In [5]:

```
# -*- coding: utf-8 -*-
"""
Author: Nathan Rice
0-1 Knapsack Problem using best first branch and bound method
Returns maxprofit with list storing the index position of the items in the best solutio
The profit is maximized while staying under the weight limit.
This program uses a priority queue to store the nodes ordered by best bound,
the node with the highest bound value is returned when removing from the priority queue
The best first approach arrives at an optimal solition faster than breadth first search
"""
#examples
# W = 13
# i  pi  wi pi/wi
# 1 $20  2   10
# 2 $30  5   6
# 3 $35  7   5
# 4 $12  3   4
# 5 $3   1   3
#problem definition
# n = 5 #items given
# W = 13 # capacity of knapsack
# p = [0, 20, 30, 35, 12, 3] # profit of each item (starts with item 0 = $0)
# w = [0, 2, 5, 7, 3, 1] # weight of each item
# p_per_weight = [0, 10, 6, 5, 4, 3] #price per weight

#example 6.1
# items are ordered by price per weight
n = 4
W = 16
p = [40, 30, 50, 10]
w = [2, 5, 10, 5]
p_per_weight = [20, 6, 5, 2]


class Priority_Queue:
    def __init__(self):
        self.pqueue = []
        self.length = 0

    def insert(self, node):
        for i in self.pqueue:
            get_bound(i)
        i = 0
        while i < len(self.pqueue):
            if self.pqueue[i].bound > node.bound:
                break
            i+=1
        self.pqueue.insert(i,node)
        self.length += 1

    def print_pqueue(self):
        for i in list(range(len(self.pqueue))):
            print ("pqueue",i, "=", self.pqueue[i].bound)

    def remove(self):
        try:
            result = self.pqueue.pop()
            self.length -= 1
```

```
        except:
            print("Priority queue is empty, cannot pop from empty list.")
        else:
            return result

class Node:
    def __init__(self, level, profit, weight):
        self.level = level
        self.profit = profit
        self.weight = weight
        self.items = []


def get_bound(node):
    if node.weight >= W:
        return 0
    else:
        result = node.profit
        j = node.level + 1
        totweight = node.weight
        while j <= n-1 and totweight + w[j] <= W:
            totweight = totweight + w[j]
            result = result + p[j]
            j+=1
        k = j
        if k<=n-1:
            result = result + (W - totweight) * p_per_weight[k]
        return result


nodes_generated = 0
pq = Priority_Queue()

v = Node(-1, 0, 0) # v initialized to be the root with level = 0, profit = $0, weight =
nodes_generated+=1
maxprofit = 0 # maxprofit initialized to $0
v.bound = get_bound(v)
#print("v.bound = ", v.bound)


pq.insert(v)

while pq.length != 0:

    v = pq.remove() #remove node with best bound
#    print("\nNode removed from pq.")
#    print("Priority Queue: ")
#    pq.print_pqueue()

#    print("\nmaxprofit = ", maxprofit)
#    print("Parent Node: ")
#    print("v.level = ", v.level, "v.profit = ", v.profit, "v.weight = ", v.weight, "v.
    if v.bound > maxprofit: #check if node is still promising
        #set u to the child that includes the next item
        u = Node(0, 0, 0)
        nodes_generated+=1
        u.level = v.level + 1
        u.profit = v.profit + p[u.level]
        u.weight = v.weight + w[u.level]
```

```
        #take v's list and add u's list
        u.items = v.items.copy()
        u.items.append(u.level) # adds next item
#         print("child that includes the next item: ")
#         print("Child 1:")
#         print("u.level = ", u.level, "u.profit = ", u.profit, "u.weight = ", u.weight)
#         print("u.items = ", u.items)
        if u.weight <= W and u.profit > maxprofit:
            #update maxprofit
            maxprofit = u.profit
#             print("\nmaxprofit updated = ", maxprofit)
            bestitems = u.items
#             print("bestitems = ", bestitems)
        u.bound = get_bound(u)
#         print("u.bound = ", u.bound)
        if u.bound > maxprofit:
            pq.insert(u)
#             print("Node u1 inserted into pq.")
#             print("Priority Queue : ")
#             pq.print_pqueue()
        #set u to the child that does not include the next item
        u2 = Node(u.level, v.profit, v.weight)
        nodes_generated+=1
        u2.bound = get_bound(u2)
        u2.items = v.items.copy()
#         print("child that doesn't include the next item: ")
#         print("Child 2:")
#         print("u2.level = ", u2.level, "u2.profit = ", u2.profit, "u2.weight = ", u2.w
#         print("u2.items = ", u2.items)
        if u2.bound > maxprofit:
            pq.insert(u2)
#             print("Node u2 inserted into pq.")
#             print("Priority Queue : ")
#             pq.print_pqueue()


print("\nEND maxprofit = ", maxprofit, "nodes generated = ", nodes_generated)
print("bestitems = ", bestitems)
```

```
END maxprofit =  90 nodes generated =  11
bestitems =  [0, 2]
```

In [ ]: