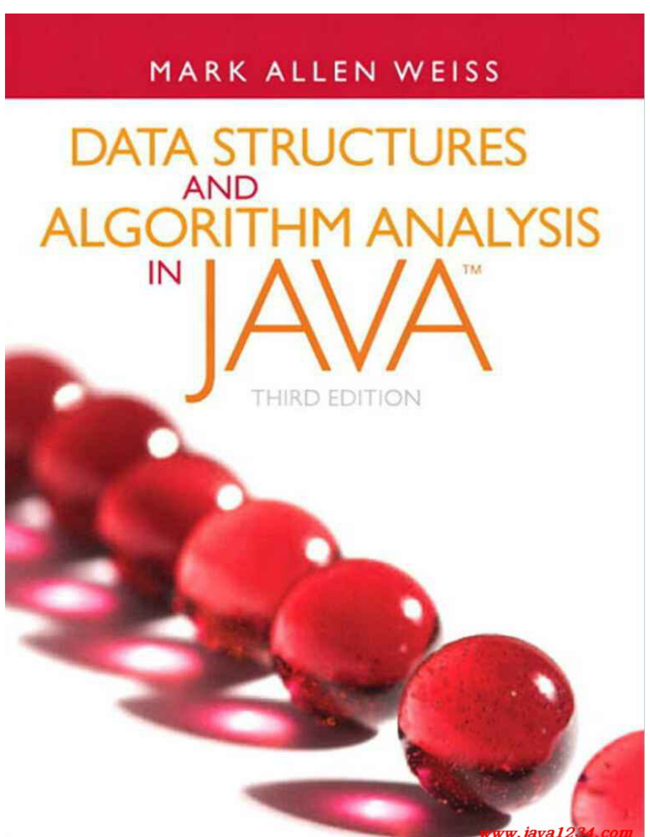
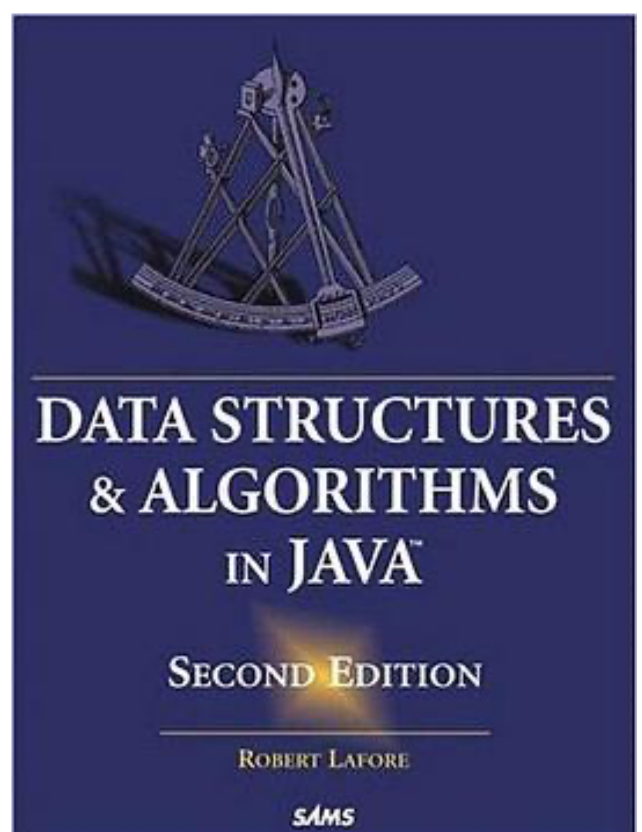
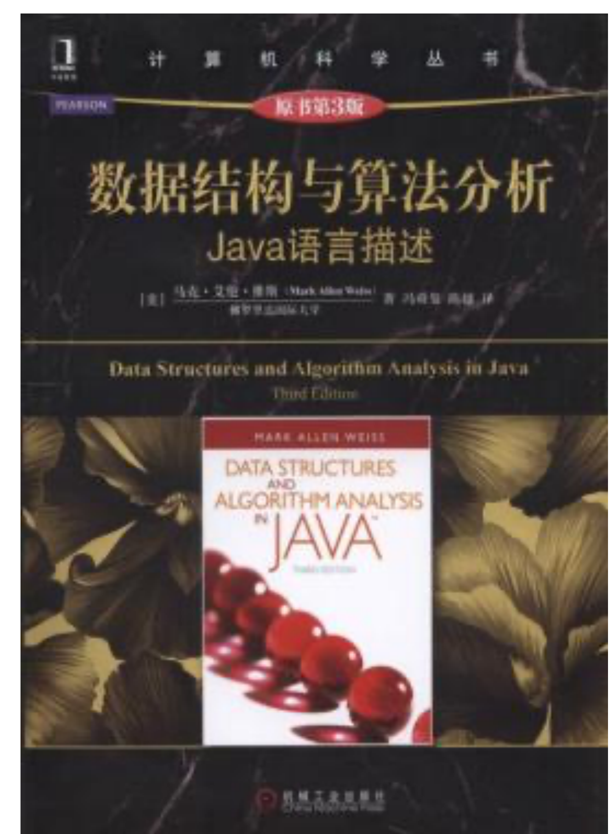
Topic 17 –
Greedy Algorithm







- **Greedy Algorithm** 贪心算法
- **Examples of Greedy Algorithm** 三个实例
    - **Minimum rotations to unlock a circular lock**
    - **Minimum product subset of an array**
    - **Find minimum number of coins**

Greedy Algorithm

选择当前最好的

- A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment, without worrying about the future result it would bring. 选择
- In other words, the **locally best choices** aim at producing globally best results.

Greedy Algorithm

- **This algorithm may not be the best option for all the problems.**
- **It may produce wrong results in some cases.**

单向，向前找

- This algorithm never goes back to reverse the decision made. This algorithm works in a top-down approach.
- The main advantage of this algorithm is:
  - The algorithm is **easier to describe**.
  - This algorithm can **perform better** than other algorithms (but, not in all cases).

Feasible Solution

- A feasible solution is the one that provides the optimal solution to the problem.

Examples of Greedy Algorithm

- Minimum rotations to unlock a circular lock
- Minimum product subset of an array 圆锁.
- Find minimum number of coins

Minimum rotations to unlock a circular lock

- You are given a lock which is made up of n-different circular rings and each ring has 0-9 digit printed serially on it.
- Initially all n-rings together show a n-digit integer but there is particular code only which can open the lock.
- You can rotate each ring any number of time in either direction.
- You have to find the minimum number of rotation done on rings of lock to open the lock.

Minimum rotations to unlock a circular lock

- **Examples:**
  - Input :
    - Input = 2345, Unlock code = 5432
  - Output : Rotations required = 8
  - Explanation :
    - 1st ring is rotated thrice as 2->3->4->5
    - 2nd ring is rotated once as 3->4
    - 3rd ring is rotated once as 4->3
    - 4th ring is rotated thrice as 5->4->3->2

Minimum rotations to unlock a circular lock

- **Examples:**
  - Input :
    - Input = 1919, Unlock code = 0000
  - Output : Rotations required = 4
  - Explanation :
    - 1st ring is rotated once as 1->0
    - 2nd ring is rotated once as 9->0
    - 3rd ring is rotated once as 1->0
    - 4th ring is rotated once as 9->0

Minimum rotations to unlock a circular lock

- For a single ring we can rotate it in any of two direction forward or backward as:
  - 0->1->2….->9->0
  - 9->8->….0->9

Greedy

- But we are concerned with minimum number of rotation required so we should choose ***min (abs(a-b), 10-abs(a-b))*** as *a-b* denotes the number of forward rotation and *10-abs(a-b)* denotes the number of backward rotation for a ring to rotate from a to b.
- Further we have to find minimum number for each ring that is for each digit.
- So starting from right most digit we can easily the find minimum number of rotation required for each ring and end up at left most digit.

Minimum rotations to unlock a circular lock

```
// Java program for min rotation to unlock
class GFG {

   // function for min rotation
   static int minRotation(int input, int unlock_code)
   {
      int rotation = 0;
      int input_digit, code_digit;

      // iterate till input and unlock code become 0
      while (input>0 || unlock_code>0) {

            //Next page

      }

      return rotation;
   }
```

Minimum rotations to unlock a circular lock

```
// iterate till input and unlock code become 0
while (input>0 || unlock_code>0) {

   // input and unlock last digit as reminder
   input_digit = input % 10;
   code_digit = unlock_code % 10;

   // find min rotation
   rotation += Math.min(Math.abs(input_digit
           - code_digit), 10 - Math.abs(
             input_digit - code_digit));

   // update code and input
   input /= 10;
   unlock_code /= 10;
}
```

5

Minimum rotations to unlock a circular lock

```
    // driver code
    public static void main (String[] args) {
    int input = 28756;
    int unlock_code = 98234;
    System.out.println("Minimum Rotation = "+
                minRotation(input, unlock_code));
    }
}
```

Output:

Minimum Rotation = 12

Examples of Greedy Algorithm

- **Minimum rotations to unlock a circular lock**
- **Minimum product subset of an array**
- **Find minimum number of coins**

Minimum product subset of an array

- Given an array a, we have to find minimum product possible with the subset of elements present in the array.
- The minimum product can be single element also.
- **Examples:**
  - **Input_1:** a[] = { -1, -1, -2, 4, 3 }
    - **Output_1:** -24
    - **Explanation:** (-2 * -1 * -1 * 4 * 3) = -24
  - **Input_2:** a[] = { -1, 0 }
    - **Output_2:** -1
    - **Explanation:** -1(single element)
  - **Input_3:** a[] = { 0, 0, 0 }
    - **Output_3:** 0

Minimum product subset of an array

- A simple solution is to generate all subsets, find product of every subset and return minimum product.
  - List all possible
- A better solution is to use the below facts.
  1. If there are **even number of negative numbers** and **no zeros**, the result is the product of all except the largest valued negative number.
  2. If there are an **odd number of negative numbers** and **no zeros**, the result is simply the product of all.
  3. If there are **zeros** and positive, **no negative**, the result is 0.
  4. The exceptional case is when there is no negative number and all other elements positive then our result should be the first minimum positive number.

Minimum product subset of an array

```
// Java program to find maximum product of a subset.
class GFG {

    static int minProductSubset(int a[], int n)
    {
        if (n == 1)
            return a[0];

        int negmax = Integer.MIN_VALUE;
        int posmin = Integer.MAX_VALUE;
        int count_neg = 0, count_zero = 0;
        int product = 1;
```

Minimum product subset of an array

```
for (int i = 0; i < n; i++)
{
    // count the zero numbers
    if(a[i] == 0) count_zero++;
    else {
        // count the negetive numbers
        if(a[i] < 0) {
            count_neg++;
            if(a[i] > negmax) negmax = a[i] ;
        }
        // find the minimum positive number
        if(a[i] > 0 && a[i] < posmin) posmin = a[i];

        product *= a[i];
    }
}
```

Minimum product subset of an array

题似不对??)

```
// main function
public static void main(String[] args) {
    int a[] = { -1, -1, -2, 4, 3 };
    int n = 5;

    System.out.println(minProductSubset(a, n));
}
}
```

Output: -24

Minimum product subset of an array

```
// if there are all zeroes
// or zero is present but no
// negetive number is present
if (count_zero > 0)
    if(count_neg == 0)
        return 0;
    else {
        if (count_neg % 2 == 1)  //odd neg.
            return product;
        else
            retuen product / negmax;
    }
else
    return posmin;
}
```

Examples of Greedy Algorithm

- **Minimum rotations to unlock a circular lock**
- **Minimum product subset of an array**
- **Find minimum number of coins**

Find Minimum Number of Coins

- Given a value V, if we want to make a change for V Rs, and we have an infinite supply of each of the denominations in Indian currency, i.e., we have an infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change?
- **Examples:**
  - **Input_1:** V = 70
    - **Output_1:** 2
    - We need a 50 Rs note and a 20 Rs note.
  - **Input_2:** V = 121
    - **Output_2:** 3
    - We need a 100 Rs note, a 20 Rs note and a 1 Rs coin.

Find Minimum Number of Coins

- **Solution:** Greedy Approach.
- **Approach:**
  - A common intuition would be to take coins with greater value first. This can reduce the total number of coins needed.
  - Start from the largest possible denomination and keep adding denominations while the remaining value is greater than 0.

Find Minimum Number of Coins

- **Algorithm:**
  1. Sort the array of coins in decreasing order.
  2. Initialize result as empty.
  3. Find the largest denomination that is smaller than current amount.
  4. Add found denomination to result. Subtract value of found denomination from amount.
  5. If amount becomes 0, then print result.
  6. Else repeat steps 3 and 4 for new value of V.

Find Minimum Number of Coins

```
// Java program to find minimum number of denominations
import java.util.Vector;

class GFG{

    // All denominations of Indian Currency
    static int deno[] = {1, 2, 5, 10, 20, 50, 100, 500, 1000};
    static int n = deno.length;
```

Find Minimum Number of Coins

```
static void findMin(int V) {
    Vector<Integer> ans = new Vector<>(); // Initialize result
    // Traverse through all denomination
    for (int i = n - 1; i >= 0; i--) {
        // Find denominations
        while (V >= deno[i]){
            V -= deno[i];
            ans.add(deno[i]);
        }
    }
    // Print result
    for (int i = 0; i < ans.size(); i++){
        System.out.print(" " + ans.elementAt(i));
    }
}
```

Find Minimum Number of Coins

```
    // Driver code
    public static void main(String[] args)
    {
        int n = 93;
        System.out.print("Following is minimal number "
            +"of change for " + n + ": ");
        findMin(n);
    }
}
```

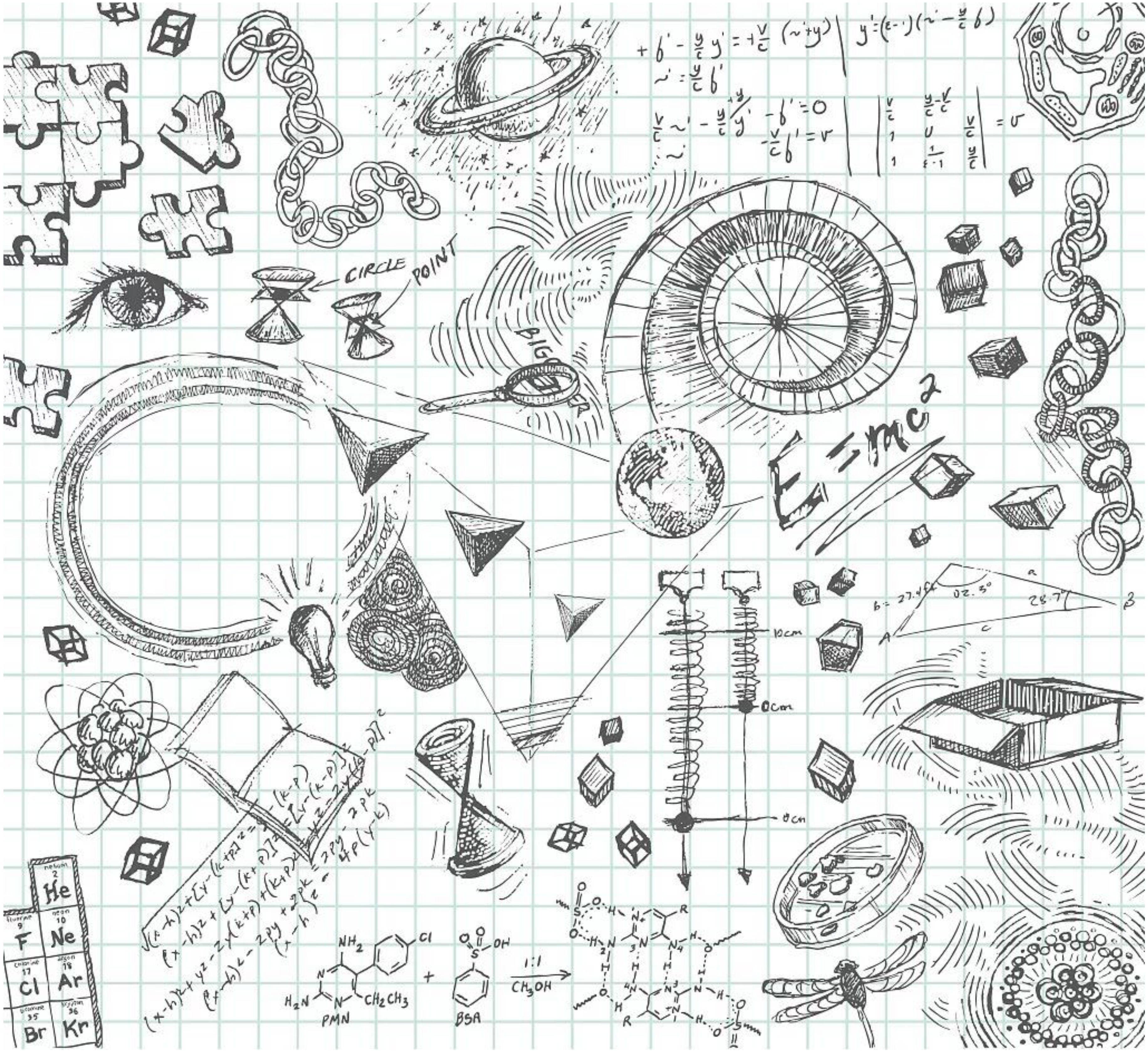
Output:

Following is minimal number of change for 93: 50 20 20 2 1

Find Minimum Number of Coins

- **Complexity Analysis:**
  - **Time Complexity:** O(V).
  - **Auxiliary Space:** O(1) as no additional space is used.

- **Note:**
  - The above approach may not work for all denominations.
  - For example, it doesn't work for denominations {9, 6, 5, 1} and V = 11.
  - The above approach would print 9, 1 and 1.
  - But we can use 2 denominations 5 and 6.
  - For general input, dynamic programming approach can solve it.



CIRCLE
POINT
$E=mc^2$
10cm
0cm
He
Ne
F
Cl
Ar
Br
Kr
PMN
BSA
CH_3OH