**Class Set: Merge Function**

**Prototype: Set Merge (Set sDash)**

**Input:** Sk (currentSet), S’

**Output:** Sk+1

**Merge function:**

Let t1 represent a pointer to the first element of SkLet t2 represent a pointer to the first element of S’

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

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

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 Sk is traversed completely, but S’ is not. ( Similarly for S’ traversed completely, Sk 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:**

S0 = (0,0) : Trivially sorted  
Note that (0,0) will never be purged and is trivially contained in all Sk.

Induction Hypothesis: Sk is sorted

Let Sk = { (0,0) (p1,w1) (p1,w2) . . . . . . . . (pn,wn)}

In worst case, n = 2k

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

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

To show : Sk+1 is also sorted.

Without loss of generalisation, suppose t1 is being added to the list. We will show that t1 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 t1 is being added, t2.weight is larger ( as seen in Code 1 ). Also, for all items in S’ after t2, weight is higher than t2. Thus, t1 is less than all unexplored items in S’. In Sk, all items after t1 have greater weight.

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

**Corollary : All sets S0, S1 … SN 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) (P1,W1) (P2,W2) . . . . . . . . (Pn,Wn)}

By Above Corollary, Sk will be sorted in both weight and profit.  
Since S’ is constructed by taking sum of Sk with (Pk+1,Wk+1)

S’ = { (Pk+1, Wk+1) (Pk+1+P1, Wk+1+W1) (Pk+1+P2, Wk+1+W2) . . . . . . . . (Pk+1+Pn, Wk+1+Wn)}

Since Sk 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 Sk,

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: Basically, Merge takes a union of two sorted lists while purging appropriate items.**

**Claim : At each step, Output Set Sk+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**

S1 = { (0,0) , (P1,W1) }  
Satisfies constraint 1

Induction Hypothesis : Sk satisfies constraint 1.

Showing that constraint holds in Sk+1  
Without loss of generality, consider the instant when t1 is being added.   
We need to show that addition of t1 would not lead to violation of purge conditions.

When t1 is added, t1.weight < t2.weight.   
All items from list Sk added before t1 will not lead to violation as Sk satisfies constraint.  
For an item from S’ is added before (say t20),   
Since all Sk are sorted, t20.weight < t1.weight  
Constraint will be violated only if t20.profit ≥ t1.profit.  
However if this was the case, during the addition of t20, all items before t1 and t1 itself would have been purged because (t1.profit ≤ t20.profit) holds for items including t1 and before.   
Hence, **contradiction that** t20.profit ≥ t1.profit.  
Therefore, **addition of a new item would not violate constraint.  
Hence, the constraint is satisfied by the complete set Sk+1**

**Thus, Merge function produces the next set Sk+1 which is the optimal arrangement considering first k elements.**