Question 1:

Method 1:

We can start to analyze the problem from destination B. At B, we should have at least 0 gas. So with tank size we have, we can figure out the last gas station we should stop $G_{n.}$ At G_n, we should add enough gas for us to arrive destination B. Also, at G_n, we can figure out the farthest gas station G_{n-1} with full tank size. Then repeat this process until our tank size is bigger than the distance between A and gas station G_1.

Method 2:

We can start to analyze the problem from A. We need to find the farthest gas station G_1 that my car can arrive with full gas. At G_1, fill up the full tank and find the farthest gas station G_2 with full tank. Then repeat this process until our tank size is bigger than the gas need to use for the distance between B and $G_{n.}$

```
def gasstation_stops(distance, tankSize, gasList):
      currentLocation = 0;
      gasornot = [False]*distance
      #assign the gas station to correct elements
      for x in gasList:
            gasornot[i] = True

      stopCount = 0

      while tankSize + currentLocation < distance:
            updateLocation = currentLocation + tankSize
            #find the farthest gas station it can reach
            while gasornot[updateLocation] is False:
                  updateLocation -= 1

            stopCount += 1

            currentLocation = updateLocation
     return stopCount
```

Question 2

Let $p_1, p_2, \ldots, p_n$ be the values of the items and $w_1, w_2, \ldots, w_n$ be the weights of corresponding items. Also, W is the capacity of the sack. We can make $\frac{p_i}{w_i}$ be the weight value per unit, and make the items be in decreasing order according to $\frac{p_i}{w_i}$. We can take items from beginning until the bag is full by using the greedy strategy.

We can use contradiction to prove this question. After we take 1st item, the following subproblem becomes $p_{2,}, \ldots, p_n$ with weights $w_2, \ldots, w_n$. Also, the capacity becomes $W - w_1$. Then the subproblem has the same solution as the previous problem. The process will continue until there are no items left in the bag or the weight *w* becomes 0.

Suppose our solution is $s_1, s_2, \ldots s_n$ is the solution where $s_1 < \min(w_1, W)$. So we have more space for item S_x, we need to prove that same weight of S_x has more value than S_1. However, we have the decreasing sort which means 1st item has the highest value. So we have a contraction here. Thus, the problem is a greedy choice property problem.

At first, I couldn't think of any way to prove this question. After I reviewed the online solution, I just figure out the contraction method is a good way to prove it. Then I tried my way to prove this question by using contradiction. Also, the online solution used the increasing order while my solution used the decreasing order.

Question 3:

We can use contradiction to prove this problem.

As described in Problem1 method2, the best solution is to stop at gas station G1, G2……Gm……Gn-1, Gn (Gm is any gas station between G1 and Gn).

<u>Assume we stop at Gmx rather than Gm and Gmx is before Gm, is better than stopping at Gm. Which means at Gmx I did not utilize the full tank gas which is added at Gm-1.</u>

So if we fill up the full tank at Gmx, our gas will not allow us to go to gas station Gmx+1 which is closer to B comparing with Gm+1.

Also, at Gmx+1, our full tank gas cannot let us get to gas station Gmx+2 which is closer to B comparing with Gm+2.

So repeat this process.

.
.
.
.
.
At Gnx-1, with our full tank gas, we cannot reach to gas station Gnx which is closer to B comparing with Gn.

We have two conditions here:

1) the distance between Gxn and B is larger than the distance of tank size;
   in this condition, we need to stop 1 or more stops to fill up gas. Then we will stop at least n+1 times.
2) the distance between Gxn and B is not larger than the distance of tank size;
   in this condition, we will stop n times.

<u>Thus, stopping at Gmx is not better than stopping at Gm. This is contradiction to our assumption that stopping at Gmx is better than Gm.</u>

Question 4:

```
class MULNUM{
      static int minProduct(int arr[], int n){

            int negativeMax = Integer.MIN_VALUE;
            int positiveMin = Integer.MAX_VALUE;
            int neg_count = 0;
            int zero_count = 0;
            int mulnumber = 1;

            for(int i = 0; i<n; i++ ){
                  //count how many zero in arr[i]
                  if(arr[i] == 0){
                        zero_count++;
                        continue;
                  }
                  //count how many negative numbers in arr[i]
                  //find max negative value
                  if(arr[i] < 0){
                        neg_count++;
                        negativeMax = Math.max(negativeMax,
                                          a[i])
                  }
                  //find minimum positive value
                  if(arr[i] > 0 && a[i] < positiveMin)
                  {
                        positiveMin = arr[i];
                  }

                        mulnumber *= arr[i];
            }

                  //if all numbers >= 0 and arr[i] has zero
                  if(zero_count == n || (zero_count > 0 &&
                          neg_count==0 )) {
                        return 0;
                  }
                  // if no numbers are negative in arr[i]
                  if(neg_count == 0){
                        return positiveMin;
                  }
                  // if mulnumber >= 0 and a[i] has negative
                   //numbers
                  if(neg_count != 0 && mulnumber >= 0){
                       mulnumber = mulnumber / negativeMax;
```

```
                        }
                return  mulnumber;

        }
}
```

The complexity is O(N).

Comparing with online solution, I forgot to discuss the condition: if only one number in arr[i], then just return it.

Also, the online solution makes “int posmin = Integer.MIN_VALUE”. With my solution, int positiveMin = Integer.MAX_VALUE. I think the online solution has a typo here.