# Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

**Synoptic End Semester Examination**
April 2018

**Duration: 3 Hrs**
**Semester: IV**
**Branch: IT/COMP**

**Max. Marks: 100**
**Class: S.E.**
**Course Code: IT41 / CE41**
**Name of the Course: Design and Analysis of Algorithm**

**Instructions:**
(1) **All Questions are Compulsory**
(2) **Draw neat diagrams**
(3) **Assume suitable data if necessary**

| Question No. | Question |
|---|---|
| *Q. 1 a)* | **Binary Search Algorithms:** |

```
Procedure binary_search
   A ← sorted array
   n ← size of array
   x ← value to be searched
   Set lowerBound = 1
   Set upperBound = n

   while x not found
      if upperBound < lowerBound
         EXIT: x does not exists.

      set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

      if A[midPoint] < x
         set lowerBound = midPoint + 1

      if A[midPoint] > x
         set upperBound = midPoint - 1

      if A[midPoint] = x
         EXIT: x found at location midPoint

   end while

end procedure
```

**Time Complexity:**

Call T(n) the time of binary search when the array size is n.

$T(n) = T(n/2) + c$, where c is some constant representing the time of execution of instructions like computing mid, return statement etc.

Assume for simplicity that $n= 2^k$. (so $k=\log_2 n$)

$T(2^k)=T(2^{k-1})+c=T(2^{k-2})+c+c=T(2^{k-3})+c+c+c = \ldots = T(2^0)+c+c+\ldots c \quad =T(1)+kc$

$=O(k)$ $\qquad =O(\log n)$ $\qquad$ Therefore, $T(n)=O(\log n)$.

**Marks Distribution:**

Binary Search algorithm -------- **03 mks**

Derived Time complexity.----------- **02mks**

*Q. 1 b)*

i) $T(n)=\Theta(n^2 \log n)$ Case 3 applicable

ii) Does not apply as $a=2^n$ which is not constant,

**Marks Distribution:**

Applicable case stated for solvable problem and given correct answer ----- 2.5 mks

Justified properly the non-solvable problem -------2.5 mks

*Q. 1 c)*

***Quick Sort algorithm;***

```
QUICKSORT(A, p, r)
  if p < r
      q = PARTITION(A, p, r)
      QUICKSORT(A, p, q - 1)
      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
  x = A[r]
  i = p - 1
  for j = p to r - 1
      if A[j] ≤ x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1
```

***Time Complexity:***

**Worst Case Analysis:**

*The worst case for quicksort can arise if the orginal array* is already sorted.

If x[a] is in its correct position, the original array is split into subfiles of size '0' and 'n-1'.

If the process continues, a total of n-1 subfiles are sorted, the first of size n, the second of size n-1, the third of size n-2, & so on...

$T(n)= (n-1)+(n-2)+ \text{---------} +1$

$T(n)= \sum_{i=1}^{n-1} i \quad =(n(n-1))/2 \quad = O(n^2)$

*Best Case Analysis:*

Assume that the file size 'n' is power of 2, say $2^X$, so that $x= \log_2 n$.

Assume also that the pivot element is always at the middle of the sub-array.

In that case there will be approximately 'n' comparisions on 1st pass, after which the file is split into two subfiles each of size n/2.

N/2 comparisons is required & which results into n/4 subfiles and n/4 comparisons is required and .........

Thus the total no. of comparisons for entire sort is approximately

T(n)= O(n+ 2*n/2 + 4*n/2 +........+ n*n/n)

= O(n+n+n+ . .. ......+ n) (X terms)

=O(n*X)

=O(nlogn)

***Marks Distribution:***
Quick Sort algorithm -------- **03mks**
Analyze it's time complexity -------- **02mks**

**OR**

**Mergesort Algorithm**

```
Algorithm MergeSort(low, high)
// a[low : high] is a global array to be sorted.
// Small(P) is true if there is only one element
// to sort. In this case the list is already sorted.
{
    if (low < high) then // If there are more than one element
    {
        // Divide P into subproblems.
            // Find where to split the set.
                mid := ⌊(low + high)/2⌋;
        // Solve the subproblems.
            MergeSort(low, mid);
            MergeSort(mid + 1, high);
        // Combine the solutions.
            Merge(low, mid, high);
    }
}
```

```
Algorithm Merge(low, mid, high)
// a[low : high] is a global array containing two sorted
// subsets in a[low : mid] and in a[mid + 1 : high]. The goal
// is to merge these two sets into a single set residing
// in a[low : high]. b[ ] is an auxiliary global array.
{
    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];
}
```

## Time Complexity of Merge Sort:

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

When $n$ is a power of 2, $n = 2^k$, we can solve this equation by successive substitutions:

$$\begin{aligned} T(n) &= 2(2T(n/4) + cn/2) + cn \\ &= 4T(n/4) + 2cn \\ &= 4(2T(n/8) + cn/4) + 2cn \\ &\vdots \\ &= 2^k T(1) + kcn \\ &= an + cn \log n \end{aligned}$$

It is easy to see that if $2^k < n \leq 2^{k+1}$, then $T(n) \leq T(2^{k+1})$. Therefore

$$T(n) = O(n \log n)$$

***Marks Distribution:***

Merge Sort algorithm -------- **03mks**

Analyze it's time complexity -------- **02mks**

| | |
|---|---|
| *Q. 1 d)* | **Divide and Conquer v/s Dynamic Programming**<br>• Both techniques split their input into parts, find subsolutions to the parts, and synthesize larger solutions from smaller ones.<br>• Divide and Conquer splits its input at pre-specified deterministic points (e.g., always in the middle)<br>• Dynamic Programming splits its input at every possible split points rather than at a pre-specified points. After trying all split points, it determines which split point is optimal.<br>**Greedy Method v/s Dynamic Programming**<br>• Both techniques are optimization techniques, and both build solutions from a collection of choices of individual elements.<br>• The greedy method computes its solution by making its choices in a serial forward fashion, never looking back or revising previous choices.<br>• Dynamic programming computes its solution bottom up by synthesizing them from smaller sub-solutions, and by trying many possibilities and choices before it arrives at the optimal set of choices.<br>• There is no priori test by which one can tell if the Greedy method will lead to an optimal solution.<br>• By contrast, there is a test for Dynamic Programming, called the principle of optimality<br>***Marks Distribution:***<br>Atleast two comparison of Divide and Conquer v/s Dynamic Programming --------02mks<br>Atleast three comparison of Greedy Method v/s Dynamic Programming ---------- 03mks |
| *Q. 2 a)* | ***Steps of Sequence:***<br>*To solve a problem by using dynamic* programming:<br>• Find out the recurrence relations.<br>• Represent the problem by a multistage graph.<br>0/1 knapsack using Dynamic Programming:<br>1. There are no items in the knapsack, or the weight of the knapsack is 0 - the benefit is 0<br>2. The weight of $item_i$ exceeds the weight w of the knapsack - $item_i$ cannot be included in the knapsack and the maximum benefit is B[i-1, w]<br>3. Otherwise, the benefit is the maximum achieved by either not including $item_i$ ( i.e., B[i-1, w]), or by including $item_i$ (i.e., $B[i-1, w-w_i]+b_i$) |

| | |
|---|---|
| | ***Marks Distribution:***<br>Steps ------- **06mks**<br>Solve an example correctly ------- **04mks** |
| *Q2. b)* | **Algorithm of Kruskal Method** |

```
MST_Kruskal()
begin
        // Input is simple connected graph represented by array of edges edge[]
        // Output is list of edges T in MST
        // Create a partition for the set of vertices
        foreach vertex v ∈ V
            Cv := {v}

        // create a minHeap h from array of edges E
        h := new Heap( E)

        // let T be an initially empty tree
        T := ∅
        while size(T) < n-1
            (u, v, wgt) :=  h.removeMin()
            Cv := findSet(v)
            Cu := findSet(u)
            if Cu ≠ Cv
                union(Cu , Cv)
                T := T ∪ {(u, v, wgt)}
    return T
end
```

**Time Complexity:**

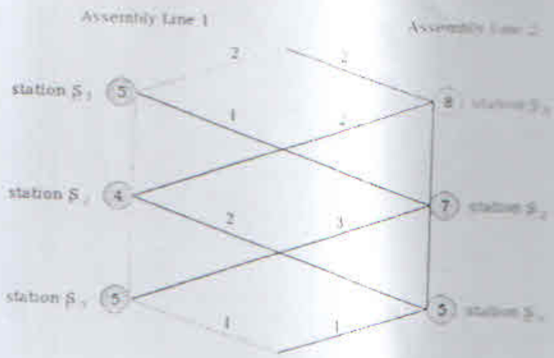
Kruskal's algorithm can be shown to run in $O(E \log V)$ time.
By way of comparison Prim complexity is $O(E \log V)$ for sparse and $O(V^2) = O(E)$ for dense.

***Marks Distribution:***
Algorithm of Kruskal's method ------- **08mks**
Analyzed time complexity -------- **02mks**

OR

Algorithm for Knapsack Problem ------ **05mks**
Time complexity ------- **01mk**

| | |
|---|---|
| | Solve problem correctly ------- **04mks** |
| *Q.3 a)* | |




| | Station 1 | Station 2 | Station 3 | TotalCost |
|---|---|---|---|---|
| $cost_1(j)$ | 6 | 14 | 22 | 24 |
| $cost_2(j)$ | 10 | 13 | 17 | 20 |

| | Station 2 | Station 3 | Finish |
|---|---|---|---|
| $line_1(j)$ | 1 | 1 | 2 |
| $line_2(j)$ | 1 | 2 | 2 |

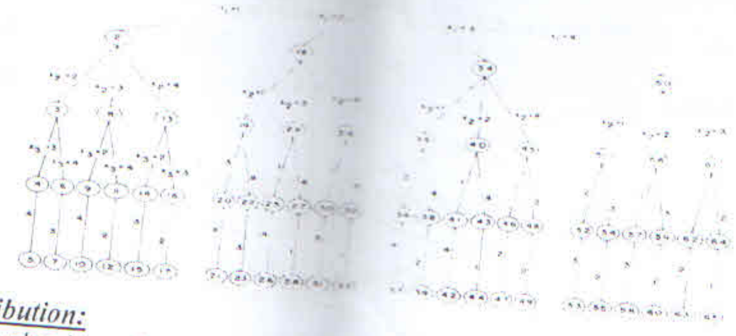
***Marks Distribution:***

Solved correctly with calculation, table of cost and the final selected path shown------ **10mks**

Solved correctly with calculation and table of cost ------ **08mks**

Solved correctly with no calculation shown------ **04 mks**

| | |
|---|---|
| *Q.3 b)* | ***State Space tree for 4 Queen*** |

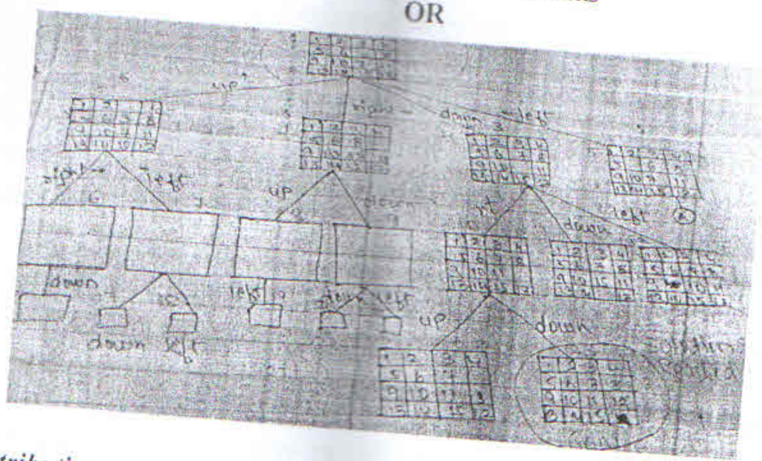


***Marks Distribution:***

Backtracking algorithm for N queen Problem --- — 05mks

State Space Tree ---- 05mks

**OR**



***Marks Distribution:***

Checking for solvable or not solvable ----- 02mks

cost function calculation of each node generated ------ 05mks

State Space Tree -------03 mks

*Q.4 a)*

Algo for Prefix function

1. Start
2. Read Pattern (P) with m characters
3. Let i=1, j=0, f(0)=0
4. while (i<m)

```
{
  if (p[j] = p[i])
  {
      f(i) = j+1;
      i = i+1;
      j = j+1;
  }
  else if (j>0) then j = f(j-1)
  else
      f(i)=0
      i = i+1
} // end of while loop
```

5. Stop

i) cocacola $\xrightarrow[\text{prefix function}]{}$ 0 0 1 0 1 2 0 0

ii) bababba $\xrightarrow[\text{prefix function}]{}$ 0 0 1 2 3 1 2

***Marks Distribution:***

KMP-Prefix Function algorithm --- **03mks**

Correctly solved the prefix function for the given pattern ------- **01mk for each pattern**

OR

Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

**Rabin Karp : Solve correctly------- 05mks**

- Given T = 31415926535 and P = 26
- We choose q = 11
- P mod q = 26 mod 11 = 4

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **3** | **1** | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 |

31 mod 11 = 9 not equal to 4

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | **1** | **4** | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 |

14 mod 11 = 3 not equal to 4

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | **4** | **1** | 5 | 9 | 2 | 6 | 5 | 3 | 5 |

41 mod 11 = 8 not equal to 4

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 4 | **1** | **5** | 9 | 2 | 6 | 5 | 3 | 5 |

15 mod 11 = 4 equal to 4 -> spurious hit

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 4 | 1 | **5** | **9** | 2 | 6 | 5 | 3 | 5 |

59 mod 11 = 4 equal to 4 -> spurious hit

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 4 | 1 | 5 | **9** | **2** | 6 | 5 | 3 | 5 |

92 mod 11 = 4 equal to 4 -> spurious hit

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 4 | 1 | 5 | 9 | **2** | **6** | 5 | 3 | 5 |

26 mod 11 = 4 equal to 4 -> an exact match!!

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 4 | 1 | 5 | 9 | 2 | **6** | **5** | 3 | 5 |

65 mod 11 = 10 not equal to 4

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | **5** | **3** | 5 |

53 mod 11 = 9 not equal to 4

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | **3** | **5** |

35 mod 11 = 2 not equal to 4

As we can see, when a match is found, further testing is done to insure that a match has indeed been found.

*Q.4 b)*

**Branch and Bound Techniques:**
**Explanation need to be with respect to an example**

**FIFO BB:** Nodes are extracted from the list of live nodes in the same order as they are put into it. Children of E-node are inserted in a queue as shown below

Example Sum of Subsets State space tree generated using LIFO BB:

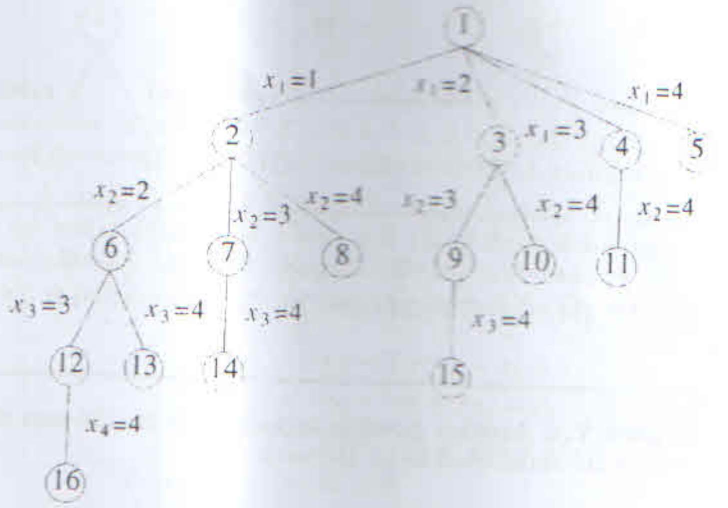



**Figure 7.3** A possible solution space organization for the sum of subsets problem. Nodes are numbered as in breadth-first search.

**LIFO BB:** Nodes are extracted from the list of live nodes in the same order as they are put into it. Children of E-node are inserted in a stack as shown below

Example Sum of Subsets State space tree generated using LIFO BB:

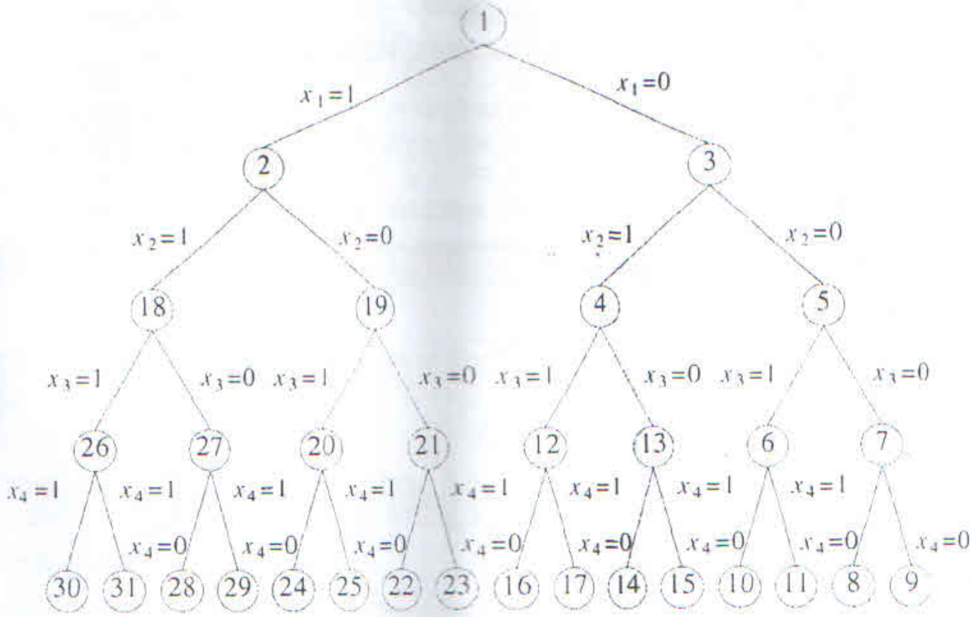



**Figure 7.4** Another possible organization for the sum of subsets problems. Nodes are numbered as in *D*-search.

**Least Cost BB:**
There is a cost or profit associates with each node. A min or max heap is used. The selection rule for the next E-node in FIFO or LIFO branch-and-bound is sometimes "blind". i.e. the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.

***Marks Distribution:***
State all three correct Branch and Bound Techniques? -------- **01 mks**
Explained any two techniques with the help of node generation in state space tree. ----- **02 mks for each technique**

*Q4 c)*

***Marks Distribution:***
Algorithm for Finite-Automata-Matcher---- 02
Transition function of string matching---- 03
Table ------ 02mks
State diagram---- 02mks
sequences of states it enters in for the given Text----- 01mk

The states will be $\{0, 1, 2, 3, 4, 5, 6\}$ and having a transition function given by

| state | $a$ | $b$ |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 2 | 0 |
| 2 | 2 | 3 |
| 3 | 4 | 0 |
| 4 | 2 | 5 |
| 5 | 1 | 0 |

The sequence of states for $T$ is $0, 1, 2, 2, 3, 4, 5, 1, 2, 3, 4, 2, 3, 4, 5, 1, 2, 3$, and so finds two occurrences of the pattern, one at $s = 1$ and another at $s = 9$.

*Q.5a)*

First, we will multiply the second and third inequalities by minus one to make it so that they are all $\leq$ inequalities. We will introduce the three new variables $x_4, x_5, x_6$, and perform the usual procedure for rewriting in slack form

$$x_4 = 7 - x_1 - x_2 + x_3$$
$$x_5 = -8 + 3x_1 - x_2$$
$$x_6 = -x_1 + 2x_2 + 2x_3$$
$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$

where we are sill trying to maximize $2x_1 - 6x_3$. The basic variables are $x_4, x_5, x_6$ and the nonbasic variables are $x_1, x_2, x_3$.

***Marks Distribution:***

Standard form and Slack Form
How to convert linear Program into standard and slack form -------- **02 mks**
Problem solved correctly as shown above -----------**03mks**
Stated the basic and non-basic variables ---------- **04mks**
----------- **01mk**

**OR**

By just transposing $A$, swapping $b$ and $c$, and switching the maximization to a minimization, wa want to minimize $20y_1 + 12y_2 + 16y_3$ subject to the constraints

$$y_1 + y_2 \geq 18$$
$$y_1 + y_3 \geq 12.5$$
$$y_1, y_2, y_3 \geq 0$$

***Marks Distribution:***

Duality Explained ------- 06mks
Solved Problem correctly -------- 04mks

| | |
|---|---|
| *Q5 b)* | ***Marks Distribution:***<br>Solved Correctly with all Table and calculation shown along with the general formula --------- **10mks**<br>Solved Correctly with all Table and calculation shown but no formula given--------- **08mks**<br>Solved Correctly with all values filled in the Iteration table table but no calculation steps shown--------- **04mks** |

S-b

# Simplex Method

n : max $Z = 2x_1 - x_2 + 2x_3 + 0S_1 + 0S_2 + 0S_3$

S.T.C. : $2x_1 + x_2 + S_1 = 10$

$x_1 + 2x_2 - 2x_3 + S_2 = 20$

$x_1 + 2x_3 + S_3 = 15$

$x_1, x_2, x_3, S_1, S_2, S_3 \geq 0$

## Initial Table

| $C_B$ | $c_j$ | 2 | -1 | 2 | 0 | 0 | 0 | Soln | Ratio |
|---|---|---|---|---|---|---|---|---|---|
| | B.V. | $x_1$ | $x_2$ | $x_3$ | $S_1$ | $S_2$ | $S_3$ | | |
| | $S_1$ | 2 | 1 | 0 | 1 | 0 | 0 | 10 | $10/2 = 5$ |
| | $S_2$ | 1 | 2 | -2 | 0 | 1 | 0 | 20 | $20/1 = 20$ |
| | $S_3$ | 1 | 0 | 2 | 0 | 0 | 1 | 15 | $15/1 = 15$ |
| | $Z_j$ | 0 | 0 | 0 | 0 | 0 | 0 | | |
| | $c_j - Z_j$ | 2 | -1 | 2 | 0 | 0 | 0 | | |

→ $Z_j = \sum_{i=1}^{n} (C_{B_i}) * (a_{ij})$

→ for max : $c_j - Z_j \leq 0$

→ for min : $c_j - Z_j \geq 0$

→ New value = Old Value − $\left[\dfrac{\text{Correct Key Row} * \text{Corr. Key Column}}{\text{Key Element}}\right]$

## Iteration - I

| $C_B$ | $C_j$ / B.V. | $x_1$ (2) | $x_2$ (-1) | $x_3$ (2) | $S_1$ (0) | $S_2$ (0) | $S_3$ (0) | Soln | Ratio |
|---|---|---|---|---|---|---|---|---|---|
| 2 | $x_1$ | 1 | 1/2 | 0 | 1/2 | 0 | 0 | 10/0 = ∞ | ∞ |
| 0 | $S_2$ | 0 | 3/2 | -2 | -1/2 | 1 | 0 | 15 | -7.5 |
| 0 | $S_3$ | 0 | -1/2 | 2 | -1/2 | 0 | 1 | 10 | 05 |
| | $Z_j$ | 2 | 1 | 0 | 1 | 0 | 0 | 10 | |
| | $C_j - Z_j$ | 0 | -2 | 2 | -1 | 0 | 0 | | |

$\Rightarrow 1 - \frac{1*2}{2} = 0$

$\Rightarrow 2 - \frac{1*1}{2} = 3/2$ - - - - -

## Iteration - II

| $C_B$ | $C_j$ / B.V. | $x_1$ (2) | $x_2$ (-1) | $x_3$ (2) | $S_1$ (0) | $S_2$ (0) | $S_3$ (0) | Soln | Ratio |
|---|---|---|---|---|---|---|---|---|---|
| 2 | $x_1$ | 1 | 1/2 | 0 | 1/2 | 0 | 0 | 5 | |
| 0 | $S_2$ | 0 | 1 | 0 | -1 | 1 | +1 | 25 | |
| 2 | $x_3$ | 0 | -1/4 | 1 | -1/4 | 0 | 1/2 | 05 | |
| | $Z_j$ | 2 | 1/2 | 2 | 1/2 | 0 | 1 | 20 | |
| | $C_j - Z_j$ | 0 | -3/2 | 0 | -1/2 | 0 | -1 | | |

Optimal solution $Z_j = 20$

$x_1 = 2$

$S_2 = 0$

$X_3 = 2$