COMP108 Algorithmic Foundations
Tutorial 8 (Suggested solution and Feedback)
w/c 27th March 2017

1. (a) c, f, e, h and g

   (b) 2

   (c) 5

   (d)

$$\begin{pmatrix} & a & b & c & d & e & f & g & h & k \\ a & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ b & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ c & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ d & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ e & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ f & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ g & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ h & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ k & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

   (e) A graph has an Euler circuit if and only if it is connected and all vertices have even degree.

   (f) G does not contain an Euler circuit because the vertices a, c, g and k have odd degree. Two edges are needed to add to the graph for an Euler circuit exists (e.g., (a, k) and (c, g)).

2.
```
// merge sort ascendingly
public void msort() {
   int[] data2 = new int[count];  // data2.length = count

   copy(data, 0, data2, 0, count);  // copy data[] to data2[]
   rec_msort(data2);  // invoke rec_msort on data2[]
   printAll(LONG, data2, count);
}
```

```
// recursively merge sort array[] assuming
// array.length gives the number of values in array[]
public void rec_msort(int [] array) {
   int [] a1, a2;
   int n, mid;

   n = array.length;
   mid = n/2;
   a1 = new int[mid];  // a1.length = mid
   a2 = new int[n-mid];  // a2.length = n-mid

   if (n > 1) {
       copy(array, 0, a1, 0, mid);
       copy(array, mid, a2, 0, n-mid);
       rec_msort(a1);
       rec_msort(a2);
       merge(a1, a2, array);
   }
}

// merge the array a1[] and a2[] into array[]
// assuming a1[] and a2[] have already been sorted ascendingly
public void merge(int[] a1, int[] a2, int [] array) {
   int i, j, k, p, q;

   p = a1.length;
   q = a2.length;
   i = j = k = 0;
   while (i<p && j<q) {
      if (a1[i] <= a2[j]) {
          array[k] = a1[i];
          i++;
      } else {
          array[k] = a2[j];
          j++;
      }
      k++;
   }
   if (i == p) {
        copy(a2, j, array, k, q-j);
   } else {
        copy(a1, i, array, k, p-i);
   }
}
```

3. The total number of handshakes of all people must be even because a handshake involves two people. People with an even (odd, respectively) number of handshakes will contribute an even value (odd value, respectively) to this total. To ensure that the total is even, the number of people with odd number of handshakes must be even.

   (Note, we can model the puzzle as an undirected graph in the following way. We represent every person as a vertex. Two vertices are connected by an edge if the two corresponding people shake hands. Then the number of handshakes a person has is equivalent to the degree of the corresponding vertex. The proof of "the number of vertices having odd degree is even" can be done similarly.)

   Alternatively,

   we can prove this using mathematical induction on the number of pairs of handshakes.

   Base case: When there is one pair of handshake, it is between two persons and each person has one handshake. Therefore, the number of people having odd number of handshake is two, which is even.

   Assume that the property holds when there are k pairs of handshakes. Now consider one more pair of handshake between two persons.

   There are three cases: (1) If both of them had even number of handshakes before, then the number of people having odd number of handshakes increases by two and hence stays even. (2) If both of them had odd number of handshakes before, then the number of people having odd number of handshakes decreases by two and hence stays even. (3) If one of them had even number of handshakes and the other odd number of handshakes, then the number of people having odd number of handshakes remain the same, and therefore stays even.

   In all three cases, the number of people having odd number of handshakes remain the same and the property holds for $k + 1$ pairs of handshakes.