ISBN: 978-93-91932-37-4

ALGORITHMS FOR LINEAR DATA STRUCTURES

Dr. Shilpa Sharma

Editor
Dr. Anita Jeph

INSPIRA
JAIPUR - INDIA

ALGORITHMS FOR LINEAR DATA STRUCTURES
Dr. Shilpa Sharma

ALGORITHMS FOR LINEAR DATA STRUCTURES

DR. SHILPA SHARMA
Associate Professor-Senior Scale
Department of Computer Applications
Manipal University Jaipur
Jaipur, Rajasthan, India

INSPIRA™
Reg. No. SH-481 R- 9-V P-76/2014
JAIPUR • DELHI (INDIA)

Published by
INSPIRA
25, Modi Sadan, Sudama Nagar
Tonk Road, Jaipur-302018
Rajasthan, India

ISBN: 978-93-91932-37-4

Edition: 2022

Price: Rs. 550/-

Laser Type Setting by
INSPIRA
Tonk Road, Jaipur
Ph.: 0141-2710264

Printed at
Shilpi Computer and Printers
Jaipur

PREFACE

This book provides a comprehensive introduction to the systematic study of algorithms for linear data structures. It presents many algorithms and covers linear data structures such as stack, queues and linked lists and makes their design and analysis accessible to all levels of readers. Author has tried to keep explanations elementary without sacrificing depth of coverage or mathematical rigor. Each chapter presents an algorithm for each kind of data structure with the algorithm for their applications. Algorithms are described in English and in a "pseudocode" designed to be readable by anyone who has done a little programming. The text is intended primarily for use in undergraduate or graduate courses in algorithms or data structures

Dr. Shilpa Sharma

ACKNOWLEDGEMENT

I thank all who in one way or another contributed to the completion of this book. First, I give thanks to God for the ability to do work.

I am so grateful to the Manipal University Jaipur leadership for providing the constant support while writing this book. My special and heartily thanks to my supervisor, Professor (Dr.) Maya Ingle who encouraged and directed me. I am also deeply thankful to my colleagues Dr. Vaibhav Bhatnagar and Dr. Linesh Raja to bring this work towards a completion. I am also so thankful to my fellow students whose challenges and productive critics have provided new ideas to the work.

I also thank my family who encouraged me and prayed for me throughout the time of my writing.

Dr. Shilpa Sharma

CONTENTS

♦❒♦

1

STACKS AND QUEUES

1. Algorithm to push and pop operations on stack

Push (Stack[MAX], ITEM)

1. Start
2. Check overflow condition

   if(TOS== MAX-1)

   Print: Stack Overflows and Exit
3. Set TOS=TOS+1
4. Set Stack[TOS]= ITEM
5. Exit

Pop (Stack [MAX])

1. Start
2. Check underflow condition

   if (TOS==-1)

   Print: Stack underflows and Exit

   Else

   Set ITEM= Stack(TOS)
3. Set TOS= TOS-1
4. Return ITEM
5. Exit

2. Algorithm for Infix to Postfix

Postfix(Q,P) [Where Q is infix Expression and P is equivalent expression]

1. Start
2. Scan Q from Left to Right and repeat steps 3 to 6 for each element of Q until the stack is

   empty.
3. If an operand is encountered, add it to P.
4. If left parenthesis is encountered, push it onto the stack.
5. If an operator × is encountered and Stack is empty then

   Push × onto the Stack.

   (end if)

   else

   Repeatedly pop from Stack and add to P each operator (i.e. from TOS) which has same orhigher precedence than × and Push lower precedence operator onto the Stack.
6. If a right parenthesis is encountered then
   a. Repeatedly pop from Stack and add to P each operator (i.e. from TOS) until the left

      parenthesis is encountered.
   b. Remove the right parenthesis.(end if)

end of Step 2 loop.

7. Exit.

3. Algorithm for Infix to Prefix

Prefix(Q,P) [Where Q is infix Expression and P is equivalent expression]

1. Start
2. Reverse the input string.
3. If an operand is encountered, add it to P.
4. If closing parenthesis is encountered, push it onto the stack.
5. If an operator × is encountered, then
   a. If Stack is empty then push operator × onto the Stack.
   b. If TOS is closing parenthesis push operator × onto the Stack.
   c. If operator × has same priority than TOS, push operator × onto the stack

      else

      Pop the operator × from the Stack and add to P.
6. If an opening parenthesis is encountered, Pop operators from the Stack and add them

   to P until closing parenthesis is encountered and discard the closing parenthesis.
7. If there are no input, unstack the remaining operators and add them to P.
8. Reverse the P.
9. Exit

4. Evaluation of Postfix Expression Algorithm

1. Start

2. Scan P from Left to Right and Repeat steps 3 to 4 for each element in P till end of P.

3. If an operand is encountered, Push it onto the Stack.

4. If an operator × is encountered then

   a. Remove the two elements A and B from the Stack, Where A is TOS and B is TOS-1 .

   b. Evaluate B X A

   c. Push the result of b step onto the Stack.

Endif

End of Step 2 loop

5. Set the resultant value to the TOS of the Stack.

6. Exit.

5. Algorithm for linear queue operations

Insert (Queue [MAX], ITEM)

1. Start
2. Initialize Front =-1 and Rear=-1
3. If (Rear== MAX-1) then

Print: Queue overflows and return. else

Set Rear= Rear+1

Set Queue[Rear]= ITEM

4. If (Front ==-1) then

Set Front =0

5. Exit

```
Delete (Queue[MAX])
1.      Start
2.      If (Front == -1) then
          Print: Queue is empty and Return
        else
        Set ITEM= Queue[Front] Set Queue[Front]=0
3.      If (Front==Rear) then
          Set Front=-1 Set Rear =-1 else
          Set Front= Front+1
4.      Return the deleted ITEM
5.      Exit
```

6. Algorithm for Circular Queue Operations

InsertCQ (Queue[MAX], ITEM)

1. Start
2. Initialize Front =-1 and Rear=-1
3. If (((Rear== MAX-1) AND (Front==0)) OR

(Rear+1= Front)) then

Print: Queue overflows and return.

4. If (Rear== MAX-1) then

Set Rear= 0

else

Set Rear= Rear+1

5. Set Queue[Rear]= ITEM
6. If (Front ==-1) then

Set Front =0

7. Exit

DeleteCQ (Queue[MAX])

1. Start
2. If (Front == -1) then

   Print: Queue is empty and Return
3. Set DATA= Queue[Front]
4. Set Queue[Front]=0
5. If (Front==Rear) then

   Set Front=-1 Set Rear =-1 else

   If (Front=MAX-1) then Set Front=0

   Else

   Set Front= Front+1

   endelse
6. Return the deleted DATA
7. Exit

7. Algorithm for Priority Queue Operations

1. START
2. Declare Struct Pque q
3. Declare Struct Data dt, TEMP
4. Declare variable int i, j and initialize j=0
5. Call Procedure InitPQueue(&q)
6. Repeat Step 7 to Step 10 until i< MAX
7. Read dt.job
8. Read dt. prno
9. Set dt.ord= j+1
10. Call Procedure AddPQueue(&q, dt)
11. Repeat Step 12 to Step 14 until i< MAX
12. Temp= Call Procedure DelPQueue(&q)
13. Print: TEMP.job
14. Print: TEMP.prno
15. STOP

InitPQueue(Struct pque pq)

1. START
2. Initialize variable int i=0
3. Initialize pq->Front= -1 and pq->Rear=-1
4. Repeat Step 5 to Step 8 until i<MAX
5. Initialize pq-> d[i].job to NULL
6. Set pq->d[i].prno=0
7. Set pq->d[i].ord=0
8. Set i=i+1
9. STOP

```
AddPQueue (Struct Pque pq, Struct data dt )
1.      START
2.      Declare Struct Data TEMP
3.      Declare varibles int i, j
4.      If (pq->Rear== MAX-1) then
Print: Queue is full
else
Set pq->Rear= pq-> Rear+1
5.      Set pq-> d[pq->Rear]= dt
6.      If (pq-> Front== -1) then
Set pq-> Front=0
7.              Initialize i= pq-> Front and Repeat Step 8 to 19 until i<= pq->Rear
8.              Initialize j= i+1 and Repeat Step 9 to 18 until j <= pq->Rear
9.      If (pq->d[i].prno > pq->d[j].prno) then
10.     Set TEMP= pq->d[i]
11.     Set pq-> d[i]= pq-> d[j] 12.Set pq-> d[j]= TEMP else
13.     if (pq->d[i].prno == pq->d[j].prno) then
14.     if (pq->d[i].ord > pq->d[j].ord) then
15.     Set Set TEMP= pq->d[i]
16.     Set pq-> d[i]= pq-> d[j]
17.     Set pq-> d[j]= TEMP
Endif of Step 14
Endif of Step 13
endelse
18.     Set j=j+1 Endloop of Step 8
19.     Set i=i+1 Endloop of Step 7
20.     STOP
```

DelPQueue(Struct pque pq)

1. START
2. Declare Struct Data t
3. Initialize t.job to NULL
4. Set t.prno= 0
5. Set t.ord = 0
6. If (pq->Front== -1) then Print: Queue is empty. else

Set t= pq->d [pq-> Front]

if (pq-> Front== pq-> Rear) then

Set pq-> Front= -1 Set pq-> Rear=-1 else

Set pq-> Front= pq-> Front+1

7. Return t
8. STOP

2

LINKED LIST

8. Algorithm to Append the LL

```
Append (START, ITEM)
1.      Start
2.      (if the LL is empty, create the first node)
         if (Start== NULL) then
        TEMP= new node
        Set Temp-> data= ITEM Set Temp-> link= NULL Set Start= TEMP
        else
        Set TEMP= Start
3.      Repeat step 4 till TEMP->link!= NULL
4.      TEMP= TEMP-> link
5.      (If the LL already exists)
NEW= new node
6.      Set NEW-> data= ITEM
7.      Set NEW-> link= NULL
8.      Set TEMP-> link= NEW
9.      Stop
```

9. Algorithm to Add node at beginning of the LL

AddatBeg (START, ITEM)

1. Start
2. (Creating the node to insert at the beginning of LL i.e. TEMP)
   TEMP= malloc (Sizeof (Struct node))
3. Set TEMP-> data= ITEM
4. Set TEMP-> link= START
5. Set START= TEMP
6. Stop

10. Algorithm to Add node at specific location in the LL

Addafter (START, LOC, ITEM)

1. Start
2. Set TEMP= START
3. Initialize the counter variable i =0
4. Repeat step 5 till i < loc
5. Set TEMP= TEMP-> link and i= i+1
6. (Create a new node to insert after location reached)

NEW = malloc (Sizeof (Struct node))

7. Set NEW-> data= ITEM
8. Set NEW-> link= TEMP-> link
9. Set TEMP-> link = NEW
10. Stop

11. Algorithm for deletion of a node from LL

```
Del (START, ITEM)
1.    Start
2.    Initialize TEMP= START
3.    Repeat Step 4 till TEMP!= NULL
4.    if (TEMP-> data== ITEM) then
(if node to be deleted is the first node of LL)
if (TEMP== START) then Set START= TEMP-> link else
(deletes the intermediate node of LL)
Set OLD-> link= TEMP->link
free (TEMP)
else
(traverse the LL till last node reached) Set OLD= TEMP
Set TEMP= TEMP-> link
5.    Stop
```

12. Algorithm for Linked Implementation of Stack

LinkedPush (TOP, ITEM)

1. Start
2. TEMP= malloc (sizeof(Struct node))
3. If (TEMP == NULL) then

   Print: Stack is Full.
4. Set TEMP->data= ITEM
5. Set TEMP-> link = TOP
6. TOP= TEMP
7. STOP

LinkedPop (TOP)

1. Start
2. If(TOP== NUll) then

   Print: Stack is empty.
3. Set TEMP= TOP
4. Set ITEM= TEMP-> data
5. Set TOP= TOP-> link
6. Free (TEMP)
7. Return ITEM
8. STOP

13. Algorithm for Linked Implementation of Queue

LinkedAddQ (q, ITEM)

1. Start
2. Declare node TEMP
3. TEMP= malloc (Sizeof(Struct node))
4. If (TEMP== NULL) then Print: Queue is full
   Set TEMP-> data= ITEM
   Set TEMP-> link= NULL
5. If (q-> Front== NULL) then
   Set q-> Rear= q-> Front= TEMP
6. else
7. Set q-> Rear-> link = TEMP
8. Set q-> Rear= q-> Rear-> link
9. Stop

LinkedDelQ (q)

1. Start
2. Declare node TEMP
3. If (q-> Front== Null) then
   Print: Queue is empty.
4. Set ITEM= q->Front->data
5. Set TEMP= q->Front
6. Set q-> Front= q-> Front->link
7. Free (TEMP)
8. Return ITEM
9. Stop

14. Algorithm for Reverse of the LL

Reverse (START)

1. Start
2. Declare node q, r and s
3. Initialize q= START and r= NULL
4. Repeat Step 5 to 8 till q!= NULL
5. Set s= r
6. Set r= q
7. Set q= q->link
8. Set r-> link = s
9. Set START= r
10. Stop

15. Algorithm for Merging of the LL

Merge (START)

1. Start
2. Declare Z and Initialize Third= Null, p= First and q= Second
3. (if both the lists are empty)

if ((First== NULL) AND (Second == NULL)) then

return NULL and Exit

4. (Traverse both LL till end. If any one LL is reached, loop is terminated)

Repeat Step 5 to 6 till First != NULL and Second != NULL

5. (If node being added in LL is first node) if (Third == NULL) then

Third= malloc (Sizeof (Struct node))

Set Z= Third

else

Set Z-> link= malloc (Sizeof (Struct node))

Set Z= Z-> link

6. If (First-> data < Second-> data) then

Set z-> data = First-> data

Set First = First-> link

else

If (Second-> data < First-> data) then Set z-> data = Second-> data

Set Second = Second-> link

endelse

endelse Endloop of Step 4

```
else
 if (First-> data== Second-> data) then Set Z-> data= Second-> data
Set First = First-> link
Set Second= Second-> link endif
 endelse
endelse Endloop of Step 4
7. Repeat Step 8 to 11 till (First != NULL)
(if the end of first list not reached)
8. Z-> link = malloc (Sizeof (Struct node))
9. Set Z = Z-> link
10. Set Z-> data= First -> data
11. Set First = First-> link
Endloop of Step 7
12. Repeat Step 13 to 16 till (Second != NULL)
(if the end of second list not reached)
13. Z-> link = malloc (Sizeof (Struct node))
14. Set Z = Z-> link
15. Set Z-> data= Second -> data
16. Set Second = Second-> link
Endloop of Step 12
17.    Set  Z-> link = NULL
18.    Stop
```

16. Algorithm for Sorting of the LL

Sort (n)

1. Start
2. Initialize int variable i=0, j=1 and temp
3. Initialize nodes p and q
4. Initialize k=n
5. Repeat Step 6 to 13 till i< n-1
6. Set p = START
7. Set q= p->link
8. Repeat steps 9 to 12 till j< k
9. If (p-> data > q -> data) then

Set temp= p->data Set p-> data = q->data Set q->data= temp

endif

10 Set p= p-> link 11 Set q= q->link

12 Set j= j+1

Endloop of Step 8

13 Set i= i+1 and k= k-1

Endloop of Step 5

14. Stop

17. Algorithm to Append the DLL

DAppend (START, NUM)

1. Start

2. (if DLL is empty)

if (START== NULL) then

START= malloc (Sizeof (Struct dnode))

Set START-> Prev= NULL

Set START-> data= NUM Set START-> Next= NULL

else

3. (Traverse the DLL till the last node is reached)

Repeat Step 4 till TEMP-> Next!= NULL

4. TEMP= TEMP-> Next

5. (Add new node at the end)

NEW= malloc (Sizeof (Struct dnode))

6. Set NEW-> data= NUM

7. Set New-> Next= NULL

8. Set NEW-> Prev= TEMP

9. Set TEMP-> Next= NEW

10. Stop

18. Algorithm to Add node at beginning of DLL

DAddatBeg (START, NUM)

1. Start
2. (Creating the node to insert at the beginning of LL i.e. TEMP)

TEMP= malloc (Sizeof (Struct dnode))

3. Set TEMP-> Prev= NULL
4. Set TEMP-> data= NUM
5. Set TEMP-> Next= START
6. Set START-> Prev= TEMP
7. Set START= TEMP
8. Stop

19. Algorithm to Add node at specific location in DLL

DAddafter (START, LOC, NUM)

1. Start
2. Initialize the counter variable i =0
3. (Skip to desired postion)

Repeat step 4 to 6 till i < loc

4. Set START= START-> Next
5. (if end of DLL is encountered)

if (START== NULL) then

Print: There are less number of elements and return

endif

6. Set i= i+1

endloop of Step 3

7. (Insert a new node) START= START-> Prev

Set TEMP = malloc (Sizeof (Struct dnode))

8. Set TEMP-> data= NUM
9. Set TEMP-> Prev= START
10. Set Temp-> Next= START-> Next
11. Set TEMP-> Next-> Prev= TEMP
12. Set START-> Next= TEMP
13. Stop

20. Algorithm for deletion of a node from DLL

DDel (START)

1. Start
2. Initialize TEMP= START
3. Repeat Step 4 to 5 till TEMP!= NULL
4. (if node to be deleted is found) if (TEMP-> data== NUM) then

(if node to be deleted is the first node of DLL)

if (TEMP== START) then Set START= START-> Next Set START-> Prev= NULL

endif

else

(if the node to be deleted is the last node of DLL) if (TEMP-> Next == NULL) then

Set TEMP-> Prev-> Next == NULL

endif else

(deletes the intermediate node of dLL) Set TEMP-> Prev-> Next = TEMP-> Next Set TEMP-> Next-> Prev= TEMP-> Prev endelse

free (TEMP) endelse

5. Set TEMP= TEMP-> Next
6. Stop

21. Algorithm to Append the CLL

```
CAppend (FRONT, REAR, ITEM)
1.    Start
2.    (Create a new node)
TEMP= malloc (Sizeof (Struct node))
3.    TEMP-> data = ITEM
4.    (if the CLL is empty)
if (FRONT == NULL) then
FRONT= TEMP
else
REAR-> link= TEMP
5.    REAR= TEMP
6.    REAR->link= FRONT
7.    Stop
```

22. Algorithm for deletion of a node from CLL

CDel (FRONT, REAR)

1. Start
2. Declare ITEM variable
3. (if CLL is empty)

if (FRONT== NULL) then

Print: CLL is empty.

else

if (FRONT== REAR) then

Set ITEM= FRONT-> data

free (FRONT)

Set FRONT= NULL

Set REAR = NULL

else

(delete the node)

Set TEMP= FRONT

Set ITEM= TEMP-> data Set FRONT= FRONT-> link Set REAR-> link= FRONT free (TEMP)

4. Return the ITEM
5. Stop

23. Algorithm for Polynomial Creation

Poly_append(START, x, y)

1. Start
2. Declare polynode TEMP
3. Initialize TEMP= START
4. Create a new polynode if the list is empty

if (START== NULL) then

START= malloc (sizeof(struct polynode))

TEMP= START

else

Repeat the step 5 till TEMP->link!= NULL

5. Set TEMP= TEMP-> link
6. Set TEMP-> link = malloc (sizeof(struct polynode))
7. Set TEMP= TEMP-> link

Endelse

8. Set TEMP-> coeff= x
9. Set TEMP-> exp= y
10. Set TEMP-> link= NULL
11. STOP

24. Algorithm for Polynomial Addition

Poly_add(First, second, Total)

1. Start

2. Declare polynode Third

3. If both the lists are empty

if (First== NULL and Second == NULL) then return

4. Repeat step 5 to 11 to till First!= NULL and Second

!= NULL

5. If (Third == NULL) then

Third = malloc (sizeof(struct polynode))

Set Third= Total

endif

6. else

Third->link= malloc (sizeof(struct polynode)) Third= Third->link

Endelse step 6

7. If (First->exp < Second->exp) then Set Third->coeff = Second->coeff Set Third->exp = Second-> exp Set Second = Second-> link

endif step 7

8. else

9. If (First->exp > Second->exp) then Set Third->coeff = First->coeff Set Third->exp = First-> exp

Set First = First-> link

endif step 9

10. else

```
11.   If (First->exp == Second->exp) then
Set Third->coeff = First->coeff + Second->coeff
Set Third->exp = First->exp
Set First = First-> link
Set Second = Second-> link
endif step 11
endelse step 10
endelse step 8
12.   Endloop of step 4
13.   Repeat step 14 to 18 till First!= NULL
14.   If (Third== NULL) then
Set Third = malloc (sizeof(struct polynode))
Set Third= Total
endif
15.   else
Third-> link = malloc (sizeof(struct
polynode)) Third= Third->link endelse
16.   Set Third->coeff = First->coeff
17.   Set Third->exp = First-> exp
18.   Set First = First-> link
Endloop of step 13
```

19. Repeat step 20 to 24 till Second!= NULL

20. If (Third== NULL) then

Set Third = malloc (sizeof(struct polynode))

Set Third= Total

endif

21. else

Third-> link = malloc (sizeof(struct

polynode)) Third= Third->link endelse

22. Set Third->coeff = second->coeff

23. Set Third->exp = Second-> exp

24. Set Second = Second-> link

Endloop of step 19

25. Third->link= NULL

26. STOP

25. Algorithm for Polynomial Multiplication

Poly_multiply(First, Second, Mult)

1. Start
2. Declare polynode Second1
3. Declare coeff1 and exp1
4. Initialize Second1 = Second
5. if (First== NULL and Second == NULL) then return
6. if (First== NULL) then Set Mult= Second
7. else
8. if (Second==NULL)

Set Mult= First

9. else
10. Repeat step 11 to 18 till First!= NULL
11. Repeat step 12 to 15 till Second!=NULL
12. Set coeff1 = First-> coeff* Second->coeff
13. Set exp1= First-> exp+ Second-> exp
14. Second= Second-> link
15. Call Procedure Padd (coeff1, exp1, Mult)
16. Endloop of step 11
17. Set Second= Second1
18. First = First-> link
19. Endloop of step 10
20. Endelse of step 9
21. Endelse of Step 7
22. Stop

```
Padd (c, e, START)
1.   Start
2.   Declare Polynode r, TEMP= START
3.   If (START== NULL OR (e> START-> exp)) then
4.   Set r = malloc (sizeof (Struct polynode))
5.   Set START= r
6.   Set START-> coeff= c
7.   Set START-> exp= e
8.   Set START-> link= TEMP endif
9.   else
10.  Repeat step 11 to 13 till TEMP!= NULL
11.  If (TEMP-> exp== e) then
Set TEMP-> coeff = TEMP->coeff +c
return
endif of step 11
If (TEMP-> exp>e AND (TEMP-> link-> exp < e OR TEMP-> link== NULL)) then
Set r = malloc(sizeof(Struct polynode))
Set r-> coeff = c
Set r-> exp= e
Set r->link= TEMP->link Set TEMP-> link= r return
endif of step 12
12.  Set TEMP= TEMP-> link
Endloop of step 10
13.  Set r-> link= NULL
14.  Set TEMP->link = r
Endelse of step 9
15.  Stop
```

3

SEARCHING AND SORTING

26. Algorithm for Sequential Search

```
Seq Search (ARR, n, ITEM)
1.    Start
2.    Initialize i=0
3.    Repeat Step 4 till i<=n-1
4.    If (ARR[i]== ITEM) then
break
Set i=i+1
endloop of Step 3
5.    If (i==n) then
Print: Number is not present in an array
else
Print: Number is present at ith position in an array
6.    Stop
```

27. Algorithm for Binary Search

Binary Search (ARR, n, ITEM)

1. Start
2. Declare mid
3. Initialize lower= 0, upper=n-1, flag=1
4. Initialize mid= (lower+ upper)/2
5. Repeat Step 6 till lower<= upper
6. If (ARR[mid]== ITEM) then

Print: Number is present at mid position

Set flag=0

endif

7. If (ARR[mid]> ITEM) then upper= mid-1

else

lower=mid+1

8. if (flag) then

Print: Number is not present in an array

9. Stop

28. Algorithm for Interpolation Search

Interpolation Search(ARR, n, ITEM)

1. Start
2. Declare mid
3. Initialize lower= 0, upper=n-1, flag=1
4. Initialize mid= lower+ ((upper-lower)*(ITEM-ARR[lower])) / (ARR[upper]-ARR[lower])
5. Repeat Step 6 till lower<= upper
6. If (ARR[mid]== ITEM) then

Print: Number is present at mid position

Set flag=0

endif

7. If (ARR[mid]> ITEM) then upper= mid-1

else

lower=mid+1

8. if (flag) then

Print: Number is not present in an array

9. Stop

29. Algorithm for Bubble Sort

```
Bubble Sort (ARR)
1.    Start
2.    Declare i, j, temp, n
3.    Initialize i=0 and repeat Step 4 to 5 till i<=n-2
4.    Initialize j=0 and repeat Step 5 till j<= (n-2)-i
5.    If (ARR[j]> ARR[j+1]) then
Set temp= ARR[j]
Set ARR[j]= ARR[j+1]
Set ARR[j+1]= temp
endif
Set j= j+1 endloop of Step 4 Set i=i+1
endloop of Step 3
6.    Stop
```

30. Algorithm for Selection Sort

Selection Sort (ARR)

1. Start
2. Declare i, j, temp, n
3. Initialize i=0 and repeat Step 4 to 5 till i<=n-2
4. Initialize j=i+1 and repeat Step 5 till j< n
5. If (ARR[i]> ARR[j]) then

Set temp= ARR[i]

Set ARR[i]= ARR[j]

Set ARR[j]= temp

endif

Set j= j+1 endloop of Step 4 Set i=i+1

endloop of Step 3

6. Stop

31. Algorithm for Insertion Sort

Insertion Sort (ARR)

1. Start
2. Declare i, j, k, temp, n
3. Initialize i=1 and repeat Step 4 to 11 till i<=n-1
4. Initialize j=0 and repeat Step 5 to 10 till j<i
5. If (ARR[j]> ARR[i]) then

Set temp= ARR[j] Set ARR[j]= ARR[i]

6. Initialize k=i and repeat steps 7 and 8 till k>j
7. Set ARR[k] = ARR[k-1]
8. Set k= k-1

endloop of Step 6

9. Set ARR[k+1]= temp

endif

10. Set j=j+1

Endloop of Step4

11. Set i=i+1

endloop of Step 3

12. Stop

32. a. Algorithm for Quick Sort

Quick Sort (arr, lower, upper)

1. Start
2. Declare i, n
3. If (upper> lower) then

Call procedure Split // for dividing array

3.a i= Split (arr, upper, lower)

Call QuickSort recursively for first half array

3b. QuickSort(arr, lower, i-1)

call QuickSort recursively for second half array

3C. QuickSort(arr, i+1, upper) endif

4. Stop

32.b. Algorithm for Split of Quick Sort

Split (arr, lower, upper)

1. Start
2. Declare i, p, q, t as integer variables
3. Set p= lower+1
4. Set q= upper
5. Set i= arr[lower]
6. Repeat Step 7 to 11 till q>=p
7. Repeat Step 8 till arr[p] < i
8. Set p= p+1 Endloop of Step 7
9. Repeat Step 10 till arr[q] > i
10. Set q= q-1 Endloop of Step 9
11. If (q>p) then

Set t=arr[p]

Set arr[p]= arr[q] set arr[q]= t endif

Endloop of Step 6

12. Set t= arr[lower]
13. Set arr[lower]=arr[q]
14. Set arr[q] = t
15. Return q to Step 3a.
16. Stop

33. Algorithm for Merge Sort

```
MergeSort (arr, brr, crr)
1.  Start
2.  Declare i, j, k, temp, n and MAX
3.  Initialize i= 0 and repeat step 4 to 6 till i<=n-2
4.  Initialize j= i+1 and repeat step 5 to 6 till j<=n-1
5.  If (arr[i]> arr[j]) then Set temp =arr [i] Set arr[i]= arr[j] Set arr[j]= temp
Endif
6.  If (brr[i]> brr[j]) then
Set temp brr [i] Set brr[i]= brr[j] Set brr[j]= temp
Endif, Set j=j+1
Endloop of step 4, set i=i+1
Endloop of step 3
7.  Initialize i=j=k=0 and repeat step 8 to 9 till i<= MAX
8.  If (arr[j]<= brr[k]) then Set crr[i++]= arr[j++] else
Set crr[i++]= brr[k++]
Endif
9.  If (j==n OR k==n) then
break
Endloop of Step 7
10. Repeat Step 11 till j<=n-1
11. Set crr[i++]= arr[j++] endloop 10
12. Repeat Step 13 till k<=n-1
13. Set crr[i++]= brr[k++] endloop 12
14. STOP
```

34. Algorithm for Radix Sort

1. Start
2. Find the largest element of the array
3. Find the total number if digits num in the largest digit.

Set digit= num

4. Repeat step 5 and 6 for pass 1 to num
5. Initialize buckets

for i=1 to (n-1)

Set num = obtain digit number pass of arr[i]

end of for loop

6. Calculate all numbers from the buckets in order
7. Exit

35. Algorithm for Shell Sort

```
Shell Sort (ARR)
1.  Start
2.  Declare i, j, TEMP and initialize increment= 3
3.  Repeat steps 4 to 11 till increment> 0
4.  Set i =0 and Repeat steps 5 to 10 till i< MAX
5.  Set j= i
6.  Set TEMP= ARR[i]
7.  Repeat step 8 to 9 till ((j> Increment) AND (ARR[j- increment] > TEMP))
8.  ARR[j]= ARR[j- increment]
9.  Set j= j-increment
10. ARR[j] = TEMP
11. If (increment/2 !=0) then
Set increment = increment/2
Else
if( increment== 1) then
Set increment =0 else
Set increment= 1
12. Stop
```

36. Algorithm for Heap Sort

1. The user inputs the size of the heap(within a specified limit).The program generates a corresponding binary tree with nodes having randomly generated key Values.

2. Build Heap Operation: Let n be the number of nodes in the tree and i be the key of a tree. For this, the program uses operation Heapify. when Heapify is called both the left and right subtree of the i are Heaps. The function of Heapify is to let i settle down to a position(by swapping itself with the larger of its children, whenever the heap property is not satisfied)till the heap property is satisfied in the tree which was rooted at (i).

3. Remove maximum element: The program removes the largest element of the heap(the root) by swapping it with the last element.

4. The program executes Heapify(new root) so that the resulting tree satisfies the heap property.

5. Goto step 3 till heap is empty.

Dr. Shilpa Sharma is presently working as Associate Professor- Senior Scale in the department of Computer Applications and Deputy Director Quality and Compliance, at Manipal University Jaipur. She is PhD (Computer Science) in Ontology Based Software Engineering. Her area of specialization includes software engineering, data structures and algorithms and Artificial Intelligence. She has around 16+ years of rich experience in academics and 9+ years of post-PhD experience. She has more than more than 25 publications in International / national journals/conference proceedings. Her research interests include Software Engineering, AI and its applications and Image Processing. She has guided 3 PhD and guiding 3 PhD students in Deep Learning and Cyber Security. She is also member of reviewer board of various journals and technical program committee of several reputed conferences.

INSPIRA™

Reg. No. SH-481 R- 9-V P-76/2014

Published by:
INSPIRA
25, Sudama Nagar, Opp. Glass Factory, Tonk Road, Jaipur - 302018 (Raj.)
Phone No.: 0141-2710264 Mobile No.: 9829321067
Email: profdrssmodi@gmail.com

₹550/-

ISBN: 978-93-91932-37-4

Printed at:
Shilpi Computer and Printers
6/174, Malviya Nagar, Jaipur
Mobile No.: 92148 68868

Website : inspirajournals.com