School of Computer Engineering
KIIT deemed to be University
Laboratory Lesson Plan – Autumn 2024 (5th Semester)

Discipline: CS / IT / CSCE / CSCE

Course Name and Code: Algorithms Laboratory-CS2098 (L-T-P-Cr: 0-0-2-1)

Pre-requisite: **Data Structures (CC21001/ CS21001/ CM21001/ IT21001) / Data Structures and Algorithms (CS-2001)/ Data Structures Using C (CS3040)/ Concepts of Data Structures and Algorith (CS20001)**

Course Contents :

- **Review of Fundamentals of Data Structure like** Array, Link List, Sorting Algorithms.
- **Fundamentals of Algorithmic Problem Solving-I:** Analysis of time complexity of small algorithms through step/frequency count method. (GCD, Prime Number, Sorting Algorithms)
- **Divide and Conquer Method :** Binary Search, Merge Sort, Quick Sort, Randomized QuickSort.
- **Heap & Priority Queues:** Building a heap, Heap sort algorithm, Min-Priority queue, Max-Priority queue.
- **Greedy Technique:** Fractional knapsack problem, Activity selection problem, Huffman's cod, Minimum spanning trees (Kruskal's Algorithm, Prim'sAlgorithm), Dijkstra algorithm (Single Source Shortest Path).
- **Dynamic Programming**: Knapsack problem, Matrix Chain Multiplication, longest common subsequence Multistage graphs, All pair's shortest paths (Floyd-Warshall Algorithm), Optimal binary search trees, Travelling salesman problem.
- **Amortization :** Randomized Algorithms and Amortized Analysis, Las Vegas and Monte Carlo types, Randomized quick sort andits analysis, Min-Cut algorithm.

Course Objectives:

- Review of fundamentals of Data Structures
- Fundamentals of Problem Solving and Analysis: Sorting algorithms, GCD, Binary Conversion, etc.)
- Divide and Conquer Method: Binary Search, Merge Sort, Quick Sort, Randomized Quick Sort
- Heap & Priority Queues: Building a heap, Heap sort algorithm, Min-Priority queue, Max-Priority queue
- Greedy Techniques: Fractional knapsack problem, Activity selection problem, Huffman's code, Single Source Shortest Path (Dijkstra's Algorithm, Bellman-Ford Algorithm)
- Dynamic Programming: Matrix Chain Multiplication, Longest Common Subsequence
- Graph Algorithms: Application of Graph Traversals, All Pair Shortest Path (Floyd-Warshall Algorithm), Minimum Cost Spanning Tree (Kruskal's Algorithm, Prim's Algorithm)
- **Amortization :** Randomized Algorithms and Amortized Analysis, Las Vegas and Monte Carlo types, Randomized quick sort and its analysis, Min-Cut algorithm.

List of Experiments (Lab-Day wise)

Lab Day 1: Revision of Data Structures

1.1 *Aim of the program:* Write a program to find out the second smallest and second largest element stored in an array of n integers.
Input: Size of the array is 'n' and read 'n' number of elements from a disc file.
Output: Second smallest, Second largest

1.2 *Aim of the program:* Given an array arr[] of size N, find the prefix sum of the array. A prefix sum array is another array prefixSum[] of the same size, such that the value of prefixSum[i] is arr[0] + arr[1] + arr[2] . . . arr[i].

Input Array:	3	4	5	1	2
Output Array:	3	7	12	13	15

1.3 *Aim of the program:* Write a program to read 'n' integers from a disc file that must contain some duplicate values and store them into an array. Perform the following operations on the array.

a) Find out the total number of duplicate elements.
b) Find out the most repeating element in the array.

Input:
Enter how many numbers you want to read from file: 15

Output:
The content of the array: 10 40 35 47 68 22 40 10 98 10 50 35 68 40 10
Total number of duplicate values = 4
The most repeating element in the array = 10

1.4 *Aim of the program:* Write a function to ROTATE_RIGHT(p1, p2) right an array for first p2 elements by 1 position using EXCHANGE(p, q) function that swaps/exchanges the numbers p & q. Parameter p1 be the starting address of the array and p2 be the number of elements to be rotated.

Input:

Enter an array A of size N (9): 11 22 33 44 55 66 77 88 99
Call the function ROTATE_RIGHT(A, 5)

Output:

Before ROTATE: 11 22 33 44 55 66 77 88 99
After ROTATE: 55 11 22 33 44 66 77 88 99

Lab Day 2: Fundamentals of Algorithmic Problem Solving

2.1 *Aim of the program:* Write a program in C to convert the first 'n' decimal numbers of a disc file to binary using recursion. Store the binary value in a separate disc file.

Note# Read the value of 'n', source file name and destination file name from command line arguments. Display the decimal numbers and their equivalent binary numbers from the output file.

Give the contents of the input disc file "inDec.dat" as
30 75 2564 …

Contents of the output disc file "outBin.dat" as
The binary equivalent of 30 is 0000000000011110
The binary equivalent of 75 is 0000000001001011
The binary equivalent of 2564 is 0000101000000100

Terminal Input:
$gcc lab2q1.c -o lab2q1
$./lab2q1 150 inDec.dat outBin.dat

Output: Content of the first 'n' decimal and their equivalent binary numbers

2.3 *Aim of the program:* Write a program in C to find GCD of two numbers using recursion. Read all pair of numbers from a file and store the result in a separate file.

Note# Source file name and destination file name taken from command line arguments. The source file must contain at least 20 pairs of numbers.

Give the contents of the input disc file "inGcd.dat" as 8 12 20 45 30 80

Contents of the output disc file "outGcd.dat" as
The GCD of 8 and 12 is 4
The GCD of 20 and 45 is 5
The GCD of 30 and 80 is 10

Terminal Input:
$gcc lab2q2.c -o lab2q2
$./lab2q2 inGcd.dat outGcd.dat

Output: Display the gcd stored in the output file outGcd.dat

Lab Day 3: Divide and Conquer Method

3.1 *Aim of the program:* Write a menu driven program to sort list of array elements using Merge Sort technique and calculate the execution time only to sort the elements. Count the number of comparisons.

Note#

- To calculate execution time, assume that single program is under execution in the CPU.
- Number of elements in each input file should vary from 300 to 500 entries.
- For ascending order: Read data from a file "inAsce.dat" having content 10 20 30 40….., Store the result in "outMergeAsce.dat".
- For descending order: Read data from a file "inDesc.dat" having content 90 80 70 60…., Store the result in "outMergeDesc.dat".
- For random data: Read data from a file "inRand.dat" having content 55 66 33 11 44 …, Store the result in "outMergeRand.dat"

Sample Input from file:
MAIN MENU (MERGE SORT)
1. Ascending Data
2. Descending Data
3. Random Data
4. ERROR (EXIT)

Output:
Enter option: 1
Before Sorting: Content of the input file
After Sorting: Content of the output file
Number of Comparisons: Actual
Execution Time: lapse time in nanosecond

3.2 *Aim of the program:* Write a menu driven program to sort a list of elements in ascending order using Quick Sort technique. Each choice for the input data has its own disc file. A separate output file can be used for sorted elements. After sorting display the content of the output file along with number of comparisons. Based on the partitioning position for each recursive call, conclude the input scenario is either best-case partitioning or worst-case partitioning.

Note#

- *The* worst-case behavior for quicksort occurs when the partitioning routine produces one subproblem with n-1 elements and one with 0 elements. The best-case behaviour occurred in most even possible split, PARTITION produces two subproblems, each of size no more than n/2.
- Number of elements in each input file should vary from 300 to 500 entries.

- For ascending order: Read data from a file “inAsce.dat” having content 10 20 30 40….., Store the result in “outQuickAsce.dat”.
- For descending order: Read data from a file “inDesc.dat” having content 90 80 70 60…., Store the result in “outQuickDesc.dat”.
- For random data: Read data from a file “inRand.dat” having content 55 66 33 11 44 …, Store the result in “outQuickRand.dat”

Sample Input from file:
MAIN MENU (QUICK SORT)
1. Ascending Data
2. Descending Data
3. Random Data
4. ERROR (EXIT)

Output:
Enter option: 1
Before Sorting: Content of the input file
After Sorting: Content of the output file
Number of Comparisons: Actual
Scenario: Best or Worst-case

Lab Day 4: Heap

4.1 *Aim of the program:*

Define a *struct person* as follows:

```
struct person
{
    int id;
    char *name;
    int age;
    int height;
    int weight;
};
```

Write a menu driven program to read the data of ‘n’ students from a file and store them in a dynamically allocated array of *struct person*. Implement the min-heap or max-heap and its operations based on the menu options.

Sample Input/Output:
MAIN MENU (HEAP)
1. Read Data

2. Create a Min-heap based on the age
3. Create a Max-heap based on the weight
4. Display weight of the youngest person
5. Insert a new person into the Min-heap
6. Delete the oldest person
7. Exit

Enter option: 1

Id	Name	Age	Height	Weight(pound)
0	Adarsh Hota	39	77	231
1	Levi Maier	56	77	129
2	Priya Kumari	63	78	240
3	Dorothy Helton	47	72	229
4	Florence Smith	24	75	171
5	Erica Anyan	38	73	102
6	Norma Webster	23	75	145

Enter option: 4
Weight of youngest student: 65.77 kg

Note#: Other menu choices are similarly verified.

Lab Day 5: Greedy Techniques

5.1 *Aim of the program:* Write a program to find the maximum profit nearest to but not exceeding the given knapsack capacity using the Fractional Knapsack algorithm.

Notes# Declare a structure ITEM having data members item_id, item_profit, item_weight and profit_weight_ratio. Apply heap sort technique to sort the items in non-increasing order, according to their profit /weight.

Input:

Enter the number of items: 3
Enter the profit and weight of item no 1: 27 16
Enter the profit and weight of item no 2: 14 12
Enter the profit and weight of item no 3: 26 13

Enter the capacity of knapsack:18

Output:

Item No profit Weight Amount to be taken

3	26.000000	13.000000	1.000000
1	27.000000	16.000000	0.312500
2	14.000000	12.000000	0.000000

Maximum profit: 34.437500

5.2 *Aim of the program:* Huffman coding assigns variable length code words to fixed length input characters based on their frequencies or probabilities of occurrence. Given a set of characters along with their frequency of occurrences, write a c program to construct a Huffman tree.

Note#

- Declare a structure SYMBOL having members alphabet and frequency. Create a Min-Priority Queue, keyed on frequency attributes.
- Create an array of structures where size=number of alphabets.

Input:

Enter the number of distinct alphabets: 6

Enter the alphabets:	a	b	c	d	e	f
Enter its frequencies:	45	13	12	16	9	5

Output:

In-order traversal of the tree (Huffman): a c b f e d

Lab Day 6: Greedy Techniques (Cont…) -- Minimum Cost Spanning Tree (Kruskal's Algorithm, Prim'sAlgorithm)

6.1 *Aim of the program:* Given an undirected weighted connected graph G(V, E) and starring vertex 's'. Maintain a Min-Priority Queue 'Q' from the vertex set V and apply Prim's algorithm to

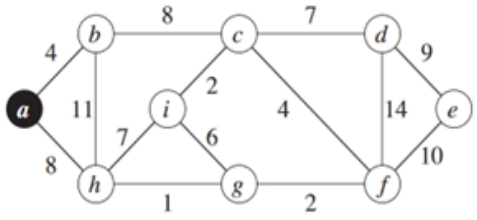
- Find the minimum spanning tree T(V, E'). Display the cost adjacency matrix of 'T'.
- Display total cost of the minimum spanning tree T.

Note# Nodes will be numbered consecutively from 1 to n (user input), and edges will have varying weight. The graph G can be read from an input file "inUnAdjMat.dat" that contains cost adjacency matrix. The expected output could be displayed as the cost adjacency matrix of the minimum spanning tree and total cost of the tree.

Content of the input file "inUnAdjMat.dat" could be

0 4 0 0 0 0 0 8 0

11 0



4 0 8 0 0 0 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0

Input:
Enter the Number of Vertices: 9

Enter the Starting Vertex: 1

Output:
0 4 0 0 0 0 0 8 0
4 0 0 0 0 0 0 0 0
0 0 0 7 0 4 0 0 2
0 0 7 0 9 0 0 0 0
0 0 0 9 0 0 0 0 0
0 0 4 0 0 0 2 0 0
0 0 0 0 0 2 0 1 0
8 0 0 0 0 0 1 0 0
0 0 2 0 0 0 0 0 0

Total Weight of the Spanning Tree: 37

6.2 *Aim of the program:* Given an undirected weighted connected graph G(V, E). Apply Krushkal's algorithm to

- Find the minimum spanning tree T(V, E') and Display the selected edges of G.
- Display total cost of the minimum spanning tree T.

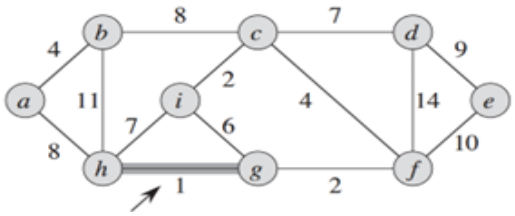
Note# Nodes will be numbered consecutively from 1 to n (user input), and edges will have varying weight. The weight matrix of the graph can be represented from the user's input in the given format. The expected output could be the selected edge and the corresponding cost of the edge as per the sample output.

Input Format:

- The first line contains two space-separated integers ‘n’ and ‘m’, the number of nodes and edges in the graph.
- Each line ‘i’ of the ‘m’ subsequent lines contains three space-separated integers ‘u’, ‘v’ and ‘w’, that describe an edge (u, v) and weight ‘w’.

Input:

9 14
1 2 4
1 8 8
2 3 8
2 8 11
3 4 7
3 6 4
3 9 2
4 5 9
4 6 14
5 6 10
6 7 2
7 8 1
7 9 6
8 9 7



Output:

Edge	Cost
8--7	1
7--6	2
3--9	2
1--2	4
3--6	4
3--4	7
1--8	8
4--5	9

Total Weight of the Spanning Tree: 37

Lab Day 7 : Greedy Techniques (Cont…) – Single Source Shortest path

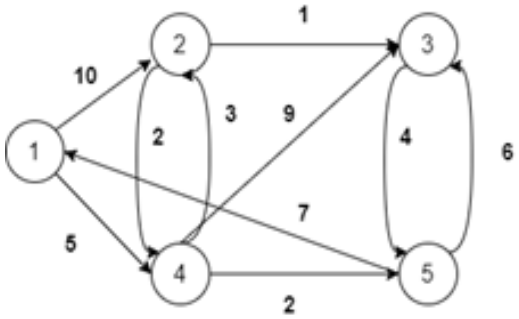
7.1 *Aim of the program:* Given a directed graph G (V, E) and a starting vertex ‘s’.

- Determine the lengths of the shortest paths from the starting vertex ‘s’ to all other vertices in the graph G using Dijkstra’s Algorithm.
- Display the shortest path from the given source ‘s’ to all other vertices.

Note# Nodes will be numbered consecutively from 1 to n (user input), and edges will have varying distances or lengths. The graph G can be read from an input file "inDiAdjMat1.dat" that contains non-negative cost adjacency matrix. The expected output could be as per the sample format.

Content of the input file "inDiAdjMat1.dat" could be

	1			
0	0	0	5	0
0	0	1	2	0
0	0	0	0	4
3	0	9	0	2
7	0	6	0	0



Input:

Enter the Number of Vertices: 5

Enter the Source Vertex: 1

Output:

Source	Destination	Cost	Path
1	1	0	-
1	2	8	1->4->2
1	3	9	1->4->2->3
1	4	5	1->4
1	5	7	1->4->5

Lab Day 8: Dynamic Programming

8.1 *Aim of the program:* Write a program to implement the matrix chain multiplication problem using M-table & S-table to find optimal parenthesization of a matrix-chain product. Print the number of scalar multiplications required for the given input.

Note# Dimensions of the matrices can be inputted as row and column values. Validate the dimension compatibility.

Input:

Enter number of matrices: 4
Enter row and col size of A1: 30 35
Enter row and col size of A2: 35 15
Enter row and col size of A3: 15 5
Enter row and col size of A4: 5 10

Output:

M Table:

0	15750	7875	9375
0	0	2625	4375
0	0	0	750
0	0	0	0

S Table:

0	1	1	3
0	0	2	3
0	0	0	3
0	0	0	0

Optimal parenthesization: ((A1 (A2 A3)) A4)

The optimal ordering of the given matrices requires 9375 scalar multiplications.

8.2 *Aim of the program:* Write a program to find out the Longest Common Subsequence of two given strings. Calculate length of the LCS.

Input:

Enter the first string into an array: 10010101
Enter the second string into an array: 010110110

Output:

LCS: 100110
LCS Length: 6

Lab Day 9: Dynamic Programming - All Pair Shortest path

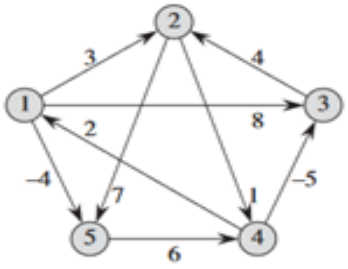
9.1 *Aim of the program:* Given a directed weighted graph G (V, E) where weight indicates distance. Vertices will be numbered consecutively from 1 to n (user input), and edges will have varying distances or lengths.

- Determine the length of the shortest path between every pair of vertices using Floyd-Warshall's algorithm.
- Display the intermediate vertices on the shortest-path from the given pair of vertices (u,v).

Note# The graph G can be read from an input file "inDiAdjMat2.dat" that contains cost adjacency matrix. The expected output could be a shorted-path weight matrix and the path consisting of intermediate vertices.

Content of the input file "inDiAdjMat2.dat" could be

0	3	8	0	-4
0	0	0	1	7
0	4	0	0	0
2	0	-5	0	0
0	0	0	6	0



Input:

Number of Vertices: 5

Enter the source and destination vertex: 2 5

Output:

0	1	-3	2	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

Shortest Path from vertex 2 to vertex 5: 2-->4-->1-->5

Path weight: -1

Lab Day 10: Amortization:

10.1 *Aim of the program:* Write a program to implement randomized quicksort. Explain the expected time complexity of this algorithm.

10.2 *Aim of the program:* We represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of fits, where the ith least significant fit indicates whether the sum includes the ith Fibonacci number Fi .

For example, the fitstring 101110 represents the number F6 + F4 + F3 + F2 = 8 + 3 + 2 + 1 = 14. Write a program to increment and decrement a single fitstring in constant amortized time. [Hint: Most numbers can be represented by more than one fitstring]

10.3 *Aim of the program* : we want to store a big binary counter in an array A. All the entries start at 0 and at each step we will be simply incrementing the counter. Let's say our cost model is: whenever we increment the counter, we pay 1 for every bit we need to flip. Now we not only wish to increment a counter but also to reset it to 0 (i.e., make all bits in it 0). Counting the time to examine or modify a bit as $\theta(1)$ show how to implement a counter as an array of bits so that any sequence of n INCREMENT and RESET operations takes O(n) time on an initially zero counter. (Hint: Keep a pointer to the high-order 1.)

Example: A binary counter with 5 bits.

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Operation	Cost
0	0	0	0	0	Intial	0
0	0	0	0	1	Increment	1
0	0	0	1	0	Increment	2
0	0	0	0	0	Reset	3
0	0	0	0	1	Increment	4
0	0	0	0	0	Reset	5

Practice Problem Sets:

Fundamentals of Algorithmic Problem Solving (contd…)

1 *Aim of the program:* Write a program to implement Binary Search to give the position of leftmost appearance of the element in the array being searched. Display the number of comparisons made while searching.

Input:

Enter size of array: 10
Enter elements of the array: 2 3 7 7 7 11 12 12 20 50
Enter the key to be searched: 7

Output:

7 found at index position 2
Number of comparisons: 3

2. *Aim of the program:* Write a menu driven program to sort a list of elements in ascending order using Insertion Sort technique. The nature of the input data is choice based and a distinct file is considered for each choice. The sorted elements can be stored in a separate output file. After sorting display the content of the output file along with number of comparisons. Based on the number of comparisons, conclude the input scenario is either best or worst case.

Note#

- Number of elements in each input file should vary from 300 to 500 entries.
- For ascending order: Read data from a file "inAsce.dat" having content 10 20 30 40 ….., Store the result in "outInsAsce.dat".
- For descending order: Read data from a file "inDesc.dat" having content 90 80 70 60…., Store the result in "outInsDesc.dat".
- For random data: Read data from a file "inRand.dat" having content 55 66 33 11 44 …, Store the result in "outInsRand.dat"

Sample Input from file:

MAIN MENU (INSERTION SORT)
1. Ascending Data
2. Descending Data
3. Random Data
4. ERROR (EXIT)

Output:

Enter option: 1
Before Sorting: Content of the input file

After Sorting: Content of the output file
Number of Comparisons: Actual
Scenario: Best or Worst-case

3. *Aim of the program:* Consider an undirected graph where each edge weights 2 units. Each of the nodes is labeled consecutively from 1 to n. The user will input a list of edges for describing an undirected graph. After representation of the graph, from a given starting position

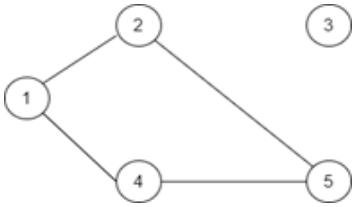
- Display the *breadth-first search traversal.*
- Determine and display the shortest distance to each of the other nodes using the *breadth-first search* algorithm. Return an array of distances from the start node in node number order. If a node is unreachable, return -1 for that node.

Input Format:

- The first line contains two space-separated integers ‘n’ and ‘m’, the number of nodes and edges in the graph.
- Each line ‘i’ of the ‘m’ subsequent lines contains two space-separated integers ‘u’ and ‘v’, that describe an edge between nodes ‘u’ and ‘v’.
- The last line contains a single integer ‘s’, the node number to start from.

Output Format:

- The first line shows the result of the BFS traversal.
- The last line shows an array of distances from node ‘s’ to all other nodes.



Input:

5 4
1 2
1 4
4 5
2 5
1

Output:

BFS Traversal: 1 2 4 5

Distance [2 -1 2 4]

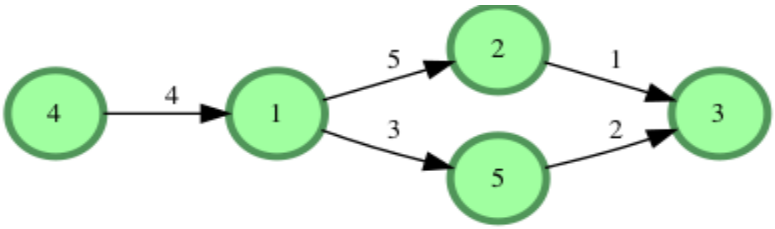
4.Aim of the program: Let us call any directed graph G = (V, E) one-way-connected iff for all pair of vertices u and v at least one of the following holds:

(a) There is a path from vertex u to v,

(b) There is a path from vertex v to u.

Design an algorithm to check if a given directed graph is one-way-connected. Give running time analysis

Input: The graph can be taken in the form of adjacency matrix or adjacency linked list.



Output: The given graph is one way connected

5. *Aim of the program:* Let n be a positive integer. We want to reduce n to 1 by applying a sequence of simple operations on n. The operations permitted are division of n by two (provided that n is even), and addition of a small positive or negative integer to n. Your objective is to reach the final goal 1 with as few applications of these operations as possible

a) Assume that the only operations permitted are decrement (by 1) and division by 2 (allowed if and only if n is even). For example, consider this reduction of 125 to 1, which uses 12 steps: 125 → 124 → 62 → 31 → 30 → 15 → 14 → 13 → 12 → 6 → 3 → 2 → 1. This sequence is not optimal. An obvious greedy strategy is based on the fact that division by 2 produces the best possible local reduction. However, if n is odd, this operation is not allowed, so decrement n to make it even, and then divide by 2. The optimal solution involves 11 steps only:

125 → 124 → 62 → 31 → 30 → 15 → 14 → 7 → 6 → 3 → 2 → 1.

Implement this greedy algorithm in a function greedy1

b) Now, suppose that besides decrement by 1 and division by 2, you are permitted to increment n by 1. A greedy approach in this case is to keep on dividing n by 2 so long as it remains even. Once n becomes odd, compute n − 1 and n + 1, and choose the one which has a larger number of factors of 2. This is illustrated by the next example. This reduction has nine steps.

125 → 124 → 62 → 31 → 32 → 16 → 8 → 4 → 2 → 1.

A long sequence of consecutive increments/decrements does not help:

125 → 126 → 127 → 128 → 64 → 32 → 16 → 8 → 4 → 2 → 1.

Changing 125 to 128 by increments is not helpful. A shorter sequence is achieved at a reduced level by one increment of 31 to 32. This greedy strategy fails only for n = 3. The greedy reduction is 3 → 4 → 2 → 1, whereas the optimal reduction is 3 → 2 → 1. Write a function greedy2 to implement these observations.

Grading Policies:

- ***Continuous Evaluation components:*** Continuous evaluation (60 **Marks**)
  - **Lab participation (10 Marks)**: Students' participation in the lab based on their attendance and engagement. (Evaluated by Faculty)
  - **Lab records (10 Marks):** Only input output of each assignment may be written in the lab records.(Verified and Evaluated by TA)
  - **Continuous evaluation (based on Lab skills, 20 Marks)**: Students' lab skills will be assessed through hands-on activities and involvements in doing assignments during the lab hour. (Evaluated by TA)
  - **Practice Set and Quiz (10+10=20 Marks)**: student learning through practice set & quiz**.** (Evaluated by Faculty)
- **End semester evaluation:** Comprehensive *assessment of student learning and performance. **(40 Marks)***
  - **Programming Test (20 Marks):** Execution of the program. (Evaluated by Faculty/TA)
  - **Viva (10 Marks):** Understanding of the course. (Evaluated by Faculty)
  - **Quiz (10 Marks):** MCQ type questions. (Evaluated by Faculty)

Reference Materials

1. T. H. Coreman, C. E. Leiserson, R. L. Rivest, "Introduction to Algorithms, PHI
2. Sara Basse, A. V. Gelder, "Computer Algorithms", Eddison Wesley.
3. Michael Goodrich, Roberto Tamassia, "Algorithm Design: Foundations, Analysis & Internet Examples", John Wiley & Sons.
4. Fundamentals of comp Alg E.Harwitz,S. sahani,S.Rajsekharan,Galgotia

NOTE: Faculty members are advised to add one problem in each of the week which is not in the manual. This will test their problems solving skill and response to unknown problems.