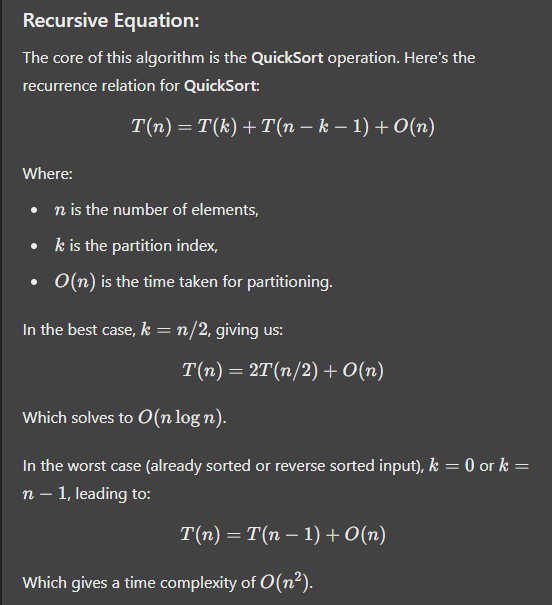
#### Steps of the Algorithm:

1. Calculate the value-to-weight ratio for each item.
2. Sort the items in descending order based on this ratio using a modified **QuickSort**.
3. Iterate through the sorted items:
   1. Add the item fully if its weight is less than or equal to the remaining knapsack capacity.
   2. If the item exceeds the remaining capacity, add a fraction of the item to fill the knapsack.
4. Output the maximum value the knapsack can hold.



### Time Complexity:

1. **QuickSort**: Sorting the items based on their value-to-weight ratio takes O(nlogn)O(n \log n)O(nlogn) in the average case.
2. **Knapsack Filling**: After sorting, the loop through the items takes O(n)O(n)O(n).

Thus, the overall time complexity of the algorithm is:

O(nlogn)O(n \log n)O(nlogn)

### Discussion:

* The **fractional knapsack** problem allows taking fractions of items, unlike the **0/1 knapsack**. Hence, a greedy solution that sorts by value-to-weight ratio works optimally.
* The key challenge here is sorting the items, which is handled efficiently using **QuickSort**.
* The complexity of the approach, O(nlogn)O(n \log n)O(nlogn), is better than the O(nW)O(nW)O(nW) time of the dynamic programming solution for the 0/1 knapsack, making this approach faster, especially when WWW is large. However, fractional knapsack only works in problems where fractional values are allowed.