QUESTION 1 SOLUTION

```
public int removeDuplicates(int[] nums) {
    if(nums == null || nums.length == 0)
    {
      return 0;
    }

    int slow = 0;
    int fast = 1;
    int currentValue = nums[0];

    while(fast < nums.length)
    {
      while(fast < nums.length && nums[fast] == currentValue)
      {
        fast++;
      }

      if(fast < nums.length)
      {
        slow++;
        nums[slow] = nums[fast];
        currentValue = nums[fast];
      }
    }
```

```
        return slow+1;
    }
```

QUESTION 2 SOLUTION

```
class Solution {
    public void solveSudoku(char[][] board) {
        boolean[][] row = new boolean[9][10];
        boolean[][] col = new boolean[9][10];
        boolean[][][] square = new boolean[3][3][10];
        for(int i=0;i<9;i++) {
            for(int j=0;j<9;j++) {
                if(board[i][j] == '.') continue;
                int v = board[i][j] - '0';
                row[i][v] = true;
                col[j][v] = true;
                square[i/3][j/3][v] = true;
            }
        }
        solve(board,0,row,col,square);
    }
    public boolean solve(char[][] board,int index,boolean[][] row,boolean[][] col,boolean[][][] square) {
        if(index > 80) return true;
        int i = index/9, j = index%9;
        if(board[i][j] != '.') {
            return solve(board,index+1,row,col,square);
        }
        for(int v=1;v<=9;v++) {
            if(row[i][v] || col[j][v] || square[i/3][j/3][v]) continue;
            row[i][v] = true;
            col[j][v] = true;
            square[i/3][j/3][v] = true;
            board[i][j] = (char) (v + '0');
            if(solve(board,index+1,row,col,square)) return true;
            row[i][v] = false;
            col[j][v] = false;
            square[i/3][j/3][v] = false;
            board[i][j] = '.';
        }
        return false;
    }
}
```

QUESTION 3 SOLUTION

```
public int maxProfit(int[] prices) {
  int l=prices.length;


  int maxProfit = 0;
  int minPrice = prices[0];
  for(int i=1; i<l; i++) {
    maxProfit = Math.max(maxProfit, prices[i]-minPrice);
    minPrice = Math.min(minPrice, prices[i]);
  }
```

```
 return maxProfit;

}
```

QUESTION 4 SOLUTION

```
class Solution {

  public int searchInsert(int[] nums, int target) {

    for(int i =0;i<nums.length;i++){

      if(target<=nums[i]){

        return i;

      }

    }

  return nums.length;

}

}
```

QUESTION 5 SOLUTION

```
class Solution {

  public int canCompleteCircuit(int[] gas, int[] cost) {

    int startPointIdx = 0;

    int additionalGasNeeded = 0;

    int totalGasInTask = 0;

    for (int i = 0; i < gas.length; i++) {

      totalGasInTask = totalGasInTask + gas[i] - cost[i];

      if (totalGasInTask < 0) {

        additionalGasNeeded = additionalGasNeeded + totalGasInTask;

        totalGasInTask = 0;
```

```
                startPointIdx = i + 1;
            }
        }
        if (totalGasInTask + additionalGasNeeded >= 0) {
            return startPointIdx;
        } else {
            return -1;
        }
    }
}
```

QUESTION 6 SOLUTION

```
class Solution {
	public int rob(int[] nums) {
		if(nums.length == 0) return 0;
		if(nums.length == 1) return nums[0];
		if(nums.length == 2) return Math.max(nums[0], nums[1]);

		int[] dp = new int[nums.length];
		dp[0] = nums[0];
		dp[1] = Math.max(nums[0], nums[1]);

		for(int i = 2; i < nums.length; i++) {
			dp[i] = Math.max(dp[i - 1], dp[i - 2] + nums[i]);
		}
```

```
                    return dp[nums.length - 1];

          }

}
```