1. (i) Show the output. (ii) Repeat if the print statement is moved to be after the for loop.

```
def dfs(G):
  seen = {}
  for v in G: seen[v] = False
  for v in sorted(G):
    if not seen[v]: explore(G,v,seen)
def explore(G,v,seen):
  print v,
  seen[v] = True
  for nbr in G[v]:
    if not seen[nbr]:
      explore(G,nbr,seen)
G = {'A': ['F', 'E', 'C'],
     'B': ['H', 'G'],
     'C': ['F', 'A'],
     'D': [],
     'E': ['F', 'A'],
     'F': ['E', 'C', 'A'],
     'G': ['H', 'B'],
     'H': ['G', 'B']}
dfs(G)
```

2. Here is a simple form of quicksort: Use the first list element as splitter, and assume splitting preserves the relative order of each sublist. Complete the trace. Show the list after every partition.

```
def quicksort(list, start, end):
  if start < end: # so at least two elements
    split = partition(list, start, end)
    quicksort(list, start, split-1)
    quicksort(list, split+1, end)

[88 11 00 33 99 22 44 77 66 55]

QS(0,9)
++++++++++++++++++++++++++++++++
[11 00 33 22 44 77 66 55] 88 [99]

. QS(0,7)

??????????????????????

. QS(9,9)
```

3. A F E C B H G D
   E C F A G H B D

4. [88 11 00 33 99 22 44 77 66 55]

```
QS(0,9)
++++++++++++++++++++++++++++++++
[11 00 33 22 44 77 66 55] 88 [99]

. QS(0,7)
++++++++++++++++++++++++++
[00] 11 [33 22 44 77 66 55] 88 99

. . QS(0,0)

. . QS(2,7)
      ++++++++++++++++++++
00 11 [22] 33 [44 77 66 55] 88 99

. . . QS(2,2)

. . . QS(4,7)
            ++++++++++++++++
00 11 22 33 [] 44 [77 66 55] 88 99

. . . . QS(4,3)

. . . . QS(5,7)
                ++++++++++++
00 11 22 33 44 [66 55] 77 [] 88 99

. . . . . QS(5,6)
                ++++++++++
00 11 22 33 44 [55] 66 [] 77 88 99

. . . . . . QS(5,5)

. . . . . . QS(7,6)

. . . . . QS(8,7)

. QS(9,9)
```