**Time-Complexity notation Counting-Sort(A, B, k)**

**for** i=0 **to** k **do** C[i]=0  
 **for** j=1 **to** n **do** C[A[j]0]+=1  
 **for** i=1 **to** k **do** C[i]+=C[i-1]  
**Master Method**   
Case 1: then, **Compute-Next(P, m)**  
Case 2: then, next[1]=0, q=0  
Case 3: then, **for** i = 2 **to** m **do**   
 **while** q>0 **and** P[i]!=P[q+1] **do** q = next(q)  
**Red-Black:** Search=Insertion=Deletion=Color Changes , Rotations **if** P[i]==P[q+1] **then** q++   
 next[i]=q;  
**Heaps**  
Top Down, Bottum Up  
Heapification: , Update=Extract-Minimum: , Minimum-Index=Get-Minimum=Get-Key:   
  
**Sorting String-Matching**(T, n, P, m)

Bucket Sort: , space i ← 1, q ← 0  
Radix Sort: **while** i <= n **do**

Countering Sort: **if** T[i] = P[q + 1] **then** i ← i + 1, q ← q + 1   
Insertion Sort: Data Movement, Comparisons. **else**  
Selection Sort: Data Movement, Comparisons. **if** q = 0 **then** i ← i + 1  
Merge Sort: **else** q = next [q]  
Quick Sort: **if** q = m **then**

print “pattern found at position” i – m  
**Union Find** union: , find: q = next [q]  
**Huffman Codes:**   
**Selection**: **Bellman-Ford(G, w, s)**   
 Initialize-Single-Source(G, s)   
**String Matching**: **for** i=1 **to** V – 1 **do**   
 **for** each edge (u, v) that belongs to E **do**  
**Graphs** Realx(u, v, w)

**DFS and BFS**: list, Matrix **for** each edge (u, v) that belongs to E **do**   
**Bellman-Fort**: , works for everything but negative cycles. **If** d[v]>d[u]+w(u, w) **then** **return** False

**Acyclic**: , doesn’t have cycles, works with negative weights has to sorted in a topological order.

**Dijkstra**: Binary heap, delete, update

Fibonacci heap, delete, update

No heap, delete, update

**Prims**: , delete, update

**Kruskal**:, heap operations , union find  
**Floyd-Warshall:** , no negative cycle , no negative weights

**Max Flow:**

**Ford-Fulkerson**:

**Edmonds-Krap**:

**P, NP, NPC**

P = Polynomial Time Solvable, Nondeterministic Polynomial Time Solvable.

, as long as k is a constant then, the problem is P.   
A problem is called NP-Hard if every problem is reducible to that problem.

A problem is called NO-Complete if the problem belong to NP, and it is an NP-Hard problem.