EXPERIMENT NO: 1
SEARCH ALGORITHMS

1.1 IMPLEMENTATION OF LINEAR SEARCH

Input :- Array of numbers of size n and search element, key

Output :- The index of key element if present.

Algorithm

1. Start
2. Enter size of array, N
3. Input array elements into array 'a'
4. Input number to be searched, m
5. i=0, flag=0
6. while (i<N)
   if (a[i] == m) then
   Display ("Element found")
   Flag = 1
   break
7. If flag == 0 then
   Display ("Element not found")
8. Stop

Result

The program ran successfully and output is obtained whether key is found or not.

1.2. Implementation Of Binary Search.

Input :- Sorted array of number of size, n and the search element, Key.

Output :- Index of the key element if present.

Algorithm

1. Start
2. Enter size of array, N
3. Enter array elements in sorted manner, a
4. Enter the integer to be searched, n
5. Initialize flag = 0, first = 0, last = N-1,
   mid = (first + last) / 2
6. while (first <= last)
   if (a[mid] == n) then
   mid = (first + last) / 2
   flag = 1
   break
   else if (a[mid] < n) then
   first = mid + 1
   mid = (first + last) / 2
   else
   last = mid - 1
   mid = (first + last) / 2
7. If (flag == 1) then
   Display "Element found at mid + 1"

else

Display ("Element not found")

8. Stop.

Result.

Program ran successfully and output is obtained whether the search element is present or not.

EXPERIMENT NO: 2
SORTING ALGORITHMS

2-1. IMPLEMENTATION OF BUBBLE SORT

Input :- Array of n elements
Output :- Sorted array.

Algorithm

1. Start
2. Input size of array, N
3. Input array elements, a
4. Bubble sort ()
5. Display the sorted array
6. Stop

Bubble sort : Algorithm

1. for i = 0 to n-1, i++
   for j = 0 to n-i-1; j++
   if (a[j] > a[j+1]) then
   temp = a[j]
   a[j] = a[j+1]
   a[j+1] = temp.
2. Display the sorted array

Result

Program ran successfully & output is obtained.

8.2. IMPLEMENTATION OF INSERTION SORT

Input: An array of 'n' element
Output :- Sorted array.

Algorithm

1. Start
2. Input size of the array, N
3. Input array elements, a
4. Insertion sort()
5. Display sorted array
6. Stop.

Algorithm of Insertion Sort Function

1. for i=1 to n-1; i++
   temp = a[i]
   j = i-1
   while (j>=0 and a[j] > temp)
   a[j+1] = a[j]
   j = j-1
   a[j+1] = temp

Result

Program ran successfully and sorted array obtained.

2.3 IMPLEMENTATION OF SELECTION SORT

Input :- An array of n elements
Output :- Sorted array

Algorithm

1. Start
2. Input size of array, N
3. Input array elements, a
4. Selection sort ()
5. Display the sorted array
6. Stop.

Selection Sort Function - Algorithm

```
1. for i=0 to n-1 ; i++
       min=i
       for j=i+1 to n-1 ; j++
           if ( a[j] < a[min] ) then
                   min=j
       end for
       Swap a[min] and a[j]
   end for.
```

Result

Program ran successfully and sorted array is obtained.

EXPERIMENT No: 3
POLYNOMIAL ADDITION

Write a program to read two polynomials and store them in an array. Calculate sum of the two polynomials and display the 1st and 2nd polynomial and the resultant polynomial.

Input :- Two polynomials
Output :- Sum of the 2 polynomials.

Algorithm

1. Start.
2. Input two polynomials and store them in array of structures P_1 and P_2
3. Read the 2 polynomials
4. Let P_3 be an array for holding the result.
5. PolynomialAdelition()
6. Display the sum
7. Stop.

Algorithm of the PolynomialAddition Function

1. i=0, j=0, K=0
2. Traverse the two arrays P_1 & P_2 until one is empty

if ($P_1[i]$.exp == $P_2[j]$.exp) then

$P_3[K]$.coeff = $P_1[i]$.coeff + $P_2[j]$.coeff

$P_3[k].exp = P_1[i].exp$

$i = i+1;\ j = j+1;\ k = k+1;$

else if ($P_1[i].exp > P_2[j].exp$) then

$P_3[k].coeff = P_1[i].coeff$

$P_3[k].exp = P_1[i].exp$

$i = i+1;\ k+1$

else

$P_3[k].coeff = P_2[j].coeff$

$P_3[k].exp = P_2[j].exp$

$j = j+1;\ k = k+1$

3. while ($P_1 \neq$ NULL)

$P_3[k].coeff = P_1[i].coeff$

$P_3[k].exp = P_1[i].exp$

$i = i+1;\ k = k+1$

4. while ($P_2 \neq$ NULL)

$P_3[k].coeff = P_2[j].coeff$

$P_3[k].exp = P_2[j].exp$

$j = j+1;\ k = k+1$

Result

Program ran successfully and sum of polynomials obtained.

EXPERIMENT - 4

STACK USING ARRAY

Aim:- Implementation of Stack using arrays.

Input:- Array of size 'n'

Output:- Operations on Stack:

Algorithm

1. Start
2. Input the size of the stack size, n
3. Create an array of size n.
4. T=-1
5. Repeat the steps
   - 5.i) Enter the option needed 1. PUSH 2. POP. 3. EXIT
   - 5.ii) If option == 1
     PUSH ()
   - 5.iii) Else if option == 2
     POP()
   - 5.iv) Else.
     goto step 6
6. Display the content of stack
7. Stop.

Push ()

1. If (T == size - 1)
   Print "Stack Overflow".
   exit.

Else

T = T+1

Input assay elements

ass[T] = element.

Pop()

1. If (T == -1)

Display "Stack Underflow"

2. Else

delete ass[T]

T = T-1

Result

Program ran successfully and operations are done on the stack.

EXPERIMENT - 5

QUEUE USING ARRAYS

Aim:
Implementation of Queue using Arrays

Input:- Array of size 'n'.
Output:- Operations in a queue.

Algorithm

1. Start
2. Input the size of queue, n
3. Create an array size, n.
4. r = -1, f = -1
5. Repeat these steps:
   5-i) Choose an option 1. ENQUEUE 2- DEQUEUE
   5.ii) If (option == 1) then
   ENQUEUE ();
   5iii) else if (option == 2) then
   DEQUEUE ();
6. Display the queue.
7. Stop.

ENQUEUE ()
1. If (r == n-1) then
   Display "Queue is full"

goto step 5. ii)

2. Enter the element to enqueue, item.

3. if (f == r == -1) then
   f = 0

4. else.
   r = r + 1
   q[r] = item.

Dequeue ()

1. If (f == r == -1) or f == r+1)
   Display "Queue is empty".

2. else.
   item = q[f]
   f = f + 1

Result:

Program seen succesfully & operation in a queue performed

EXPERIMENT - 6

CIRCULAR QUEUE USING ARRAYS

AIM :-

Implementation of Circular queue using arrays.

Input :- Array of size 'n'

Output :- Operations on circular queue.

Algorithm

1. Start
2. Input the size of the queue, n
3. Create an array of size n
4. f = -1; r = -1
5. Repeat these steps while (s == 0)
   - 5.i) Choose an option 1 - ENQUEUE 2 - DEQUEUE
   - 5.ii) If (option == 1) then
     ENQUEUE()
   - 5.iii) If (option == 2) then
     DEQUEUE()
6. Display the queue.

ENQUEUE()

1. If ((r+1)%n == f)
   Display "Queue is full"
   Go to step 5. ii)

2. else (f == r == -1)

f = 0

r = (r+1) % l

q[r] = item.

DEQUEUE()

1. If (f == r == -1) then

Display "Queue is empty"

2. else if ((r+1) % n == f) then

item = q[f]

f = (f+1) % l

3. else if (f == r) then

item = q[f]

f = r = -1

RESULT

Program ran successfully and operations on circular queue performed.

EXPERIMENT-7

STUDENT DATA USING SINGLY LINKED LIST

AIM

Create a singly linked list to store name, roll no. and marks obtained by n students. Implement the following functions i) Insert student at front of list ii) Insert student at end of list iii) Insert student after a particular student iv) delete student at front of list v) delete student at end of list vi) delete a student having a particular roll no.

Input: Data of Student
Output: Operations on linked list.

ALGORITHM

1. Start
2. Create a structure 'student info' containing data types for name, roll no. and marks and struct student info * next.
3. struct studentinfo * start == NULL
4. Input the no. of nodes needed, n
5. From the menu choose the option needed using Switch case method 1) Create 2) Display 3) Insert at beg

4) Insert at end 5) delete beg 6) delete end
7) delete from middle 8) Insert from middle
9) Exit.

6.

6.1. If case 1 then
Create a node (create())

6.2. If case 2 then
Display student details (Display())

6.3. If case 3 then
Insert student at beginning (Insert_beg())

6.4. If case 4 then
Insert student at end (Insert_end())

6.5. If case 5 then
Delete from beginning (del_beg())

6.6. If case 6 then
Delete from end (del_end())

6.7. If case 7 then
Delete from middle of the roll no. (del_pos())

6.8. If case 8 then
Insert from middle (Insert_pos())

6.9. If case 9 then
exit.

7. Stop

Create()

1. Struct student info * student, *ptr.
2. Input name, roll no. and marks

3- Student.name = name
4. Student.roll no. = roll no.
5. student.mark = marks
6. student.next = NULL
7. If (start == NULL) then
start = student
8. else
ptr = start
while (ptr.next ≠ NULL)
ptr = ptr.next.
ptr.next = student.

Display()

1. Struct studentinfo * ptr;
2. If (start == NULL) then
Display "List is empty"
3 else
ptr = start
while (ptr ≠ NULL)
Display "ptr.name", "ptr.rollno", "ptr.marks"
ptr = ptr.next

Insertbeg()

1. struct studentinfo * student.
2 Input the info of the student (name, roll no., mark)
3. student.name = name
4. student.roll no = roll no.

5. student.mark = mark
6. student.next = NULL
7. If (start == NULL) then
   start = student
8. else
   student.next = start
   start = student.

Insert_end()

1. struct studentinfo *student, *ptr;
2. Input the data of the student (name, roll no. & mark)
3. student.name = name.
4. student.roll no = roll no.
5. student.mark = marks
6. student.next = NULL.
7. If (start == NULL) then
   start = student
   else
   ptr = start
   while (ptr.next ≠ NULL)
   ptr = ptr.next
   ptr.next = student

Delete beg()

1 structure studentinfo *ptr

2. If (ptr == NULL) then
   Display "List is empty"
else
   ptr = start
   start = start.next
   free(ptr)

Delete_end()

1 structure studentinfo * student *ptr
2. If (start == NULL) then
   Display "List is empty"
3. else if (start.next == NULL) then
   ptr = start
   start = NULL
   free(ptr)
4. else
   ptr = start
   while (ptr.next ≠ NULL) then
      student = ptr
      ptr = ptr.next
   student.next = NULL
   free(ptr)

Delete_pos()

1. structure studentinfo * student, *ptr

2. If (start == NULL) then
   Display "List is Smply"
3. else.
   Input the position of the node to delete, p
   If (p == 0) then
      ptr = start
      start = start - next
      free (ptr)
   else.
      ptr = start
      for (i = 0; i < p ; i++)
         student = ptr
         ptr = ptr - next
         if (ptr == NULL) then
            Display "Position not found"
      student. next = ptr - next
      free (ptr)

Insert_pos()

1. struct studentinfo *ptr *student.
2. Enter the position of the new node to insert, p
3. Input the student details (name, roll no, marks)
4. student. name = name
5. student. roll no = roll no
6. student. marks = marks
7. student. next = NULL
8. If (p == 0) then
      student. next = start
      start = student

9. else

```
for(i=0, ptr = start, i<p-1; i++)
    ptr = ptr.next
    if (ptr == NULL) then
        Display "Position not found"
    student.next = ptr.next
    ptr.next = student.
```

RESULT

Program ran succesfully and opeeations were peiformed on the linked list.

EXPERIMENT - 8

PALINDROME USING DOUBLY LINKED LIST

AIM

Store a string using doubly-linked list and check whether palindrome or not.

Input :- A word
Output :- check whether palindrome or not.

ALGORITHM

1. Start
2. Create a structure 'node' with datatype 'char' for the data and struct node* next, struct node* prev.
3. struct node* header = NULL
4. Input the size of the word, n
5. Input the word, str.
6. If (isPalindrome (header, str, n) == 1) then
   Display "Its a palindrome."
   else.
   Display "Not a palindrome"
7. Stop.

IsPalindeome (struct node * head, char str[], int length)

1. Allocate memory for a new node 'new'
2. new.data = str[0]
3. new.prev = NULL
4. new.next = NULL
5. head = new
6. struct node * ptr = new
7. for(i=1; i< length, i++)
   - 7.1. new data = str[i]
   - 7.2. new.next = NULL
   - 7.3. new.prev = NULL
   - 7.4. while (ptr.next != NULL)
     ptr = ptr.next
   - 7.5. ptr.next = new
   - 7.6. new.prev = ptr.
8. while (ptr.next != NULL)
   ptr = ptr.next.
9. struct node * right = ptr.
10. ptr = head.
11. while (head != right)
    - 11.1. if (head.data != right.data) then
      c=0
    - 11.2. head = head.next
    - 11.3. right = right.prev.
12. return 1

EXPERIMENT-9

STACK USING LINKED LIST

AIM

Implement a stack using singly linked list.

Input: Set of integers

Output:- Operations on stack

ALGORITHM

1. Start.
2. Create a structure 'node' containing datatypes int for the data and struct node *next.
3. Choose an option from the menu. 1-Push 2-Pop 3-Display 4-Exit.
   3.1. If option 1 then
   push()
   3.2. If option 2 then
   pop()
   3.3. If option 3 then
   Display()
   3.4. If option 4 then
   exit.
4. Repeat step 3 till option = 4
5. Stop.

Push()

1. Enter the value to push, data
2. if (head == NULL) then
   ptr.data = data
   ptr.next = NULL
   head = ptr.
3. else.
   ptr.data = data
   ptr.next = head.
   head = ptr.
4. Display "Item pushed".

Pop()

1. struct node *ptr
2. if (head == NULL) then
   Display "Stack Empty"
3. else.
   item = head.data
   ptr = head
   head = head.next.
   free(ptr)
4. Display item popped

Display()

1. struct node *ptr
2. ptr = head.
3. if (ptr == NULL) then
   Display "Stack is empty"

4. else.

while (ptr != NULL)

Display (ptr. data)

ptr = ptr. next.

Result

Program ran successfully and operations performed on stack.