```
/** Second algorithm - Greedy */
public void greedyAlgorithm() {
    item[] copyItems = new item[totalItem];
    int totalValue = 0;
    int totalWeight = 0;
    int index = 0;
    int[] resultIndex = new int[totalItem];

    // get the copy array of items
    for (int i = 0; i < totalItem; i++)
        copyItems[i] = itemArray[i];
    // sort the array of item in order of descending ratio of value/weight
    // the ratio of value/weight define how dense that item is. The more density, the better.
    Arrays.sort(copyItems);

    // Keep picking items from the sorted array until sack is full. If next best item too large to put in sack, skip it.
    while (index < totalItem) {
        if(totalWeight + copyItems[index].weight <= capacity){
            resultIndex[index] = copyItems[index].index;
            totalValue += copyItems[index].value;
            totalWeight += copyItems[index].weight;
        }
        index++;
    }
    // print result
    System.out.println("Greedy solution (not necessarily optimal): "
       + totalValue + " " + totalWeight);
    Arrays.sort(resultIndex);
    for (int ind : resultIndex)
        if (ind != 0)
            System.out.print(ind + " " );
    System.out.println();
}
/** Inner class item which represent an item with value, weight, and its index in the test file */
public class Item implements Comparable {
    int index; int value; int weight;
    public item (int ind, int val, int wei) {
        index = ind; value = val; weight = wei;
    }
     // implement compareTo method so that I can use Arrays.sort later.
    //  comparing the ratio of (value/weight). Return 1 if this Item has lesser ratio than other.
    public int compareTo(Object other) {
        return (double)(this.value)/(double)(this.weight) <
         (double)(((item)other).value)/(double)(((item)other).weight) ? 1 :
         (double)(this.value)/(double)(this.weight) >
         (double)(((item)other).value)/(double)(((item)other).weight) ? -1 : 0;
    }
}
```