Q1. Provide the recurrence relation, and the time complexity for the function (Q1) shown below.

```c
void Q1(int n){ f(n); } 
void f(int n){ if(n>1){ g(n/2); cout << n << endl;} } 
void g(int n){ if(n>1){ f(n/4); cout << n << endl;} }
```

Q2. The following algorithm takes a sorted array of floats (a) and determines if there exist two elements in whose sum is equal to val. The algorithm invokes FindSum(a, f, l, val), where n is the number of elements in the array. You are asked to provide time (show the running time function T(n)) and space analysis of this algorithm.

```c
FindSum(float a[], int f, int l, float val){
   if(f == l-1){ return val==a[f]+a[l]; }
   else{
      if (binsearch(a, f+1, l, val-a[f])) return true;
      else return FindSum(a, f+1, l, val);
   }
}
```

Q3. The following function draws recursive squares (called a fractal star). The drawing primitive `Box(x, y, n)` requires $4n$ steps to draw a square.

```c
void STAR( int x, int y, int n){
   if (n > 1){
      STAR(x-n , y+n , n/2);
      STAR(x+n , y+n , n/2);
      STAR(x-n , y-n , n/2);
      STAR(x+n , y-n , n/2);
      Box(x , y , n);
   }
}
```

Provide the recurrence relations and time and space complexity of the `STAR` function.

Q4. Devise a recursive “ternary” search in an array algorithm that first tests the element at position n/3 for equality with some key value (x), and then checks the element at 2n/3 and either discovers (x) or reduces the set size to one-third of the original. Model the algorithm as a recurrence relation and solve it to find the number of operations as a function of the array size (n) in the worst case.
Q5. Ancient Egyptians used the fact that each fraction can be expressed as a sum of different fractions with *unit numerators* to represent fractions (e.g., \( \frac{87}{110} = \frac{1}{2} + \frac{1}{5} + \frac{1}{11} \)).

You are asked to devise a greedy algorithm to find the Egyptian representation of a given fraction.

Q6. Given a graph \( G \), you are asked to design an algorithm to determine if \( G \) contains any loops. Extend your algorithm to count the number of loops in \( G \). Provide time and space analysis.

Q7. Repeat question 5 for hyper graphs. Where a hyper-graph is a graph like structure, given by: \( HG=(V, E) \). Where \( V \) is a finite set of nodes, and \( E \) is a finite set of hyper-edges. Unlike edges in simple graphs (that simply connect two nodes), hyper-edges in a hyper-graph may connect \( m \)-nodes. In general we say that for \( e \in E \) we have: \( e \neq \emptyset \) and \( e \subseteq V \).

Q8. Program 4.13 (page 236) generates optimal two-way merge trees. A generated tree can be used to optimally merge \( n \)-lists provided that the merge process merges two lists at a time. You are asked to:
1. Develop an algorithm to generate an optimal three-way merge pattern (i.e. all internal nodes have degree 3). The tree you generate should be used to optimally merge \( n \)-lists provided that the merge process mergers three lists at a time.
2. Can those three-way trees be used to generate Huffman-like codes for compressing data on a binary channel? Briefly explain your answer.

Q9. Provide the recurrence relation, and the time and space complexity for the function \( Q7 \) shown below.

```c
int Q7(int a[], int f, int l){
    int s=0;
    if((l-f)/10 < 10){
        for(int i=f; i<l; i++) s = s + a[i];
        return s;
    }
    else {
        for(int i=0; i<10; i = i+2)
        s = s + Q7(a, i*(((l-f)/10), (i+1)*(((l-f)/10)))
        return s;
    }
}
```
Q10. You are asked to design an algorithm for “making change.” The algorithm will be used in a money change machine that can accept any number of notes of 20, 10, 5, and 1 EGP, and can produce change in the form of EGP 0.50, 0.25, 0.10, 0.05 EGP coins. The algorithm must find the smallest number of coins that make up the required amount.

Q11. You are asked to devise a greedy algorithm for creating voting districts in a country. You are given a map of the country showing the locations and population of its cities. You are asked to create N districts such that there is minimum deviation in population between voting districts and that cities making up a voting district are connected.

Q12. Suppose you are given an array of size \( n \) sorted numbers that has been circularly shifted \( k \) positions to the right. For example, \([35, 42, 5, 15, 27, 29]\) is a sorted array that has been circularly shifted by \( k=2 \) positions, and \([27, 29, 35, 42, 5, 15]\) has been shifted \( k=4 \) positions.

(a) If the value of \( k \) is known, devise an \( O(1) \) algorithm to find the largest number in \( A \).

(b) If the value of \( k \) is unknown, devise an \( O(\log n) \) algorithm to find the largest number in \( A \).

Q13. The set-covering-problem is described as follows: you are given a finite set \( X \) and a collection \( F \) of subsets of \( X \). The problem is finding the minimum number of subsets in \( F \) whose union covers \( X \) (i.e., whose union equals \( X \)). For example, consider \( X = \{a, b, c, d\} \) and \( F = \{\{a, d\}, \{a, b, c\}, \{a\}, \{b\}, \{c\}\} \), then the minimum set cover is equal to \( \{a, d\} \cup \{a, b, c\} \). The set-cover-problem is a known hard problem, however you are asked to devise a greedy algorithm to provide the best possible solution using a greedy strategy. What is the complexity of your algorithm?

Q14. A vertex cover of a graph \( G = (V, E) \) is a subset of \( V \) such that the nodes in the subset are attached to all the edges in the graph. For example, for a graph \( G \) given by \( G = (\{(1,2,3,4), (1,2), (2,3), (2,4), (3,4)\}) \), such a vertex cover can be given by the set \( \{2,3\} \) (because node 2 touches the edges (1,2), (2,3), and (2,4); and node 3 touches edge (3,4)).

You are asked to devise a greedy algorithm to obtain a vertex cover (VC) of a graph \( G \) such that the \( |VC| \) is the smallest possible (using a greedy method). What is the time complexity of your solution?

Q15. You are given a set of strings (e.g., \( ss = \{abc, defa, faxy\} \)) and you are asked to devise a greedy algorithm to compute the shortest possible common superstring (using a greedy method). Where a superstring is a string that contains all the elements of the given set as substrings (for the example above, both \( s1 = abcdedafaxy \), \( s2 = abcddefaxy \) are superstrings for the set \( ss \); \( s1 \) is the trivial superstring that results from concatenating all the elements of \( ss \)).