**Merge Sort O(NlogN)**

**MergeSort(arr[], l, r)**

If r > l

**1.** Find the middle point to divide the array into two halves:

middle m = (l+r)/2

**2.** Call mergeSort for first half:

Call mergeSort(arr, l, m)

**3.** Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

**4.** Merge the two halves sorted in step 2 and 3:

Call merge(arr, l, m, r)

O(nLogn)



**Quicksort O(NlogN)**

/\* low --> Starting index, high --> Ending index \*/

quickSort(arr[], low, high)

{

if (low < high)

{

/\* pi is partitioning index, arr[pi] is now

at right place \*/

pi = partition(arr, low, high);

quickSort(arr, low, pi - 1); // Before pi

quickSort(arr, pi + 1, high); // After pi

}

}



**Selection Sort O(N^2)**

arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]

// and place it at beginning

**11** 25 12 22 64

// Find the minimum element in arr[1...4]

// and place it at beginning of arr[1...4]

11 **12** 25 22 64

// Find the minimum element in arr[2...4]

// and place it at beginning of arr[2...4]

11 12 **22** 25 64

// Find the minimum element in arr[3...4]

// and place it at beginning of arr[3...4]

**Insertion Sort O (N^2)**

// Sort an arr[] of size n  
insertionSort(arr, n)  
Loop from i = 1 to n-1.  
……a) Pick element arr[i] and insert it into sorted sequence arr[0…i-1]

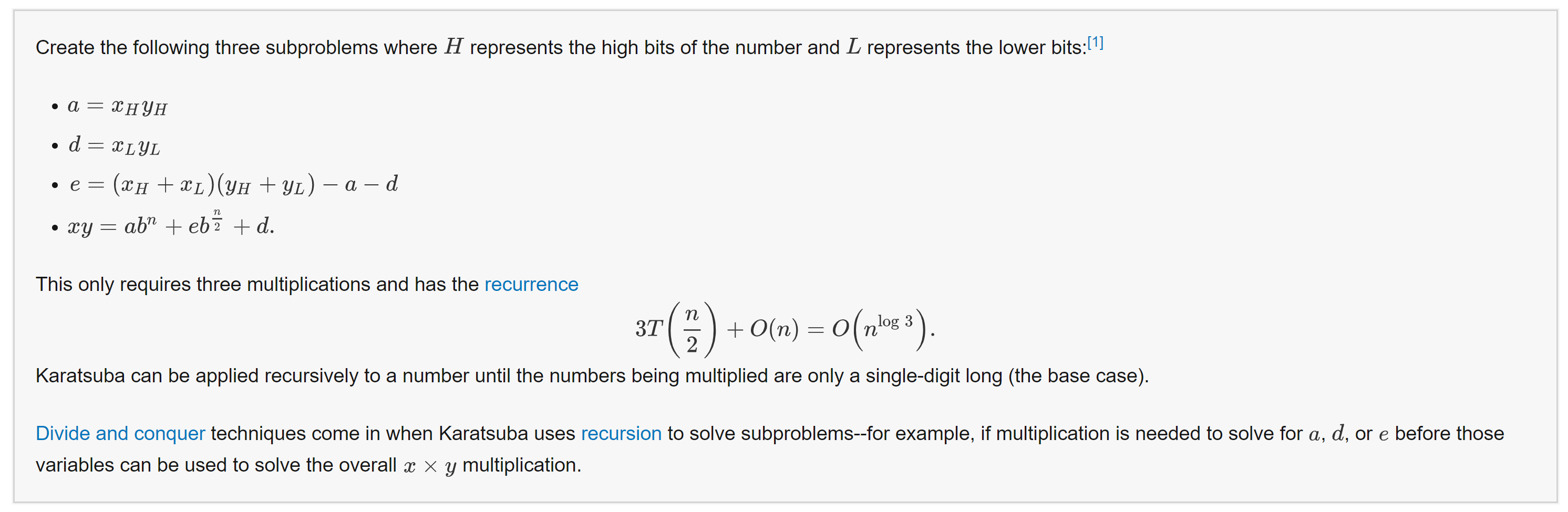


**Binary Search** **O(log n)**

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.



**Karatsubas Algo O(n^Log3)**



**Linear Selection of Medians**

**kthSmallest(arr[0..n-1], k)**

**1)** Divide arr[] into ⌈n/5⌉ groups where size of each group is 5

except possibly the last group which may have less than 5 elements.

**2)** Sort the above created ⌈n/5⌉ groups and find median

of all groups. Create an auxiliary array 'median[]' and store medians

of all ⌈n/5⌉ groups in this median array.

// Recursively call this method to find median of median[0..⌈n/5⌉-1]

**3)** medOfMed = kthSmallest(median[0..⌈n/5⌉-1], ⌈n/10⌉)

**4)** Partition arr[] around medOfMed and obtain its position.

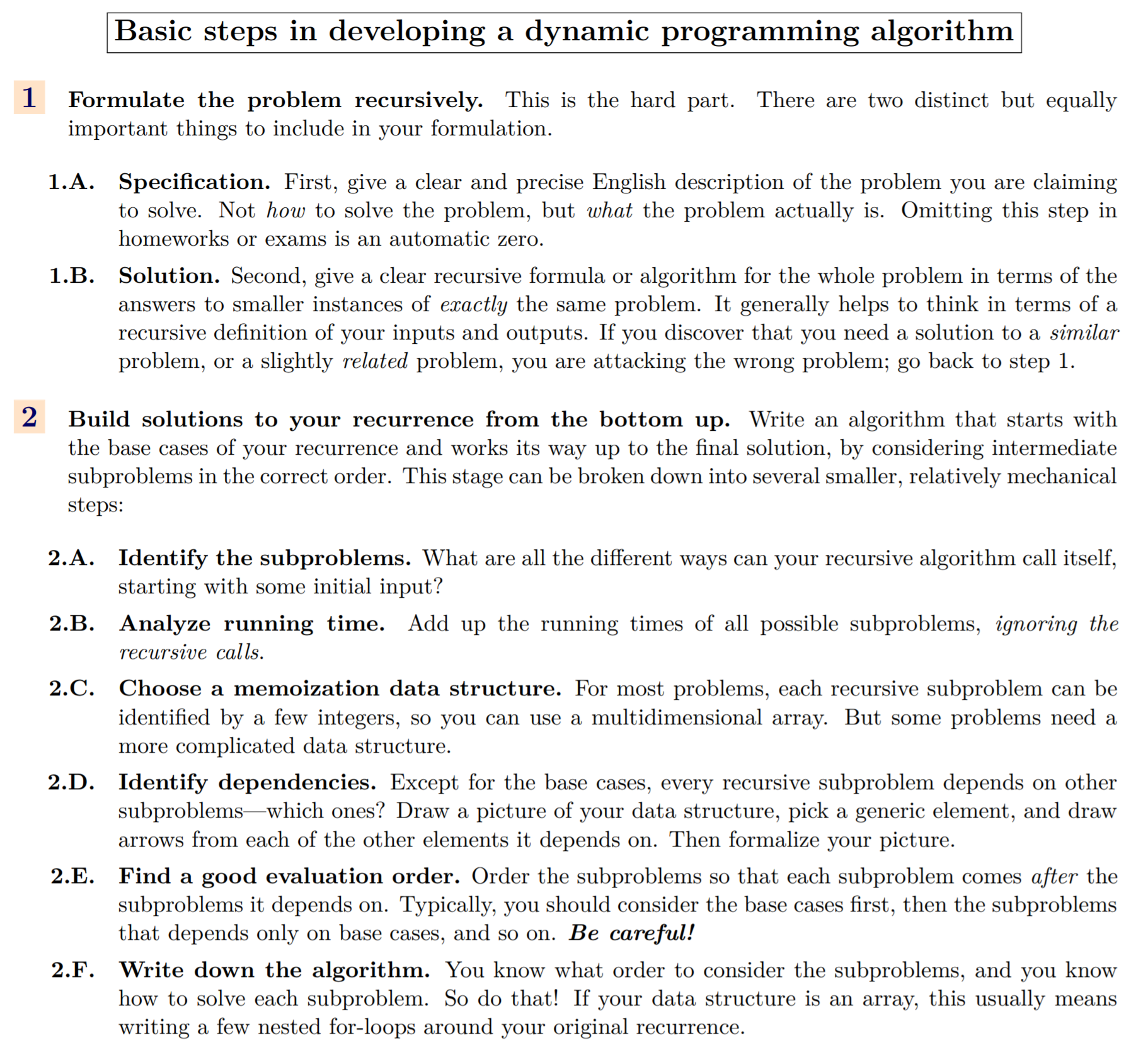
pos = partition(arr, n, medOfMed)

**5)** If pos == k return medOfMed

**6)** If pos > k return kthSmallest(arr[l..pos-1], k)

**7)** If pos < k return kthSmallest(arr[pos+1..r], k-pos+l-1)

**DP framework**

****

**BFS**

BFS (G, s) //Where G is the graph and s is the source node

let Q be queue.

Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.

mark s as visited.

while ( Q is not empty)

//Removing that vertex from queue,whose neighbour will be visited now

v = Q.dequeue( )

//processing all the neighbours of v

for all neighbours w of v in Graph G

if w is not visited

Q.enqueue( w ) //Stores w in Q to further visit its neighbour

mark w as visited.

**DFS**

DFS(G, u)

u.visited = true

for each v ∈ G.Adj[u]

if v.visited == false

DFS(G,v)

init() {

For each u ∈ G

u.visited = false

For each u ∈ G

DFS(G, u)

}

**BFS paths**

create a queue which will store path(s) of type vector

initialise the queue with first path starting from src

Now run a loop till queue is not empty

get the frontmost path from queue

check if the lastnode of this path is destination

if true then print the path

run a loop for all the vertices connected to the

current vertex i.e. lastnode extracted from path

if the vertex is not visited in current path

a) create a new path from earlier path and

append this vertex

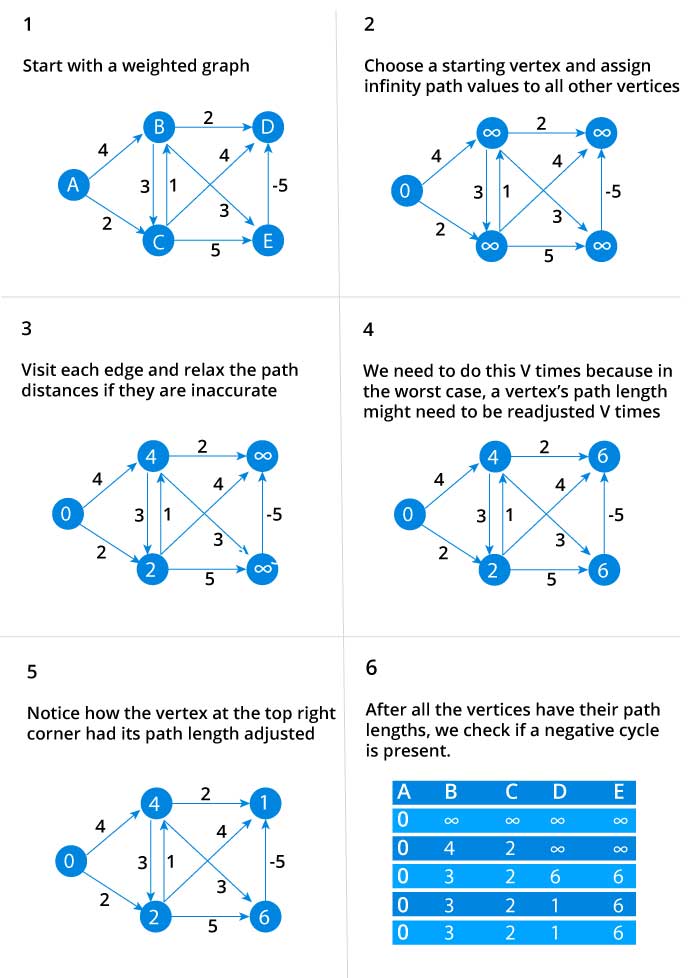
b) insert this new path to queue

**Djikstras algo**  *O*(*V* lg *V* +*E* lg *V*),

No negative weights

Algorithm  
**1)** Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.  
**2)** Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.  
**3)** While sptSet doesn’t include all vertices  
….**a)** Pick a vertex u which is not there in sptSet and has minimum distance value.  
….**b)** Include u to sptSet.  
….**c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

**Bellman ford** O(V E)



function bellmanFord(G, S)

for each vertex V in G

distance[V] <- infinite

previous[V] <- NULL

distance[S] <- 0

for each vertex V in G

for each edge (U,V) in G

tempDistance <- distance[U] + edge\_weight(U, V)

if tempDistance < distance[V]

distance[V] <- tempDistance

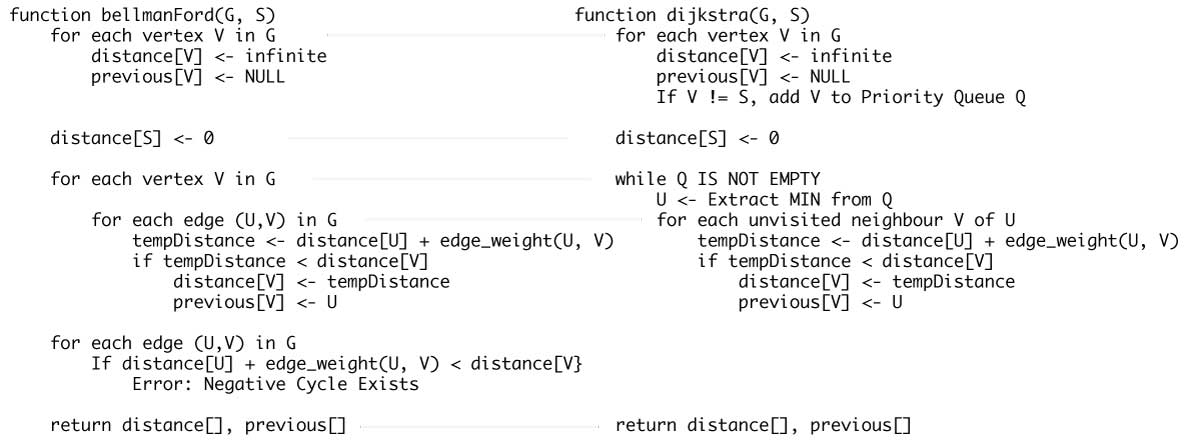
previous[V] <- U

for each edge (U,V) in G

If distance[U] + edge\_weight(U, V) < distance[V}

Error: Negative Cycle Exists

return distance[], previous[]



**Cases**

Shortest path, dir unweighted G – BFS, DAG – DP, Weighted – dijkstras

Topological sort: DFS All

Strongly connected components: DFS topological sort related

Finding a cycle: BFS, DFS

DFS trees and meta graph creation

**Example Problems**

**Smallest Subset that sum to a number: weighted, k elements, smallest # elements**

**Max Independent sets: weight based**

**Fib #’s**

**Split string into palindromes**

**Max independent sets**

**Longest x Subsequence**

**Edit Distance**

**See if a subset of an array can sum to a target number, can also be weight based, or subset of size k to sum to some target**