Workbook

Data Structure Algorithm & Application (CT – 157)

First Year

Name	
Roll No	
Batch	
Year	
Department	

Department of Computer Science & Information Technology
NED University of Engineering & Technology

Workbook

Data Structure Algorithm & Application (CT – 157)

First Year

Prepared by

Nazish Saleem
I.T Manager, CS&IT

Approved by

Chairman
Department of Computer Science & Information Technology

Department of Computer Science & Information Technology
NED University of Engineering & Technology

Contents

Lab # 1

Object:
To insert and delete in Word Processing.

Theory:
- **<u>Insertion</u>**
  Suppose in a given text T. We want to insert a string P so that P begins in position R in text T. This operation is denoted as

  INSERT (T, R, P)

  For example, if T = " I am a student of second year science",

  R=30, and P= "computer". Then INSERT (T ,R, P) will result in the string "I am a student of second year computer science".

- **<u>Deletion</u>**
  Suppose in a given text T. We want to delete the substring that begins in position R in text T and has length L. This operation is denoted as
  DELETE (T, R, L)

  For example, if T= ABCDEFGH, R=4 & L=3.

  Then DELETE (T, R, L) will result in the string "ABCGH".

Algorithm:
Algorithm A; INSERT (T, R, P)
This algorithm uses strings temp 1 and temp2.

```
1. Temp 1= substring (T,1 ,R-1).
2. Temp 2= substring (T, R, Length (T) –R+1).
3. Concatenate (Temp 1, P).
4. Concatenate (Temp 1, Temp 2).
5. T= temp 1.
6. EXIT.
```

Algorithm B; DELETE (T, R, L)
This algorithm uses temporary strings Temp 1 and Temp 2

```
1. temp 1= substring (T, 1, R-1)
2. temp 2 = substring (T, R+L, length (T) –R –L +1).
3. T = concatenate (temp 1, temp 2).
4. Exit.
```

Exercise:

(a) Implement algorithms A and B in C Language.

(b) Run your algorithm A with the following
T= “Department of Science”. R=14 and
P= “Computer”.

(c) Run your algorithm B with the following
T= “Master of Computer Science”, R=10, and L=8.

Lab # 2

Object:
To find the pattern in text (Pattern matching Algorithm).

Theory:
- **Index**
  To find the position where a string pattern P first appears in a given string T. This operation is called indexing or pattern matching, denoted by INDEX (T, P).

Algorithm:
INDEX (T, P).

P & T are strings with lengths M & N respectively. This algorithm finds the index of P in T.

```
1.    Set K= 1   and    MAX=  N-M + 1.
2.    Repeat step 3 to 5 while K <= MAX.
3.    Repeat for L =  1 to M.
          If P [L] ≠ T [K+L-1], then go to step 5.
4.    Set INDEX= K and exit.
5.    Set  K= K+1.
6.    Set INDEX= 0     (failure)
7.    Exit.
```

Exercise:

(a) Implement the above algorithm in C Language.

(b) Run your algorithm with the following test cases

(i). T= “Computer Science” and P= “put”.

(ii). T= “INFORMATION” and P= “FORN”

Lab # 3

Object:
To find the string replace function in Word Processing.

Theory:

- **Replacement**
  Replacement of pattern P by Q in text T.

  For example, if T= “ABCDEFG”, P= “DE” and Q= “X”, then REPLACE (T, P, Q) will result as T= “ABCXFG”.

Algorithm:
REPLACE (T, P, Q).
This algorithm replaces every occurrence of P in T by Q.

```
1. Set K= INDEX (T, P).
2. Repeat while  K≠ 0
                 (a) Set T= REPLACE (T, P, Q)
                 (b) [Update index]. Set K= INDEX (T, P)
   [End of loop]
3. Write : T
4. Exit.
```

Exercise:

(a) Implement the above algorithm in C Language.

(b) Run your algorithm on the following test cases
T= “We are Muslims”, P= “We” and Q= “All”

Lab # 4

Object:
To find the largest element in array.

Theory:
A linear array is a list of a finite number n of homogenous data elements (that is data elements of same type) such that:

a) The elements of the array are referenced respectively by an index set consisting of n consecutive numbers

b) The elements of the array are stored respectively in successive memory locations

The number n of elements is called the length or size of array, it can be obtained from the index set by the formula"

Length = upper bound – lower bound

The elements of the array A may be denoted by the subscript notation:
A[1], A[2],……..A[N]

Where:
A is the name of array and the number K in A[K] is called the subscript or an index and A[K] is called a subscripted variable. Subscripts allow any element of A to be referenced by its relative position A.

Algorithm:
Given a non empty array DATA with N numerical values. This algorithm finds the location LOC and value MAX of the largest element of DATA.

```
1. Set  K= 1, LOC= 1 and MAX= DATA [1].
2. Repeat step 3 and 4 while K<= N.
3.      if MAX < DATA  [K], then Set LOC = K and
                MAX= DATA [K].
        [End of if structure]
4. Set    K= K+1.
   [end of step 2 loop].
5. Write:  LOC, MAX.
6. Exit.
```

Exercise:

(a) Implement the above algorithm in C Language.

(b) Run the algorithm with some examples.

Lab # 5

Object:
Performing linear search on a list of elements stored in an array.

Theory:
Let DATA be collection of N element in memory and suppose a specific ITEM of information is given. We want either to find the location LOC of ITEM in array DATA or to send some message, such as LOC=0 to indicate that ITEM does not appear in DATA. We compare ITEM with DATA [1], then DATA [2] and so on, until we find LOC such that   ITEM=DATA [LOC].

Algorithm:
This algorithm finds the location LOC of ITEM in array DATA with N elements or set LOC =0.

```
1.  Set  K=1 and  LOC= 0.
2.  Repeat step 3 & 4 while LOC=0 and K<= N.
3.      If  ITEM=DATA[K], then set LOC=K.
4.      Set K= K+1.
    [End of step 2 loop]
5.  if LOC=0  then write:  ITEM is not in array DATA
    ELSE
            Write LOC is the location of ITEM
    [end of if structure].
6.  Exit.
```

Exercise:

(a) Implement the above algorithm in C Language.

(b) Run the algorithm with some examples.

Lab # 6

Object:
Performing Binary search on a list of elements stored in an array.

Theory:
Suppose DATA is an array which is stored in increasing (or decreasing) or alphabetically order. Then there is an extremely efficient searching algorithm called Binary search, which can be used to find the location LOC of a given ITEM of information in DATA. The Binary search applied to our array DATA works as follows: During each stage of our algorithm, our search for ITEM is reduced to a segment of elements of DATA:

DATA [BEG], DATA[BEG+1], ………, DATA [END]

Note that the variables BEG and END denote, respectively, the beginning and end locations of the segment under consideration.

The algorithm compares ITEM with the middle elements DATA[MID] of the segment, where MID is compared as MID=INT ((BEG + END) / 2) (INT(A) refers to the integer value of A.). If DATA[MID]=ITEM, then the search is successful and we set LOC+MID. Otherwise a new segment of DATA is obtained as follows:

(a) If ITEM < DATA[MID], then ITEM can appear only in left half of the segment. So we reset END= MID-1 and begin search again.
(b) If ITEM > DATA[MID], then ITEM can appear only in the right half of the segment. So we reset BEG-= MID+1 and begin search again.

Initially we begin with the entire array DATA: i.e., we begin with BEG=1 and END=n or more generally, with BEG=LB and END=UB. If ITEM is not in DATA, then eventually we obtain BEG> END. This condition signals that the search is unsuccessful, and in such a case we assign LOC = NULL. Here, NULL is a value that lies outside the set of indices of DATA.

Algorithm:

```
1. Set BEG =LB, END= UB and MID= INT (( BEG + END) /2)
2. Repeat step 3 & 4 while BEG <= END and DATA [MID]≠ITEM.
3. If  ITEM < DATA[MID], then Set END=Mid-1
   ELSE
       Set  BEG = MID+1
4. Set MID = INT ((BEG + END) /2)          [End of step 2 loop]
5. If DATA [MID] = ITEM then
      Set LOC = MID
   Else
      Set LOC = NULL.                      [End of if structure].
6. Exit.
```

Exercise:

(a) Implement the above algorithm in C Language

(b) Run your algorithm with some examples.

Lab # 7

Object:
Performing bubble sorting on a list of elements stored in an array.

Theory:
Suppose the list of numbers A[1], A[2], ………,A[N] is in memory. The bubble sort algorithm works as follows:

Step 1
Compare A[1] and A[2] and arrange them in the desired order, so that A[1] < A[2]. Then compare A[2] and A[3] and arrange them so that A[S]< A[3]. Continue till we compare A[N-1] with A[N] and arrange them so that A[N-1] <A[N]. Observe that step 1 involves n-1 comparisons. When step 1 is completed, A[N] will contain the largest element.

Step 2
Repeat step 1 with one less comparison; that is, now we stop after we compare and possibly rearrange A[N-2] and A[N-1].
(Step 2 involves N-2 comparisons and, when step 2 is completed, the second largest element will occupy A[N-1].)

Step 3
Repeat step 1 with two fewer comparisons; that is, we stop after we compare and possibly rearrange A[N-3] and A[N-2].

.

.

.

Step N-1
Compare A[1] with A[2] and arrange them so that A[1] < A[2]. After n-1 steps, the list will be stored in increasing order.

Algorithm:
Here, DATA is an array with N elements. This algorithm sorts the elements in DATA.

```
1.  Repeat step 2 & 3 for K= 1 to N-1.
2.        Set PTR = 1
3.        Repeat while PTR<= N-K
             (a) if DATA  [PTR] > DATA [PTR+1], then
                     Interchange DATA[PTR] and DATA [PTR+1]
             (b) Set PTR = PRT+1
           [End of inner loop]
    [End of step 1 outer loop]
4.  Exit.
```

Exercise:

(a) Implement the above algorithm in C Language

.

(b) Run your algorithm with some examples.

Lab # 8

Object:
Performing matrix multiplication.

Theory:
Suppose A is an m x p matrix and suppose B is a p x n matrix. The product of A and B written AB is then m x n matrix C whose ijth element Cij is given by P

$$C_{ij} = A_{i1} B_{1j} + A_{i2} B_{2j} + \ldots\ldots\ldots + = \sum_{K=1}^{P} A_{ik} B_{kj}$$

i.e. Cij is equal to the scalar product of row i of A and Column j of B.

Algorithm:
Let A be an M x P matrix array and let B be P x N matrix array. This algorithm stores the product A & B is an M x N matrix array C.

```
1.  Repeat step 2 to 4 for   I = 1 to M.
2.        Repeat step 3 & 4 for J = 1 to N.
3.                  Set    C[ I, J] =0.
4.                  Repeat for       K = 1 to P.
                              C[I, J] =  C[I, J] + A[I, K] * B[K, J]
                    [End of inner loop]
          [End of step 2 middle loop]
    [End of step 1 outer loop]
5.  Exit.
```

Exercise:

(a) Implement the above algorithm in C Language

(b) Run your algorithm with some examples.

Lab # 9

Object:

Stack – PUSH and POP.

Theory:

A stack is a linear list of elements in which elements may be inserted or deleted at only one end, called the **top**. It means that elements are removed from a stack in the reverse order of that in which they were inserted into the stack. This is the reason stacks are called last-in first-out **(LIFO)**.

Special terminology is used for two basic operations associated with stacks:

a) "PUSH" is the term used to insert an element in the stack.
b) "POP" is the term used to delete an element in the stack.

Stack may be represented in the computer in various ways, usually by means of a one-way list or a linear array.

In the algorithms stack will be maintained by a linear array STACK, a pointer variable TOP which contains the location of the top element of stack and a variable MAXSTR which gives maximum number of the elements that can be held by the stack.

The condition TOP = 0 or TOP = NULL will indicate that stack is empty.

Algorithm:

PUSH (STACK, TOP, MAXSTR, ITEM)

```
1. [Stack already filled?]
   If TOP = MAXSTR, then: PRINT: OVERFLOW and return.
2. Set TOP := TOP + 1. [Increases TOP by 1].
3. Set STACK[TOP] := ITEM. [Inserts ITEM in new TOP position].
4. Return.
```

POP (STACK, TOP, ITEM)

```
1. [Stack has an item to be removed?]
   If TOP = 0, then: PRINT: UNDERFLOW and return.
2. Set ITEM := STACK[TOP]. [Assigns TOP element to ITEM].
3. Set TOP := TOP -  1. [Decreases TOP by 1].
4. Return.
```

Exercise:

(b) Implement the above algorithm in C Language

(b) Run your algorithm with some examples.

Lab # 10

Object:
Queues – INSERT and DELETE.

Theory:
A queue is a linear list of elements in which deletions can take place at only one end, called the **front** and insertions can take place only at other end called **rear**. This is the reason queues are called first-in first-out **(FIFO)**. Queue may be represented in the computer in various ways, usually by means of a one-way list or a linear array.

In the algorithms queue will be maintained by a linear array QUEUE and two pointer variables FRONT; containing the location of the front element in queue and REAR which contains the location of the rear element of queue. The condition FRONT = NULL will indicate that queue is empty.

Algorithm:
QINSERT (QUEUE, N, FRONT, REAR, ITEM)

```
1. [Queue already filled?]
   If FRONT = 1 and REAR = N or if FRONT = REAR + 1 then:
       Write: OVERFLOW and Return.
2. [Find new value of REAR].
   If FRONT := NULL then [Queue initially empty]
       Set FRONT :=1 and REAR := 1
   Else if REAR :=N then
       Set REAR := 1
   Else
       Set REAR : = REAR + 1
   [End of If structure]
3. Set QUEUE [REAR] := ITEM. [Inserts new element].
4. Return.
```

QDELETE (QUEUE, N, FRONT, REAR, ITEM)

```
1. [Queue already empty?]
   If FRONT = NULL  then : Write: UNDERFLOW and Return.
2. Set ITEM  := QUEUE[FRONT}
3. [Find new value of FRONT].
   If FRONT := REAR then: [Queue has only one element to start]
               Set FRONT := NULL  and REAR := NULL
   Else if FRONT  :=  N then
               Set FRONT  := 1
   Else
               Set FRONT : = FRONT + 1
   [End of If structure]
4. Return.
```

Exercise:

(a) Implement the above algorithm in C Language

(b) Run your algorithm with some examples.