# Design and Analysis of Algorithm

- BY VIDHITA BAIS (A1 B1 12)

CONTENTS:

# LONGEST INCREASING SUBSEQUENCE

## PROBLEM EXPLAINATION:-

Problem: Given an array of size n , we have to find out the length of largest increasing subsequence and print the subsequence,

The longest increasing subsequence is described as a subsequence of an array where:

- All elements of the subsequence are in increasing order.
- This subsequence itself is of the longest length possible.
- The subsequence not necessary be contiguous.

Suppose, we have arr = {1,5,2,7,4}
So, here subsets are {1},{1,5},{1,5,2},{5,2,7},…
But subsequence are {1},{1,7},{1,2,7},{5,7,4},{1,5,7,4},…

Example:-

arr = [3, 10, 2, 1, 20, 4, 6, 7, 8, 5, 9, 11]

i,j : indexing variables where for every i, j starts from 0

arr:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

dp:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

prev:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

We are given the array containing elements, we are using dp (array) which will store the max length till the ith element, and prev (array) which will help in printing the subsequence.
So, initially dp[i] will be 1 for i (1 to n) because each element in the array is occurring once so it's LIS will be 1 and prev[i]=-1 as there is no subequence initially.

| | j | i | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | ~~1~~ 2 | 1 | 1 | 1 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | ~~-1~~ 0 | -1 | -1 | -1 | -1 | -1 | -1 |

If arr[1] > arr[0] && dp[1] < dp[0]+1
(10 > 3) && (1 < 2)
True: dp[1]=dp[0]+1
dp[1]=1+1=2
prev[1]=0

Check if (arr[i] > arr[j] && dp[i] < dp[j] +1)
True : dp[i]=dp[j]+1
prev[i]=j
False: do nothing

j i

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 |

Check if (arr[i] > arr[j] && dp[i] < dp[j] +1)
arr[2] > arr[0] : (2 >3)
False j++

| | | j | i | | | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 |

Check if (arr[i] > arr[j] && dp[i] < dp[j] +1)
arr[2] > arr[0] : (2 >3)
False: j++

arr[2] > arr[1] : (2 >10)
False: i++

j i

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 |

Check if (arr[i] > arr[j] && dp[i] < dp[j] +1)
arr[3] > arr[0] : (1 >3) && dp[3] < dp[0] +1: (1 < 2)
False: no changes
j++

| | | j | | i | | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 |

Check if (arr[i] > arr[j] && dp[i] < dp[j] +1)

1. arr[3] > arr[0] : (1 >3) && dp[3] < dp[0] +1: (1 < 2)
   False: no changes
   j++
2. arr[3] > arr[1] : (1 >10) && dp[3] < dp[1] +1: ( < 2)
   False: j++

| | | | j | i | | | |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 |

Check if (arr[i] > arr[j] && dp[i] < dp[j] +1)

1. arr[3] > arr[0] : (1 >3) && dp[3] < dp[0] +1: (1 < 2)
   False: j++
2. arr[3] > arr[1] : (1 >10) && dp[3] < dp[1] +1: (2 < 3)
   False: j++
3. arr[3] > arr[2] : (1 >2) && dp[3] < dp[2] +1: (1 < 2)
   False:
   next iteration

| | j | | | | i | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 |

1. arr[4] > arr[0] && dp[4] < dp[0] +1
   (20>3) && (1 < 2)
   True: dp[4]=2 prev[4]=0

| | | j | | | i | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 2 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 0 | -1 | -1 | -1 |

1. arr[4] > arr[0] && dp[4] < dp[0] +1
   (20>3) && (1 < 2)
   True: dp[4]=2 prev[4]=0

2. arr[4] > arr[1] && dp[4] < dp[1] +1
   (20>10 && (2 < 3)
   True: dp[4]=3 prev[4]=1

| | | | j | | i | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | -1 | -1 | -1 |

1. arr[4] > arr[0] && dp[4] < dp[0] +1
   (20>3) && (1 < 2)
   True: dp[4]=2 prev[4]=0

2. arr[4] > arr[1] && dp[4] < dp[1] +1
   (20>10 && (2 < 3)
   True: dp[4]=3 prev[4]=1

3. arr[4] > arr[2] && dp[4] < dp[2] +1
   (20>2) && (3 < 2)
   False: no changes

| | | | | j | i | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | -1 | -1 | -1 |

1. arr[4] > arr[0] && dp[4] < dp[0] +1
   (20>3) && (1 < 2)
   True: dp[4]=2 prev[4]=0

2. arr[4] > arr[1] && dp[4] < dp[1] +1
   (20>10 && (2 < 3)
   True: dp[4]=3 prev[4]=1

3. arr[4] > arr[2] && dp[4] < dp[2] +1
   (20>2) && (3 < 2)
   False: no changes

4. arr[4] > arr[3] && dp[4] < dp[3] +1
   (20>1) && (3 < 3)
   False: no changes

j i

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 1 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | -1 | -1 | -1 |

1. arr[5] > arr[0] && dp[5] < dp[0] +1
   (4>3) && (1 < 2)
   True: dp[5]=2 prev[5]=0

| | j | | | | i | | |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | -1 | -1 |

1. arr[5] > arr[0] && dp[4] < dp[0] +1
   (4>3) && (1 < 2)
   True: dp[5]=2 prev[5]=0

2. arr[5] > arr[1] && dp[5] < dp[1] +1
   (4>10) && (2 < 3)
   False: no changes

| | | | j | | | i | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | -1 | -1 |

1. arr[5] > arr[0] && dp[4] < dp[0] +1
   (4>3) && (1 < 2)
   True: dp[5]=2 prev[5]=0

2. arr[5] > arr[1] && dp[5] < dp[1] +1
   (4>10) && (2 < 3)
   False: no changes

3. arr[5] > arr[2] && dp[5] < dp[2] +1
   (4>2) && (2 < 2)
   False

| | | | | j | | i | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | -1 | -1 |

1. arr[5] > arr[0] && dp[4] < dp[0] +1
   (4>3) && (1 < 2)
   True: dp[5]=2 prev[5]=0

2. arr[5] > arr[1] && dp[5] < dp[1] +1
   (4>10) && (2 < 3)
   False: no changes

3. arr[5] > arr[2] && dp[5] < dp[2] +1
   (4>2) && (2 < 2)
   False

4. arr[5] > arr[3] && dp[5] < dp[3] +1
   (4>1 && (2 < 2)
   False

| | | | | | j | i | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | -1 | -1 |

1. arr[5] > arr[0] && dp[4] < dp[0] +1
   (4>3) && (1 < 2)
   True: dp[5]=2 prev[5]=0

2. arr[5] > arr[1] && dp[5] < dp[1] +1
   (4>10) && (2 < 3)
   False: no changes

3. arr[5] > arr[2] && dp[5] < dp[2] +1
   (4>2) && (2 < 2)
   False

4. arr[5] > arr[3] && dp[5] < dp[3] +1
   (4>1 && (2 < 2)
   False

5. arr[5] > arr[3] && dp[5] < dp[4] +1
   (4>20) && (2 < 4)
   False

j i

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 1 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | -1 | -1 |

1. arr[6] > arr[0] && dp[6] < dp[0] +1
   (6>3) && (1 < 2)
   True: dp[6]=2 prev[6]=0

j i

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 2 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 0 | -1 |

1. arr[6] > arr[0] && dp[6] < dp[0] +1
   (6>3) && (1 < 2)
   True: dp[6]=2 prev[6]=0

2. 1. arr[6] > arr[1] && dp[6] < dp[1] +1
   (6>10) && (2 < 3)
   False:

| | | | j | | | | i | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 2 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 0 | -1 |

1. arr[6] > arr[0] && dp[6] < dp[0] +1
   (6>3) && (1 < 2)
   True: dp[6]=2 prev[6]=0

2. 1. arr[6] > arr[1] && dp[6] < dp[1] +1
   (6>10) && (2 < 3)
   False:

3. arr[6] > arr[2] && dp[6] < dp[2] +1
   (6>2) && (2 < 2)
   False

j i

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 2 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 0 | -1 |

1. arr[6] > arr[0] && dp[6] < dp[0] +1
   (6>3) && (1 < 2)
   True: dp[6]=2 prev[6]=0

2. 1. arr[6] > arr[1] && dp[6] < dp[1] +1
   (6>10) && (2 < 3)
   False:

3. arr[6] > arr[2] && dp[6] < dp[2] +1
   (6>2) && (2 < 2)
   False

4. arr[6] > arr[3] && dp[6] < dp[3] +1
   (6>1) && (2 < 2)
   False

| | | | | j | | i | |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 2 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 0 | -1 |

1. arr[6] > arr[0] && dp[6] < dp[0] +1
   (6>3) && (1 < 2)
   True: dp[6]=2 prev[6]=0

2. 1. arr[6] > arr[1] && dp[6] < dp[1] +1
   (6>10) && (2 < 3)
   False:

3. arr[6] > arr[2] && dp[6] < dp[2] +1
   (6>2) && (2 < 2)
   False

4. arr[6] > arr[3] && dp[6] < dp[3] +1
   (6>1) && (2 < 2)
   False

5. arr[6] > arr[4] && dp[6] < dp[4] +1
   (6>20) && (3 < 4)
   False

| | | | | | | j | i |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 3 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 5 | -1 |

1. arr[6] > arr[0] && dp[6] < dp[0] +1
   (6>3) && (1 < 2)
   True: dp[6]=2 prev[6]=0

2. 1. arr[6] > arr[1] && dp[6] < dp[1] +1
   (6>10) && (2 < 3)
   False:

3. arr[6] > arr[2] && dp[6] < dp[2] +1
   (6>2) && (2 < 2)
   False

4. arr[6] > arr[3] && dp[6] < dp[3] +1
   (6>1) && (2 < 3)
   True: dp[6]=3 prev[6]=3

5. arr[6] > arr[4] && dp[6] < dp[4] +1
   (6>20) && (2 < 4)
   False

6. arr[6] > arr[5] && dp[6] < dp[5] +1
   (6>4) && (2 < 3)
   True: dp[6]=3 prev[6]=5

j i

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 3 | 1 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 5 | -1 |

1. arr[7] > arr[0] && dp[7] < dp[0] +1
   (6>3) && (1 < 2)
   True: dp[7]=2 prev[7]=0

| | j | | | | | | i |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 3 | 2 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 5 | 0 |

1. arr[7] > arr[0] && dp[7] < dp[0] +1
   (7>3) && (1 < 2)
   True: dp[7]=2 prev[7]=0

2. arr[7] > arr[1] && dp[7] < dp[1] +1
   (7>10) && (2 < 3)
   False

| | | | j | | | | i |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 3 | 2 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 5 | 0 |

1. arr[7] > arr[0] && dp[7] < dp[0] +1
   (7>3) && (1 < 2)
   True: dp[7]=2 prev[7]=0

2. arr[7] > arr[1] && dp[7] < dp[1] +1
   (7>10) && (2 < 3)
   False

3. arr[7] > arr[2] && dp[7] < dp[2] +1
   (7>2) && (2 < 2)
   False

| | | | | j | | | i |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 3 | 2 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 5 | 0 |

1. arr[7] > arr[0] && dp[7] < dp[0] +1
   (7>3) && (1 < 2)
   True: dp[7]=2 prev[7]=0

2. arr[7] > arr[1] && dp[7] < dp[1] +1
   (7>10) && (2 < 3)
   False

3. arr[7] > arr[2] && dp[7] < dp[2] +1
   (7>2) && (2 < 2)
   False

4. arr[7] > arr[3] && dp[7] < dp[3] +1
   (7>1) && (2 < 2)
   False

| | | | | j | | | i |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 3 | 2 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 5 | 0 |

1. arr[7] > arr[0] && dp[7] < dp[0] +1
   (7>3) && (1 < 2)
   True: dp[7]=2 prev[7]=0

2. arr[7] > arr[1] && dp[7] < dp[1] +1
   (7>10) && (2 < 3)
   False

3. arr[7] > arr[2] && dp[7] < dp[2] +1
   (7>2) && (2 < 2)
   False

4. arr[7] > arr[3] && dp[7] < dp[3] +1
   (7>1) && (2 <2)
   False

5. arr[7] > arr[4] && dp[7] < dp[4] +1
   (7>20) && (2 < 4)
   False

| | | | | | j | | i |
|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 3 | 2 |

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 5 | 0 |

1. arr[7] > arr[0] && dp[7] < dp[0] +1
   (7>3) && (1 < 2)
   True: dp[7]=2 prev[7]=0

2. arr[7] > arr[1] && dp[7] < dp[1] +1
   (7>10) && (2 < 3)
   False

3. arr[7] > arr[2] && dp[7] < dp[2] +1
   (7>2) && (2 < 2)
   False

4. arr[7] > arr[3] && dp[7] < dp[3] +1
   (7>1) && (2 <2)
   False

5. arr[7] > arr[4] && dp[7] < dp[4] +1
   (7>20) && (2 < 4)
   False

6. arr[7] > arr[5] && dp[7] < dp[5] +1
   (7>4) && (2 < 3)
   True: dp[7]=3 prev[7]=5

| | | | | | | | j | i |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| arr: | 3 | 10 | 2 | 1 | 20 | 4 | 6 | 7 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| dp: | 1 | 2 | 1 | 1 | 3 | 2 | 3 | 4 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| prev: | -1 | 0 | -1 | -1 | 1 | 0 | 5 | 6 |

1. arr[7] > arr[0] && dp[7] < dp[0] +1
   (7>3) && (1 < 2)
   True: dp[7]=2 prev[7]=0

2. arr[7] > arr[1] && dp[7] < dp[1] +1
   (7>10) && (2 < 3)
   False

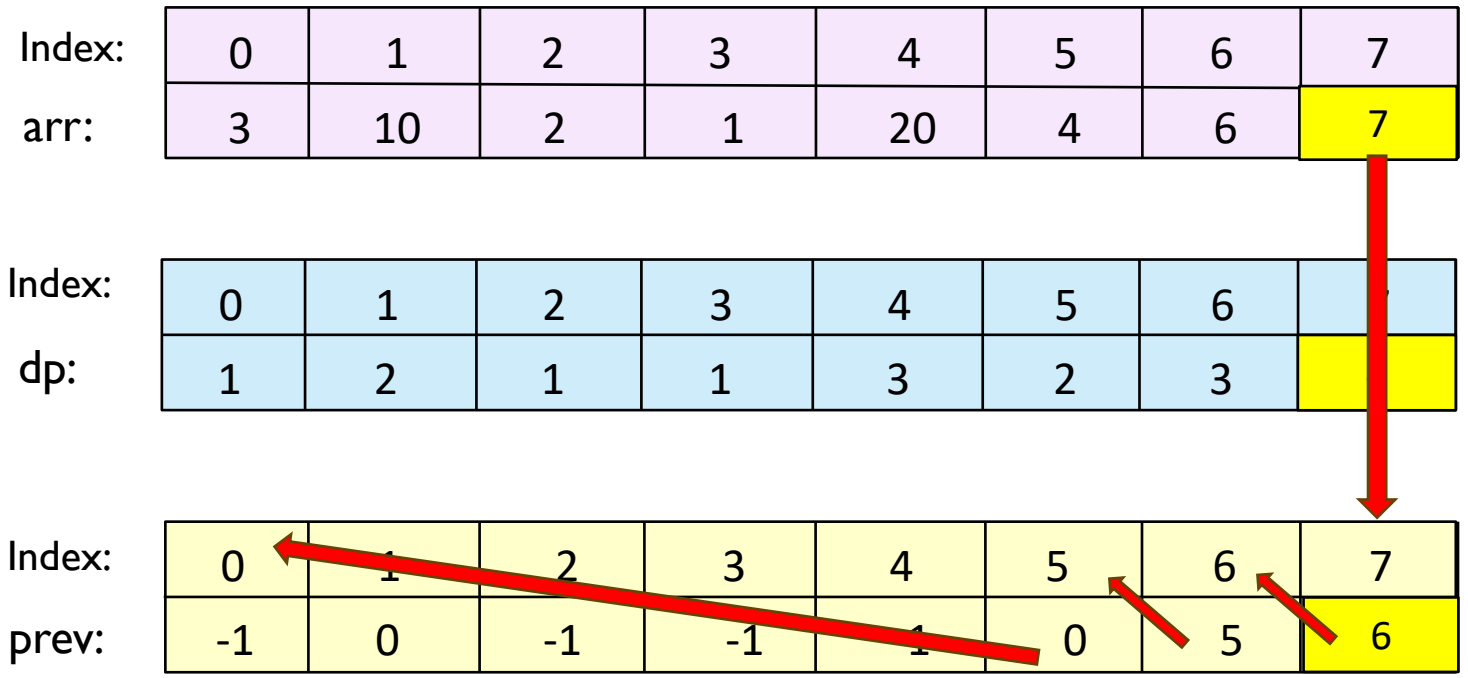
3. arr[7] > arr[2] && dp[7] < dp[2] +1
   (7>2) && (2 < 2)
   False

4. arr[7] > arr[3] && dp[7] < dp[3] +1
   (7>1) && (2 <2)
   False

5. arr[7] > arr[4] && dp[7] < dp[4] +1
   (7>20) && (2 < 4)
   False

6. arr[7] > arr[5] && dp[7] < dp[5] +1
   (7>4) && (2 < 3)
   True: dp[7]=3 prev[7]=5

7. arr[7] > arr[6] && dp[7] < dp[6] +1
   (7>6) && (3 < 4)
   True: dp[7]=4 prev[7]=6




Hence, LIS = dp[7]=4

| 0 | I | 2 | 3 |
|---|---|---|---|
| 3 | 4 | 6 | 7 |

CODE:-

```c
#include<stdio.h>
#include<stdlib.h>
void lis(int arr[20],int size){
    int prev[size];
    int dp[size];
    int len,i,j;
    for(i=0;i<size;i++){
        dp[i]=1;
        prev[i]=-1;
    }
    for(i=1 ;i<size;i++){
        for(j=0;j<i;j++){
            if(arr[j]< arr[i] && dp[i]<dp[j]+1){
                dp[i]=dp[j]+1;
                prev[i]=j;
            }
        }
    }
    int maxL=dp[0];
    int max_idx=0;
    for(i=0;i<size;i++){
        if(maxL<dp[i]){
            maxL=dp[i];
            max_idx=i;

        }
    }
    printf("Longest increasing subsequence: %d\n",maxL);
    printf("Subsequence: ");
    int k=max_idx;
    i=maxL-1;
    int seq[size];
    while(k!=-1){
        seq[i]=arr[k];
        i--;
        k=prev[k];
    }
    for(j=0;j<maxL;j++){
        printf("%d ",seq[j]);
    }
}
int main(){
    int arr[20];
    int size,i;
    printf("Enter the size of the array:\n");
    scanf("%d",&size);
    printf("Enter the array:\n");
    for(i=0;i<size;i++){
        scanf("%d",&arr[i]);
    }
    lis(arr,size);
    return 0;
}
```

EXECUTION:-

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\LIS> gcc lis.c
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\LIS> ./a.exe
Enter the size of the array:
12
Enter the array:
3
10
2
1
20
4
6
7
8
5
9
11
Longest increasing subsequence: 7
Subsequence: 3 4 6 7 8 9 11
```

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\LIS> ./a.exe
Enter the size of the array:
10
Enter the array:
1
2
3
4
5
6
7
8
9
10
Longest increasing subsequence: 10
Subsequence: 1 2 3 4 5 6 7 8 9 10
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\LIS>
```

EXECUTION:-

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\LIS> ./a.exe
Enter the size of the array:
10
Enter the array:
10
9
8
7
6
5
4
3
2
1
Longest increasing subsequence: 1
Subsequence: 10
```

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\LIS> ./a.exe
Enter the size of the array:
10
Enter the array:
0
8
4
12
2
10
6
14
1
9
Longest increasing subsequence: 4
Subsequence: 0 8 12 14
```

**TIME COMPLEXITIES:-**

- We maintain a dp[i] array where dp[i] stores the length of the longest increasing subsequence ending at index i.

- For every element i, we check all previous elements j where $j < i \rightarrow O(n^2)$.
  Time Complexity = $O(n^2)$

# LONGEST DIVISIBLE SUBSET

**PROBLEM EXPLAINATION:-**

Problem:
Given an array of distinct positive integers, We have to find the largest subset such that every pair of elements in the subset must be divisible by each other.
Return any one of the largest divisible subsets.

## Difference between Subsequence and Subset:-

**Subset:-**

- A subset is any selection of elements from a set, without considering their order.
- The elements need not be contiguous.
- Any combination of elements can form a subset.
- The size of subsets can be from 0 to n, where n is the number of elements in the set.
- Order is not important.

**Subsequence:-**

- A subsequence is a sequence derived from the original array by deleting zero or more elements, while keeping the remaining elements in their original order.
- The elements must appear in the same relative order as in the original array, but they do **not** need to be contiguous
- Order is important.

Example:
For arr = [1, 2, 3],

subsets :
[1], [2], [3], [1, 2], [2,1],[2, 3], [3, 2],[1, 2, 3],…..

Subsequence :
[1], [2], [3], [1,3],[2,3], [1,2,3],[1,2]

Let's move to our problem:

Example:
arr={5,15,3,30}

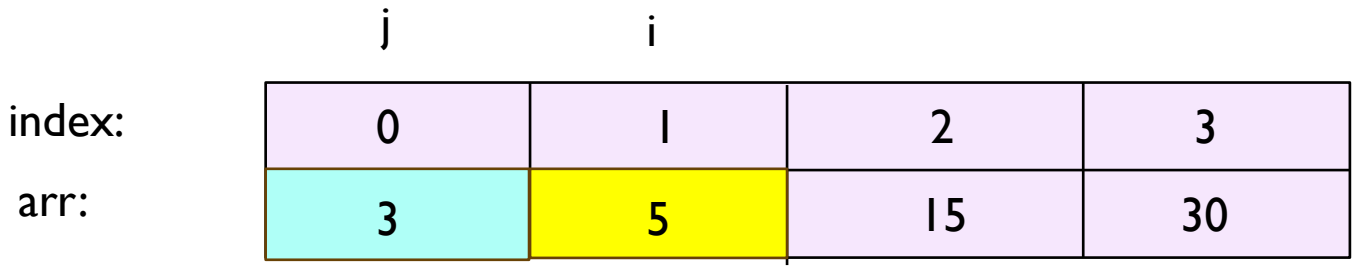
For finding out longest divisible subset we have to first sort the array and then perform the operations on it.

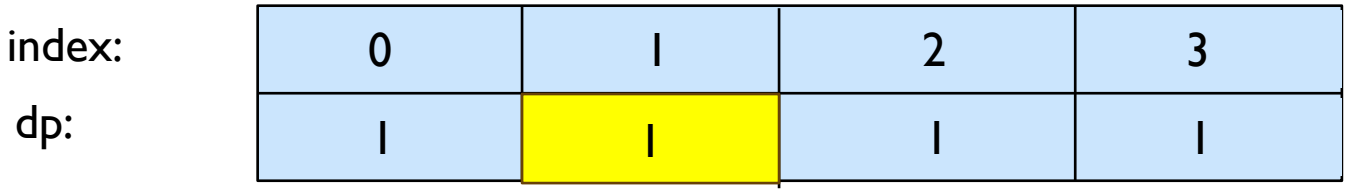
Condition:

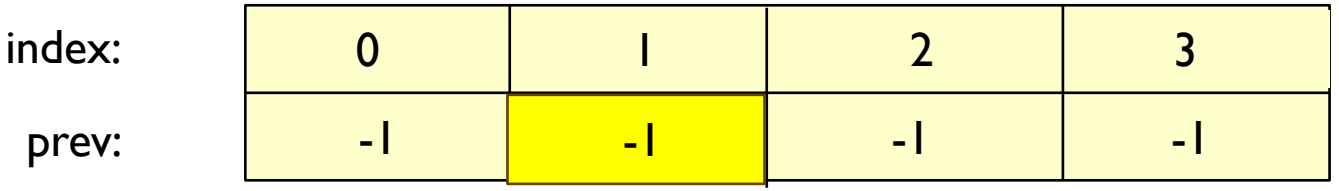
If( (arr[i] % arr[j] ==0) && dp[i] <dp[j] +1)

True: dp[i]=dp[j]+1
prev[i]=j
False: no changes (do nothing)

Example:-

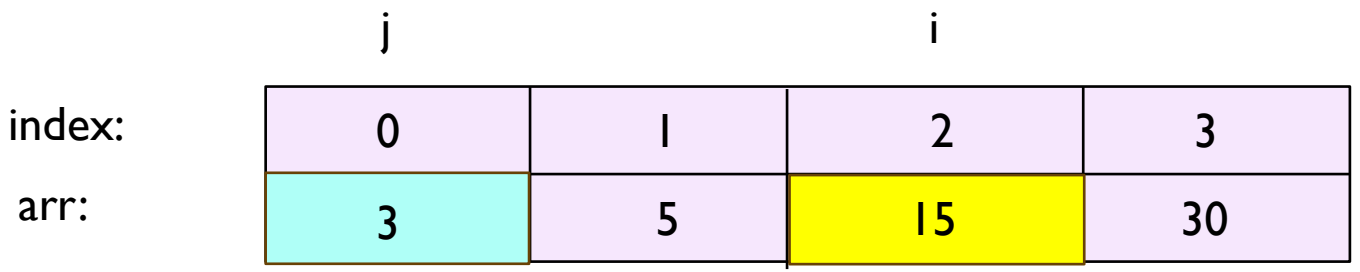


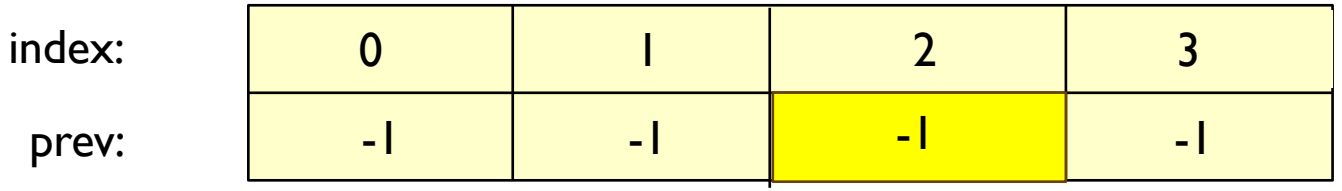




If( (arr[1] % arr[0] ==0) && dp[1] <dp[0] +1)
False: no changes (do nothing)

Example:-

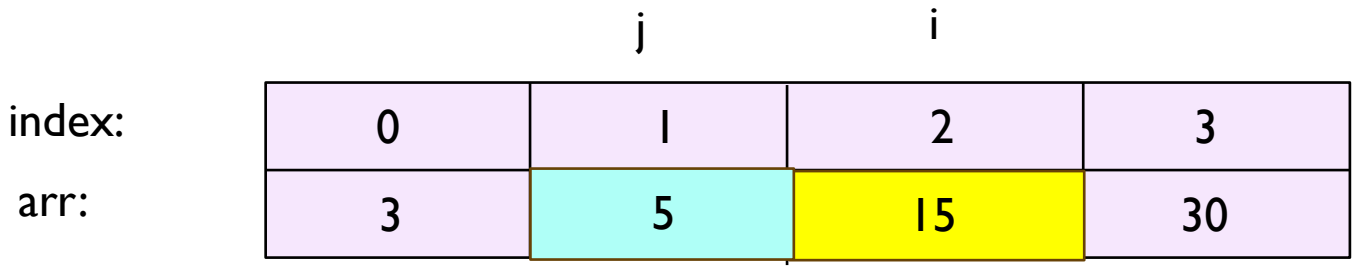




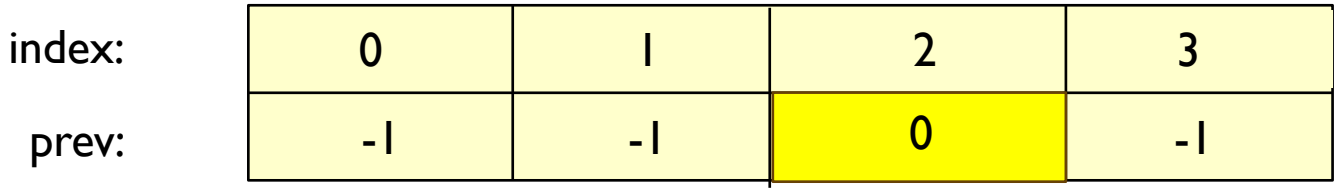



If( (arr[2] % arr[0] ==0) && dp[2] <dp[0] +1)
(15%5==0) && (1<2)
True: dp[2]=2  prev[2]=0

Example:-








If( (arr[2] % arr[0] ==0) && dp[2] <dp[0] +1)
(15%3==0) && (1<2)
True: dp[2]=2  prev[2]=0

If( (arr[2] % arr[1] ==0) && dp[2] <dp[1] +1)
(15%5==0) && (2<2)
False : do nothing

Example:-

j i

| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| arr: | 3 | 5 | 15 | 30 |

| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| dp: | 1 | 1 | 2 | 1 |

| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| prev: | -1 | -1 | 0 | -1 |

If( (arr[3] % arr[0] ==0) && dp[3] <dp[0] +1)
(30%3==0) && (1<2)
True: dp[3]=2  prev[3]=0

Example:-

j i

| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| arr: | 3 | 5 | 15 | 30 |

| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| dp: | 1 | 1 | 2 | 2 |

| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| prev: | -1 | -1 | 0 | 0 |

If( (arr[3] % arr[0] ==0) && dp[3] <dp[0] +1)
(30%3==0) && (1<2)
True: dp[3]=2 prev[3]=0

If( (arr[3] % arr[1] ==0) && dp[3] <dp[1] +1)
(30%5==0) && (2<2)
False

Example:-

j i

| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| arr: | 3 | 5 | 15 | 30 |

| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| dp: | 1 | 1 | 2 | 2 |

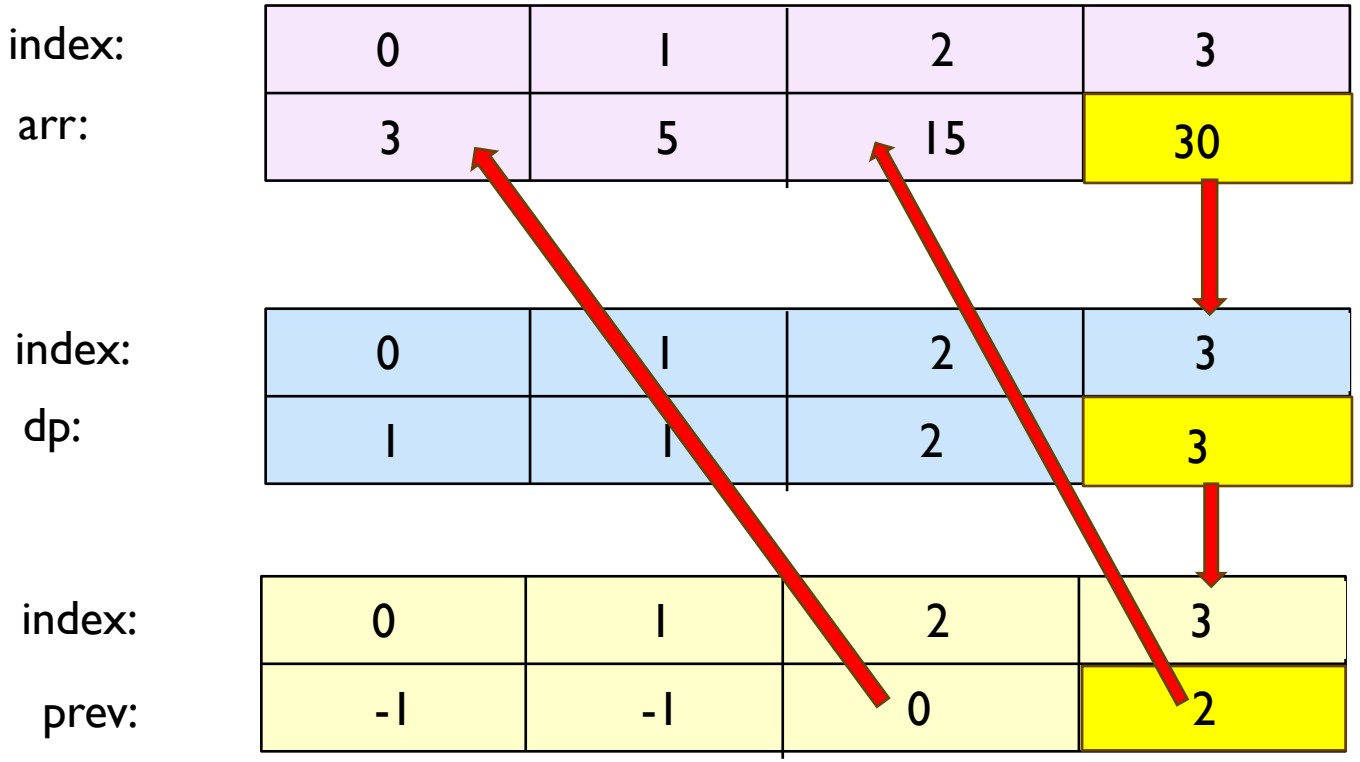
| index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| prev: | -1 | -1 | 0 | 0 |

If( (arr[3] % arr[0] ==0) && dp[3] <dp[0] +1)
(30%3==0) && (1<2)
True: dp[3]=2 prev[3]=0

If( (arr[3] % arr[1] ==0) && dp[3] <dp[1] +1)
(30%5==0) && (2<2)
False

If( (arr[3] % arr[2] ==0) && dp[3] <dp[2] +1)
(30%15==0) && (2<3)
True: dp[3]=3 prev[3]=2

Example:-




Hence, LDS=3 means length of longest divisible subset is 3

| 0 | 1 | 2 |
|---|---|---|
| 3 | 15 | 30 |

## CODE:-

C: > Users > Vidhita > OneDrive > Desktop > DAA TA2 > Longest Divisible Subset >

```c
#include<stdio.h>
#include<stdlib.h>
void sort(int arr[20],int size){
    for(int i=0;i<size-1;i++){
        for(int j=0;j<size-1-i;j++){
            if(arr[j]>arr[j+1]){
                int temp= arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
}
void lds(int arr[20],int size){
    sort(arr,size);
    int i,j;
    int dp[size];
    int prev[size];
    for(i=0;i<size;i++){
        dp[i]=1;
        prev[i]=-1;
    }
    for(i=1;i<size;i++){
        for(j=0;j<i;j++){
            if((arr[i] % arr[j]==0) && dp[i]<dp[j]+1){
                dp[i]=dp[j]+1;
                prev[i]=j;
            }
        }
    }
    int maxL=dp[0];
    int max_idx=0;
    for(i=0;i<size;i++){
        if(maxL<dp[i]){
            maxL=dp[i];
            max_idx=i;

        }
    }
    printf("Longest Divisible Subset Length: %d\n",maxL);
    printf("Longest Divisible Subset: ");
    int k=max_idx;
    i=maxL-1;
    int seq[size];
    while(k!=-1){
        seq[i]=arr[k];
        i--;
        k=prev[k];
    }
    for(j=0;j<maxL;j++){
        printf("%d ",seq[j]);
    }
}

int main(){
    int arr[20];
    int size,i,j;
    printf("Enter the size of array:\n");
    scanf("%d",&size);
    printf("Enter the array:\n");
    for(i=0;i<size;i++){
        scanf("%d",&arr[i]);
    }
    lds(arr,size);
    return 0;
}
```

Execution:-

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\Longest Divisible Subset> gcc lds.c
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\Longest Divisible Subset> ./a.exe
Enter the size of array:
6
Enter the array:
1
2
3
4
9
8
Longest Divisible Subset Length: 4
Longest Divisible Subset: 1 2 4 8
```

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\Longest Divisible Subset> ./a.exe
Enter the size of array:
5
Enter the array:
2
3
5
7
11
Longest Divisible Subset Length: 1
Longest Divisible Subset: 2
```

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\Longest Divisible Subset> ./a.exe
Enter the size of array:
5
Enter the array:
2
3
5
7
11
Longest Divisible Subset Length: 1
Longest Divisible Subset: 2
```

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\Longest Divisible Subset> gcc lds.c
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\Longest Divisible Subset> ./a.exe
Enter the size of array:
4
Enter the array:
3
5
15
30
Longest Divisible Subset Length: 3
Longest Divisible Subset: 3 15 30
```

## Time Complexity:-

1. Sorting the array
- Sorting n elements → O(n log n)

2. DP computation
- For each element i from 1 to n-1, we check all earlier elements → nested loop
- $O(n^2)$

3. Reconstructing the subset
- backward→ at most `n` steps → `O(n)`

Total Time Complexity:

$O(n \log n) + O(n^2) + O(n)$

Since $O(n^2)$ dominates, the overall time complexity is: $O(n^2)$

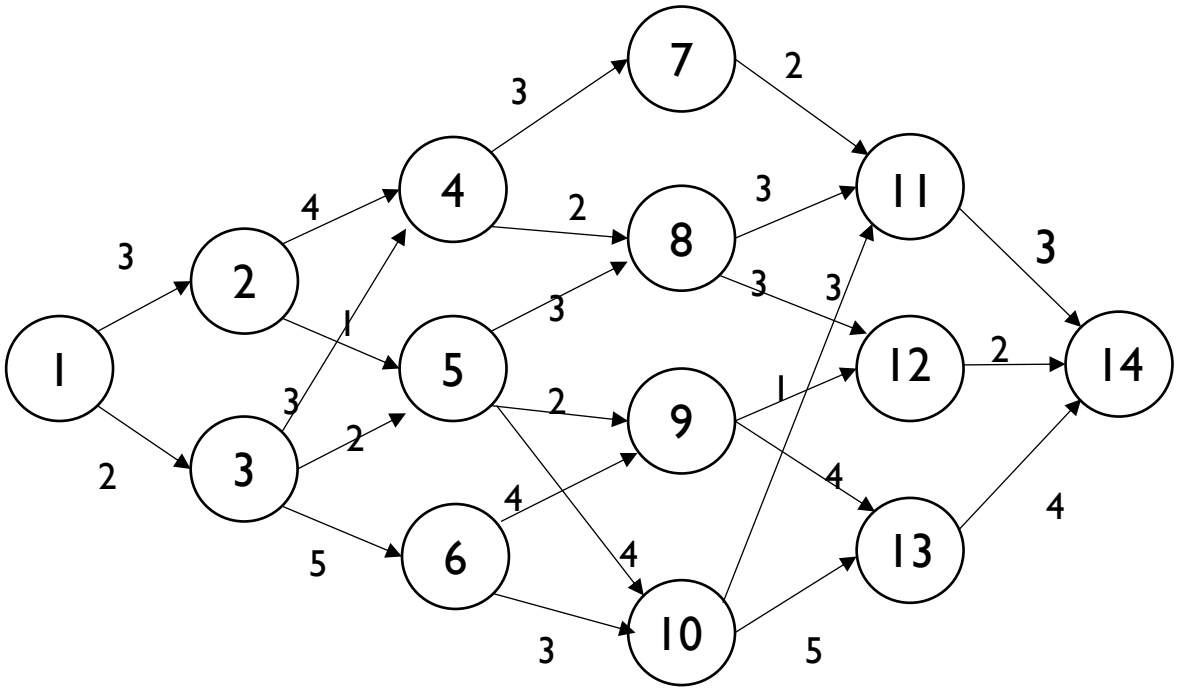
# MULTISTAGE GRAPH

**PROBLEM EXPLAINATION:-**

Given is a directed acyclic graph where nodes are divided into stages.
We start from source node , move through intermediate nodes in stages, and end at a destination node.
Each edge has a cost , and we want the shortest path from source to destination

Formula:-
bcost[i,j] = min[ bcost[i-1,l] + cost[l,j] ]
Where,
i :- stage
j :- vertex on stage i(destination)
l :- vertex through which we go to dest

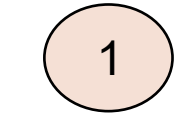
STAGES:- S1 S2 S3 S4 S5 S6




Stages:-
S1 : [ 1 ]
S2 : [ 2, 3 ]
S3 : [ 4, 5, 6 ]
S4 : [ 7, 8, 9, 10 ]
S5 : [ 11, 12, 13 ]
S6 : [ 14 ]

There are total 14 nodes and 6 stages.
Source: 1 Destination: 14

Stage 1: bcost[1,1]=0



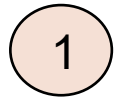

bcost:

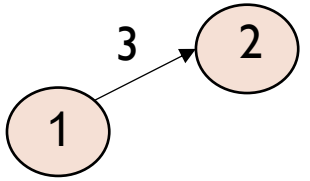
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | |

l:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | |

Stage 1: bcost[1,1]=0




Stage 2: bcost[2,2]=min[ bcost[1,1]+cost[1,2] ]=0+3=3

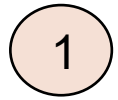
bcost:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | | | | | | | | | | | | |

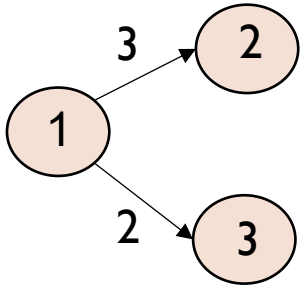
l:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | | | | | | | | | | | | |

Stage 1: bcost[1,1]=0




Stage 2: bcost[2,2]=min[ bcost[1,1]+cost[1,2] ]=0+3=3
bcost[2,3]=min[ bcost[1,1]+cost[1,3] ]=0+2=2




bcost:

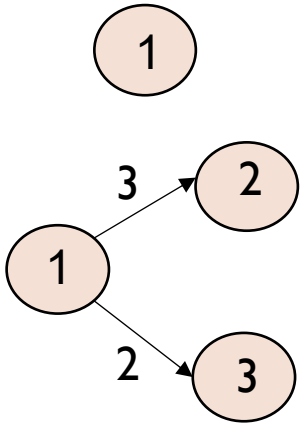
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | | | | | | | | | | | |

l:

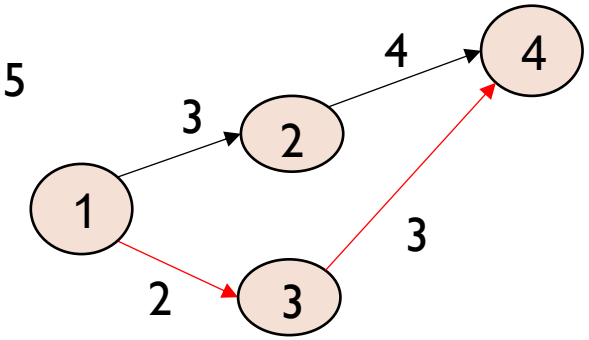
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | | | | | | | | | | | |

Stage 1: bcost[1,1]=0

Stage 2: bcost[2,2]=min[ bcost[1,1]+cost[1,2] ]=0+3=3
bcost[2,3]=min[ bcost[1,1]+cost[1,3] ]=0+2=2




Stage 3: bcost[3,4]=min[ bcost[2,2]+cost[2,4]=3+4=7
l=2,3 bcost[2,3]+cost[3,4] ]=2+3=5
Hence, bcost[3,4]=5, l=3




bcost:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | | | | | | | | | | |

l:

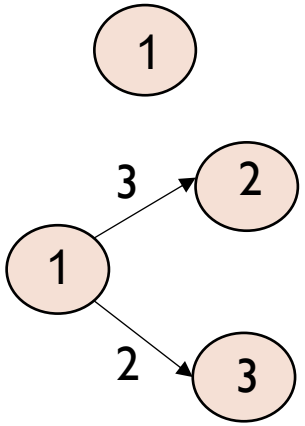
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | | | | | | | | | | |

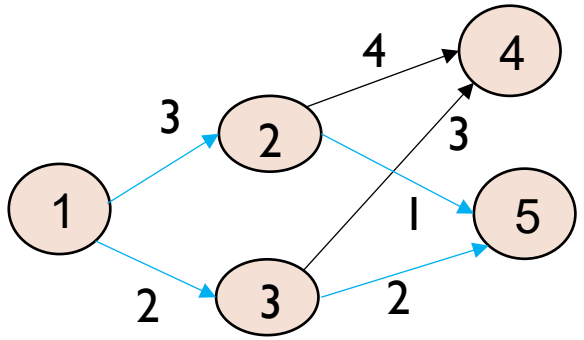
Stage 1: bcost[1,1]=0

Stage 2: bcost[2,2]=min[ bcost[1,1]+cost[1,2] ]=0+3=3
bcost[2,3]=min[ bcost[1,1]+cost[1,3] ]=0+2=2

Stage 3: bcost[3,4]=min[ bcost[2,2]+cost[2,4]=3+4=7
l=2,3 bcost[2,3]+cost[3,4] ]=2+3=5
Hence, min=5, l=3

bcost[3,5]=min[ bcost[2,2]+cost[2,5]=3+1=4
l=2,3 bcost[2,3]+cost[3,5] ]=2+2=4
Hence, min=4, l=2,3




bcost:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | | | | | | | | | |

l:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| l | l | l | 3 | 2,3 | | | | | | | | | |

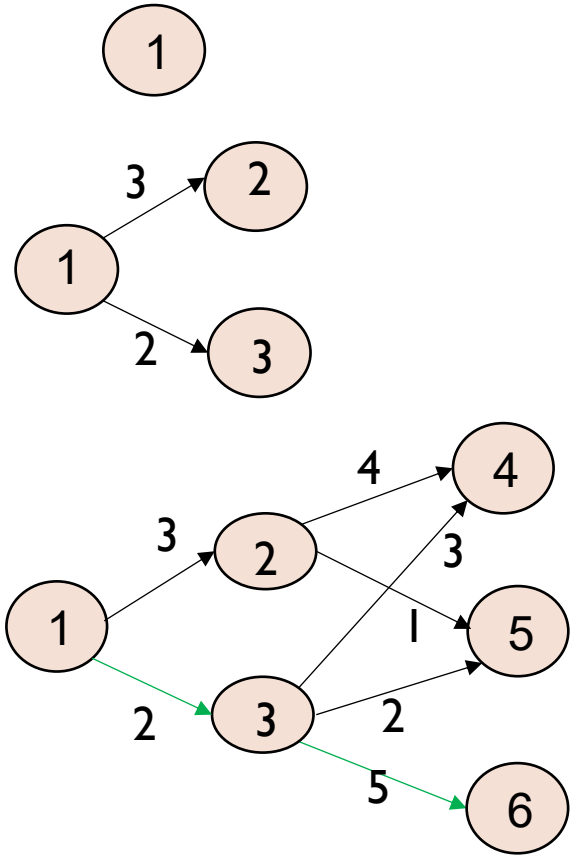
Stage 1: bcost[1,1]=0

Stage 2: bcost[2,2]=min[ bcost[1,1]+cost[1,2] ]=0+3=3
bcost[2,3]=min[ bcost[1,1]+cost[1,3] ]=0+2=2

Stage 3: bcost[3,4]=min[ bcost[2,2]+cost[2,4]=3+4=7
l=2,3 bcost[2,3]+cost[3,4] ]=2+3=5
Hence, bcost[3,4]=5, l=3

bcost[3,5]=min[ bcost[2,2]+cost[2,5]=3+1=4
l=2,3 bcost[2,3]+cost[3,5] ]=2+2=4
Hence, bcost[3,5]=4, l=2,3

bcost[3,6]=min[ bcost[2,3]+cost[3,6] ]=2+5=7
Hence, bcost[3,6]=7, l=3




bcost:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 4 | | | | | | | | |

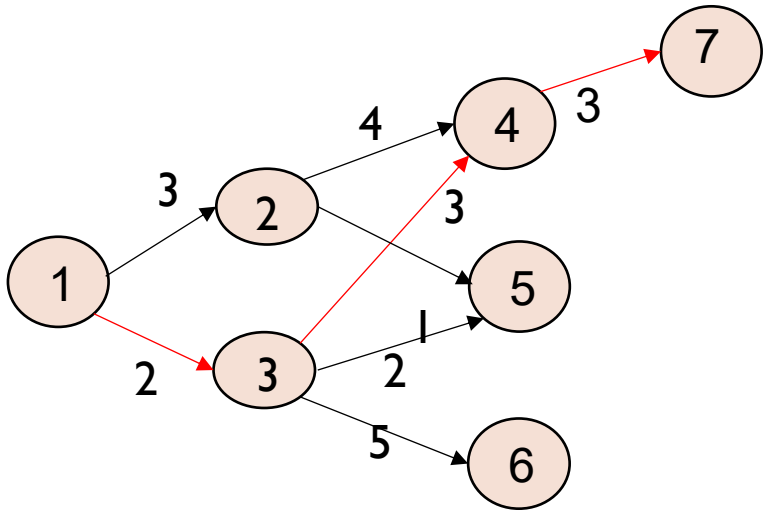
l:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | 2,3 | 3 | | | | | | | | |

Stage 4:

bcost[4,7]=min[ bcost[3,4]+cost[4,7]=5+3=8 ]
l=4

Hence, bcost[4,7]=8, l=4




bcost:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 4 | 8 | | | | | | | |

l:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | 2,3 | 3 | 4 | | | | | | | |

Stage 4: bcost[4,7]=min[ bcost[3,4]+cost[4,7]=5+3=8 ]

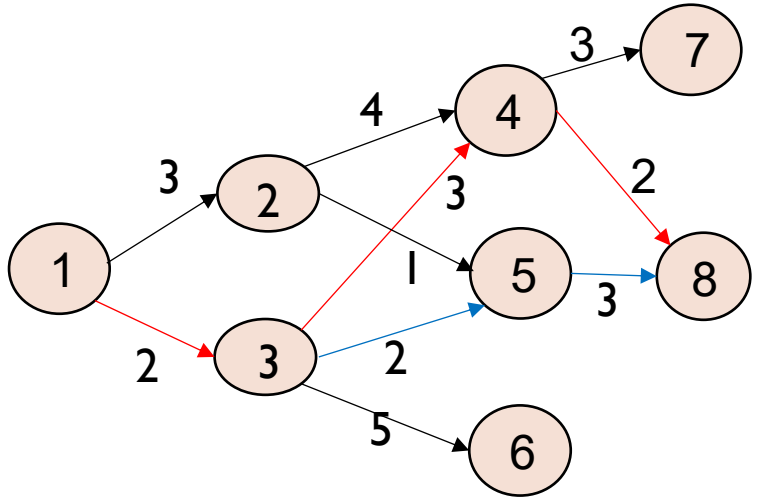
l=4

Hence, bcost[4,7]=8, l=4

bcost[4,8]=min[ bcost[3,4]+cost[4,8]=5+2=7 ]

l=4,5 bcost[3,5]+cost[5,8]=4+3=7

Hence, bcost[4,8]=7, l=4,5




bcost:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 8 | 7 | | | | | | |

l:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| l | l | l | 3 | 2,3 | 3 | 4 | 4,5 | | | | | | |

Stage 4:

bcost[4,7]=min[ bcost[3,4]+cost[4,7]=5+3=8 ]
l=4

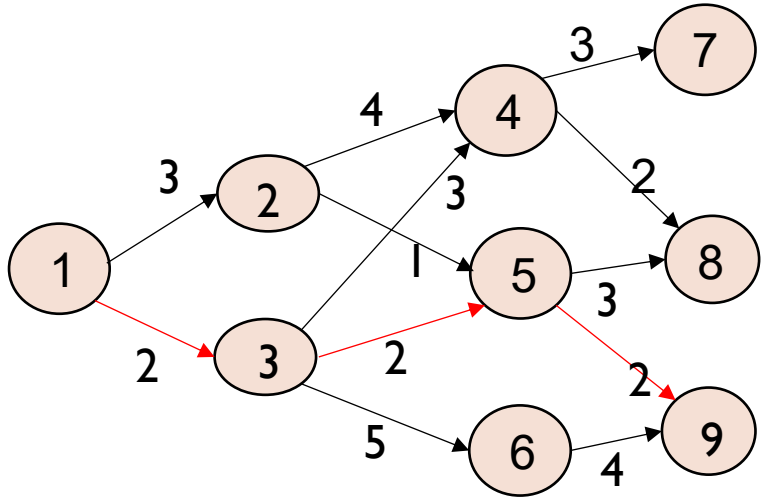
Hence, bcost[4,7]=8, l=4

bcost[4,8]=min[ bcost[3,4]+cost[4,8]=5+2=7
l=4,5 bcost[3,5]+cost[5,8]=4+3=7 ]

Hence, bcost[4,8]=8, l=4,5

bcost[4,9]=min[ bcost[3,5]+cost[5,9]=4+2=6
l=5,6 bcost[3,6]+cost[6,9]=7+4=11 ]

Hence, bcost[4,9]=6, l=5




bcost:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 8 | 7 | 6 | | | | | |

l:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| l | l | l | 3 | 2,3 | 3 | 4 | 4,5 | 5 | | | | | |

Stage 4:

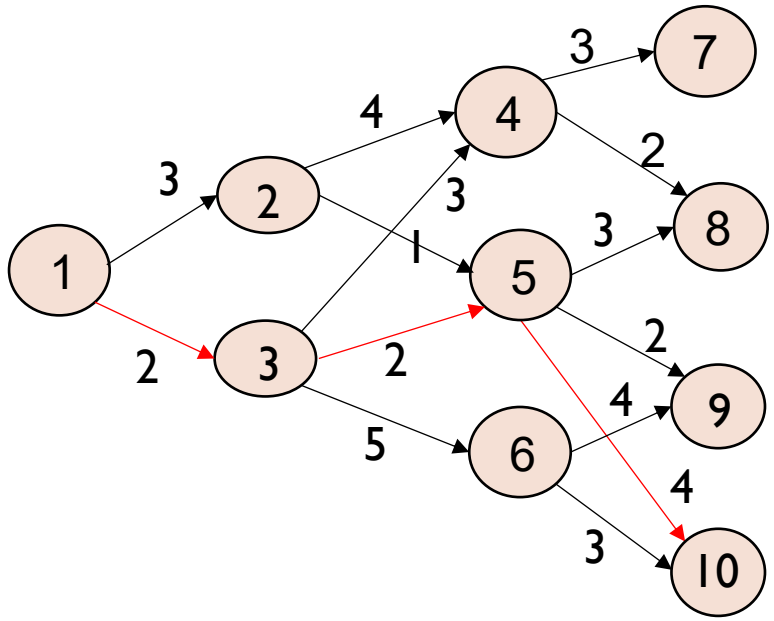
bcost[4,7]=min[ bcost[3,4]+cost[4,7]=5+3=8 ]
l=4

Hence, bcost[4,7]=8, l=4

bcost[4,8]=min[ bcost[3,4]+cost[4,8]=5+2=7
l=4,5 bcost[3,5]+cost[5,8]=4+3=7 ]
Hence, bcost[4,8]=8, l=4,5

bcost[4,9]=min[ bcost[3,5]+cost[5,9]=4+2=6
l=5,6 bcost[3,6]+cost[6,9]=7+4=11 ]
Hence, bcost[4,9]=6, l=5

bcost[4,9]=min[ bcost[3,5]+cost[5,10]=4+4=8
l=5,6 bcost[3,6]+cost[6,10]=7+3=10 ]
Hence, bcost[4,10]=8, l=5




bcost:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 8 | 7 | 6 | 8 | | | | |

l:

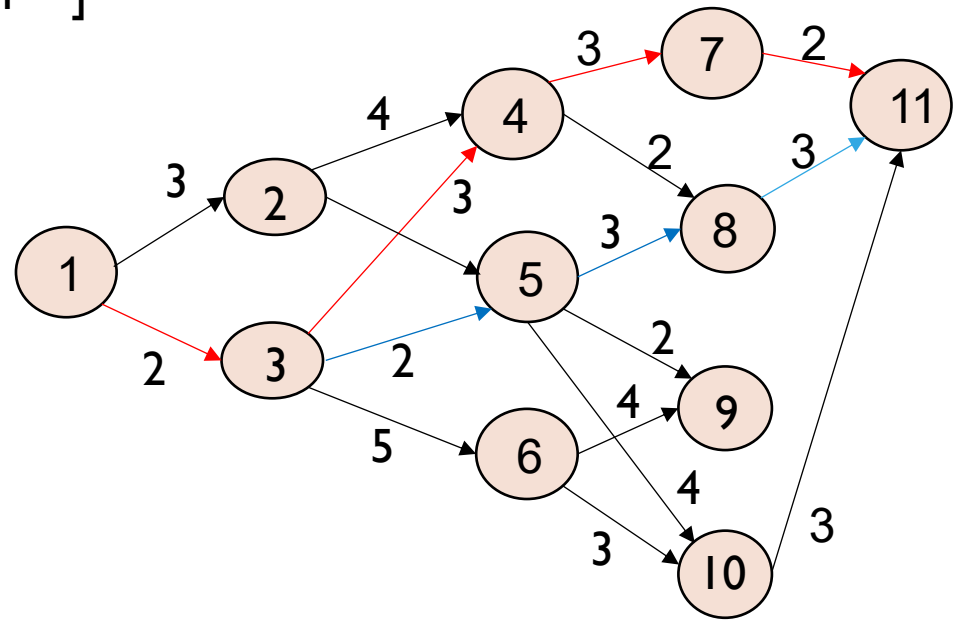
| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| l | l | l | 3 | 2,3 | 3 | 4 | 4,5 | 5 | 5 | | | | |

Stage 5:

bcost[5,11]=min[ bcost[4,7]+cost[7,11]=8+2=10
bcost[4,8]+cost[8,11]=7+3=10
bcost[4,10]+cost[10,11]=8+3=11 ]

l=7,8,10

Hence, bcost[5,11]=10, l=7,8




bcost:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 8 | 7 | 6 | 8 | 10 | | | |

l:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| l | l | l | 3 | 2,3 | 3 | 4 | 4,5 | 5 | 5 | 7,8 | | | |

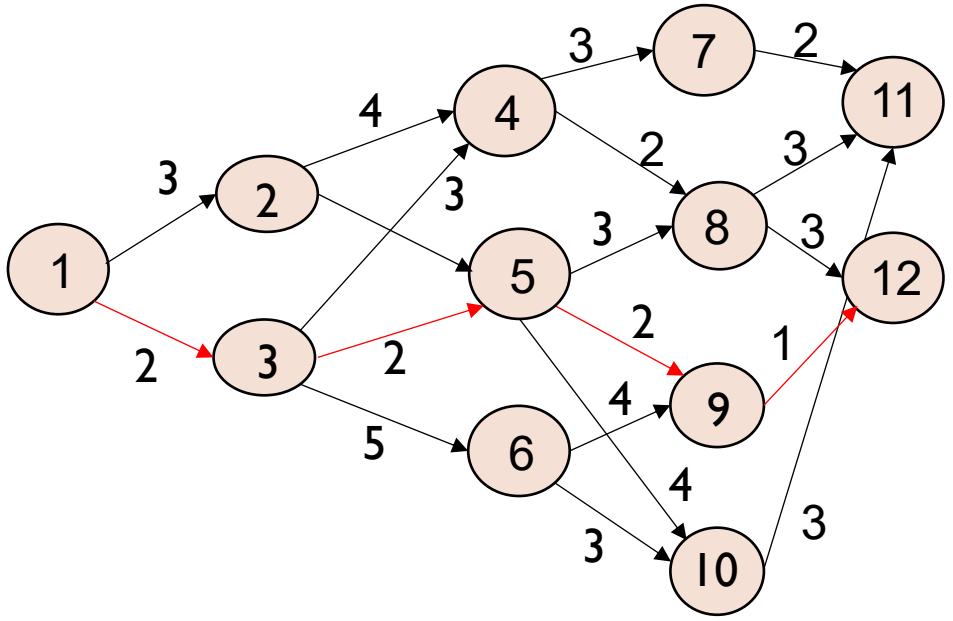
Stage 5: bcost[5,11]= min[ bcost[4,7]+cost[7,11]=8+2=10
bcost[4,8]+cost[8,11]=7+3=10
bcost[4,10]+cost[10,11]=8+3=11 ]

l=7,8,10

Hence, bcost[5,11]=10, l=7,8

bcost[5,12]= min[ bcost[4,8]+cost[8,12]= 7+3=10
l=8,9 bcost[4,9]+cost[9,12] = 6+1=7 ]

Hence, bcost[5,12]=7, l=9




bcost:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 8 | 7 | 6 | 8 | 10 | 7 | | |

l:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | 2,3 | 3 | 4 | 4,5 | 5 | 5 | 7,8 | 9 | | |

Stage 5: bcost[5,11]= min[ bcost[4,7]+cost[7,11]=8+2=10
bcost[4,8]+cost[8,11]=7+3=10
bcost[4,10]+cost[10,11]=8+3=11 ]

l=7,8,10

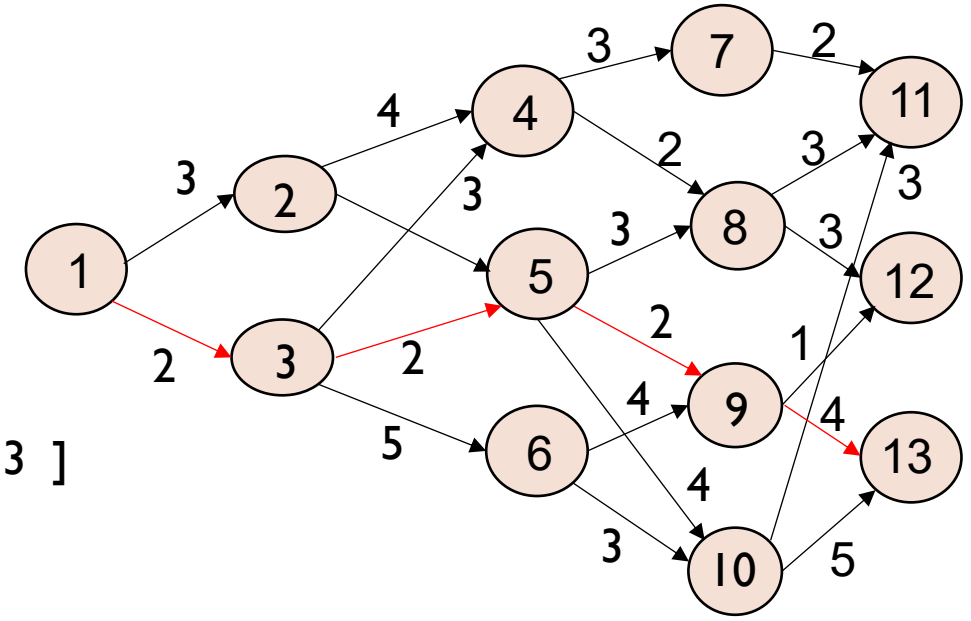
Hence, bcost[5,11]=10, l=7,8

bcost[5,12]= min[ bcost[4,8]+cost[8,12]= 7+3=10
l=8,9 bcost[4,9]+cost[9,12] = 6+1=7 ]

Hence, bcost[5,12]=7, l=9

bcost[5,13]= min[ bcost[4,9]+cost[9,13]= 6+4=10
l=9,10 bcost[4,10]+cost[10,13] = 8+5=13 ]

Hence, bcost[5,13]=10, l=9




bcost:

| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 8 | 7 | 6 | 8 | 10 | 7 | 10 | |

l:

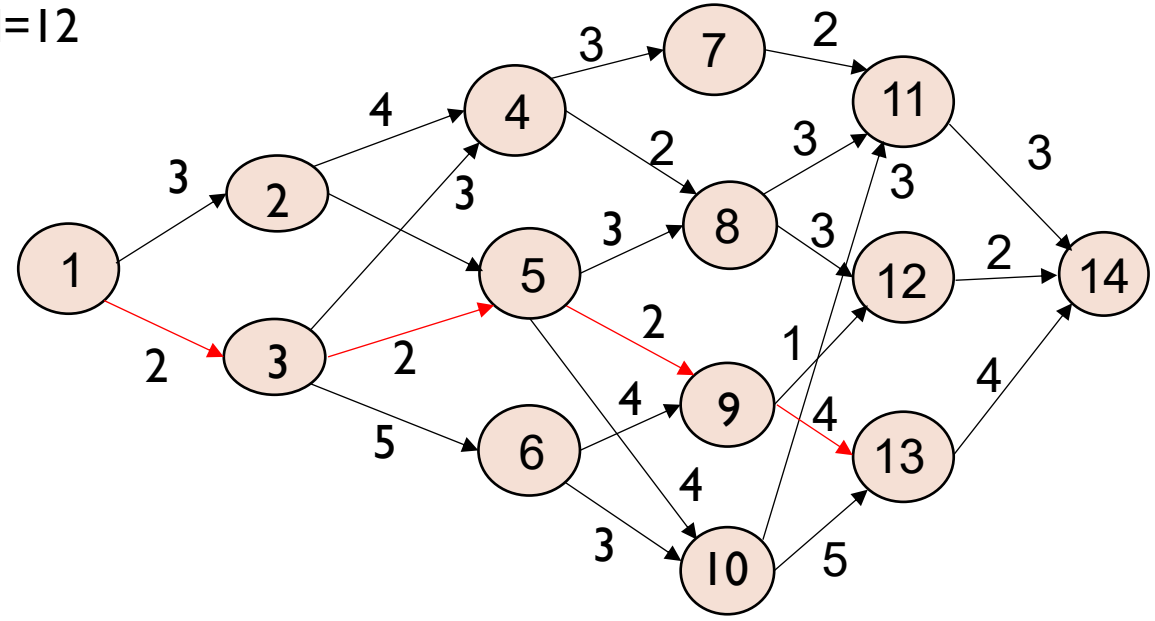
| l | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| l | l | l | 3 | 2,3 | 3 | 4 | 4,5 | 5 | 5 | 7,8 | 9 | 9 | |

Stage 6:

bcost[6,14]=min[ bcost[6,11]+cost[11,14]=10+3=13
bcost[6,12]+cost[1214]=7+2=9
bcost[6,13]+cost[13,14]=10+4=14 ]

l=11,12,13

Hence, bcost[6,14]=9, l=12

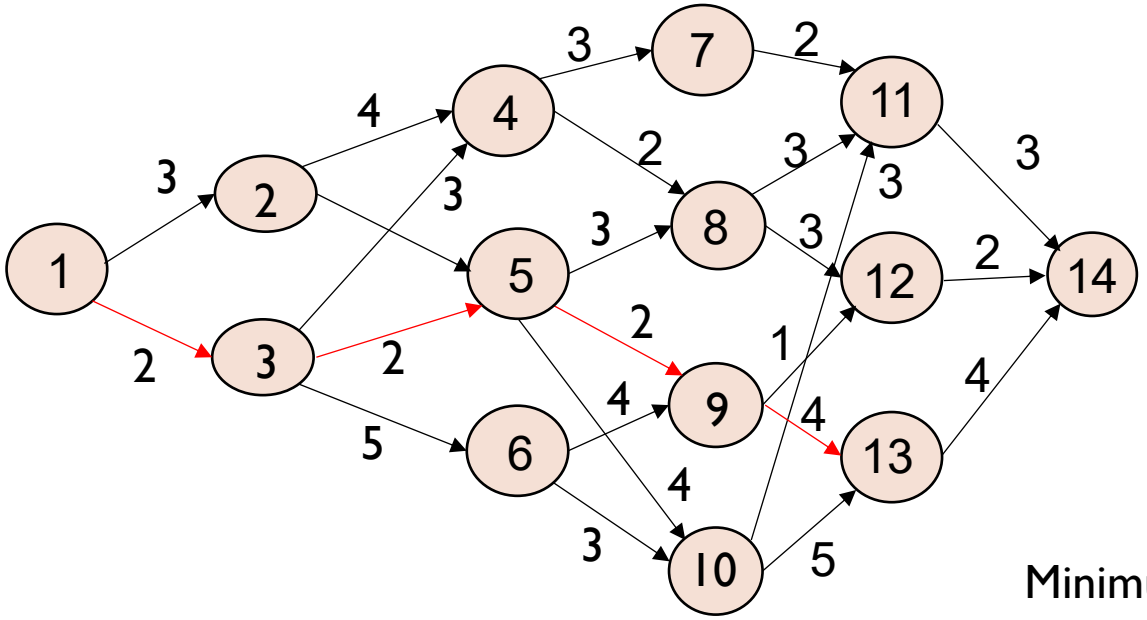



bcost:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 8 | 7 | 6 | 8 | 10 | 7 | 10 | 9 |

l:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | 2,3 | 3 | 4 | 4,5 | 5 | 5 | 7,8 | 9 | 9 | 12 |




Minimum path length: 12

bcost:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 8 | 7 | 6 | 8 | 10 | 7 | 10 | 9 |

l:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | 2,3 | 3 | 4 | 4,5 | 5 | 5 | 7,8 | 9 | 9 | 12 |

path 1:

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | 2 | 5 | 9 | 12 | 14 |

path 2:

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | 3 | 5 | 9 | 12 | 14 |

ALGORITHM (BACKWARD):-

Input:-
G: represents graph
n: total no. of vertices
cost[n*n]: represents graph with cost values

Output:-
bcost[1…..n]: cost of each vertex
d[1….n]: intermediate for each vertex
p[1….k]: path with minimum cost

## Time Complexities:-

$O(n^2)$ where n is total no. of vertices
This is the worst case where every vertex is connected with its previous vertex

```
Algorithm MultiStageBackward( G, n, cost: bcost, d, p)
{
  bcost[1]=0
  d[1]=0
  for j=2 to n do
  {
     #select a vertex 'l' such that <l,j> is an edge in the graph
       && bcost[l] + cost[l,j] is minimum
       bcost[j] = bcost[l]+cost[l,j]
       d[j]=l
  }
  p[1]=1
  p[k]=n

  for j=k-1 downto 2 do
  {
       p[j]=d[ p[j+1] ]
  }
}
```

Time Complexities:-

$O(n^2)$ where n is total no. of vertices
This is the worst case where every vertex is connected with its previous vertex

CODE:-

```
#include<stdio.h>

#include<stdlib.h>

#include<limits.h>
void multi(int n,int cost[][15],int stages){
    int bcost[n+1];
    int d[n+1];
    int p[stages+1];
    int j,l;
    bcost[0]=0;
    bcost[1]=0;
    d[0]=0;
    d[1]=1;
    for(int i=2;i<=n;i++){
        d[i]=-1;
    }
    int min;
    for(j=2;j<=n;j++){
        min=INT_MAX;
        for(l=j-1;l>=1;l--){
            if(cost[l][j]!=0 && (bcost[l]+cost[l][j] < min)){
                min=bcost[l]+cost[l][j];
                d[j]=l;
            }
        }
```

CODE (Continued..)

```
bcost[j]=min;
    }
    p[0]=0;
    p[1]=1;
    p[stages]=n;
    for(j=stages-1;j>=2;j--){
        p[j]=d[p[j+1]]; }
    printf("                Minimum path sum: %d\n",bcost[n]);
    printf("                PATH: ");
    for(j=1;j<=stages;j++){
        printf(" %d ",p[j]);
    }}
```

```c
int main(){
int cost[15][15] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 3, 2, 5, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 3, 2, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 3, 2, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 3, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 5, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}   };
    int n=14;
    int stages=6;
    printf("\n||-----------------MULTI STAGE GRAPH-----------------||\n");
    multi(n,cost,stages);
    printf("\n||---------------------------------------------------||\n");
    return 0;
}
```

OUTPUT:-

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\MultiStage graph> gcc multiStage.c
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\MultiStage graph> ./a.exe

  ||----------------MULTI STAGE GRAPH-----------------||
                  Minimum path sum: 9
                  PATH:  1  3  5  9  12  14
  ||--------------------------------------------------||
```

# MINIMUM PATH SUM IN A GRID

**Problem:**
You are given a 2D grid of size N × M where each cell contains a non-negative integer representing the cost to step into that cell.
You need to find the path from the top-left cell (0, 0) to the bottom-right cell (N-1, M-1) such that the sum of the cell values along the path is minimized.

**Rules:**
1. You can only move either:
    1. Right → from (i, j) to (i, j+1)
    2. Down → from (i, j) to (i+1, j)
2. You must start at cell (0, 0) and end at cell (N-1, M-1).
3. The path should include both the starting and ending cells.

# Minimum Path Sum in a Grid (Using Greedy Approach )

Firstly, we will find minimum path sum in a grid (using greedy approach)

The greedy approach always picks the smallest immediate neighbor. (min right or min left)

Let us take an example:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

N=3, M=3

Condition:-
i=0 j=0
Sum=arr[0][0]
While(i<m-1 || j<n-1){
    if(i==m-1)
        j++
    if(j==n-1)
        i++
    else
        Check if arr[i][j+1] <arr[i+1][j]
            True: j++
            False:i++
    Sum+=arr[i][j]
}

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

1] i=0,j=0

i<3 || j<3

Here arr[0][1] < arr[1][0]?

8 <10

True: j++ j=1

Sum+=arr[0][1]

Sum=10+8=18

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

1] i=0,j=0
i<3 || j<3
Here arr[0][1] < arr[1][0]?
8 <10
True: j++ j=1
Sum+=arr[0][1]
Sum=10+8=18

2] i=0,j=1
i<3 || j<3
Here arr[0][2] < arr[1][1]?
2 <5
True: j++ j=2
Sum+=arr[0][2]
Sum=18+2=20

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

1] i=0,j=0
i<3 || j<3
Here arr[0][1] < arr[1][0]?
8 <10
True: j++: j=1
Sum+=arr[0][1]
Sum=10+8=18

2] i=0,j=1
i<3 || j<3
Here arr[0][2] < arr[1][1]?
2 <5
True: j++ j=2
Sum+=arr[0][2]
Sum=18+2=20

3] i=0,j=2
i<3 || j<3
Here j==m-1=2
i++ : i=1
Sum+=arr[1][2]
Sum=20+100=120

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

1] i=0,j=0
i<3 || j<3
Here arr[0][1] < arr[1][0]?
8 <10
True: j++: j=1
Sum+=arr[0][1]
Sum=10+8=18

2] i=0,j=1
i<3 || j<3
Here arr[0][2] < arr[1][1]?
2 <5
True: j++ j=2
Sum+=arr[0][2]
Sum=18+2=20

3] i=0,j=2
i<3 || j<3
Here j==m-1=2
i++ : i=1
Sum+=arr[1][2]
Sum=20+100=120

4] i=1,j=2
i<3 || j<3
Here j==m-1=2
i++ : i=2
Sum+=arr[2][2]
Sum=120+2=122

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

1] i=0,j=0
i<3 || j<3
Here arr[0][1] < arr[1][0]?
8 <10
True: j++: j=1
Sum+=arr[0][1]
Sum=10+8=18

2] i=0,j=1
i<3 || j<3
Here arr[0][2] < arr[1][1]?
2 <5
True: j++ j=2
Sum+=arr[0][2]
Sum=18+2=20

3] i=0,j=2
i<3 || j<3
Here j==m-1=2
i++ : i=1
Sum+=arr[1][2]
Sum=20+100=120

4] i=1,j=2
i<3 || j<3
Here j==m-1=2
i++ : i=2
Sum+=arr[2][2]
Sum=120+2=122

Hence, the minimum path according to greedy approach is 122, which is not true as we can see the min path is 10+10+1+1+2=24

Code:-

```
//Q4 MINIMUM PATH SUM IN A GRID (USING GREEDY APPROCH)
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
void min_path_sum_greedy(int m,int n,int arr[m][n]){
    int i,j;
    int sum=0;
    sum=arr[0][0];
    i=0,j=0;
    while(i < m-1 || j < n-1){
        if(i==m-1){
            j++;
        }
        else if(j==n-1){
            i++;
        }
        else{
            if(arr[i][j+1] < arr[i+1][j] ){
                j++;
            }
            else{
                i++;
            }}
        sum+=arr[i][j];
    }
```

## Code(Continued) :-

```
    printf("-----------------------------------------\n");
    printf("Minimum Path Sum(Using Greedy Approach): %d\n",sum);
    printf("-----------------------------------------");
}
int main(){
    int m,n;
    int arr[3][3]={
        {10,8,2},
        {10,5,100},
        {1,1,2}
    };
    m=3;
    n=3;
    min_path_sum_greedy(m,n,arr);
    return 0;
}
```

OUTPUT:-

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\min path sum> gcc Min_path_sum_greedy.c
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\min path sum> ./a.exe
-----------------------------------------
Minimum Path Sum(Using Greedy Approach): 122
-----------------------------------------
```

**Greedy Algorithm Time Complexity**

- At each step, greedy only looks at the immediate neighbors (right and down).
- For a grid of size N × M, it makes decisions at each cell = O(N × M)
- Time Complexity = O(N × M)
- But this doesn't mean it finds the correct answer — it just runs fast.

## Drawback of Greedy:-

- **Greedy chooses the immediate smallest cost**
  At each step, it looks only at the next cell — it doesn't think ahead.

- **It ignores future consequences**
  Even if a bigger cost at the current step leads to a smaller total cost later, greedy misses it

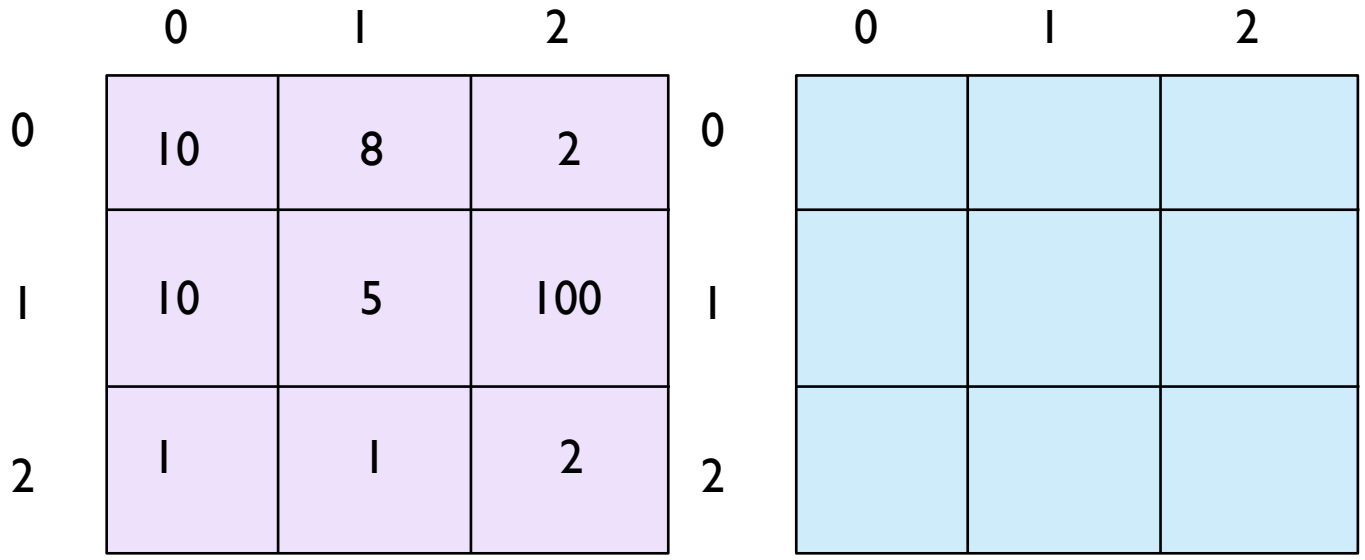
.

- **It gets stuck in a locally optimal but globally suboptimal path**
  It picks the best-looking option at every step without evaluating the full path

Hence to overcome this drawback we use DP approach which helps in finding out the best possible minimum path in a grid

# Minimum Path Sum in a Grid (Using DP Approach )

This is the optimal way to find out the minimum path sum in a grid

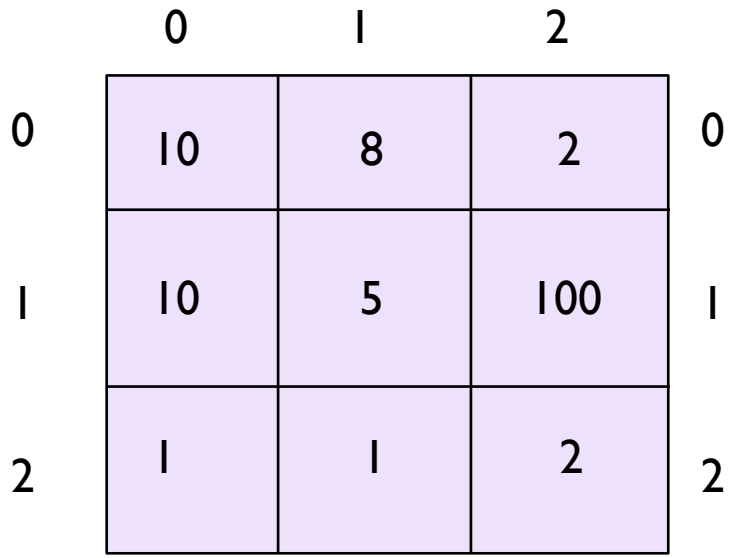



1] i=0 j=0

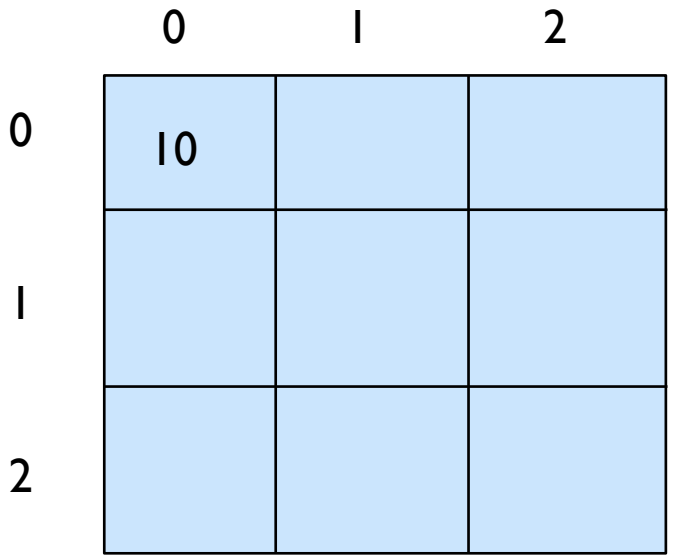
dp[0][0]=arr[0][0]

# Minimum Path Sum in a Grid (Using DP Approach )

This is the optimal way to find out the minimum path sum in a grid




arr[M][N]




dp[M][N]

1] i=0 j=0
dp[0][0]=arr[0][0]

2] i=0 j=1
up=arr[0][1]
if (i>0)
False: up=INT_MAX
left=arr[0][1]=8
if(j>0)
True: left+=dp[0][0]
left=10+8=18
if(left<=up)
True: dp[0][1]=left=18

arr[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

dp[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 18 | |
| 1 | | | |
| 2 | | | |

1] i=0 j=0
dp[0][0]=arr[0][0]

2] i=0 j=1
up=arr[0][1]
if (i>0)
False: up=INT_MAX
left=arr[0][1]=8
if(j>0)
True: left+=dp[0][0]
left=10+8=18
if(left<=up)
True: dp[0][1]=left=18

3] i=0 j=2
up=arr[0][2]
if (i>0)
False: up=INT_MAX
left=arr[0][2]=2
if(j>0)
True: left+=dp[0][1]
left=2+8=20
if(left<=up)
True: dp[0][2]=left=20

arr[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

dp[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 18 | 20 |
| 1 | | | |
| 2 | | | |

1] i=0 j=0
dp[0][0]=arr[0][0]

2] i=0 j=1
up=arr[0][1]
if (i>0)
False: up=INT_MAX
left=arr[0][1]=8
if(j>0)
True: left+=dp[0][0]
left=10+8=18
if(left<=up)
True: dp[0][1]=left=18

3] i=0 j=2
up=arr[0][2]
if (i>0)
False: up=INT_MAX
left=arr[0][2]=2
if(j>0)
True: left+=dp[0][1]
left=2+8=20
if(left<=up)
True: dp[0][2]=left=20

4] i=1 j=0
up=arr[1][0]
if (i>0)
True: up+=dp[0][0]
up=10+10=20
left=arr[1][0]=10
if(j>0)
False: left=INT_MAX
if(up<left)
True: dp[1][0]=up=20

arr[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

dp[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 18 | 20 |
| 1 | 20 | | |
| 2 | | | |

1] i=0 j=0
dp[0][0]=arr[0][0]

2] i=0 j=1
up=arr[0][1]
if (i>0)
False: up=INT_MAX
left=arr[0][1]=8
if(j>0)
True: left+=dp[0][0]
left=10+8=18
if(left<=up)
True: dp[0][1]=left=18

3] i=0 j=2
up=arr[0][2]
if (i>0)
False: up=INT_MAX
left=arr[0][2]=2
if(j>0)
True: left+=dp[0][1]
left=2+8=20
if(left<=up)
True: dp[0][2]=left=20

4] i=1 j=0
up=arr[1][0]
if (i>0)
True: up+=dp[0][0]
up=10+10=20
left=arr[1][0]=10
if(j>0)
False: left=INT_MAX
if(up<left)
True: dp[1][0]=up=20

5] i=1 j=1
up=arr[1][1]=5
if (i>0)
True: up+=dp[0][1]
up=5+18=23
left=arr[1][1]=5
if(j>0)
True: left+=dp[1][0]
left=5+20=25
if(up<left)
True: dp[1][1]=up=23

arr[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

dp[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 18 | 20 |
| 1 | 20 | 23 | |
| 2 | | | |

1] i=0 j=0
dp[0][0]=arr[0][0]

2] i=0 j=1
up=arr[0][1]
if (i>0)
False: up=INT_MAX
left=arr[0][1]=8
if(j>0)
True: left+=dp[0][0]
left=10+8=18
if(left<=up)
True: dp[0][1]=left=18

3] i=0 j=2
up=arr[0][2]
if (i>0)
False: up=INT_MAX
left=arr[0][2]=2
if(j>0)
True: left+=dp[0][1]
left=2+8=20
if(left<=up)
True: dp[0][2]=left=20

4] i=1 j=0
up=arr[1][0]
if (i>0)
True: up+=dp[0][0]
up=10+10=20
left=arr[1][0]=10
if(j>0)
False: left=INT_MAX
if(up<left)
True: dp[1][0]=up=20

5] i=1 j=1
up=arr[1][1]=5
if (i>0)
True: up+=dp[0][1]
up=5+18=23
left=arr[1][1]=5
if(j>0)
True: left+=dp[1][0]
left=5+20=25
if(up<left)
True: dp[1][1]=up=23

6] i=1 j=2
up=arr[1][2]=100
if (i>0)
True: up+=dp[0][2]
up=100+20=120
left=arr[1][2]=100
if(j>0)
True: left+=dp[1][1]
left=100+23=123
if(up<left)
True: dp[1][2]=up=120

arr[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

dp[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 18 | 20 |
| 1 | 20 | 23 | 120 |
| 2 | | | |

7] i=2 j=0
up=arr[2][0]=1
if (i>0)
True: up+=dp[1][0]
up=1+20=21
left=arr[2][0]=1
if(j>0)
False: left=INT_MAX
if(up<left)
True: dp[2][0]=up=21

arr[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

dp[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 18 | 20 |
| 1 | 20 | 23 | 120 |
| 2 | 21 | | |

7] i=2 j=0
up=arr[2][0]=1
if (i>0)
True: up+=dp[1][0]
up=1+20=21
left=arr[2][0]=1
if(j>0)
False: left=INT_MAX
if(up<left)
True: dp[2][0]=up=21

8] i=2 j=1
up=arr[2][1]=1
if (i>0)
True: up+=dp[1][1]
up=1+23=24
left=arr[2][1]=1
if(j>0)
True: left+=dp[2][0]
left=1+21=22
if(up>=left)
True: dp[2][1]=left=22

arr[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

dp[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 18 | 20 |
| 1 | 20 | 23 | 120 |
| 2 | 21 | 22 | |

7] i=2 j=0
up=arr[2][0]=1
if (i>0)
True: up+=dp[1][0]
up=1+20=21
left=arr[2][0]=1
if(j>0)
False: left=INT_MAX
if(up<left)
True: dp[2][0]=up=21

8] i=2 j=1
up=arr[2][1]=1
if (i>0)
True: up+=dp[1][1]
up=1+23=24
left=arr[2][1]=1
if(j>0)
True: left+=dp[2][0]
left=1+21=22
if(up>=left)
True: dp[2][1]=left=22

6] i=2 j=2
up=arr[2][2]=2
if (i>0)
True: up+=dp[1][2]
up=2+120=122
left=arr[2][2]=2
if(j>0)
True: left+=dp[2][1]
left=2+22=24
if(up<left)
True: dp[2][2]=left=24

# Minimum Path Sum in a Grid (Using DP Approach )

arr[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 8 | 2 |
| 1 | 10 | 5 | 100 |
| 2 | 1 | 1 | 2 |

dp[M][N]

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 18 | 20 |
| 1 | 20 | 23 | 120 |
| 2 | 21 | 22 | 24 |

Hence, Minimum path sum in a grid= dp[M-1][N-1]
= dp[2][2]=24

CODE:-

```
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
void min_path_sum_non_greedy(int m,int n,int arr[m][n]){
    int dp[m][n];
    int up,left,i,j;
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            if(i==0 && j==0){
                dp[0][0]=arr[0][0];
            }else{
                up=arr[i][j];
                if(i>0){
                        up+=dp[i-1][j];
                }
                else{
                    up=INT_MAX;
                }
                left=arr[i][j];
                if(j>0){
                    left+=dp[i][j-1];
                }
                else{
                    left=INT_MAX;
                 }
```

CODE(Continued):-

```
                if(up<left ){
                    dp[i][j]=up;
                }
                if(left<=up ){
                    dp[i][j]=left;
                }
            }
        }
    }
    printf("-----------------------------------------------\n");
    printf("Minimum path sum(Using Non-Greedy Approach): %d\n",dp[m-1][n-1]);
    printf("-----------------------------------------------\n");
}
int main(){
    int m,n;
    int arr[3][3]={
        {10,8,2},
        {10,5,100},
        {1,1,2}
    };
    m=3;
    n=3;
    min_path_sum_non_greedy(m,n,arr);
    return 0;
}
```

OUTPUT (USING DP Approach):-

```
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\min path sum> gcc Min_path_sum_nonGreedy.c
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\min path sum> ./a.exe
 ------------------------------------------
 Minimum path sum(Using Non-Greedy Approach): 24
 ------------------------------------------
PS C:\Users\Vidhita\OneDrive\Desktop\DAA TA2\min path sum>
```

**Dynamic Programming (DP) Time Complexity**

- We systematically calculate the minimum path sum for every cell.
- For each cell (i, j), we only need to look at two neighbors → top (i-1,j) and left (i,j-1).
- We fill every cell once.
- Time Complexity = O(N × M)
- It guarantees the correct answer.

# THANK YOU!!