```
public class StriverGreedy {
    //**********************N meetings in a room***********************************8 */
    class meeting{
        int start;
        int end;
        int pos;

        meeting(int s,int e,int p){
            this.start=s;
            this.end=e;
            this.pos=p;
        }
    }
    class meetingComparator implements Comparator<meeting>
    {
        @Override
        public int compare(meeting o1,meeting o2)
        {
            if(o1.end<o2.end)
                return -1;
            else if(o1.end>o2.end)
                return 1;
            else if(o1.pos<o2.pos)
                return -1;
            return 1;
        }
    }
    class Solution
    {
        public static int maxMeetings(int start[], int end[], int n)
        {
            // add your code here
            ArrayList<meeting> meet=new ArrayList<>();
            for(int i=0;i<n;i++){
                meet.add(new meeting(start[i],end[i],i+1));
            }
            meetingComparator mc=new meetingComparator();
            Collections.sort(meet,mc);
            ArrayList<Integer> ans=new ArrayList<>();
            ans.add(meet.get(0).pos);
            int limit=meet.get(0).end;
            for(int i=1;i<n;i++){
                if(meet.get(i).start>limit){
                    limit=meet.get(i).end;
                    ans.add(meet.get(i).pos);
                }
            }
            return ans.size();
        }
```

```
    }

    //****************Minimum number of platforms required for a Railway*****************/
    class Solution
    {
    //Function to find the minimum number of platforms required at the
    //railway station such that no train waits.
    static int findPlatform(int arr[], int dep[], int n)
    {
        // add your code here
        Arrays.sort(arr);
        Arrays.sort(dep);
        int plat=1,res=1;
        int i=1,j=0;
        while(i<n && j<n){
            if(arr[i]<=dep[j]){
                plat++;
                i++;
            }
            else if(arr[i]>dep[j]){
                plat--;
                j++;
            }
            if(res<plat){
                res=plat;
            }
        }
        return res;
    }
}
    //*****************Job Scheduling************************************************** */
    class Solution
{
    //Function to find the maximum profit and the number of jobs done.
    int[] JobScheduling(Job arr[], int n)
    {
        // Your code here
        Arrays.sort(arr,(a,b)->(b.profit-a.profit));
        int maxi=0;
        for(int i=0;i<n;i++){
            maxi=Math.max(maxi,arr[i].deadline);
        }
        int result[]=new int[maxi+1];
        for(int j=0;j<maxi+1;j++){
            result[j]=-1;
        }
        int profit=0;
        int countJobs=0;
        for(int k=0;k<n;k++){
```

```
            for(int m=arr[k].deadline;m>=1;m--){
                if(result[m]==-1){
                    result[m]=k;
                    countJobs++;
                    profit+=arr[k].profit;
                    break;
                }
            }
        }
        int fin[]={countJobs,profit};
        return fin;
    }
}
    //******************Fractional KnapSack********************************************** */
    class itemComparator implements Comparator<Item>{
        @Override
        public int compare(Item a,Item b){
            double r1=(double)a.value/(double)a.weight;
            double r2=(double)b.value/(double)b.weight;
            if(r1<r2){
                return 1;
            }
            else if(r1>r2){
                return -1;
            }
            else return 0;
        }
    }
    class Solution
    {
        //Function to get the maximum total value in the knapsack.
        double fractionalKnapsack(int W, Item arr[], int n)
        {
            // Your code here
            Arrays.sort(arr,new itemComparator());
            int currWeight=0;
            double maxProfit=0.0;
            for(int i=0;i<n;i++){
                if(currWeight+arr[i].weight<=W){
                    currWeight+=arr[i].weight;
                    maxProfit+=arr[i].value;
                }
                else{
                    int remain=W-currWeight;
                    maxProfit+=((double)arr[i].value/(double)arr[i].weight)*(double)remain;
                    break;
                }
            }
            return maxProfit;
```

```
        }
    }
    //**************Minimum Number of Coins********************************************** */
    public class Solution
{
public static int findMinimumCoins(int V)
    {
        // Write your code here.
      ArrayList<Integer> ans=new ArrayList<>();
      int coins[]={1,2,5,10,20,50,100,500,1000};
      int n = coins.length;
      for (int i=n-1;i>=0;i--){
      while(V>=coins[i]) {
        V-=coins[i];
        ans.add(coins[i]);
      }
    }
      return ans.size();
    }
}
```