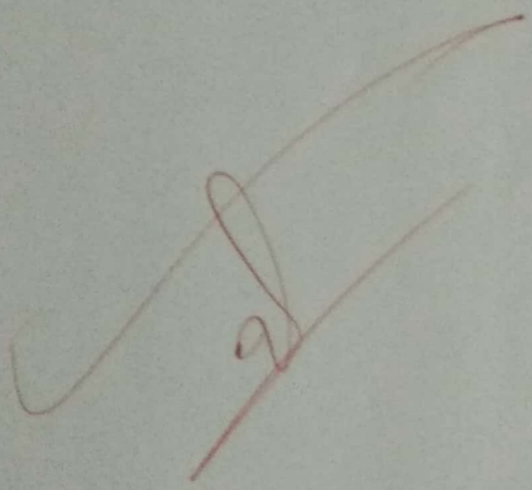
Name → Satyam Rawat

Section → G

Roll No → 16

Univ. R No → 2017006

Sem → 4th



Week 1

Given an array of non negative integers, design a linear algorithm & implement it using a program to search for element in array. Also find no of comparisons. T.C. = O(n)

Algorithm →

Take input array from user
Take element that is to be searched from user
Set flag = 0
Loop a from i=0 to size-1
  if no == array element
    increment flag
End Loop
After loop check if flag = 0 print "Not found"
  else print "Element found"

STOP

Given an array (sorted) design an algorith to search for a key in O(logn).

Algorithm →

Take input array from user and key
Initialise l=0, r = size-1
LOOP - while (l ≤ r)
  mid = (l+r)/2
  if (a[mid] == key) then
    return mid
  else if (a[mid < key) then
    l = mid+1
  else r = mid-1
End Loop
return -1

> Given a sorted array of integer, Search for a key using Jumpsort TC. < O(n)

Algorithm →

Start

c = √size

s = 0, e = blocksize

while (a[end] <= key and end < size):

s = e ;

e = e + blocksize

if (end > size-1) then

end = size

End Loop

FOR (i = s to e-1) :

if (a[i] = = key) then

return i

End Loop

return -1

STOP

Week 2

1) Given sorted array, design an algorithm & find if an element is present. Also find its duplicates TC = O(logn)

Algorithm =

Use Binary Search to get index of first occurence of key element present in arr[ ].

Let index = i.

Use Binary Search to get index of last occurence of key element occurence of arr[i].

Let index of last ocurrence be j.

return (j - i + 1).

2) Given sorted array of positive integer, Find 3 indices i, j, k so that a[i] + a[j] = a[k].

Algorithm -

Enter a sorted array

For (i=0 to n, i++) :

Set k=1 & for every] > 1.

then incerrent k until A[A] <= a[i] + a[j]

and incerrent count if equality is achieved.

STOP.

Q Given an array of non negative integers, design an algorithm to count number of pairs such that their difference equals k.

Algorithm –

```
Start
Enter key k #
Sort the array
l ← 0 , r ← 0
while (r < size of array)
    if (arr[r] - arr[l] == k )
      then  c++, l++, r++

    else if  arr[r] - arr[l] > k)
      then  l++

    else  r++
return c
Stop
```

ck 3

Given an unsorted array. Design an algo to sort it using Insertion sort. Also Find Comparisons & Shifts.

Algorithm –

If its first element, return 1
Pick next element
Compare with all elements in sorted sub array.
Shift all the elements in sorted sub array
for which value is greater than value to be sorted.
Insert value
Repeat until array is sorted

> Given an unsorted array, sort it using Selection Sort. Also Find Comparisons & Shifts.

Algorithm –

```
Selection Sort(a, n)
Repeat step 2 & 3 for i = 0 to n-1
Call smallest (a, i, n, pos)
Swap a[i] & a[pos]
End Loop
End
Smallest (a, i, n, pos)
Set small = a[i]
pos = i
Repeat for j = i+1 to n
if (small > a[j]) set small = a[j]
Set pos = j
End if
End Loop
return pos
```

Given unsorted array. Design an algorithm to find if array has duplicate elements or not. TC(O(nlogn)).

Algorithm →

Start
Input array
Sort the array using insert sort
c ← 0
for (i=1 to n ; i++)
    if (a[i-1] == a[i]) print "Duplicate found"
                        return

print "No duplicate found"
Stop

WEEK 4

1) Given unsorted array. Design an algorithm & sort the array using merge sort. Also find no of comparisons & shifts.

Algorithm

Find middle index of array as:
mid = (lb + ub) / 2
Divide the array from middle by recursive call
Call merge sort for first half of array:
mergesort (a, lb, mid)
Call merge sort for second half of array:
mergesort (a, mid+1, ub)
Merge two sorted halves into single array

2) Given an unsorted array, Sort it using Quicksort. Also find comparisons & shifts.

Algorithm

Choose highest index as pivot
Take 2 variables to point first & last location.
While value at left < pivot move right
While value at right > pivot move left
If both step 5 or step 6 don't match, swap left & right
If left ≥ right, the point where next is, is the new pivot.
Swap element so that pivot reaches its original location
Recursively call the function again

3) Given an unsorted array. find kth smallest element (TC. = O(N))

Algorithm →

Start

Find middle index of the array

mid = (lb + ub)/2

Divide the array from middle

Call merge sort for first half of array & then for second half

MergeSort (a, first, mid)

MergeSort (a, mid+1, last)

Merge the 2 sorted halves into single array

return element at index k-1 in sorted array.

Prob 5

Given unsorted array of alphabets. Find the alphabet with maximum occurence TC = O(N).

Algorithm

Start
Create function count(a, size, k)
count[k+1] = 0 & flag = 0
For (i = 0 to n-1; i++):
Increment count[a[i] - 'a'] by 1
End loop
for (ch = 97; ch <= 122; ch++):
If (count[ch-97] > 1 then
flag++
End Loop
Max = count[0]
for (i = 0 to k-1; i++):
if (count[i] > max) then
max = count[i]
key = i
End if
End Loop
If flag == 0 print "No Duplicate present"
Else print (char)(key + 97), print max
Stop

> Given unsorted array of integer. Design an algorithm & find 2 elements whose sum is more than given element TC = O(nlogn).

Algorithm

Start
Sort the array using Quick Sort
k = size-1, j = 0, flag = 0
While (j < n/2 and k >= n/2):
If (a[j] + a[k] >= num)
then k--
else j++;
If (a[j] + a[k] == num)
then print("a[j] & a[k]
flag++;
End Loop
If flag = 0 then
print("No such pair exist")
STop

Q) Given 2 sorted integer arrays of size m & n. Design an algorithm to find elements which are common to both.
T(C) = O(m+n)

Algorithm

Start
For (i=0 to m-1, i++) :
    flag = 0
    For (j=0 to n-1; j++)
        if (a[i] == b[j]) then
            Print (a[i])
        Endif
    End Loop
End Loop
Stop

WEEK - 6

Given a graph, design an algorithm & implement it using a program to find path, if exists, between 2 given nodes.

START
Input V
If i >= V go to step 7
Input temp
adj[i].push_back(temp)
If i < v, go to step 3
input s, d
res = checkpath (adj, V, s-1, d-1)
If (res == 1) print "path exists"
else print "Path does not exist"
STop

Design an algorithm to find if a graph is Bipartite or not

Start

Input v

If i >= V go to step 7

input temp

input g[i].push_back(temp)

If i < v go to step 3

res = (Bipartite (G, V))

if(res == 1) print "Yes, Bipartite"

else print, "No not Bipartite"

Stop

Given a directed graph, design an algorithm to find whether cycle exists or not.

Start
Input V
Input u, v in adj[v]
res = DFS (adj, V)
If (res == 1) print "Cycle exists"
else print "No cycle exists"
Stop

WEEK - 7

> Apply Dijkstra's Algorithm to print shortest path & distance from given source to destination.

Start
Input V
If v >=0 go to step 6
if j >=v go to step 7
input adj[i][j]
if j<v go to step 4
if i<v go to step 3
input S
dijkstra(adj, V, S-1)
Stop

Design an algorithm to find shortest path & distance between 2 nodes using Bellmanford algorithm.

Start
Input m
If i >= m go to step 8
If j >= m go to step 7
Input graph[i][j]
If j < m go to step 4
If i < m go to step 3
Input s
findpath (graph, m, s-1)
Stop

Given a directed graph with two vertices, design an algorithm to find weight of shortest path from source to destination with exactly k edges on the path

Start
input var
if v i >= ver go to step 6
if j >= ver go to step 7
input graph[i][j]
if j < ver go to step 4
if i < ver go to step 3
input u, v, k
ans = shortest_wt(graph, ver, u-1, v-1, k)
print ans
Stop

WEEK - 8

Design an algorith which will find maximum cost required to connect cities (use Prim's Algo)

Start
Input n
if i>=n go to step 8
if j>=n go to step 7
Input a[i][j]
if j<n go to step4
if i<n go to step3
prims(a, n)
STOP

2) Design an algorithm to find minimum cost required to connect cities (kruskal's algorithm)

1) Start
2) Input n
3) If i>=n go to step 8
4) If j>=n go to step 7
5) Input graph[i][j]
6) If j<n go to step 4
7) If i<n go to step 5
8) kruskal(graph, n)
9) Stop

Design an algorithm & implement it using program for finding maximum budget required for a project

START
Input n
if i>=n go to step8
if j>=n ~~goto~~ to step7
input graph[i][j]
if j<n go to step 4
if j<n go to step 3
kruskals(graph, n)
Stop

Week 9

Design an algorithm to implement Floyd - Warshall's all pair shortest path algorithm.

1) Start
2) Input n
3) if i >= n go to step 11
4) if j >= n go to step 10
5) input temp
6) if(temp != "INF")
7) graph[i][j] = stoi(temp)
8) graph[i][j] = 108
9) if j < n go to step 4
10) if i < n go to step 3
11) if k >= n go to step 19
12) if i >= n go to step 18
13) if j >= n go to step 17
14) if(graph[i][k] + graph[k][j] < graph[i][j])
15) graph[i][j] = graph[i][k] + graph[k][j]
16) if j < n go to step 13
17) if i < n go to step 12
18) if k < n go to step 11
19) to print shortest path matrix
20) if i >= n go to step 26
21) if j >= n go to step 25
22) if (graph[i][j] >= 108) print "INF"
23) else print graph[i][j]
24) if j < n go to step 21
25) if i < n go to step 20
26) Stop

Given a knapsack of maximum capacity w. N items are provided each having its own value & weight.
Design an algorithm to find subset of selected items such that final selected subset has weight w & has maximum value.

Start
Input n
if i>=n go to step6
Input items
if i<n go to step 3
if i>n go to step 11
input val[i]
Job.push_back(val[i], items[i])
double (i+1)
if i<n go to step 6
Input k
Sort (job.begin(), job.end())
profit = 0
if i>=n go to step 22
if (job[i][1] >= k)
profit += k * job[i][0]
Is.push_back({k, job[i][2]})
break
else profit += job[i][1] * job[i][0]
Is.push_back({job[i][i], job[i][2]})
k = k - job[i][1]
print profit
print "Item - weight"
for every element, it. of Ls
print it.second - it.first
Stop

Given an array of elements. Assume a[i] represents size of file i. Write an algorithm to merge all these files into single file with minimum computation. For given 2 files A & B with sizes m & n, computation cost of merging them is O(m+n).

Start
Input n
If i>=n go to step 6
Input a[i]
If i<n go to step 3
i=0
If i>=n go to step 10
minheap.push(a[i])
If i<n go to step 7
ans=0
while (minheap.size() > 1)
(i) e1 = minheap.top(), minheap.pop()
(ii) e2 = minheap.top(), minheap.pop()
(iii) ans += e1 + e2
(iv) minheap.push(e1+e2)
print ans
Stop

WEEK 10

Given a list of activities with their starting time & finishing time. Select maximum no of activities that can be performed by single person such that selected activities must be non conflicting.

```
start
Input n
i, s[n], f[n]
if i>n go to step 7
input s[i]
if i<n go to step 4
  if i=0
  if i>=n go to step 11
input f[i]
if i<n go to step 11
vector < vector<int> >a, vector<int> act
if i>=n go to step 14
a.push_back ( f[i], [s[i], i+1])
sort (a.begin(), a.end())
e = INT_MIN, c=0
if i>=n, go to step 19
  if (a[i][1] >=e )
    e = a[i][0]
    c++
    act.push_back (a[i][2])
if i<n go to step 16
print "No of non conflicting ", c
print "No of activities selected"
if i>=n go to step
print act[i]
```

> Given a long list of tasks. Each task takes specific time
, accomplish it & each task has a deadline associated with
t. Design an algorithm to find maximum no of tasks
hat can be completed without crossing their deadlines &
bo find list of selected tasks.

Start
Input n
i, t[n], y[n]
if i>=n go to step 7
, Input t[i]
if i<n go to step 4
> i=0
> if i>n go to skp 11
> Input y[i]
> if i<n go to step 8
> vector<vector<int>>a , vector<int> act
> if i>=n go to step14
> a.push_back({y[i] , y[i]- t[i], i+1})
+> sort (a.begin(), a.end())
:> e = INT_MIN , c=0
:) if i>=n go to step 12
7:) if (a[i][i] >=e)
e= a[i][a]
8:) e=a[i][a]
9:) e++
20> act.push_back (a[i][2])
21> if i<n go to step 16
22> sort (act.begin(), act.end())
23> Print c
24> Print "Selected Task Numbers"
25> if i>=n go to step 28
26> Print act[i]
27> if i<n go to step 25
28> Stop

Given an unsorted array, design an algorithm to find whether majority element exists or not. Also find median of array.

Start
Input n
i, a[n], i, j
if i>=n go to step 6
Input a[i]
j=0
Sort (a, a+n)
if i>=n go to step 14
c=1
j=(i+1)
while(j<n && a[j++] == a[i])
do c++
if (c>n/2) Print "yes"
do j=1
(ii) break
i=j-1
if j==0 Print "No"
if n%2!=0 Print a[n/2]
else Print a[n/2] + a[n/2-1]/2
STop

WEEK 11

ven a sequence of matrices. Write an algorithm to find erent way do multiply them.

```
start
input n
p[n+1]
if i>=n go to step 6
input p[i], p[i+1]
print. matchain order (p, n+1)
Stop

atch chain order (p, n)
m [n][n], i, j, k, l, q;
or (i= 1 to n ; i++)
    m[i][i]= 0

or (l=2 ; l<n ; l++)
{ for (i= 1; i< n-l+1 ; i++)
  { j= i+l-1 ;
    m[i][j] = INT_MAX;
    for (k=i ; k<=j-1 ; k++)
    { q= m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j]
      if (q< m[i][j])
        m[i][j]= q
    }
  }
}
return 0 m[1][n-1];
```

Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to design an algorithm to find no of ways in which these coins can be added to make sum value equals to N.

Start
Input n
i, j, a[n]
if i >= n go to step 7
Input a[i]
Input amt
i=1
if i >= amt go to step 11
ans[i] = 0
ans[0] = 1
if j >= n go to step 14
if i >= amt go to step 11
if (a[j] <= i)
ans[i] += (ans[i - a[j]])
Print ans[amt]
Stop

7. Given a set of positive elements, you have to partition the set into 2 subsets such that the sum of the elements in both is same. Design an algorithm for it.

Start
Input n
Input a[n]
i, j, sum
if i>=n go to step 6
Input a[i]
sum=0, i=0
if i>=n go to step 8
if (sum%2 != 0)
    Print "No", return 0
sum = sum/2
bool s[n+1][sum+1]
if i>=n go to step 16
if j>=sum go to step 11
if (j==0) s[i][j]=1
else if (i==0) s[i][j]=0
else
    (i) if (a[i-1]>j)
        s[i][j] = s[i-1][j]
    (ii) else s[i][j] = s[i-1][j] || s[i-1][j-a[i-1]])
if (s[n][sum]) Print "Yes"
else Print "No"
Stop