Chapter - 2

(18) program to Handle different type of Lists

Aim:

To Write a program that correctly processes

- An empty List
- A List with one element
- A List with all identical elements
- A List with negative numbers

*Pseudocode

```
Start
Read List A
 n = length of A
If n<=1
  print A
else
 SORT A in ascending order
 print A
END IF
STOP
```

*Test cases

INPUT	OUTPUT
[ ]	[ ]
[1]	[1]
[7,7,7,7]	[7,7,7,7]
[-5,-1,-3,-2,-4]	[-5,-4,-3,-2,-1]

Selection Sort

(19) Aim

To Sort a given Array of elements in ascending order using Selection Sort Algorithm

Pseudocode

```
Start
Read n
Read array A[n]
for i=0 to n-2
  min = i
 for j = i+1 to n-1
 If A[j] < A[min]
  min = j
  END If
  END For
   SWAP A[i] and A(min)
   end for
   print A
  STOP
```

* Input

[5, 2, 9, 1, 5, 6]

* output

[1, 2, 5, 5, 6, 9]

(10) Bubble Sort

Aim:

To sort a list of elements in ascending order using Bubble Sort

* pseudo code

```
Start
Read n
Read array A[n]
for i=0 to n-2
 Swapped = 0
 for j = 0 to n-i-2
 If A[j] > A[j+1]
 Swap A[j], A[j+1]
 Swapped = 1
 END IF
 END FOR
If SWAPPED == 0
Break
end if
end for
print A
STOP
```

* Input

[5, 1, 4, 2, 8]

* output

[1, 2, 4, 5, 8]

(21) Test cases:

Insertion Sort

Aim:
To implement insertion sort that correctly sorts an array containing duplicate elements and to study its behaviour

* Pseudocode:

```
InsertionSort (A,n)
 {
  for i=1 to n-1
  { key =A[i];
    j=i-1;
    While(j>=0 && A[j]>key)
    {
     A[j+1]=A[j];
       j=j-1;
    }
    A[j+1]=key;
   }
 }
```

* Input

Input	output
[64,25,12,22,11]	[11,12,22,25,64]
[29,10,14,37,13]	[10,13,14,29,37]
[3,5,2,1,4]	[1,2,3,4,5]
[1,2,3,4,5] (Already Sorted)	[1,2,3,4,5]
[5,4,3,2,1] Reverse Sorted	[1,2,3,4,5]

(2) Array with duplicates

Input:
[3,1,4,1,5,9,2,6,5,3]

Output:
[1,1,2,3,3,4,5,5,6,9]

(3) All Identical Elements

Input:
[5,5,5,5,5]

output.
[5,5,5,5,5]

Kth missing positive integer

(22)

Aim:

To find the Kth missing positive integer from a given array of strictly increasing positive integers

* **pseudocode:**

```
findKthpositive (arr,n,k)
{
 left =0;
 right =n-1;
 while (left <=right)
 {
 mid =(left +right)/2;
 missing =arr [mid] - (mid+1);
  if (missing <k)
     left = mid +1;
   else
   right =mid -1;
 }
 return k + left;
 }
```

* **Input**

arr =[2,3, 4,7, 11]

K=5

* **output**

output: 9

(23)

peak element

Aim: To find the index of a peak element in a given 0-indexed integer array using an O(log n) time complexity algorithm.

* **pseudocode:**

```
findpeak element (nums,n)
 {
  low =0;
  high =n-1;
 while (low <high)
 {
 mid = (low+high)/2;
 if (nums(mid] <nums (mid+1])
  low =mid+1;
  else
    high =mid;
 }
 return low;
 }
```

* Input:
nums = [1,2,3,1]
* output
output: 2

(24) Index of first occurrence

Aim:
To find the index of the first occurrence of the string "needle" in the string haystack. If needle is not present, return -1.

* pseudocode

```
strstr (haystack, needle)
{
 n = Length(haystack);
 m = Length (needle);
  if(m==0)
   return0;
for i=0 to n-m {
  j = 0;
  while(j < m && haystack [i+j] == needle[j])
   j++;
  if (j==m)
     return i;
 }
 return -1;
}
```

* Inputs
haystack = "sad but sad"
needle = "sad"

* output
output : 0

(27) Given an array of string words, return all strin in words that is a substring of another wo

Aim: To find and return all strings from a given array that are substrings of another string in the same array

* pseudocode;

```
findSubstrings (words, n){
 result = empty List;
  for i=0 to n-1 {
   for j=0 to n-1 {
    if (i != j && isSubstring (words [i], words[j])
  {
     add words [i] to result;
        break
```

}
}
return result;

}

Q) Brute force approach

Aim: To find the closest pair of points in a given set of 2D points using the brute-force method

* Pseudocode

```
ClosestPair(points, n)
{
minDist = INFINITY;
 for i=0 to n-2 {
 for j = i+1 to n-1 {
dist = sqrt [(points[i].x - points[j].x)^2 + (points[i].y - points(j).y)^2];
  if (dist < minDist)
   {
   minDist = dist;
   P1 = points[i];
    P2 = points[j];
    }
   }
  }
print P1, P2, minDist;
}
```

* Input

points = [(1,2), (4,7), (7,8), (3,1)]

* Output

Closest pair : (1, 2) - (3,1)

minimum distance : 1.41421356237309

closest pair of points (Bruteforce)

Aim:
To find the closest pair of points in a given set of 2D points using the bruteforce approach and analyze its time complexity

Euclidean distance function

$$Distance = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$$

* pseudocode:

```
distance (P1,P2)
{
return sqrt ((P1.x - P2.x) * (P1.x - P2.x) + (P1.y - P2.y) * (P1.y - P2.y)); }
closest pair (points, n)
{
minDist = Infinity;
for i=0 to n-2
{
for j = i+1 to n-1
{
d = distance (points[i], points [j]);
if (d < minDist) {
minDist = d;
p1 = points[i];
p2 = points [j];
}
}
}
print p1, p2, minDist;
}
```

* Input
points = [(1,2), (4,5), (7,8), (3,1)]

* output
closest pair : (1,2) - (3,1)
minimum distance: 1.4142135623730951

28 Convex Hull

Aim:

To write a program that finds the convex hull of a given set of 2D using the brute force approach

* Pseudocode

for each point i in points
 for each point j in points
  if i != j
   pos = neg = 0
  for each point k in points
   val = cross product (i, j, k)
   if val > 0 then pos++
   else if val < 0 then neg++
    if pos == 0 or neg == 0
    mark i and j as hull points
print hull points in counter-clockwise order

* Input

points = [(1,1), (4,6), (8,1), (0,0), (3,3)]

* output

Convex Hull = [(0,0), (1,1), (8,1), (4,6)]

29 Travelling Salesman

Aim:

To develop a program that solves the travelling salesman problem (TSP) using an exhaustive search (brute force) approach by generating all possible permutations of cities and find the shortest possible tour

* Pseudocode

function tsp(cities, n)
start = cities[0]
minDist = INFINITY
for each permutation P of cities (1 to n-1)
 dist = 0
  curr = start
for each city in p

```
dist + = distance (curr, city)
  curr = city
  dist + = distance (curr, start)
    if dist < minDist
      minDist = dist
      bestpath = (start, p, start)
  return minDist, bestpath
```

* Input:

cities = [(1,2), (4,5), (7,1), (3,6)]

* output

shortest distance: 7.0710678118654755

shortest path: [(1,2), (4,5), (7,1), (3,6), (1,2)]

30 Assignment problem

Aim: To develop a program that solves the Assignment problem using an exhaustive (brute force) search approach by checking all possible worker task

* pseudocode

```
function total_cost (assign[], cost, n)
      Sum = 0
   for i = 0 to n-1
      Sum + = cost [i] [assign[i]]
   return Sum
function assignment_problem (cost, n)
   mincost = INFINITY
 foreach permutation p of tasks 0 to n-1
   Curr cost = total_Cost (p, Cost, n)
   if currcost < mincost
      mincost = currcost
      best Assign = p
   return best Assign, mincost
```

* Input

cost matrix =
[[3, 10, 7],
[8, 5, 12],
[4, 6, 9]]

* output

optimal assignment:

[(worker1, task2), (worker2, task1), (worker3, task3)]

Total cost: 19

Knapsack using exhaustive search

Aim:

To develop a program that solves the 0-1 knapsack problem using exhaustive search.

* procedure

* pseudocode:

```
max = 0
for each subset S of items
    if weight (S) <= capacity
        if value(S) > max
            max = value(S)
            best = S
return best, max
```

* Input

Items : 3

weights : [2,3,1]

values : [4,5,3]

Capacity : 4

* output

optimal Selection : [0,2]

Total value : 7