ASSIGNMENT - 1

ANIKEIT SETHI(190001003)

Design and Implement an algorithm for fractional knapsack in O(n) time.

Use the weighted medians approach . That will cost you O(n) as finding the weighted median will take O(n). The code for this approach is given below.

**Weighted median approach for fractional knapsack:**

We will work on value per unit of item in the following code. The code will first find the middle value (i.e. mid of values per unit of items if given in sorted order) and place it in its correct position. We will use quick sort partition method for this. Once we get the middle (call it mid) element, following two cases need to be taken into consideration:

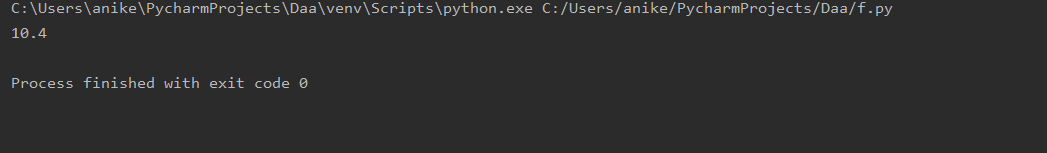
1. When sum of weight of all items present in the right side of mid is more than the value of W, we need to search our answer in right side of mid.
2. else sum all the values present in right side of mid (call it v\_left) and search for W-v\_left in the left side of mid (include mid as well)

(Use only floating point numbers everywhere):

Python code:-

from functools import reduce  
  
  
def partition(weights, values, start, end):  
 x = values[end] / weights[end]  
 i = start  
 for j in range(start, end):  
 if values[j] / weights[j] < x:  
 values[i], values[j] = values[j], values[i]  
 weights[i], weights[j] = weights[j], weights[i]  
 i += 1  
  
 values[i], values[end] = values[end], values[i]  
 weights[i], weights[end] = weights[end], weights[i]  
  
 return i  
  
  
def \_find\_kth(weights, values, start, end, k):  
 ind = partition(weights, values, start, end)  
 if ind - start == k - 1:  
 return ind  
 if ind - start > k - 1:  
 return \_find\_kth(weights, values, start, ind - 1, k)  
 return \_find\_kth(weights, values, ind + 1, end, k - ind - 1)  
  
  
def find\_kth(weights, values, k):  
 return \_find\_kth(weights, values, 0, len(weights) - 1, k)  
  
  
def fractional\_knapsack(weights, values, w):  
 if w == 0 or len(weights) == 0:  
 return 0  
  
 if len(weights) == 1 and weights[0] > w:  
 return w \* (values[0] / weights[0])  
  
 mid = find\_kth(weights, values, len(weights) / 2)  
  
 w1 = reduce(lambda x, y: x + y, weights[mid + 1:])  
 v1 = reduce(lambda x, y: x + y, values[mid + 1:])  
  
 if (w1 > w):  
 return fractional\_knapsack(weights[mid + 1:], values[mid + 1:], w)  
  
 return v1 + fractional\_knapsack(weights[:mid + 1], values[:mid + 1], w - w1)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print(fractional\_knapsack([10., 1.], [1., 10.], 5.))

Output: -



Graph: 