPOP.

- The PUSH operation insert an element to the top of the stack and POP operation deletes an element from the top of the stack. Here we are augmenting the stack data structure with a new operation "MULTIPOP".

```
MULTIPOP (S, X)
        while not stack empty (s) and k ≠ 0
        do pop (s)
        k ← k – 1
```

- Here, a stack S and number k, operation MULTIPOP removes the top k objects on the stack. MULTIPOP just call pop either k times or until stack is empty.

(2) Accounting method :

- This method is based on keeping track of operations performed on the given input.
- In this method for some initial operations on the input, additional execution time is added.
- This additional time is balanced if the same input is operated again and again.
- The additional time can be considered as time required for execution.

Example : Web page opening

(1^{st}, 2^{nd} time the page may take more time to open but next time it may open instanteously as the page is stored in cache memory.)

(3) Potential method :

- In this method the input is divided into two classes :

(i) General input.

(ii) Potential input.

- General input represents the input requiring general time for operations.
- Potential input will consume the maximum time of total time required for execution, hence if the time required for execution is to be minimized then potential input should be the main consideration.

Example : Binary incrementor

Consider binary numbers between 0-8

0000 → 0 cycle
0001 → 1 cycles
0010 → 2 cycles
0011 → 1 cycles
0100 → (3) cycles
0101 → 1 cycles
0100 → 2 cycles
0111 → 1 cycles
0000 → (4) cycles

are consuming more cycle i.e. potential input

Since there are 4 bit 8 numbers the total time for implementing binary incrementor will be

$4 \times 8 = 32$ cycles

Algorithm for binary incrementor :

```
Algorithm binary incrementor
{
    i = n
  while (i >= 1) and (a [i] = 1)
      {
        a[i] = 0
        i = i – 1
      }
  a[i] = 1
}
```

APPLICATION OF AMORTIZED ANALYSIS

Q.8. Give the applications of amortized analysis.

Ans. The various applications of amortized analysis are :

(1) Stack operation :

- In the following pseudocode, the operation STACK-EMPTY returns TRUE if there are no object currently on the stack and FALSE otherwise

MULTIPOP (s, k)

Step 1: While (NOT STACK-EMPTY (s) and k ≠ 0)

Step 2: do pop(s).

Step 3: k = k – 1.

(2) **Binary counter :**

- We use an array A [0, ..., k – 1] of bits, where length [A] = k, as the counter.
- A binary number x that is stored in the counter has its lowest order bit in A[0] and highest order bit is A[k –1], so that

$$x = \sum_{i=0}^{k-1} 2^i A[i]$$

INCREMENT (A)

Step 1: i = 0.

Step 2: while i < length [A] and A[i] = 1.

Step 3: do A[i] = 0.

Step 4: i = i + 1.

Step 5: if i < length [A].

Step 5: then A[i] = 1.

SORTING NETWORKS

Q.9. **What is sorting network? Explain.** *CS : S-13,14(2M)*

OR **Give advantages of sorting network.** *CS : W-13(2M)*

Ans. **Sorting Network :**

- A sorting network is a comparison network for which the output sequence is monotonically increasing (i.e. $b_1 \le b_2 \le ..... b_n$) for every input sequence.
- Sorting network is capable of sorting thousands of items in the order of microseconds and can be constructed with present day hardware.
- Any sorting network on n inputs has depth of atleast logn the number of comparator in any sorting network is atleast Ω (n log n)

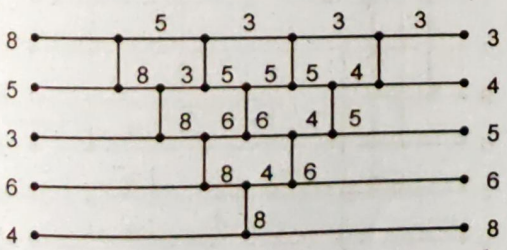


Fig. A sorting network based on insertion sort

Advantages of sorting network :

(1) Capable of sorting thousands of item in the order of microseconds.

(2) Fast sorting capability can be used to manipulate large set of data.

(3) Standard modules of convenient size can be picked up and used in any size network to lower the cost.

COMPARISON NETWORK

Q.10. **What is comparison network? Explain.**

Ans. **Comparison network :**

- Comparison network is made of wires and comparators.
- A comparator is a device with two inputs x and y and two outputs x′ and y′, where

  x′ = min (x, y)

  y′ = max (x, y)
- In comparison network inputs appear on the left and outputs on the right, with the smaller input value appearing on the top output and the larger input value appearing on the bottom output.
- Comparison network is a set of comparators interconnected by wires.

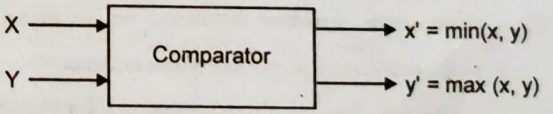


Fig. Comparison network

- Comparison network differs from RAMs in two important respects.
- First, they can only perform comparisons. Thus an algorithm such as counting sort cannot be implemented on a comparison network.
- Second, unlike RAM model in which operations occur serially i.e. one after another operation in comparison network may occur at same time or parallel.
- A comparison network contains n input wires $a_1, a_2, ... a_n$ through which the values to be sorted enter the network and n output wires $b_1, b_2, ...b_n$ which produced the results computed by the network.

BITONIC SORTING NETWORK

Q.11. Explain in detail bitonic sorting network.

Ans. **Bitonic sorting network :**

- A sequence that either monotonically increases and then monotonically decreases or else monotonically decreases and then monotonically increases is called bitonic sequence.

  Example : The sequence < 2, 5, 6, 9, 3, 1 > and < 8, 7, 5, 2, 4, 6 > are both bitonic.

- The bitonic sorter is a comparison network that sorts bitonic sequences of 0's and 1's.

(1) The half cleaner :

- A bitonic sorter is comprised of several stages, each of which is called a half-cleaner.
- Each half cleaner is a comparison network of depth 1 in which input line i is compared with line i + n/2 for i = 1, 2, ...n/2
- When a bitonic sequence of 0's and 1's is applied as input to a half cleaner, produces an output sequence in which smaller values are in the top half, larger values are in bottom half and both halves are bitonic.

(2) Bitonic sorter :

- By recursively combining half-cleaners we can build a bitonic sorter which is a network that sorts bitonic sequences.
- The first stage of bitonic sorter [n] consists of HALF-CLEANER[n], which produces two bitonic sequences of half the size such that every element in the top half is atleast as small as every element in the bottom half.

Q.12. Explain the operation of 1, 3, 5, 9, 8, 7, 4, 2 bitonic sorting network. *CS : S-14(5M)*

OR Design a 8 bit bitonic sorting network and explain its operation for following example.

1, 3, 5, 9, 8, 7, 4, 2. *CS : S-13(5M)*

Ans.

- The given sequence < 1, 3, 5, 9, 8, 7, 4, 2 > is bitonic because the sequence monotonically increases and then monotonically decreases.
- The input is as per sequences given and output is monotonically increasing sequences.

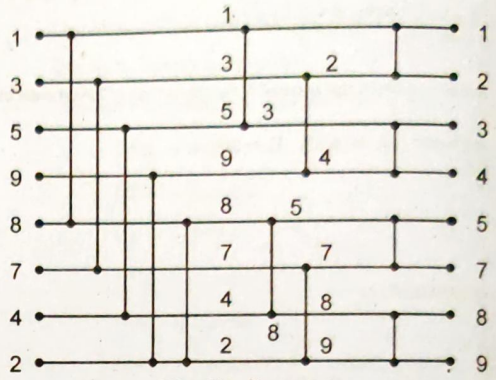


Fig. Bitonic sorter

Here n = 8

Input to the circuit, the i^{th} wire is connected to $1+\frac{n}{2}$ wire

As n = 8

$\therefore\ 1+\frac{8}{2} = 1+4 = 5$

$\therefore$ 1 wire is connected to 5^{th} wire

i.e. 8 and so on.

In next stage $1+\frac{5}{2} = 3.$

$\therefore$ 1 is connected to 5, 2 to 4, 5 to 8, 7 to 9.

Q.13. Implement biotonic sorting network for the following set of information :

1, 5, 7, 2, 8, 6, 3, 9. *CS : W-13(4M)*

Ans. Sequence :

1, 5, 7, 2, 8, 6, 3, 9

n = 8

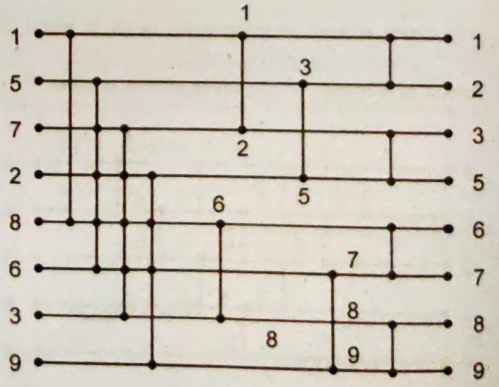


Fig. Bitonic sorter

ADVANCED DATA STRUCTURES

Q.14. Explain the structure of Fibonacci heap.

Ans. **Fibonacci heap :**

- Fibonacci heaps are linked lists of heap-ordered trees (children's keys are atleast those of their parents) with the following characteristics :

(i) The trees are not necessarily binomial.

(ii) Siblings are bi-directionally linked.

(iii) There is a pointer min[H] to the root with the minimum key.

(iv) The root degrees are not unique.

(v) A special attribute n(H) maintains the total number of nodes.

(vi) Each node has an additional Boolean label mark, indicating whether it has lost a child since the last time it was made a child of another node.

Structure of Fibonacci heap :

- Fibonacci heap like binomial heap contains collections for heap such that trees in Fibonacci heap are not necessarily a binomial tree and also are rooted but not ordered.

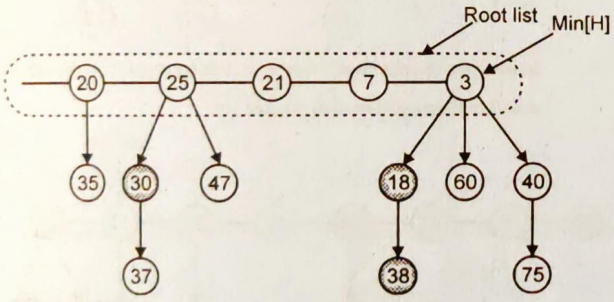


Fig. Fibonacci heap

- The root of Fibonacci heap must be the minimum of all the roots in the minimum heap collection.
- Fibonacci heap is represented by doubly circular linked list with pointer p[x], child[x], left[y], right[y] for node x with child y, left, right represents siblings.

DISJOINT SET REPRESENTATION

Q.15. Explain disjoint set representation and operations.

Ans. **Disjoint set representation :**

- A disjoint set data structure maintains a collection $s = \{s_1, s_2, \ldots\ldots s_k\}$ of dynamic sets. Each set in identified by a representative, which is some members of the set.
- If we have two sets s_x and s_y, $x \neq y$ such that $s_x = \{3, 4, 5, 6, 7\}$ and $s_y = \{1, 2\}$ these sets are called disjoint set as there is no element which is common in both sets.

Disjoint set operations :

- In the dynamic set implementations an object represents each element of a set. Letting x denote an object.

MAKE -SET(x) :

Creates a new set whose only member is pointed by x. Since the sets are disjoint we require that x not already be in a set.

UNION (x, y) :

This operations unites the dynamic sets that contains x and y say s_x and s_y into a new set that is union of these two sets. Where s_x and s_y are disjoint sets initially.

FIND-SET (x) :

This operation returns a point to the representative of the (unique) set containing x.

RED BLACK TREES

Q.16. Explain red black trees.

Ans. **RED-BLACK trees :**

- Red black tree is a binary search tree where each node is assigned a color, where the coloring scheme help us maintaining height as $\theta(\log n)$.
- Each node of the red black tree contains the following fields:

(i) Colors (Red/Black).

(ii) Key.

(iii) Left.

(iv) Right.

(v) P(parent).

Properties of Red/Black trees :

A binary search tree is a red-black tree if it satisfies the following properties :

(i) Every node is colored either red or black.

(ii) The root is black.

(iii) Every leaf is black.

(iv) If a node is red then both its children are black.

(v) Every single path from a node to a descendant leaf contains the same number of black nodes.

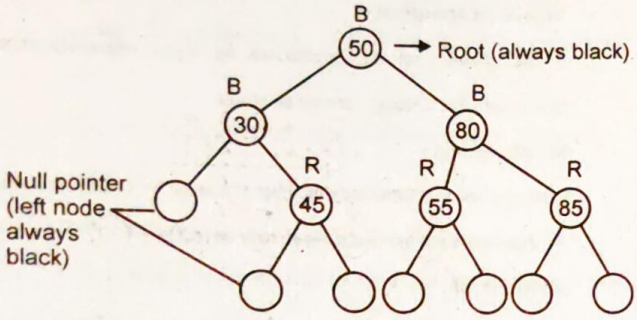


Fig. Simple Red Black face

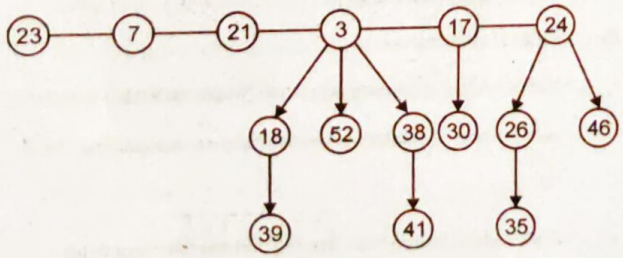
Applications :

(i) To search an element from large database.

(ii) Set of library.

(iii) Set and maths operations.

Q.17. Explain the process of deleting a node from Fibonacci Heap structure. Draw all the modification if minimum value is deleted from the tree. ***CS : S-13(6M)***



Ans. Deleting a node form Fibonacci heap structures :

- It is easy to delete a node from an n-node Fibonacci heap in O(D(n)) amortized time, as is done by the following pseudocode.
- We assume that there is no key value of $-\infty$ currently in the Fibonacci heap.

FiB - HEAP - DELETE (H, x)

(1) FiB - HEAP - DECREASE - KEY (H, x, $-\infty$).

(2) FiB - HEAP - EXTRACT - MIN (H)

FiB - HEAP - EXTRACT - MIN (H)

z ← min [H]

if z ≠ NIL

then for each child x of z.

do add x to the root list of H

P[x] ← NIL

remove z from the root list of H

if z = right[z]

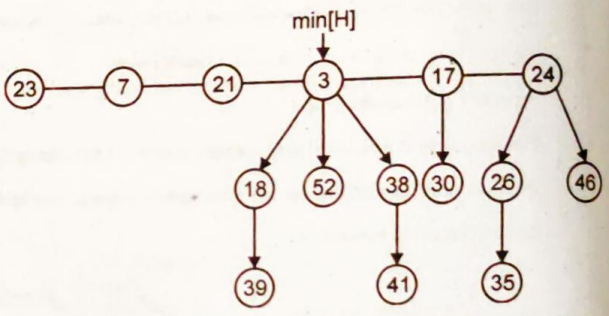
then min[H] ←NIL

else min[H] ← right [z]

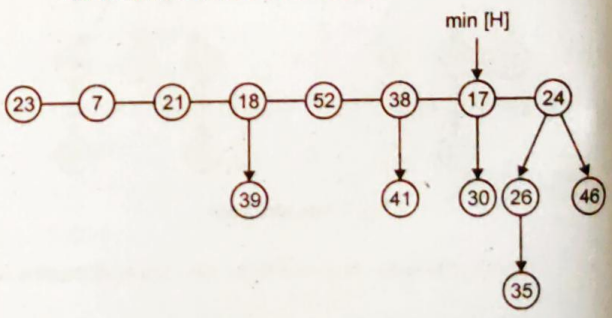
CONSOLIDATE (H)

n[H] ←n [H –1]

return z.



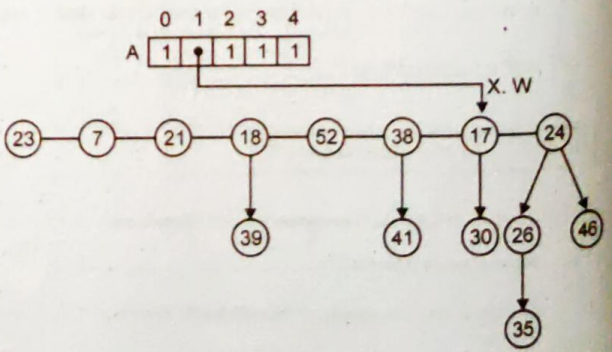
- In first step the minimum node z i.e. 3 is removed from the root list and its children are added to the root list.



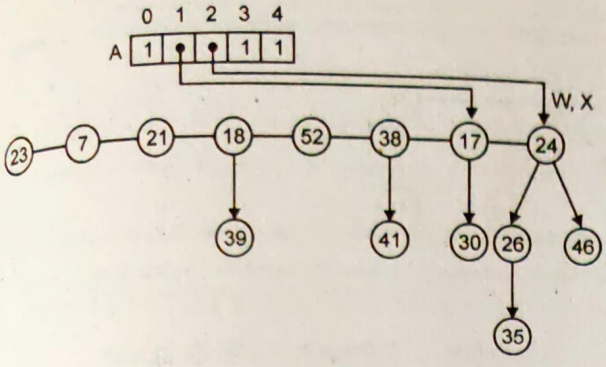
Now, min [H] = right [z]

degree [x] = 1

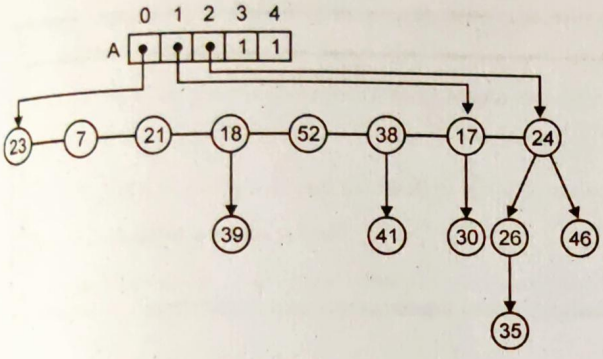
∴ d = 1

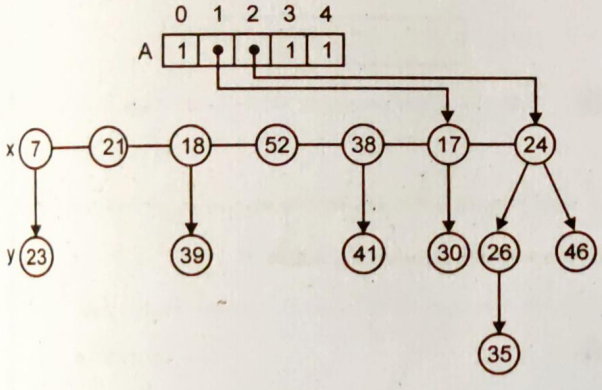


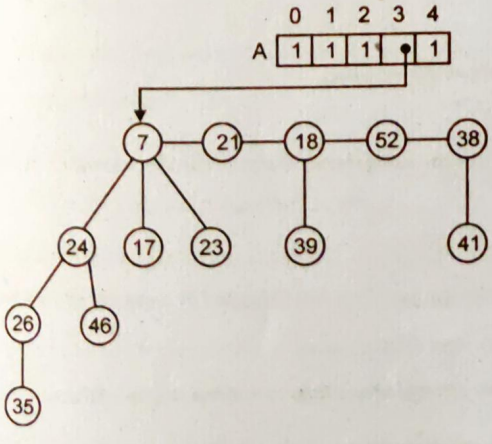
Now w is the right node i.e.
degree [x] = 2

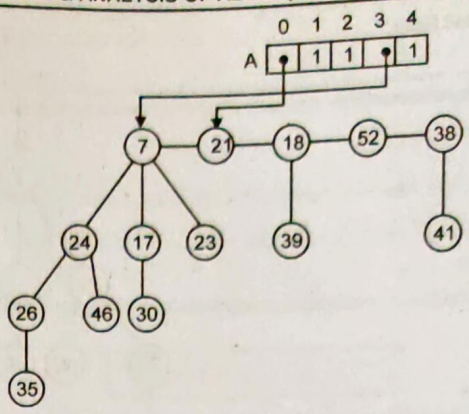


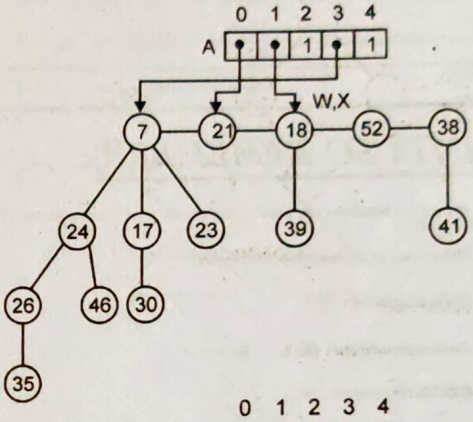
degree [x] = 0

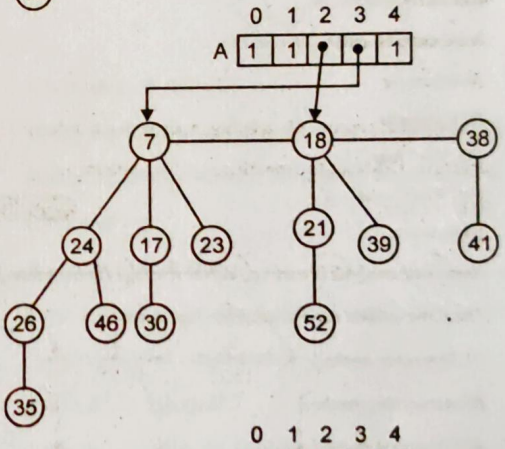


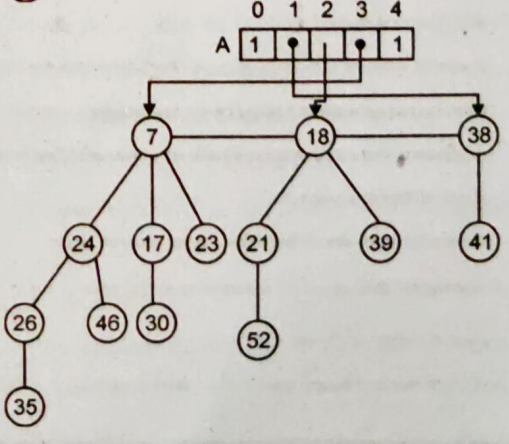




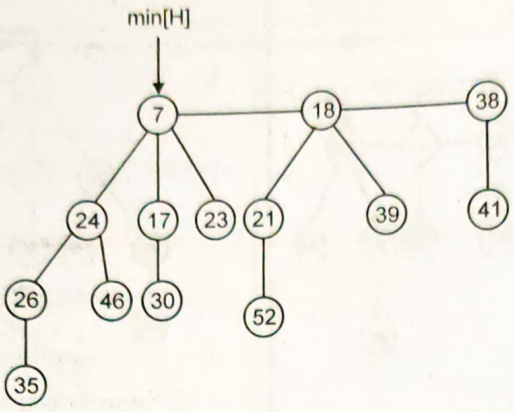








Final Heap :



POINTS TO REMEMBER :

(1) *Asymptotic notations are used to find out three phase complexity of a function.*

(2) *Three types of asymptotic notations are :*

(i) Theta notations (θ)

(ii) Omega notations (Ω)

(iii) Big Oh notations (O).

(3) *Some algorithm control structures are :*

(i) Sequencing

(ii) If-then-else

(iii) "For" loop

(iv) "While" loop

(v) Recursion

(4) *Amortized analysis is used to find the average running time per operation over a worst case sequence of operations.*

(5) *The three method used in amortized analysis are :*

(i) Aggregate method

(ii) Accounting method

(iii) Potential method

(6) *A sorting network is a comparison network for which the output sequence is monotonically increasing.*

(7) *Comparison network is a set of comparators interconnected by wires.*

(8) *A sequence that either monotonically increases and then monotonically decreases or else monotonically decreases and then motonically increases is called bitonic sequence.*

(9) *Fibonacci heaps are linked lists of heap ordered trees.*

(10) *A disjoint set data structure maintanins a collection $s = \{ s_1, s_2, \ldots\ldots s_k \}$ of dynamic sets. Each set is identified by a representative, which is some members of the set.*

(11) *Red black tree is a binary search tree where each node is assigned a color, where the coloring scheme helps to maintain height as $\theta(\log n)$.*