# LAB MANUAL

# Algorithm Laboratory

## (CS-2098) Through C/C++, Python(Opt)

FOR B.TECH STUDENTS

**Version - 2.0/13.07.2022**

Prepared By

**Prof. Anil Kumar Swain**

School of Computer Engineering

**School of Computer Engineering**

**Kalinga Institute of Industrial Technology(KIIT),**

**Deemd to be University, Bhubaneswar**

## INTRODUCTION

- Programming cannot be learned by watching others do it. Students must spend numerous hours working on programs themselves.
- This laboratory manual is a tool that will allow students to experiment with computer science & this is the beginning. As students progress through each laboratory, they may wonder how or why something works. The best way to discover the answer is to try things out.
- The **purpose of this lab. manual** is to acquaint the students to understand about writing algorithms, performance analysis of algorithms and step by step approach in **solving problems with the help of well known algorithm designs through C/C++ language.**
- This strengthen the ability of students to identify and apply suitable algorithm design techniques for the given real world problem.

## STRUCTURE OF THIS LAB. MANUAL

- This lab. manual provides study aids from programming assignments to scheduled exercises using prepared materials.
- This lab manual is divided into **10 laboratory classes**. The laboratory classes consist of the following components:
  a) **Lab. Excerise (LE):** These are the **assignments** that ask each student to independently create small programs **during the lab time**.
  b) **Home Exercise (HE):** These are the assignments to be done during lab time by each student if lab. assignments are completed before lab. time or may be assigned as **post-lab homework** and submitted in the next lab class
  c) **Round Exercise (RE):** These are the **group assignments/small projects** to be done by each group round the time and to be submitted one copy per group at any time if asked to submit after one week of RE given or at the end of the course completion.

The approach of the Lab proceedings: **LE-HE-RE**

## INSTRUCTIONS FOR STUDENTS

To make laboratory experiments effective, each student must obey the following rules:

1. **General instructions**
   - Once you create a directory named as your rollno_section under the home directory of UBUNTU OS system using command-line or by GUI.
   - In Each lab, store programs within appropriate folders named as LAB01, LAB02, LAB03...etc. which are the sub folders under your rollno_section folder.
   - Always save programs files with the meaningful name preceded by lab assignment no within specified folders. If you want solve a lab assignment no. 3.5 (3.5 means 5th assignment of 3rd lab), then name the program as 35_progname.c or 35_prog.c etc.
2. **Attendance:** Attendance is required at all labs without exception. There are no make-up labs in this course. Performance will be judged based on the experiments conducted, quality and punctual submission of the labs reports for each experiment. Faculty/Instructor will take attendance. Failure to be present for an experiment will result in loosing entire marks for the corresponding lab. However, genuine cases may be considered for repeat lab. If a student misses a lab session due to unavoidable circumstances can provide a legitimate proof as soon as possible, he/she may be then be allowed by the lab instructor, to make-it-up.
3. **Laboratory Report:** At the end of every lab student will be assigned to write-up one of the experiment's problem. Your report must present a clear and accurate account, results you obtained. Student should develop habit to submit the laboratory report/assignments continuously and progressively on the scheduled dates and should get the assessment done.
4. Read the write up of each experiment to be performed, a day in advance. Understand the purpose of experiment and its practical implications.
5. Student should not hesitate to ask any difficulty faced during conduct of practical / exercise.
6. The student shall study all the questions given in the laboratory manual and practice to write the answers to these questions.
7. Student shall develop the habit of evolving more ideas, innovations, skills etc. those included in the scope of the manual.
8. Student should develop the habit of not to depend totally on teachers but to develop self learning techniques.
9. While entering into the LAB students should wear their ID cards.
10. Shut down your system after you have finished with your experiment.

## PROCEDURE FOR EVALUATION

- The entire lab course consists of 100 marks. The marking scheme is as follows:

| | |
|---|---|
| Continuous Evaluation marks | 60 |
| End Sem. Lab Examination | 40 |
| **Total** | **100** |

## Scheme for continuous evaluation (60 Marks)

- The distribution of marks is upto the faculty, but the scheme of evaluation must be informed in advance to the students.
- One sample is given as follows:
  Students will be evaluated bi-weekly. Minimum 6 evaluations should be conducted for each student. Each evaluation carries 10 marks. The scheme is as follows:

| **Components** | **Marks** |
|---|---|
| Program & Execution | 5 |
| Lab. Record | 3 |
| Viva-Voce | 2 |
| **Total** | **10** |

## Scheme for end sem lab examination (40 Marks)

- The distribution of marks is upto the faculty, but the scheme of evaluation must be informed in advance to the students.
- One sample is given as follows:
  End sem. lab exam will be conducted after the completion of all the weekly exercises. The student will not be allowed for exam if he/she is found short of attendance and has not completed all the experiments. The marking scheme for end sem lab exam is as follows:

| **Components** | **Marks** |
|---|---|
| Write-up of program | 10 |
| Program execution & Checking Results for all inputs | 10 |
| Final Viva-Voce, Quiz | 20 |
| **Total** | **40** |

# CONTENTS

# LAB - 1

# Review of Fundamentals of Data Structure

## PROGRAM EXERCISE

## Lab. Exercise (LE)

**1.1** Write a program to store random numbers into an array of n integers and then find out the smallest and largest number stored in it. n is the user input.

**1.2** Write a program to store random numbers into an array of n integers, where the array must contains some duplicates. Do the following:

a) Find out the total number of duplicate elements.

b) Find out the most repeating element in the array.

**1.3** Write a program to rearrange the elements of an array of n integers such that all even numbers are followed by all odd numbers. How many different ways you can solve this problem. Write your approaches & strategy for solving this problem.

**1.4** Write a program that takes three variable (a, b, c) as separate parameters and rotates the values stored so that value a goes to be, b, b to c and c to a by using SWAP(x,y) function that swaps/exchanges the numbers x & y.

**1.5** Let A be n*n square matrix array. WAP by using appropriate user defined functions for the following:

a) Find the number of nonzero elements in A

b) Find the sum of the elements above the leading diagonal.

c) Display the elements below the minor diagonal.

d) Find the product of the diagonal elements.

## Home Exercise (LE)

**1.6** Write a program to find out the second smallest and second largest element stored in an array of n integers. n is the user input. The array takes input through random number generation within a given range. How many different ways you can solve this problem. Write your approaches & strategy for solving this problem.

**1.7** Write a program to swap pair of elements of an array of n integers from starting. If n is odd, then last number will be remain unchanged.

**1.8** Write a program to display an array of n integers (n>1), where at every index of the array should contain the product of all elements in the array except the element at the given index. Solve this problem by taking single loop and without an additional array.

Input Array : 3 4 5 1 2

Output Array :40 30 24 120 60

**1.9** Write a program using a function for computing ⌊√n⌋ for any positive integer. Besides assignment and comparison, your algorithm may only use the four basic arithmetic operations.

**Hints:** In number theory, the integer square root (isqrt) of a positive integer n is the positive integer m which is the greatest integer less than or equal to the square root of n,

isqrt(n)=⌊√n⌋

## Round Exercise (RE)

**1.10** Assume that you are given a rudimentary programming language which contains only four operators, viz., +, −, abs and div. + and − have their usual meanings, while div(a, b) returns the quotient of a/b and abs(a) returns the absolute value of a. Write a program to solve this problem by using a function max(a, b) that takes two integers a and b as input and returns the maximum of the two. Note that you can only use the operators provided; in particular, the constructs ”if”, “while”, or “for” are not available.

# LAB - 2
# Fundamentals of Algorithmic Problem Solving-I:

**(Analysis of time complexity of small algorithms through step/frequency count method.)**

## PROGRAM EXERCISE

## Lab. Exercise (LE)

2.1 Write a program to **test whether a number n, entered through keyboard is prime or not** by using different algorithms you know for atleast 10 inputs and note down the time complexity by step/frequency count method for each algorithm & for each input. Finally make a comparision of time complexities found for different inputs, plot an appropriate graph & decide which algothm is faster.

| Sl. No. | Input (n) | Prime Number Testing | | |
|---|---|---|---|---|
| | | Algorithm-1 (Time by frequency) | Algorithm-2 (Time by frequency) | Algorithm-3 (Time by frequency) |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

**N.B:** If you can solve more than three different ways, then add more columns right of Algorithm-3 column.

2.2 Write a program **to find out GCD (greatest common divisor)** using the following three algorithms.

a) Euclid's algorithm
b) Consecutive integer checking algorithm.
c) Middle school procedure which makes use of common prime factors. For finding list of primes implement sieve of Eratosthenes algorithm.

Write a program to find out which algorithm is faster for the following data. Estimate how many times it will be faster than the other two by step/frequency count method in each case.

i. Find **GCD of two numbers when both are very larg**e i.e.GCD(31415, 14142) by applying each of the above algorithms.

ii. Find **GCD of two numbers when one of them can be very large** i.e.GCD(56, 32566) or GCD(34218, 56) by applying each of the above algorithms.

iii. Find **GCD of two numbers when both are very smal**l i.e.GCD(12,15) by applying each of the above algorithms.

iv. Find **GCD of two numbers when both are same** i.e.GCD(31415, 31415) or GCD(12, 12) by applying each of the above algorithms.

Write the above data in the following format and decide which algorithm is faster for the particular data.

| **Sl. No.** | **Input GCD(x, y)** | **GCD Algorithm** | | | |
|---|---|---|---|---|---|
| | | **Euclid's algorithm (Frequency Count)** | **Consecutive integer checking algorithm. (Frequency Count)** | **Middle school procedure algorithm (Frequency Count)** | **Remarks (Which one Faster than other two)** |
| 1 | GCD (31415, 14142) | | | | |
| 2 | GCD (56, 32566) | | | | |
| 3 | GCD (34218, 56) | | | | |
| 4 | GCD(12,15) | | | | |
| 5 | GCD (31415, 31415) | | | | |
| 6 | GCD (12, 12) | | | | |

## Home Exercise (HE)

2.3 Write a menu driven program as given below, to sort an array of n integers in ascending order by **insertion sort algorithm** and determine the **time required (in terms of step/frequency count)** to sort the elements. Repeat the experiment for different values of n and different nature of data (i.e.apply insertion sort algorithm on the data of array that are already sorted, reversely sorted and random data). Finally plot a graph of the time taken versus n for each type of data. The elements can be read from a file or can be generated using the random number generator.

**INSERTION SORT MENU**

---

0. Quit
1. n Random numbers=>Array
2. Display the Array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using any sorting algorithm
5. Time Complexity to sort ascending of random data
6. Time Complexity to sort ascending of data already sorted in ascending order
7. Time Complexity to sort ascending of data already sorted in descending order
8. Time Complexity to sort ascending of data for all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000

---

Enter your choice:

If the choice is option 8, the it will display the tabular form as follows:

**<u>Analysis of Insertion Sort Algorithm</u>**

| **Sl. No.** | **Value of n** | **Time Complexity (Random Data)** | **Time Complexity (Data in Ascending)** | **Time Complexity (Data in Descending)** |
|---|---|---|---|---|
| 1 | 5000 | | | |
| 2 | 10000 | | | |
| 3 | 15000 | | | |
| 4 | 20000 | | | |
| 5 | 25000 | | | |
| 6 | 30000 | | | |
| 7 | 35000 | | | |
| 8 | 40000 | | | |
| 9 | 45000 | | | |
| 10 | 50000 | | | |

# LAB - 3

## Fundamentals of Algorithmic Problem Solving-II:

**(Analysis of time complexity of algorithms through the concept of asymptotic notations)**

**PROGRAM EXERCISE**

### Lab. Exercise (LE)

3. 1 Rewrite the program no-2.3 **(Insertion Sort)** with the following details.

   i. Compare the best case, worst case and average case time complexity with the same data except time complexity will count the CPU clock time.

   ii. Plot a graph showing the above comparison (n, the input data Vs. CPU times for best, worst & average case)

   iii. Compare manually program no-2.1 graph vs program no-3.1 graph and draw your inference.

3. 2 Let A be a list of n (not necessarily distinct) integers. Write a program by using User Defined Function(UDF)s to test whether any item occurs more than □ n/2□ times in A.

   a) UDF should take $O(n^2)$ time and use no additional space.

   b) UDF should take $O(n)$ time and use $O(1)$ additional space.

3. 3 Write a program by using an user defined function for computing □ √n□ for any positive integer n. Besides assignment and comparison, your algorithm may only use the four basic arithmetical operations.

3. 4 Let A be an array of n integers $a_0,a_1,\ldots ,a_{n-1}$ (negative integers are allowed), denoted, by A[i... j], the sub-array $a_i$, $a_{i+1},\ldots ,a_j$ for i≤j. Also let $S_{i-j}$ denote the sum $a_i + a_{i+1} +\cdot\cdot\cdot + a_j$. Write a programby using an udf that must run in $O(n^2)$ time to find out the maximum value of $S_{i-j}$ for all the pair i, j with $0 \le i \le j \le n-1$. Call this maximum value S. Also obtains the maximum of these computed sums. Let j < i in the notation A[i... j] is also allowed. In this case, A[i... j] denotes the empty sub-array (that is, a sub-array that ends before it starts) with sum $S_{i-j} = 0$. Indeed, if all the elements of A are negative, then one returns 0 as the maximum sub-array sum.

   a. For example, for the array A[]={1, 3, 7, 2, 1, 5, 1, 2, 4, 6, 3}.

   b. This maximum sum is $S = S_{3-8} = 2+( 1)+5+( 1)+( 2)+4 = 7$.

## Round Exercise (RE)

3. 5 Design a data structure to maintain a set S of n distinct integers that supports the following two operations:

a) INSERT(x, S): insert integer x into S

b) REMOVE-BOTTOM-HALF(S): remove the smallest □ n/2□ integers from S.

c) Write a program by using UDFs that give the worse-case time complexity of the two operations INSERT(x, S) in O(1) time and REMOVE-BOTTOM-HALF(S) in O(n) time.

3. 6 Consider an n × n matrix A = ($a_{ij}$), each of whose elements $a_{ij}$ is a nonnegative real number, and suppose that each row and column of A sums to an integer value. We wish to replace each element $a_{ij}$ with either □ $a_{ij}$□ or □ $a_{ij}$□ without disturbing the row and column sums. Here is an example:

$$\begin{pmatrix} 10.9 & 2.5 & 1.3 & 9.3 \\ 3.8 & 9.2 & 2.2 & 11.8 \\ 7.9 & 5.2 & 7.3 & 0.6 \\ 3.4 & 13.1 & 1.2 & 6.3 \end{pmatrix} \rightarrow \begin{pmatrix} 11 & 3 & 1 & 9 \\ 4 & 9 & 2 & 12 \\ 7 & 5 & 8 & 1 \\ 4 & 13 & 2 & 6 \end{pmatrix}$$

Write a program by defining an user defined function that is used to produce the rounded matrix as described in the above example. Find out the time complexity of your algorithm/function.

# LAB - 4
# Divide and Conquer Method

**(Binary Search, Merge Sort, Quick Sort, Randomized Quick Sort)**

## PROGRAM EXERCISE

### Lab. Exercise (LE)

**4.1** Write a program to search an element x in an array of n integers using **binary search** algorithm that uses divide and conquer technique. Find out the best case, worst case and average case time complexities for different values of n and plot a graph of the time taken versus n. The n integers can be generated randomly and x can be choosen randomly, or any element of the array or middle or last element of the array depending on type of time complexity analysis.

**4.2** Write a program to **sort a list** of n elements using the **merge sort** method and determine the time required to sort the elements. Repeat the experiment for different values of n and different nature of data (random data, sorted data, reversely sorted data) in the list. n is the user input and n integers can be generated randomly. Finally plot a graph of the time taken versus n.

### Home Exercise (HE)

**4.3** Write a program to use divide and conquer method to determine the time required to find the maximum and minimum element in a list of n elements. The data for the list can be generated randomly. Compare this time with the time taken by straight forward algorithm or brute force algorithm for finding the maximum and minimum element for the same list of n elements. Show the comparison by plotting a required graph for this problem.

**4.4** Write a program that uses a divide-and-conquer algorithm/user defined function for the exponentiation problem of computing $a^n$ where $a > 0$ and n is a positive integer. How does this algorithm compare with the brute-force algorithm in terms of number of multiplications made by both algorithms.

## Round Exercise (RE)

**4.5** A and B are playing a guessing game where B first thinks up an integer X (positive, negative or zero, and could be of arbitrarily large magnitude) and A tries to guess it. In response to A's guess, B gives exactly one of the following three replies:

a) Try a bigger number

b) Try a smaller number or

c) You got it.

Write a program by designing an efficient algorithm to minimize the number of guesses A has to make. An example (not necessarily an efficient one) below:

Let B thinks up the number 35

| A's guess | B's response |
|---|---|
| 10 | Try a bigger number |
| 20 | Try a bigger number |
| 30 | Try a bigger number |
| 40 | Try a smaller number |
| 35 | You got it |

**4.6** Write a program by using an user defined function to implement **merge sort** without a recursion ( bottom-up version of mergesort) by starting with merging adjacent elements of a given array, then merging sorted pairs, and so on.

**4.7** The **quick sort** algorithm is an efficient and popular sorting technique that sorts a list of keys recursively by choosing a pivot key. A pivot may be chosen as the first or last or mean or median or any random element of the list. Write a program to implement this sorting algorithm and execute the sorting programs for the following sets of data.

i. Ordered List

ii. Reverse order List

iii. A list containing the same value through out

iv. Random List

v. 50% of the List sorted

Also measure CPU time, number of partitions and number of comparisons for data sizes 1K, 50K, 1L, 1.5L, 2L, 2.5L, 3L, 3.5L, 4L, 4.5L and 1M. Present your results using tables and graphs and write a 1-page report that summarize the behavior of sorting algorithms tested and their suitability in each case as mentioned above.

**4.8** In a social gathering, there are b boys and g girls ($b > g$) of different ages. You have two unsorted arrays giving their ages (one for the boys, the other for the girls). Write a program by devising an efficient O(b log g) algorithm to find out the ages that are common between both the boys and girls.

Example:

If Arrayboy = {10, 20, 11, 89, 23, 21} and Arraygirl = {12, 30, 11, 20},

Then Arraycommon = {11, 20}

**4.9** Refer the following **new sort algorithm** for sorting an array A of n numbers. The algorithm is described below:

(i) If there is only one number, return.
(ii) If there are two numbers, perform a single comparison to determine the order.
(iii) If there are more than two numbers, then first sort the top two-thirds of the elements recursively. Follow this by sorting the bottom two-thirds of the elements recursively and then sorting the top two-thirds of the elements again.

a) Write a program that uses a recursive algorithm to implement the above strategy.
b) What is the comparison complexity of this new-sort algorithm? Formulate a recurrence relation and solve the same to justify your answer.

# LAB - 5
# Heap & Priority Queues

**(Building a heap, Heap sort algorithm, Min-Priority queue, Max-Priority queue)**

**PROGRAM EXERCISE**

## Lab. Exercise (LE)

**5.1** Write a menu (given as follows) driven program to sort an array of n integers in ascending order by heap sort algorithm and perform the operations on max heap. Determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the array to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

**MAX-HEAP & PRIORITY QUEUE MENU**

**0.** Quit
**1.** n Random numbers=>Array
**2.** Display the Array
**3.** Sort the Array in Ascending Order by using Max-Heap Sort technique
**4.** Sort the Array in Descending Order by using any algorithm
**5.** Time Complexity to sort ascending of random data
**6.** Time Complexity to sort ascending of data already sorted in ascending order
**7.** Time Complexity to sort ascending of data already sorted in descending order
**8.** Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000
**9.** Extract largest element
**10.** Replace value at a node with new value
**11.** Insert a new element
**12.** Delete an element

Enter your choice:

If the choice is option 8, the it will display the tabular form as follows:

**Analysis of Max-Heap Sort Algorithm**

| **Sl. No.** | **Value of n** | **Time Complexity (Sorted Data)** | **Time Complexity (Reversely Sorted Data)** | **Time Complexity (Random Data)** |
|---|---|---|---|---|
| 1 | 5000 | | | |
| 2 | 10000 | | | |

| | | | | |
|---|---|---|---|---|
| 3 | 15000 | | | |
| 4 | 20000 | | | |
| 5 | 25000 | | | |
| 6 | 30000 | | | |
| 7 | 35000 | | | |
| 8 | 40000 | | | |
| 9 | 45000 | | | |
| 10 | 50000 | | | |

## Home Exercise (HE)

**5.2** Similar to above program no.5.1, write a menu driven program to sort an array of n integers in **descending order by heap sort algorithm.** Hints: Use min heap and accordingly change the menu options.

## Round Exercise (RE)

**5.3** Write a program with the following two functions.

a) MaxMin() - Takes a heap, returns 'MAX' if it is a max-heap and converts it into a mean-heap in linear (to the number of elements) time.

b) MinMax() - Takes a heap, returns 'MIN' if it is a min-heap and converts it into a max-heap in linear (to the number of elements) time.