```
from queue import PriorityQueue

class Item:
    def __init__(self, weight, value):
        self.weight = weight
        self.value = value

class Node:
    def __init__(self, level, profit, weight):
        self.level = level      # Level of the node in the decision
tree (or index in arr[])
        self.profit = profit    # Profit of nodes on the path from
root to this node (including this node)
        self.weight = weight    # Total weight at the node

    def __lt__(self, other):
        return other.weight < self.weight  # Compare based on weight
in descending order

def bound(u, n, W, arr):
    # Calculate the upper bound of profit for a node in the search
tree
    if u.weight >= W:
        return 0

    profit_bound = u.profit
    j = u.level + 1
    total_weight = u.weight

    # Greedily add items to the knapsack until the weight limit is
reached
    while j < n and total_weight + arr[j].weight <= W:
        total_weight += arr[j].weight
        profit_bound += arr[j].value
        j += 1

    # If there are still items left, calculate the fractional
contribution of the next item
    if j < n:
        profit_bound += int((W - total_weight) * arr[j].value /
arr[j].weight)

    return profit_bound

def knapsack(W, arr, n):
    # Sort items based on value-to-weight ratio in non-ascending order
    arr.sort(key=lambda x: x.value / x.weight, reverse=True)

    priority_queue = PriorityQueue()
    u = Node(-1, 0, 0)  # Dummy node at the starting
```

```
    priority_queue.put(u)

    max_profit = 0

    while not priority_queue.empty():
        u = priority_queue.get()

        if u.level == -1:
            v = Node(0, 0, 0)  # Starting node
        elif u.level == n - 1:
            continue  # Skip if it is the last level (no more items to
consider)
        else:
            v = Node(u.level + 1, u.profit, u.weight)  # Node without
considering the next item

        v.weight += arr[v.level].weight
        v.profit += arr[v.level].value

        # If the cumulated weight is less than or equal to W and
profit is greater than previous profit, update maxProfit
        if v.weight <= W and v.profit > max_profit:
            max_profit = v.profit

        v_bound = bound(v, n, W, arr)
        # If the bound value is greater than current maxProfit, add
the node to the priority queue for further consideration
        if v_bound > max_profit:
            priority_queue.put(v)

        # Node considering the next item without adding it to the
knapsack
        v = Node(u.level + 1, u.profit, u.weight)
        v_bound = bound(v, n, W, arr)
        # If the bound value is greater than current maxProfit, add
the node to the priority queue for further consideration
        if v_bound > max_profit:
            priority_queue.put(v)

    return max_profit

# Driver program to test the above function
if __name__ == "__main__":
    # Driver program to test the above function
    W = 15
    arr = [
        Item(2, 10),
        Item(4, 10),
        Item(6, 12),
        Item(9, 18)
```

```
    ]
    n = len(arr)

    max_profit = knapsack(W, arr, n)
    print("Maximum possible profit =", max_profit)
```

```
Maximum possible profit = 38
```