Date_____ Page No.

LINEAR SEARCH

Linear S(a, n, val).

STEP 1 → [Initialize] Set flag = 0
STEP 2 → Initialize] Set i = 0
STEP 3 → Repeat Setep 4 while i < n
STEP 4 → If a [i] < val
Set flag = 1
Print "value present"
go to step6.
[end of if]
set i = i + 1
[end of loop]
STEP 5 → if flag = 0
Print "value not present"
[end of if]
STEP 6 → EXIT

Binary Search

Binary S (a, lowerBound, upperBound, val)

STEP I → [initialize] Set start = lowerBound
set end = upperBound

STEP II → Repeat Step III and IV while Start <= end

STEP III → SET mid = (start + end)/2

STEP IV → If a[mid] = val
set pos = val
print pos
go to step 6
else if a[mid] > value
set end = mid - 1
else
set start = mid + 1
[end of if]
[end of loop]

STEP V → If pos = -1
print "value not present"
[end of if]

STEP VI →

Date

Page No.

SELECTION SORT

Selection Sort.

STEP I → set i = 0, swap = 0.
STEP II → for i < n-1.
swap++
s = a[i]
pos = i
for (j = i+1 till j<n)
if a[j] < s
s = a[j]
pos = j
[end of if]
temp = a[i]
a[i] = a[pos]
a[pos] = temp

Step 3 => print array.
Step 4 => print swap
Step 5 => Exit

Teacher's Signature

Date_____ Page No. _____

INSERTION SORT

Insertions (a, n)
Step 1: set i = 1
step 2: for i < n
temp = a[i];
j = i - 1;
while temp < a[j] and j >= 0
a[j+1] = a[j]
j--.
[end of while]
a[j+1] = temp
i++;
[end of for].

Step 3: Print array
Step 4: exit

Teacher's Signature_____

Date

Page No.

MERGE SORT

Merge S(arr, l, m, r)

STEP I → set $n_1 = m-l-1, n_2 = r-m$

STEP II → set $l[n_1], r[n_2]$

STEP III → for $i=0, j=0$ till $i<n_1$ & $j<n_2$

$l[i] = arr[l+i]$

$r[j] = arr[m+1+j]$

$i = i+1, j = j+1$

STEP IV → Set $i=0, j=0, k=l$

STEP V → while $i<n_1$ & $j<n_2$

if $l[i] <= r[j]$

$arr[k] = l[i]$

$i++$

[end of if]

else $arr[k] = r[j]$

$j++$

[end of else]

$k++$

[end while]

STEP VI → while $i<n_1$,

$arr[k] = L[i]$.

$i++, j++$

[end while]

Huhan

Teacher's Signature

Date

Page No.

STEP 7 ⇒ while $j < n_2$

arr[k] = r[j]

j++, k++

[end while]

STEP 8 ⇒ Exit

Teacher's Signature

Date

Page No.

EXPONENTIAL SORT

Exponential S (a, n, key)

STEP I → set flag = 0

STEP II → if a[0] == key.

print "Present"

go to step 8

[end of if].

STEP III → set range = 1

STEP IV → while range < n and a[range] <= key.

range = range * 2

[end of while].

STEP V → set l = range /2, h = min(range, n-1)

STEP VI → while l <= h

mid = (h+l) /2

if a[mid] == key

flag = 1

[end of if].

else if (a[mid] > key)

h = mid - 1;

else if (a[mid] < key)

l = mid + 1

[end of while]

STEP 7 → if flag = 1, print ("Present")

STEP 8 → EXIT

Mohan

Teacher's Signature

Date

Page No.

BREADTH FIRST SEARCH

BFS

Step 1 ⇒ set status = 1 (ready state)
for each node in G.

STEP 2 ⇒ Enqueue the starting node A and set its status = 2
(waiting state)

STEP 3 ⇒ Repeat steps 4 & 5 until Queue is empty

STEP 4 ⇒ Dequeue a node N. Process it and set status = 3.

STEP 5 ⇒ Enqueue all neighbours of N that are in ready state where (status = 1) & set their status = 2

(end of loop)

STEP 6 ⇒ Exit.

Teacher's Signature

Date

Page No.

DEPTH FIRST SEARCH

DFS

STEP I → set Status = 1 for each node in G.

STEP II → Push starting node A on stack and set its status = 2

STEP III → Repeat step 4 and 5 until stack is empty.

STEP IV → Pop the top node N.
Process it and set its status = 3.

STEP V → Put on stack all the neighbours of N that are in ready state and set their status = 2.
[end of loop]

STEP VI → Exit

Mohan

Teacher's Signature

Date

Page No.

KRUSKAL'S ALGORITHM

Kruskal (E, cost, n, t)

STEP 1 → construct a head of edge cost using heapify

STEP 2 → for i = 1 to n do parent [i] = -1

i = 0, min cost = 0.0

STEP 3 → While (i < n-1) and (heap not empty) do

delete a min cost edge (u, v) from the heap

and reheapify using adjust

j = find (u), k = find (v)

if (j ≠ k)

i = i + 1

t [i, 1] = u

t [i, 2] = v.

STEP 4 → min cost = min cost + cost [u, v]

Union (j, k)

[end if]

[end while].

STEP 5 → if i ≠ n-1

print 'No Spanning Tree"

else

return min cost

STEP 6 → Exit

Mohan

Teacher's Signature

Date _____ Page No. _____

PRIMS ALGORITHM

```
Prims (G, src)
for (each vertex u bt. g. vertices)
        u.key = ∞
        u.parent = NULL

src.key = 0
Q = minHeap (g.vertices)
while Q != ϕ
        u = extractMin(Q)
    for each vertex 'v' adjacent to u
    if bt s and u(u,v) < u.key
        v.parent = u
        v.key = u(u,v)

Exit.
```


Teacher's Signature _____

Date

Page No.

DIJKSTRA'S ALGORITHM

```
Dijkstra (G,S)

for each vertex V in G
        dis[v] = infinite
        prev[v] = NULL
    if v != S, add V to priority Queue Q

    dis[s] = 0

while Q is not empty.
        u = extract min from Q
    for each unvisited neighbour v of u
        tempDist = dist[u] + edgeweight[u,v]
        if tempDist < dist[v]
            dist[v] = tempDist
            prev[v] = u

    return dist[], prev[].
```

BELLMAN FORD ALGORITHM

```
Bellman Ford(G,S)

for each vertex N in G
        dist[N] = infinite
        prev[N] = NULL
dist[S] = 0

for each vertex N in G
    for each edge (M,N) in G
        tempdist = dist[M] + edgecost(M,N)
        if (tempDist < dist[N])
            dist[N] = tempDist
            prev[N] = M

for each edge (M,N) in G
        if dist[M] + edge(M,N) < dist[N]
            error: negative cycle exists
return dist[], prev[].
```

Date _____ Page No. _____

FLOYD WARSHALL

Floyd Warshall (cost, A, n).

// cost [i=n, j=n] is cost adjacency matrix of ∞ graph with n vertices

STEP I ⇒ for i=0 to n
do for j=1 to n
do a[i,j] = cost[i,j]

STEP 2 ⇒ for k=1 to n
do for i=1 to n
do A[i,j] = min(A[i][j], A[i][k]+A[k][j]);

STEP 3 ⇒ EXIT.

Teacher's Signature _____

Date

Page No.

HUFFMAN ENCODING

Huffman Coding (string)

STEP 1 → Define a node with character, frequency left and right child of node for huffman tree.

STEP II → Create a list frequency to store frequency of each character, initially all are 0.

STEP III → for each character c in the string do, increase the frequency for character ch in frequency list.

STEP IV → for all type of character ch do, if the freq. of ch is non-zero then, add ch and its frequency as a node of priority queue Q

STEP V → while Q is not empty do, remove item from Q and assign it to left child of node.

- remove item from Q and assign to right child of node.
- traverse the node to find the assigned code.

STEP VI → Exit

Balaji

Teacher's Signature

Date _____ Page No. _____

ACTIVITY SELECTION

Activity selection (start[], end[] size).

STEP 1 → Sort all activities according to their end time.

STEP 2 → Select the first activity and note the end time call it as end time.

STEP 3 → Now, iterate through rest of the activities

```
For each currentActivity
        if startTime of currentActivity > end_time
    Select current Activity
        update end Time = end Time of current
        activity
else
    ignore the current_activity

[end for]
```

STEP 4 ⇒ EXIT.

Teacher's Signature _____

Date _____ Page No. _____

JOB SEQUENCING

Job sequencing (d, j, n)

STEP 1: d[0] = J[0] = 0

STEP 2: J[1] = 1
k = 1

STEP 3: for i = 2 to n.
do r = k
while ((d(J[r]) > d[i] and
d[J[r]]) ≠ r)
do r = r-1;
if ((d[J[r]] ≤ d[i]) and (d[i] > r))
then for a = k to (r+1) step -1
do J[a+1] = J[a]
J[r+1] = i
k = k+1
[if ends]
[for ends].

STEP 4: END

Balaji

Teacher's Signature _____

Date

Page No.

FRACTIONAL Knapsack

Greedy knapsack (m, n)

STEP 1 ⇒ for i=1 to n
do x[i]=0.0 // initialize x

STEP 2 ⇒ u=m.

STEP 3⇒ for i=1 to n
do if w[i]>u
break.
x[i]=1.0
u-w[i]

STEP 4 ⇒ if (i<=n)
x[i] = u/w[i]

STEP 5 ⇒ Exit

Teacher's Signature

Date

Page No.

SUBSET SUM

SubsetSum (n, sum, arr[])

STEP I → if sum = 0
return true.

STEP II → if n >= 0
return false

STEP III → if (arr[n-1] > sum)
return subset (n-1, sum)

STEP IV → else
return subset (n-1, sum) ||
subset (n-1, sum-arr[n-1]);

Teacher's Signature