KRISH

1

```
def knapSack(W, wt, val, n):

    # Base Case
    if n == 0 or W == 0:
        return 0

    # If weight of the nth item is
    # more than Knapsack of capacity W,
    # then this item cannot be included
    # in the optimal solution
    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)

    # return the maximum of two cases:
    # (1) nth item included
    # (2) not included
    else:
        return max(
            val[n-1] + knapSack(
                W-wt[n-1], wt, val, n-1),
            knapSack(W, wt, val, n-1))
n=int(input())
w=int(input())
wt=list(map(int, input().split()))
val=list(map(int, input().split()))
print (knapSack(w, wt, val, n))
```

OUTPUT:

2

```
def ActivitySelection(start, finish, n):
    k=[]
    j = 0

    k.append(j)

    for i in range(1,n):

        if start[i] >= finish[j]:

            k.append(i)

            j = i

    return(len(k))

n=int(input())

start=list(map(int, input().split()))

finish=list(map(int, input().split()))

print(ActivitySelection(start, finish, n))
```

3

```
#include<bits/stdc++.h>

using namespace std;


// Function to find minimum computation

int minComputation(int size, int files[])

{


    // Create a min heap

    priority_queue<int, vector<int>,

        greater<int>> pq;


    for(int i = 0; i < size; i++)

    {


        // Add sizes to priorityQueue

        pq.push(files[i]);

    }


    // Variable to count total Computation

    int count = 0;
    while(pq.size() > 1)
    {

        // pop two smallest size element
        // from the min heap
        int first_smallest = pq.top();
        pq.pop();
        int second_smallest = pq.top();
        pq.pop();

        int temp = first_smallest + second_smallest;

        // Add the current computations
        // with the previous one's
        count += temp;

        // Add new combined file size
        // to priority queue or min heap
        pq.push(temp);
    }
    return count;
}

// Driver code
int main()
{

    // No of files
    int n = 6;

    // 6 files with their sizes
    int files[] = { 5, 3, 2, 7, 9, 13 };
```

KRISH

```
   // Total no of computations

   // do be done final answer

   cout << minComputation(n, files);


   return 0;

}
```

4.

```
class ItemValue:


   """Item Value DataClass"""


   def __init__(self, wt, val, ind):

      self.wt = wt

      self.val = val

      self.ind = ind

      self.cost = val // wt


   def __lt__(self, other):

      return self.cost < other.cost

class FractionalKnapSack:


   """Time Complexity O(n log n)"""

   @staticmethod

   def getMaxValue(wt, val, capacity):

      """function to get maximum value """

      iVal = []

      for i in range(len(wt)):

         iVal.append(ItemValue(wt[i], val[i], i))


      # sorting items by value

      iVal.sort(reverse=True)


      totalValue = 0
```

KRISH

```
        for i in iVal:
            curWt = int(i.wt)
            curVal = int(i.val)
            if capacity - curWt >= 0:
                capacity -= curWt
                totalValue += curVal
            else:
                fraction = capacity / curWt
                totalValue += curVal * fraction
                capacity = int(capacity - (curWt * fraction))
                break
        return totalValue
n=int(input())
capacity=int(input())
wt=list(map(int, input().split()))
val=list(map(int, input().split()))
maxValue = FractionalKnapSack.getMaxValue(wt, val, capacity)
print(int(maxValue))
```