**[Learn more about Greedy Algorithms in DSA Self Paced Course](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=articles+greedy_lp+header_link_click&utm_campaign=dsa+course+tracker)**

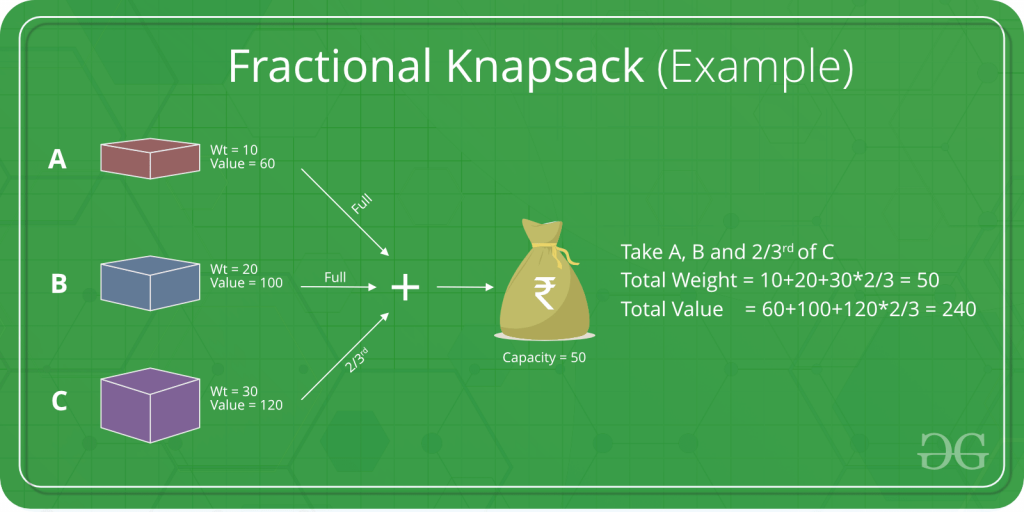
[**Practice Problems on Greedy Algorithms**](https://practice.geeksforgeeks.org/explore/?category%5B%5D=Greedy&page=1&category%5B%5D=Greedy&utm_source=geeksforgeeks&utm_medium=articles+greedy_lp+header_link_click&utm_campaign=practice+tracker)

[**Recent Articles on Greedy Algorithms**](https://www.geeksforgeeks.org/category/algorithm/greedy/)

**What is Greedy Algorithm?**

*Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are the best fit for Greedy.*

For example consider the [Fractional Knapsack Problem](https://www.geeksforgeeks.org/fractional-knapsack-problem/). The local optimal strategy is to choose the item that has maximum value vs weight ratio. This strategy also leads to a globally optimal solution because we are allowed to take fractions of an item.



**Topics:**

* [Introduction](https://www.geeksforgeeks.org/greedy-algorithms/#introduction)
* [Standard Greedy Algorithms](https://www.geeksforgeeks.org/greedy-algorithms/#standardgreedy)
* [Greedy Problems on Array](https://www.geeksforgeeks.org/greedy-algorithms/#greedyarray)
* [Greedy Problems on Operating System](https://www.geeksforgeeks.org/greedy-algorithms/#greedyos)
* [Greedy Problems on Graph](https://www.geeksforgeeks.org/greedy-algorithms/#greedygraph)
* [Approximate Greedy Algorithms for NP Complete Problems](https://www.geeksforgeeks.org/greedy-algorithms/#approximate)
* [Greedy for Special Cases of DP](https://www.geeksforgeeks.org/greedy-algorithms/#greedydp)
* [Some Practice problems on Greedy](https://www.geeksforgeeks.org/greedy-algorithms/#practiceproblems)
* [Quick Links](https://www.geeksforgeeks.org/greedy-algorithms/#quickLinks)

**Introduction:**

1. [Introduction to Greedy Algorithm – Data Structures and Algorithm Tutorials](https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/)
2. [Greedy Algorithms (General Structure and Applications)](https://www.geeksforgeeks.org/greedy-algorithms-general-structure-and-applications/)
3. [Difference between Greedy Algorithm and Divide and Conquer Algorithm](https://www.geeksforgeeks.org/difference-between-greedy-algorithm-and-divide-and-conquer-algorithm/)
4. [Greedy approach vs Dynamic programming](https://www.geeksforgeeks.org/greedy-approach-vs-dynamic-programming/)
5. [Comparison among Greedy, Divide and Conquer and Dynamic Programming algorithm](https://www.geeksforgeeks.org/comparison-among-greedy-divide-and-conquer-and-dynamic-programming-algorithm/)

**Standard Greedy Algorithms:**

1. [Activity Selection Problem](https://www.geeksforgeeks.org/greedy-algorithms-set-1-activity-selection-problem/)
2. [Job Sequencing Problem](https://www.geeksforgeeks.org/job-sequencing-problem-set-1-greedy-algorithm/)
3. [Huffman Coding](https://www.geeksforgeeks.org/greedy-algorithms-set-3-huffman-coding/)
4. [Huffman Decoding](https://www.geeksforgeeks.org/huffman-decoding/)
5. [Water Connection Problem](https://www.geeksforgeeks.org/water-connection-problem/)
6. [Minimum Swaps for Bracket Balancing](https://www.geeksforgeeks.org/minimum-swaps-bracket-balancing/)
7. [Egyptian Fraction](https://www.geeksforgeeks.org/greedy-algorithm-egyptian-fraction/)
8. [Policemen catch thieves](https://www.geeksforgeeks.org/policemen-catch-thieves/)
9. [Fitting Shelves Problem](https://www.geeksforgeeks.org/fitting-shelves-problem/)
10. [Assign Mice to Holes](https://www.geeksforgeeks.org/assign-mice-holes/)

**Greedy Problems on Array:**

1. [Minimum product subset of an array](https://www.geeksforgeeks.org/minimum-product-subset-array/)
2. [Maximize array sum after K negations using Sorting](https://www.geeksforgeeks.org/maximize-array-sum-after-k-negations-using-sorting/)
3. [Minimum sum of product of two arrays](https://www.geeksforgeeks.org/minimum-sum-product-two-arrays/)
4. [Minimum sum of absolute difference of pairs of two arrays](https://www.geeksforgeeks.org/minimum-sum-absolute-difference-pairs-two-arrays/)
5. [Minimum increment/decrement to make array non-Increasing](https://www.geeksforgeeks.org/minimum-incrementdecrement-to-make-array-non-increasing/)
6. [Sorting array with reverse around middle](https://www.geeksforgeeks.org/sorting-array-reverse-around-middle/)
7. [Sum of Areas of Rectangles possible for an array](https://www.geeksforgeeks.org/sum-area-rectangles-possible-array/)
8. [Largest lexicographic array with at-most K consecutive swaps](https://www.geeksforgeeks.org/largest-lexicographic-array-with-at-most-k-consecutive-swaps/)
9. [Partition into two subarrays of lengths k and (N – k) such that the difference of sums is maximum](https://www.geeksforgeeks.org/partition-into-two-subarrays-of-lengths-k-and-n-k-such-that-the-difference-of-sums-is-maximum/)

**Greedy Problems on Operating System:**

1. [First Fit algorithm in Memory Management](https://www.geeksforgeeks.org/program-first-fit-algorithm-memory-management/)
2. [Best Fit algorithm in Memory Management](https://www.geeksforgeeks.org/program-best-fit-algorithm-memory-management/)
3. [Worst Fit algorithm in Memory Management](https://www.geeksforgeeks.org/program-worst-fit-algorithm-memory-management/)
4. [Shortest Job First Scheduling](https://www.geeksforgeeks.org/program-shortest-job-first-sjf-scheduling-set-1-non-preemptive/)
5. [Job Scheduling with two jobs allowed at a time](https://www.geeksforgeeks.org/job-scheduling-two-jobs-allowed-time/)
6. [Program for Optimal Page Replacement Algorithm](https://www.geeksforgeeks.org/program-optimal-page-replacement-algorithm/)

**Greedy Problems on Graph:**

1. [Kruskal’s Minimum Spanning Tree](https://www.geeksforgeeks.org/greedy-algorithms-set-2-kruskals-minimum-spanning-tree-mst/)
2. [Prim’s Minimum Spanning Tree](https://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-minimum-spanning-tree-mst-2/)
3. [Boruvka’s Minimum Spanning Tree](https://www.geeksforgeeks.org/greedy-algorithms-set-9-boruvkas-algorithm/)
4. [Dijkastra’s Shortest Path Algorithm](https://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/)
5. [Dial’s Algorithm](https://www.geeksforgeeks.org/dials-algorithm-optimized-dijkstra-for-small-range-weights/)
6. [Minimum cost to connect all cities](https://www.geeksforgeeks.org/minimum-cost-connect-cities/)
7. [Max Flow Problem Introduction](https://www.geeksforgeeks.org/max-flow-problem-introduction/)
8. [Number of single cycle components in an undirected graph](https://www.geeksforgeeks.org/number-of-simple-cyclic-components-in-an-undirected-graph/)

**Approximate Greedy Algorithm for NP Complete:**

1. [Set cover problem](https://www.geeksforgeeks.org/set-cover-problem-set-1-greedy-approximate-algorithm/)
2. [Bin Packing Problem](https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/)
3. [Graph Coloring](https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/)
4. [K-centers problem](https://www.geeksforgeeks.org/k-centers-problem-set-1-greedy-approximate-algorithm/)
5. [Shortest superstring problem](https://www.geeksforgeeks.org/shortest-superstring-problem/)
6. [Approximate solution for Travelling Salesman Problem using MST](https://www.geeksforgeeks.org/approximate-solution-for-travelling-salesman-problem-using-mst/)

**Greedy for Special cases of DP:**

1. [Fractional Knapsack Problem](https://www.geeksforgeeks.org/fractional-knapsack-problem/)
2. [Minimum number of coins required](https://www.geeksforgeeks.org/greedy-algorithm-to-find-minimum-number-of-coins/)

**Some practice problems on Greedy:**

* **Easy:**
  1. [Split n into maximum composite numbers](https://www.geeksforgeeks.org/split-n-maximum-composite-numbers/)
  2. [Buy Maximum Stocks if i stocks can be bought on i-th day](https://www.geeksforgeeks.org/buy-maximum-stocks-stocks-can-bought-th-day/)
  3. [Find the minimum and maximum amount to buy all N candies](https://www.geeksforgeeks.org/find-minimum-maximum-amount-buy-n-candies/)
  4. [Maximum sum possible equal to sum of three stacks](https://www.geeksforgeeks.org/find-maximum-sum-possible-equal-sum-three-stacks/)
  5. [Divide cuboid into cubes such that sum of volumes is maximum](https://www.geeksforgeeks.org/divide-cuboid-cubes-sum-volumes-maximum/)
  6. [Maximum number of customers that can be satisfied with given quantity](https://www.geeksforgeeks.org/maximum-number-customers-can-satisfied-given-quantity/)
  7. [Minimum rotations to unlock a circular lock](https://www.geeksforgeeks.org/minimum-rotations-unlock-circular-lock/)
  8. [Minimum rooms for m events of n batches with given schedule](https://www.geeksforgeeks.org/minimum-rooms-for-m-events-of-n-batches-with-given-schedule/)
  9. [Minimum cost to make array size 1 by removing larger of pairs](https://www.geeksforgeeks.org/minimum-cost-make-array-size-1-removing-larger-pairs/)
  10. [Minimum cost for acquiring all coins with k extra coins allowed with every coin](https://www.geeksforgeeks.org/minimum-cost-for-acquiring-all-coins-with-k-extra-coins-allowed-with-every-coin/)
  11. [Minimum increment by k operations to make all elements equal](https://www.geeksforgeeks.org/minimum-increment-k-operations-make-elements-equal/)
  12. [Find minimum number of currency notes and values that sum to given amount](https://www.geeksforgeeks.org/find-number-currency-notes-sum-upto-given-amount/)
  13. [Smallest subset with sum greater than all other elements](https://www.geeksforgeeks.org/smallest-subset-sum-greater-elements/)
* **Medium:**
  1. [Maximum trains for which stoppage can be provided](https://www.geeksforgeeks.org/maximum-trains-stoppage-can-provided/)
  2. [Minimum Fibonacci terms with sum equal to K](https://www.geeksforgeeks.org/minimum-fibonacci-terms-sum-equal-k/)
  3. [Divide 1 to n into two groups with minimum sum difference](https://www.geeksforgeeks.org/divide-1-n-two-groups-minimum-sum-difference/)
  4. [Paper cut into minimum number of squares](https://www.geeksforgeeks.org/paper-cut-minimum-number-squares/)
  5. [Minimum difference between groups of size two](https://www.geeksforgeeks.org/minimum-difference-between-groups-of-size-two/)
  6. [Connect n ropes with minimum cost](https://www.geeksforgeeks.org/connect-n-ropes-minimum-cost/)
  7. [Minimum number of Platforms required for a railway/bus station](https://www.geeksforgeeks.org/minimum-number-platforms-required-railwaybus-station/)
  8. [Minimum initial vertices to traverse whole matrix with given conditions](https://www.geeksforgeeks.org/minimum-initial-vertices-traverse-whole-matrix-given-conditions/)
  9. [Largest palindromic number by permuting digits](https://www.geeksforgeeks.org/largest-palindromic-number-permuting-digits/)
  10. [Find Smallest number with given number of digits and digits sum](https://www.geeksforgeeks.org/find-smallest-number-with-given-number-of-digits-and-digit-sum/)
  11. [Lexicographically largest subsequence such that every character occurs at least k times](https://www.geeksforgeeks.org/lexicographically-largest-subsequence-every-character-occurs-least-k-times/)
* **Hard:**
  1. [Maximum elements that can be made equal with k updates](https://www.geeksforgeeks.org/maximum-elements-can-made-equal-k-updates/)
  2. [Minimize cash flow among friends](https://www.geeksforgeeks.org/minimize-cash-flow-among-given-set-friends-borrowed-money/)
  3. [Minimum Cost to cut a board into squares](https://www.geeksforgeeks.org/minimum-cost-cut-board-squares/)
  4. [Minimum cost to process m tasks where switching costs](https://www.geeksforgeeks.org/minimum-cost-to-process-m-tasks-where-switching-costs/)
  5. [Minimum time to finish all jobs with given constraints](https://www.geeksforgeeks.org/find-minimum-time-to-finish-all-jobs-with-given-constraints/)
  6. [Minimize the maximum difference between the heights of towers](https://www.geeksforgeeks.org/minimize-the-maximum-difference-between-the-heights/)
  7. [Minimum edges to reverse to make path from a source to a destination](https://www.geeksforgeeks.org/minimum-edges-reverse-make-path-source-destination/)
  8. [Find the Largest Cube formed by Deleting minimum Digits from a number](https://www.geeksforgeeks.org/find-largest-cube-formed-deleting-minimum-digits-number/)
  9. [Rearrange characters in a string such that no two adjacent are same](https://www.geeksforgeeks.org/rearrange-characters-string-no-two-adjacent/)
  10. [Rearrange a string so that all same characters become d distance away](https://www.geeksforgeeks.org/rearrange-a-string-so-that-all-same-characters-become-at-least-d-distance-away/)

-----------------------------------------------------------------------------------------------------------------------

**Easy Questions:**

**1.Split n into maximum composite numbers**

Given n, print the maximum number of [composite numbers](https://www.geeksforgeeks.org/composite-number/) that sum up to n. First few composite numbers are 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, ………

**Examples:**

Input: 90   
Output: 22  
Explanation: If we add 21 4's, then we   
get 84 and then add 6 to it, we get 90.

Input: 10  
Output: 2  
Explanation: 4 + 6 = 10

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

Below are some important observations.

1. If the number is less than 4, it won’t have any combinations.
2. If the number is 5, 7, 11, it wont have any splitting.
3. Since smallest composite number is 4, it makes sense to use maximum number of 4s.
4. For numbers that don’t leave a composite remainder when divided by 4, we do following. If remainder is 1, we subtract 9 from it to get the number which is perfectly divisible by 4. If the remainder is 2, then subtract 6 from it to make n a number which is perfectly divisible by 4. If the remainder is 3, then subtract 15 from it to make n perfectly divisible by 4, and 15 can be made up by 9 + 6.

So the main observation is to make n such that is composes of maximum no of 4’s and the remaining can be made up by 6 and 9. We won’t need composite numbers more than that, as the composite numbers above 9 can be made up of 4, 6, and 9.

Below is the implementation of the above approach

# Python3 program to split a number into

# maximum number of composite numbers.

# Function to calculate the maximum number

# of composite numbers adding upto n

**def** count(n):

    # 4 is the smallest composite number

**if** (n < 4):

**return -**1

    # stores the remainder when n

    # is divided n is divided by 4

    rem **=** n **%** 4

    # if remainder is 0, then it is

    # perfectly divisible by 4.

**if** (rem **==** 0):

**return** n **//** 4

    # if the remainder is 1

**if** (rem **==** 1):

        # If the number is less than 9, that

        # is 5, then it cannot be expressed as

        # 4 is the only composite number less

        # than 5

**if** (n < 9):

**return -**1

        # If the number is greater than 8, and

        # has a remainder of 1, then express n

        # as n-9 a and it is perfectly divisible

        # by 4 and for 9, count 1.

**return** (n **-** 9) **//** 4 **+** 1

    # When remainder is 2, just subtract 6 from n,

    # so that n is perfectly divisible by 4 and

    # count 1 for 6 which is subtracted.

**if** (rem **==** 2):

**return** (n **-** 6) **//** 4 **+** 1

    # if the number is 7, 11 which cannot be

    # expressed as sum of any composite numbers

**if** (rem **==** 3):

**if** (n < 15):

**return -**1

        # when the remainder is 3, then subtract

        # 15 from it and n becomes perfectly

        # divisible by 4 and we add 2 for 9 and 6,

        # which is getting subtracted to make n

        # perfectly divisible by 4.

**return** (n **-** 15) **//** 4 **+** 2

# Driver Code

n **=** 90

print(count(n))

n **=** 143

print(count(n))

# This code is contributed by Anant Agarwal.

Output:

22   
34

Time complexity: O(1)

Auxiliary Space: O(1)

**2.Buy Maximum Stocks if i stocks can be bought on i-th day**

In a stock market, there is a product with its infinite stocks. The stock prices are given for **N** days, where arr[i] denotes the price of the stock on the ith day. There is a rule that a customer can buy at most i stock on the ith day. If the customer has **k** amount of money initially, find out the maximum number of stocks a customer can buy.

For example, for 3 days the price of a stock is given as 7, 10, 4. You can buy 1 stock worth 7 rs on day 1, 2 stocks worth 10 rs each on day 2 and 3 stock worth 4 rs each on day 3.

Examples:

Input : price[] = { 10, 7, 19 },   
 k = 45.  
Output : 4  
A customer purchases 1 stock on day 1 for 10 rs,   
2 stocks on day 2 for 7 rs each and 1 stock on day 3 for 19 rs.Therefore total of  
10, 7 \* 2 = 14 and 19 respectively. Hence,   
total amount is 10 + 14 + 19 = 43 and number   
of stocks purchased is 4.

Input : price[] = { 7, 10, 4 },   
 k = 100.  
Output : 6

Buy Maximum Stocks if i stocks can be bought on i-th day

The idea is to use greedy approach, where we should start buying product from the day when the stock price is least and so on.

So, we will sort the pair of two values i.e { stock price, day } according to the stock price, and if stock prices are same, then we sort according to the day.

Now, we will traverse along the sorted list of pairs, and start buying as follows:

Let say, we have R rs remaining till now, and the cost of product on this day be C, and we can buy atmost L products on this day then,

total purchase on this day (P) = min(L, R/C)

Now, add this value to the answer

total\_purchase = total\_purchase + P, where P =min(L, R/C)

Now, subtract the cost of buying P items from remaining money, R = R – P\*C.

Total number of products that we can buy is equal to total\_purchase.

Below is the implementation of this approach:

# Python3 program to find maximum number of stocks

# that can be bought with given constraints.

# Returns the maximum stocks

**def** buyMaximumProducts(n, k, price):

    # Making pair of stock cost and day number

    arr **=** []

**for** i **in** range(n):

        arr.append([i **+** 1, price[i]])

    # Sort based on the price of stock

    arr.sort(key **= lambda** x: x[1])

    # Calculating the max stocks purchased

    total\_purchase **=** 0

**for** i **in** range(n):

        P **=** min(arr[i][0], k**//**arr[i][1])

        total\_purchase **+=** P

        k **-=** (P **\*** arr[i][1])

**return** total\_purchase

# Driver code

price **=** [ 10, 7, 19 ]

n **=** len(price)

k **=** 45

print(buyMaximumProducts(n, k, price))

# This code is contributed by Tharun Reddy

**Output**

4

**Time Complexity:**O(nlogn).

**Auxiliary Space**: O(n)

**3.Find the minimum and maximum amount to buy all N candies**

In a candy store, there are N different types of candies available and the prices of all the N different types of candies are provided. There is also an attractive offer by the candy store. We can buy a single candy from the store and get at most K other candies (all are different types) for free.

1. Find the minimum amount of money we have to spend to buy all the N different candies.
2. Find the maximum amount of money we have to spend to buy all the N different candies.

In both cases, we must utilize the offer and get the maximum possible candies back. If k or more candies are available, we must take k candies for every candy purchase. If less than k candies are available, we must take all candies for a candy purchase.

**Examples:**

**Input :**   
price[] = {3, 2, 1, 4}  
k = 2  
**Output :**   
Min = 3, Max = 7  
**Explanation :**  
Since k is 2, if we buy one candy we can take   
atmost two more for free.  
So in the first case we buy the candy which   
costs 1 and take candies worth 3 and 4 for   
free, also you buy candy worth 2 as well.  
So min cost = 1 + 2 = 3.  
In the second case we buy the candy which   
costs 4 and take candies worth 1 and 2 for   
free, also We buy candy worth 3 as well.  
So max cost = 3 + 4 = 7.

One important thing to note is, we must use the offer and get maximum candies back for every candy purchase. So if we want to minimize the money, we must buy candies at minimum cost and get candies of maximum costs for free. To maximize the money, we must do the reverse. Below is an algorithm based on this.

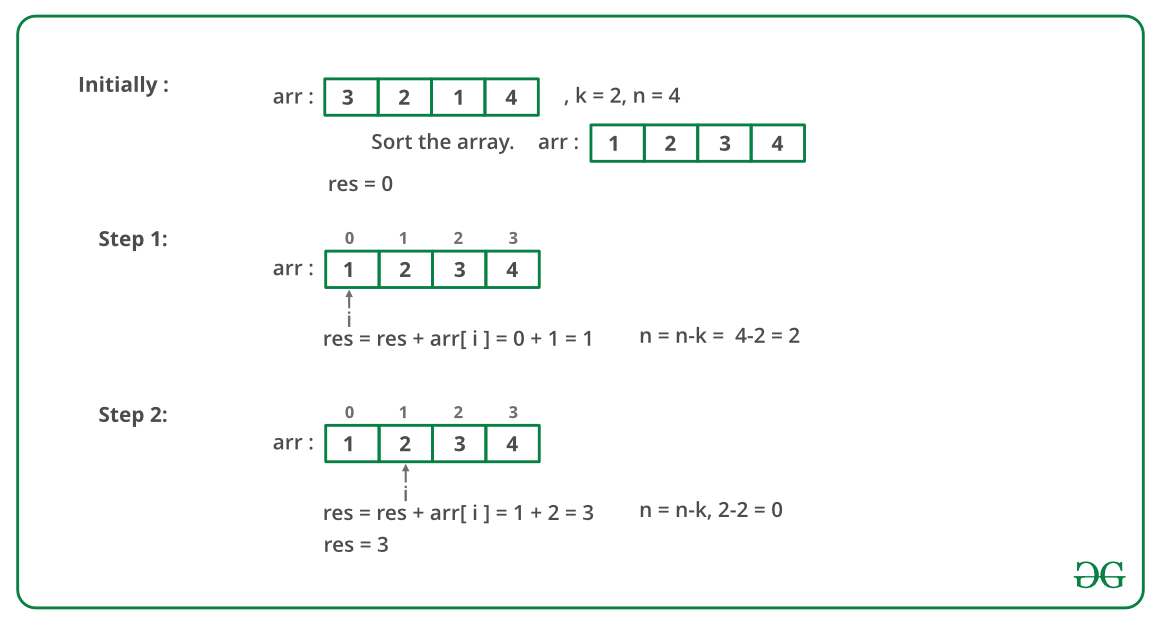
First Sort the price array.

**For finding minimum amount :**  
 Start purchasing candies from starting   
 and reduce k free candies from last with  
 every single purchase.

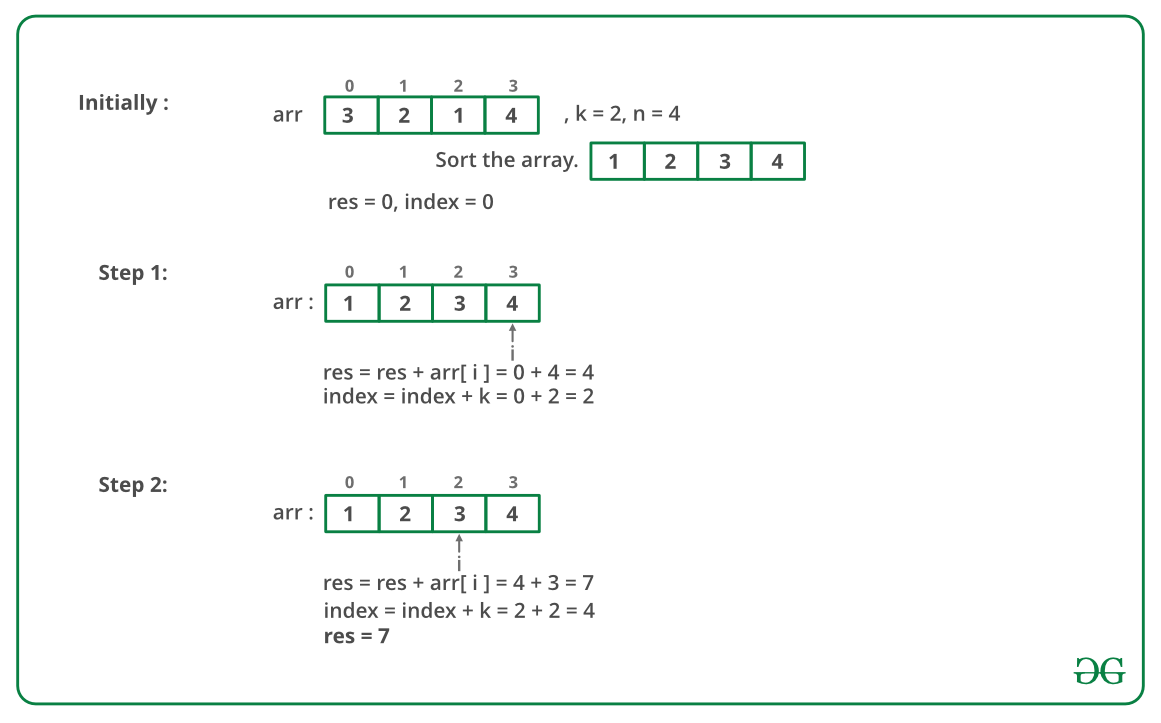
**For finding maximum amount :**   
 Start purchasing candies from the end   
 and reduce k free candies from starting   
 in every single purchase.

Below image is an illustration of the above approach:

Minimum amount :



Maximum amount :



Below is the implementation of the above approach:

# Python implementation

# to find the minimum

# and maximum amount

# Function to find

# the minimum amount

# to buy all candies

**def** findMinimum(arr, n, k):

    res **=** 0

    i **=** 0

**while**(i<n):

        # Buy current candy

        res **+=** arr[i]

        # And take k

        # candies for free

        # from the last

        n **=** n**-**k

        i **+=** 1

**return** res

# Function to find

# the maximum amount

# to buy all candies

**def** findMaximum(arr, n, k):

    res **=** 0

    index **=** 0

    i **=** n**-**1

**while**(i >**=** index):

        # Buy candy with

        # maximum amount

        res **+=** arr[i]

        # And get k candies

        # for free from

        # the starting

        index **+=** k

        i **-=** 1

**return** res

# Driver code

arr **=** [1,2,3,4,5,6,7,8,9,10]

n **=** len(arr)

k **=** 0

arr.sort()

# Function call

print(findMinimum(arr, n, k), " ",

      findMaximum(arr, n, k))

# This code is contributed

# by Anant Agarwal.

**Output**

3 7

**Time Complexity** : O(nlogn)

**Auxiliary Space:**O(1)

**Another Implementation:**

We can use the help of The Least integer function (Ceiling function) using built-in ceil() function to implement:

Below is the implementation in Python:

# Python implementation

# to find the minimum

# and maximum amount

#import ceil function

**from** math **import** ceil

# function to find the maximum

# and the minimum cost required

**def** find(arr,n,k):

    # Sort the array

    arr.sort()

    b **=** int(ceil(n**/**k))

    # print the minimum cost

**print**("minimum ",sum(arr[:b]))

    # print the maximum cost

    print("maximum ", sum(arr[**-**b:]))

# Driver Code

arr **=** [3, 2, 1, 4]

n **=** len(arr)

k **=** 2

# Function call

find(arr,n,k)

**Output**

('minimum ', 3)  
('maximum ', 7)

**Time Complexity:** O(nlog(n))

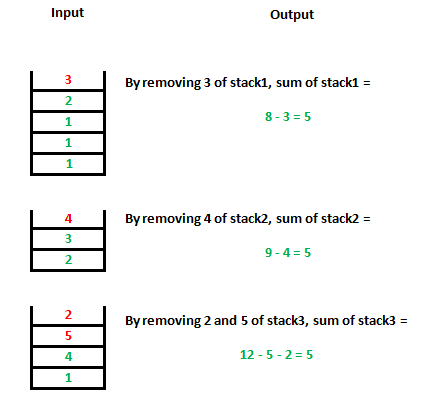
**Auxiliary Space:** O(1)

**4.Find maximum equal sum of every three stacks**

Given three stacks of the positive numbers, the task is to find the possible equal maximum sum of the stacks with the removal of top elements allowed. Stacks are represented as an array, and the first index of the array represent the top element of the stack.

**Examples:**

**Input :** stack1[] = { 3, 10}  
 stack2[] = { 4, 5 }  
 stack3[] = { 2, 1 }  
**Output :** 0  
Sum can only be equal after removing all elements   
from all stacks.



The idea is to compare the sum of each stack, if they are not same, remove the top element of the stack having the maximum sum.

Algorithm for solving this problem:

1. Find the sum of all elements of in individual stacks.
2. If the sum of all three stacks is the same, then this is the maximum sum.
3. Else remove the top element of the stack having the maximum sum among three of stacks. Repeat step 1 and step 2.

The approach works because elements are positive. To make sum equal, we must remove some element from stack having more sum, and we can only remove from the top.

Below is the implementation of this approach:

# Python program to calculate maximum sum with equal

# stack sum.

# Returns maximum possible equal sum of three stacks

# with removal of top elements allowed

**def** maxSum(stack1, stack2, stack3, n1, n2, n3):

    sum1, sum2, sum3 **=** 0, 0, 0

  # Finding the initial sum of stack1.

**for** i **in** range(n1):

        sum1 **+=** stack1[i]

  # Finding the initial sum of stack2.

**for** i **in** range(n2):

        sum2 **+=** stack2[i]

  # Finding the initial sum of stack3.

**for** i **in** range(n3):

        sum3 **+=** stack3[i]

  # As given in question, first element is top

  # of stack..

    top1, top2, top3 **=** 0, 0, 0

    ans **=** 0

**while** (1):

      # If any stack is empty

**if** (top1 **==** n1 **or** top2 **==** n2 **or** top3 **==** n3):

**return** 0

      # If sum of all three stack are equal.

**if** (sum1 **==** sum2 **and** sum2 **==** sum3):

**return** sum1

      # Finding the stack with maximum sum and

      # removing its top element.

**if** (sum1 >**=** sum2 **and** sum1 >**=** sum3):

            sum1 **-=** stack1[top1]

            top1**=**top1**+**1

**else if** (sum2 >**=** sum1 **and** sum2 >**=** sum3):

            sum2 **-=** stack2[top2]

            top2**=**top2**+**1

**else if** (sum3 >**=** sum2 **and** sum3 >**=** sum1):

            sum3 **-=** stack3[top3]

            top3**=**top3**+**1

# Driven Program

stack1 **=** [ 3, 2, 1, 1, 1 ]

stack2 **=** [ 4, 3, 2 ]

stack3 **=** [ 1, 1, 4, 1 ]

n1 **=** len(stack1)

n2 **=** len(stack2)

n3 **=** len(stack3)

print (maxSum(stack1, stack2, stack3, n1, n2, n3))

#This code is contributed by Afzal Ansari

**Output**

5

***Time Complexity : O(n1 + n2 + n3)****where n1, n2 and n3 are sizes of three stacks.*

***Auxiliary space****: O(1) because using constant space for variables*

**5.Divide cuboid into cubes such that sum of volumes is maximum**

Given the **length**, **breadth**, **height** of a cuboid. The task is to divide the given cuboid in minimum number of cubes such that size of all cubes is same and sum of volumes of cubes is maximum.

Examples:

Input : l = 2, b = 4, h = 6  
Output : 2 6  
A cuboid of length 2, breadth 4 and   
height 6 can be divided into 6 cube   
of side equal to 2.  
Volume of cubes = 6\*(2\*2\*2) = 6\*8 = 48.  
Volume of cuboid = 2\*4\*6 = 48.

Input : 1 2 3  
Output : 1 6

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

First of all, we are not allowed to waste volume of cuboid as we meed maximum volume sum. So, each side should be completely divide among all cubes. And since each of three side of cubes are equal, so each side of the cuboid need to be divisible by same number, say x, which will going to be the side of the cube. So, we have to maximize this x, which will divide given length, breadth and height. This x will be maximum only if it is greatest common divisor of given length, breadth and height. So, the length of the cube will be GCD of length, breadth and height.

Now, to compute number of cubes, we know total volume of cuboid and can find volume of one cube (since side is already calculated). So, total number of cubes is equal to (volume of cuboid)/(volume of cube) i.e (l \* b \* h)/(x \* x \* x).

Below is implementation of this approach:

 # Python3 code to find optimal way to break

# cuboid into cubes.

**from** fractions **import** gcd

# Print the maximum side and no of cube.

**def** maximizecube( l , b , h ):

    # GCD to find side.

    side **=** gcd(l, gcd(b, h))

    # dividing to find number of cubes.

    num **=** int(l **/** side)

    num **=** int(num **\*** b **/** side)

    num **=** int(num **\*** h **/** side)

    print(side, num)

# Driver code

l **=** 2

b **=** 4

h **=** 6

maximizecube(l, b, h)

# This code is contributed by "Sharad\_Bhardwaj".

Output:

2 6

**Time Complexity:** O(log2n), where n is the upper limit of b and h.

**Auxiliary Space:**O(1)

**6.Maximum number of customers that can be satisfied with given quantity**

A new variety of rice has been brought in supermarket and being available for the first time, the quantity of this rice is limited. Each customer demands the rice in two different packaging of size a and size b. The sizes a and b are decided by staff as per the demand. Given the size of the packets a and b, the total quantity of rice available d and the number of customers n, find out maximum number of customers that can be satisfied with the given quantity of rice. Display the total number of customers that can be satisfied and the index of customers that can be satisfied.

**Note:** If a customer orders 2 3, he requires 2 packets of size a and 3 packets of size b. Assume indexing of customers starts from 1.

**Input:** The first line of input contains two integers n and d; next line contains two integers a and b. Next n lines contain two integers for each customer denoting total number of bags of size a and size b that customer requires.

**Output:** Print the maximum number of customers that can be satisfied and in the next line print the space-separated indexes of satisfied customers.

**Examples:**

Input : n = 5, d = 5  
 a = 1, b = 1  
 2 0  
 3 2  
 4 4  
 10 0  
 0 1  
Output : 2  
 5 1

Input : n = 6, d = 1000000000  
 a = 9999, b = 10000  
 10000 9998  
 10000 10000  
 10000 10000  
 70000 70000  
 10000 10000  
 10000 10000  
Output : 5  
 1 2 3 5 6

**Explanation:** In first example, the order of customers according to their demand is:

Customer ID Demand  
 5 1  
 1 2  
 2 5  
 3 8  
 4 10

From this, it can easily be concluded that only customer 5 and customer 1 can be satisfied for total demand of 1 + 2 = 3. Rest of the customer cannot purchase the remaining rice, as their demand is greater than available amount.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach:** In order to meet the demand of maximum number of customers we must start with the customer with minimum demand so that we have maximum amount of rice left to satisfy remaining customers. Therefore, sort the customers according to the increasing order of demand so that maximum number of customers can be satisfied. Below is the implementation of above approach:

**Implementation:**

# Python3 program to find maximum number

# of customers that can be satisfied

v **=** []

# print maximum number of satisfied

# customers and their indexes

**def** solve(n, d, a, b, arr):

    first, second **=** 0, 1

    # Creating an vector of pair of

    # total demand and customer number

**for** i **in** range(n):

        m **=** arr[i][0]

        t **=** arr[i][1]

        v.append([a **\*** m **+** b **\*** t, i **+** 1])

    # Sorting the customers according

    # to their total demand

    v.sort()

    ans **=** []

    # Taking the first k customers that

    # can be satisfied by total amount d

**for** i **in** range(n):

**if** v[i][first] <**=** d:

            ans.append(v[i][second])

            d **-=** v[i][first]

**print**(len(ans))

**for** i **in** range(len(ans)):

        print(ans[i], end **=** " ")

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # Initializing variables

    n **=** 5

    d **=** 5

    a **=** 1

    b **=** 1

    arr **=** [[2, 0], [3, 2],

           [4, 4], [10, 0],

           [0, 1]]

    solve(n, d, a, b, arr)

# This code is contributed by PranchalK

**Output**

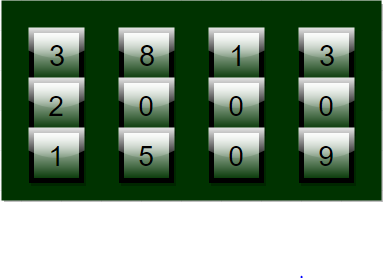
2  
5 1

**Time Complexity: O(n\*log(n))**

**Auxiliary Space: O(n)**

**7.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.



**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

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

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.

# Python3 program for min rotation to unlock

# function for min rotation

**def** minRotation(input, unlock\_code):

    rotation **=** 0;

    # iterate till input and unlock

    # code become 0

**while** (input > 0 **or** unlock\_code > 0):

        # input and unlock last digit

        # as reminder

        input\_digit **=** input **%** 10;

        code\_digit **=** unlock\_code **%** 10;

        # find min rotation

        rotation **+=** min(abs(input\_digit **-** code\_digit),

                    10 **-** abs(input\_digit **-** code\_digit));

        # update code and input

        input **=** int(input **/** 10);

        unlock\_code **=** int(unlock\_code **/** 10);

**return** rotation;

# Driver Code

input **=** 28756;

unlock\_code **=** 98234;

print("Minimum Rotation =",

       minRotation(input, unlock\_code));

# This code is contributed by mits

**Output:**

Minimum Rotation = 12

**Time Complexity:**O(log(input))

**Auxiliary Space:**O(1)

**8.Minimum rooms for m events of n batches with given schedule**

There are n student groups at the school. On each day in school, there are m time slots. A student group may or may not be free during a time slot. We are given n binary string where each binary string is of length m. A character at j-th position in i-th string is 0 if i-th group is free in j-th slot and 1 if i-th group is busy.

Our task is to determine the minimum number of rooms needed to hold classes for all groups on a single study day. Note that one room can hold at most one group class in a single time slot.

**Examples:**

***Input :****n = 2, m = 7, slots[] = {“0101010”, “1010101”}*

***Output :****1*

***Explanation :****Both group can hold their classes in a single room as they have alternative classes.*

***Input :****n = 3, m = 7, slots[] = {“0101011”, “0011001”, “0110111”}*

***Output :****3*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach used:**

Here we traverse through each character of strings we have and while traversing maintaining a count of the number of 1’s at each position of the strings and hence we know the number of coinciding classes at each particular time slot. Then we just need to find the maximum number of coinciding classes amongst all time slots.

**Implementation:**

# Python3 program to find minimum

# number of rooms required

# Returns minimum number of

# rooms required to perform

# classes of n groups in m

# slots with given schedule.

**def** findMinRooms(slots, n, m):

    # Store count of classes

    # happening in every slot.

    counts **=** [0] **\*** m;

**for** i **in** range(n):

**for** j **in** range(m):

**if** (slots[i][j] **==** '1'):

                counts[j] **+=** 1;

    # Number of rooms required is

    # equal to maximum classes

    # happening in a particular slot.

**return** max(counts);

# Driver Code

n **=** 3;

m **=** 7;

slots **=** ["0101011", "0011001", "0110111"];

print(findMinRooms(slots, n, m));

# This code is contributed by mits

**Output**

3

**Complexity Analysis:**

* **Time Complexity:** **O(m \* n)**
* **Auxiliary Space: O(m)**

**9.Minimum cost to make array size 1 by removing larger of pairs**

Given an array of n integers. We need to reduce size of array to one. We are allowed to select a pair of integers and remove the larger one of these two. This decreases the array size by 1. Cost of this operation is equal to value of smaller one. Find out minimum sum of costs of operations needed to convert the array into a single element.

**Examples:**

Input: 4 3 2   
Output: 4  
Explanation:   
Choose (4, 2) so 4 is removed, new array   
= {2, 3}. Now choose (2, 3) so 3 is removed.   
So total cost = 2 + 2 = 4

Input: 3 4  
Output: 3  
Explanation: choose 3, 4, so cost is 3.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

The idea is to always pick minimum value as part of the pair and remove larger value. This minimizes cost of reducing array to size 1.

Below is the implementation of the above approach:

# Python program to find minimum

# cost to reduce array size to 1

# function to calculate the

# minimum cost

**def** cost(a, n):

    # Minimum cost is n-1 multiplied

    # with minimum element.

**return** ( (n **-** 1) **\*** min(a) )

# driver code

a **=** [ 4, 3, 2 ]

n **=** len(a)

print(cost(a, n))

# This code is contributed by

# Smitha Dinesh Semwal

**Output**

4

**Time Complexity: O(N),**as we are using a min function which will cost O(N).

**Auxiliary Space: O(1),**as we are not using any extra space.

**10.Minimum cost for acquiring all coins with k extra coins allowed with every coin**

You are given a list of N coins of different denominations. You can pay an amount equivalent to any 1 coin and can acquire that coin. In addition, once you have paid for a coin, we can choose at most K more coins and can acquire those for free. The task is to find the minimum amount required to acquire all the N coins for a given value of K.

**Examples :**

Input : coin[] = {100, 20, 50, 10, 2, 5},   
 k = 3  
Output : 7

Input : coin[] = {1, 2, 5, 10, 20, 50},   
 k = 3  
Output : 3

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

As per the question, we can see that at a cost of 1 coin, we can acquire at most K+1 coins. Therefore, in order to acquire all the n coins, we will be choosing ceil(n/(k+1)) coins and the cost of choosing coins will be minimum if we choose the smallest ceil(n/(k+1)) ( Greedy approach). The smallest ceil(n/(k+1)) coins can be found by simply sorting all the N values in increasing order.

If we should check for time complexity (n log n) is for sorting element and (k) is for adding the total amount. So, finally Time Complexity: O(n log n).

# Python3 program to

# acquire all n coins

**import** math

# function to calculate min cost

**def** minCost(coin, n, k):

    # sort the coins value

    coin.sort()

    # calculate no. of

    # coins needed

    coins\_needed **=** math.ceil(1.0 **\*** n **//**

                            (k **+** 1));

    # calculate sum of all

    # selected coins

    ans **=** 0

**for** i **in** range(coins\_needed **-** 1 **+** 1):

        ans **+=** coin[i]

**return** ans

# Driver code

coin **=** [8, 5, 3, 10,

        2, 1, 15, 25]

n **=** len(coin)

k **=** 3

print(minCost(coin, n, k))

# This code is contributed

# by Anant Agarwal.

**Output :**

3

Time Complexity: O(n log n)

Auxiliary Space: O(1)

Note that there are more efficient approaches to find the given number of smallest values. For example, method 6 of [m largest(or smallest) elements in an array](https://www.geeksforgeeks.org/k-largestor-smallest-elements-in-an-array/) can find m’th smallest element in (n-m) Log m + m Log m).

**How to handle multiple queries for a single predefined array?**

In this case, if you are asked to find the above answer for many values of K, you have to compute it fast and our time complexity got increased as per the number of queries for k. For the purpose to serve, we can maintain a prefix sum array after sorting all the N values and can answer queries easily and quickly.

Suppose

# Python3 program to acquire all n coins at

# minimum cost with multiple values of k.

**import** math as mt

# Converts coin[] to prefix sum array

**def** preprocess(coin, n):

    # sort the coins values

    coin.sort()

    # maintain prefix sum array

**for** i **in** range(1, n):

        coin[i] **+=** coin[i **-** 1]

# Function to calculate min cost when we can

# get k extra coins after paying cost of one.

**def** minCost(coin, n, k):

    # calculate no. of coins needed

    coins\_needed **=** mt.ceil(1.0 **\*** n **/** (k **+** 1))

    # return sum of from prefix array

**return** coin[coins\_needed **-** 1]

# Driver code

coin **=** [8, 5, 3, 10, 2, 1, 15, 25]

n **=** len(coin)

preprocess(coin, n)

k **=** 3

print(minCost(coin, n, k))

k **=** 7

print(minCost(coin, n, k))

# This code is contributed

# by Mohit kumar 29

**Output :**

3  
1

Time Complexity: O(n log n)

Auxiliary Space: O(1)

**11.Minimum increment by k operations to make all elements equal**

You are given an array of n-elements, you have to find the number of operations needed to make all elements of array equal. Where a single operation can increment an element by k. If it is not possible to make all elements equal print -1.

**Example :**

**Input :** arr[] = {4, 7, 19, 16}, k = 3  
**Output :** 10

**Input :** arr[] = {4, 4, 4, 4}, k = 3  
**Output :** 0

**Input :** arr[] = {4, 2, 6, 8}, k = 3  
**Output :** -1

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

To solve this question we require to check whether all elements can became equal or not and that too only by incrementing k from elements value. For this we have to check that the difference of any two elements should always be divisible by k. If it is so, then all elements can become equal otherwise they can not became equal in any case by incrementing k from them. Also, the number of operations required can be calculated by finding value of (max – Ai)/k for all elements. where max is maximum element of array.

**Algorithm :**

// iterate for all elements  
for (int i=0; i<n; i++)  
{  
 // check if element can make equal to max  
 // or not if not then return -1  
 if ((max - arr[i]) % k != 0 )  
 return -1;

// else update res for required operations  
 else  
 res += (max - arr[i]) / k ;  
}

return res;

**Implementation:**

# Python3 Program to make all array equal

# function for calculating min operations

**def** minOps(arr, n, k):

    # max elements of array

    max1 **=** max(arr)

    res **=** 0

    # iterate for all elements

**for** i **in** range(0, n):

        # check if element can make equal to

        # max or not if not then return -1

**if** ((max1 **-** arr[i]) **%** k !**=** 0):

**return -**1

        # else update res for

        # required operations

**else**:

            res **+=** (max1 **-** arr[i]) **/** k

    # return result

**return** int(res)

# driver program

arr **=** [21, 33, 9, 45, 63]

n **=** len(arr)

k **=** 6

print(minOps(arr, n, k))

# This code is contributed by

# Smitha Dinesh Semwal

**Output**

24

**Time Complexity: O(n)**

**Auxiliary Space: O(1)**

**12Find minimum number of currency notes and values that sum to given amount**

Given an amount, find the minimum number of notes of different denominations that sum up to the given amount. Starting from the highest denomination note, try to accommodate as many notes as possible for a given amount.

We may assume that we have infinite supply of notes of values {2000, 500, 200, 100, 50, 20, 10, 5, 1}

**Examples:**

Input : 800  
Output : Currency Count   
 500 : 1  
 200 : 1  
 100 : 1

Input : 2456  
Output : Currency Count  
 2000 : 1  
 200 : 2  
 50 : 1  
 5 : 1  
 1 : 1

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem is a simple variation of [coin change problem](https://www.geeksforgeeks.org/find-minimum-number-of-coins-that-make-a-change/). Here Greedy approach works as the given system is canonical (Please refer [this](https://arxiv.org/abs/0809.0400) and [this](https://arxiv.org/pdf/0809.0400.pdf) for details)

Below is the program implementation to find the number of notes:

 # Python3 program to accept an amount

# and count number of notes

# Function to count and print

# currency notes

**def** countCurrency(amount):

    notes **=** [2000, 500, 200, 100, 50, 20, 10, 5, 1]

    notesCount **=** {}

**for** note **in** notes:

**if** amount >**=** note:

            notesCount[note] **=** amount**//**note

            amount **=** amount **%** note

**print** ("Currency Count ->")

**for** key, val **in** notesCount.items():

**print**(f"{key} : {val}")

# Driver code

amount **=** 868

countCurrency(amount)

# Code contributed by farzams101

**Output:**

Currency Count ->  
500 : 1  
200 : 1  
100 : 1  
50 : 1  
10 : 1  
5 : 1  
1 : 3

**13.Smallest subset with sum greater than all other elements**

Given an array of non-negative integers, the task is to find the minimum number of elements such that their sum should be greater than the sum of the rest of the elements of the array.

**Example:**

***Input****: arr[] = [ 3 , 1 , 7, 1 ]*

***Output****: 1*

***Explanation:****Smallest subset is {7}. Sum of this subset is greater than the sum of all other elements left after removing subset {7} from the array*

***Input****:  arr[] = [ 2 , 1 , 2 ]*

***Output****: 2*

***Explanation:****Smallest subset is {2 , 1}. Sum of this subset is greater than the sum of all other elements left after removing subset {2 , 1} from the array*

Recommended Problem

Smallest Subset with Greater Sum

**14.Smallest subset with sum greater than all other elements using Sorting**

*The**approach is to take the largest elements from the array , in that way we can decrease the size of the subset that has sum greater than the sum of rest of the elements of the array , so we sort the array in descending order, then take the largest elements, until we get strictly more than half of total sum of the given array.*

Follow the steps mentioned below to implement the idea:

* Create a variable **halfSum** to store half of the overall sum of the array **arr[]**.
* Sort the array in **descending** order.
* Create a variable **curr\_sum** and increase the value of**curr\_sum** by **arr[i]**while traversing the array from index 0 till the value of **curr\_sum** is less than **halfSum**
* When **curr\_sum** is greater than **halfSum** return**index+1**.

Below is the implementation of the above approach:

# Python3 code to find minimum number of

# elements such that their sum is greater

# than sum of remaining elements of the array.

# function to find minimum elements needed.

**def** minElements(arr, n):

    # calculating HALF of array sum

    halfSum **=** 0

**for** i **in** range(n):

        halfSum **=** halfSum **+** arr[i]

    halfSum **=** int(halfSum **/** 2)

    # sort the array in descending order.

    arr.sort(reverse**=**True)

    res **=** 0

    curr\_sum **=** 0

**for** i **in** range(n):

        curr\_sum **+=** arr[i]

        res **+=** 1

        # current sum greater than sum

**if** curr\_sum > halfSum:

**return** res

**return** res

# driver code

arr **=** [3, 1, 7, 1]

n **=** len(arr)

print(minElements(arr, n))

# This code is contributed by "Sharad\_Bhardwaj".

**Output**

1

**Time Complexity:** O(N\*log(N)).

**Auxiliary Space:**O(1)