Class Set: Merge Function

Prototype: Set Merge (Set sDash)

Input: S_k (currentSet), S’

Output: S_{k+1}

Merge function:

Let t_1 represent a pointer to the first element of S_k
Let t_2 represent a pointer to the first element of S’

If none of the S_k or S’ is traversed completely,

```
If (t1.weight < t2.weight)
   add (t1)
   while (t1.profit ≥ t2.profit)
      t2++
   t1++
```

```
If (t1.weight > t2.weight)
   add (t2)
   while (t1.profit ≤ t2.profit)
      t1++
   t2++
```

```
If (t1.weight = t2.weight)
   if (t1.profit ≤ t2.profit)
      t2++
   else t1++
```

If S_k is traversed completely, but S’ is not. (Similarly for S’ traversed completely, S_k is not)

```
while ( S’ is not empty)
   if ( item.profit ≥ lastItemProfitS )
      add(item)
   item++
```

Claim : Output of Merge function is sorted in order of increasing weights

Proof by induction:

S_0 = (0,0) : Trivially sorted
Note that (0,0) will never be purged and is trivially contained in all S_k.

Induction Hypothesis: S_k is sorted

Let $S_k = \{ (0,0)\ (p_1,w_1)\ (p_1,w_2)\ \ldots\ldots\ldots\ (p_n,w_n)\}$

In worst case, $n = 2^k$

Note : S_k denotes optimal knapsack considering first k elements. Hence, in optimal arrangement, profit must increase with weight.

Hence, By I.H. S_k is sorted in increasing order in both profit and weight.

To show : S_{k+1} is also sorted.

Without loss of generalisation, suppose t_1 is being added to the list. We will show that t_1 has the weight of all the items not traversed yet. Then at each step, the smallest weight is added implying ascending order is preserved.
NOTE : This is the reason that this is a **Greedy Algorithm**.

When t_1 is being added, t_2.weight is larger (as seen in Code 1). Also, for all items in S' after t_2, weight is higher than t_2. Thus, t_1 is less than all unexplored items in S'. In S_k, all items after t_1 have greater weight.

Hence proved by induction that **Merge function produce Sorted sets.**

Corollary : All sets S_0, S_1 … S_N are ordered in accordance to increasing weight and profit.

Claim : At each step, S' constructed in sorted in order of weight and profit.

Proof:
Let program Input be of form { (0,0) (P_1,W_1) (P_2,W_2) . . . . (P_n,W_n)}

By Above Corollary, S_k will be sorted in both weight and profit.
Since S' is constructed by taking sum of S_k with (P_{k+1},W_{k+1})

S' = { (P_{k+1}, W_{k+1}) $(P_{k+1}+P_1, W_{k+1}+W_1)$ $(P_{k+1}+P_2, W_{k+1}+W_2)$ . . . . $(P_{k+1}+P_n, W_{k+1}+W_n)$}

Since S_k is sorted and we are only adding constant doublet to each term, hence (as seen above), S' is also sorted.
Hence Proved.

Purge Constraints: In Set S_k,

1. For a pair of items p1 and p2:
   p1.weight > p2.weight and p1.profit ≤ p2.profit

2. For a pair of items p1 and p2:
   p1.weight = p2.weight and p1.profit ≤ p2.profit

3. For any pair p:
   p.weight > maximum KnapSack Capacity

Result: **<u>Basically, Merge takes a union of two sorted lists while purging appropriate items.</u>**

Claim : At each step, Output Set S_{k+1} produced by merge satisfies purge constraints.

Constraint 3
Note that **constraint 3** is handled by extend function of class Set. Hence S' never contains an infeasible term.

Constraint 1 & 2
Proof by Induction

S_1 = { (0,0) , (P_1,W_1) }
Satisfies constraint 1

Induction Hypothesis : S_k satisfies constraint 1.

Showing that constraint holds in S_{k+1}
Without loss of generality, consider the instant when t_1 is being added.
We need to show that addition of t_1 would not lead to violation of purge conditions.

When t_1 is added, t_1.weight < t2.weight.
All items from list S_k added before t1 will not lead to violation as S_k satisfies constraint.
For an item from S' is added before (say t_{20}),
Since all S_k are sorted, t_{20}.weight < t_1.weight
Constraint will be violated only if t_{20}.profit ≥ t_1.profit.
However if this was the case, during the addition of t_{20}, all items before t_1 and t_1 itself would have been purged because (t_1.profit ≤ t_{20}.profit) holds for items including t_1 and before.
Hence, **contradiction that** t_{20}.profit ≥ t_1.profit.
Therefore, **addition of a new item would not violate constraint.**
Hence, the constraint is satisfied by the complete set S_{k+1}

Thus, Merge function produces the next set S_{k+1} which is the optimal arrangement considering first k elements.