Algorithms

Stable matching

Initially all $m \in M$ and $w \in W$ are free

while there is a man m who is free and has not proposed to every woman

Choose such a man m

Let w be the highest ranked woman in m's preference list to who m has not yet proposed

If w is free then

(m, w) become engaged

else

w is currently engaged to m'

if w prefers m' to m then

m remains free

end if

end if

~~End while~~

Return the set S of engaged pairs

Time complexity :- $O(n^2)$

DFS Algorithm

DFS (u):

Mark u as "visited" and add u to R

for each edge (u, w, v) incident to u

If v is not marked "visited then

Add v to R

Recursively invoke DFS (v)

End if

End for

Time Complexity = O (V+E)

Merge - Sort Algorithm

mergesort (A[0.. n-1])

// Sorts array A [0..n-1] by recursive mergesort

// Input : An array A [0.. n-1] of orderable elements

// Output : An array A [0... n-1] sorted in non decreasing order

if n > 1

Copy A [0.. [n/2] - 1] to B [0.. [n/2] - 1]

Copy A [[n/2].. n-1] to C [0.. [n/2] - 1]

→ mergesort (B[0..⌈n/2⌉ - 1])

mergesort (C[0..⌈n/2⌉ - 1])

merge (B, C, A)

merge (B[0..p-1], C[0..q-1], A[0..p+q-1])

// merges two sorted arrays into one sorted array

// Input: Arrays B[0..p-1] and C[0..q-1] both sorted

// Output: Sorted array A[0...p+q-1] of the elements of B and C

while i < p & j < q do

if B[i] ≤ C[j]

A[k] ← B[i]; i ← i+1

else A[k] ← C[j]; j ← j+1

k ← k+1

if i = p

copy C[j..q-1] to A[k..p+q-1]

else copy B[i..p-1] to A[k..p+q-1]

Time complexity: O(n log n)

Counting inversions :-

Sort - and - Count (L)

If : the list has one element then
there are no inversions

Else :

Divide the list into two halves
A Contains the first $[n/2]$ elements
B Contains the remaining $[n/2]$ elements

(r_A, A) = Sort - and - Count (A)

(r_B, B) = Sort - and - Count (B)

(r, L) = merge - and - Count (A, B)

End if

return $r = r_A + r_B + r$, and the sorted List L

while both lists are non empty

let a_i and b_j be the elements pointed to by the current pointer Append the smaller of these two to the output list

If b_j is the smaller element then
Increment Count by the number of elements remaining in A

End if

Advance the current pointer in the list from which the smaller element was selected.

End while

Once one list is empty, append the remainder of the other list to the output

Return count and the merged list

Time complexity = $O(n \log n)$

Quick Sort

```
Quick sort (array A, start, end)
    if ( start < end)
    p = partition (A, start, end)
        Quick sort (A, start, P-1)
        Quick sort ( A, P-1, end)
Partition (arrays A., stat, end).
    Pivot A [end]
        i = start - 1
    For j = start to end - 1
        do if (A [j] < pivot ){
            then i = i+1 ;
Swap A[i] with A[j]
    Swap A[i+1] with A[end]
        return i+1
```

T. C = $O(n \log n)$

DIJKSTRA's Algorithm

Dijkstra's Algorithm (G, l)

Let S be the set of explored nodes

For each $u \in S$, we store a distance $d(u)$.

Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

Select a node $v \notin S$ with at least one edge from S for which $d'(v) = \min d(u) + l_e$ is as small as possible

Add v to S define $d(v) = d'(v)$

End while

Time comp :- $O((V+E) \log V)$

Prims algorithm

Input :- Graph $G = (V, E)$

output :- minimal spanning tree 'T'

Assume Set V is a set of explored nodes

initially $T = \phi$

$U = \{1\}$ where 1 is any arbitrary vertex $\in V$

while $U \neq V$ - no cycles

let (u, v) be lowest cost edge

such that $u \in V$, $v \in V - u$

$T = T \cup \{u, v\}$

$U = U \cup \{v\}$

end while

return T

$T.C = O((V+E) \log V)$

Kruskals Algorithm

Input : $G(V, E)$

output : Minimum spanning tree T

$T = \phi$

-> E sorted ← Sort the edges according to cost / weight

-> Select an edge C_K from E sorted

while (count $(|u| - 1)$

if $C_K \cup T$ is acyclic

count ++

else

discard C_K

end while

return T