In [75]:

```
#assignment-2
def printjobschedule(array, t):
  m = len(array)
# Sort all jobs accordingly
  for j in range(m):
    for q in range(m - 1 - j):
      if array[q][2] < array[q + 1][2]:
         array[q], array[q + 1] = array[q + 1], array[q]
         res = [False] * t
# To store result
         job = ['-1'] * t
  for q in range(len(array)):
# Find a free slot
   for q in range(min(t - 1, array[q][1] - 1), -1, -1):
    if res[q] is False:
      res[q] = True
      job[q] = array[q][0]
    break
# print
  print(job)
# Driver
array = [['a', 7, 202],
['b', 5, 29],
['c', 6, 84],
['d', 1, 75],
['e', 2, 43]]
print("Maximum profit sequence of jobs is- ")
printjobschedule(array, 3)
```

```
Maximum profit sequence of jobs is-
['a', 'c', 'd']
```

In [76]:

```
#Assignment-3
class Solution:
 def solve(self, weights, values, capacity):
  res = 0
  for pair in sorted(zip(weights, values), key=lambda x: -x[1]/x[0]):
   if not bool(capacity):
    break
   if pair[0] > capacity:
     res += int(pair[1] / (pair[0] / capacity))
     capacity = 0
   elif pair[0] <= capacity:
    res +=pair[1]
    capacity -= pair[0]
   return int(res)
ob = Solution()
weights = [6, 7, 3]
values = [110, 120, 2]
capacity = 10
print(ob.solve(weights, values, capacity))
```

```
110
```

In [79]:

```
#Assignment-4
def knapSack(W, wt, val, n):
#Base Case
   if n == 0 or W == 0:
      return 0

   if (wt[n-1] > W):
      return knapSack(W, wt, val, n-1)

   else:
      return max(
       val[n-1] + knapSack(
       W-wt[n-1], wt, val, n-1),
       knapSack(W, wt, val, n-1))
val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print;knapSack(W, wt, val, n)
```

Out[79]:

```
220
```

In [ ]: