Code for Knapsack Algorithm

import java.util.\*;

public class EXPT\_05\_Knapsack {

    // Sort by profit in descending order

    void Merge\_Arrays\_Profit(Item arr[], int start, int mid, int end) {

        int leftarr\_end = mid;

        int rightarr\_start = mid + 1;

        int l = start;

        while ((l <= leftarr\_end) && (rightarr\_start <= end)) {

            if (arr[start].pf > arr[rightarr\_start].pf) {

                start++;

            } else {

                Item temp = arr[rightarr\_start];

                for (int j = rightarr\_start - 1; j >= start; j--) {

                    arr[j + 1] = arr[j];

                }

                arr[start] = temp;

                start++;

                leftarr\_end++;

                rightarr\_start++;

            }

        }

    }

    void Merge\_Sort\_Profit(Item arr[], int start, int end) {

        if (start >= end) {

            return;

        }

        int mid = (start + end) / 2;

        Merge\_Sort\_Profit(arr, start, mid);

        Merge\_Sort\_Profit(arr, mid + 1, end);

        Merge\_Arrays\_Profit(arr, start, mid, end);

    }

    // sort by weight in ascending order

    void Merge\_Arrays\_Weight(Item arr[], int start, int mid, int end) {

        int leftarr\_end = mid;

        int rightarr\_start = mid + 1;

        int l = start;

        while ((l <= leftarr\_end) && (rightarr\_start <= end)) {

            if (arr[start].wt < arr[rightarr\_start].wt) {

                start++;

            } else {

                Item temp = arr[rightarr\_start];

                for (int j = rightarr\_start - 1; j >= start; j--) {

                    arr[j + 1] = arr[j];

                }

                arr[start] = temp;

                start++;

                leftarr\_end++;

                rightarr\_start++;

            }

        }

    }

    void Merge\_Sort\_Weight(Item arr[], int start, int end) {

        if (start >= end) {

            return;

        }

        int mid = (start + end) / 2;

        Merge\_Sort\_Weight(arr, start, mid);

        Merge\_Sort\_Weight(arr, mid + 1, end);

        Merge\_Arrays\_Weight(arr, start, mid, end);

    }

    // sort by profit by weight in descending order

    void Merge\_Arrays\_Profit\_Weight(Item arr[], int start, int mid, int end) {

        int leftarr\_end = mid;

        int rightarr\_start = mid + 1;

        int l = start;

        while ((l <= leftarr\_end) && (rightarr\_start <= end)) {

            if (arr[start].pw > arr[rightarr\_start].pw) {

                start++;

            } else {

                Item temp = arr[rightarr\_start];

                for (int j = rightarr\_start - 1; j >= start; j--) {

                    arr[j + 1] = arr[j];

                }

                arr[start] = temp;

                start++;

                leftarr\_end++;

                rightarr\_start++;

            }

        }

    }

    void Merge\_Sort\_Profit\_Weight(Item arr[], int start, int end) {

        if (start >= end) {

            return;

        }

        int mid = (start + end) / 2;

        Merge\_Sort\_Profit\_Weight(arr, start, mid);

        Merge\_Sort\_Profit\_Weight(arr, mid + 1, end);

        Merge\_Arrays\_Profit\_Weight(arr, start, mid, end);

    }

    void Max\_Profit(Item arr[], float max\_size) {

        fresh(arr);

        float current\_size = 0, max\_profit = 0;

        int i = 0;

        while (i < arr.length && current\_size < max\_size) {

            float space\_available = max\_size - current\_size;

            if (arr[i].wt <= space\_available) {

                arr[i].wei\_cons = 1;

                current\_size = current\_size + arr[i].wt;

                max\_profit = max\_profit + arr[i].pf;

            } else if (arr[i].wt > space\_available) {

                float size\_constant = space\_available / arr[i].wt;

                arr[i].wei\_cons = size\_constant;

                current\_size = max\_size;

                max\_profit = max\_profit + (arr[i].pf \* size\_constant);

            }

            i++;

        }

        System.out.print("\nThe Maximum Profit is " + max\_profit);

        String solution = "", weight = "";

        for (i = 0; i < arr.length; i++) {

            solution = solution + (arr[i].wei\_cons + " ");

            weight = weight + (arr[i].wt + " ");

        }

        System.out.print("\nWeights = { " + weight + " }");

        System.out.print("\nSolution = { " + solution + " }\n");

    }

    void fresh(Item arr[]) {

        for (int i = 0; i < arr.length; i++)

            arr[i].wei\_cons = 0;

    }

    public static void main(String args[]) {

        Scanner in = new Scanner(System.in);

        int num;

        float weight, profit, size;

        System.out.print("Enter the number of items in knapsack: ");

        num = in.nextInt();

        Item arr[] = new Item[num];

        System.out.print("\nEnter the details of each item.");

        for (int i = 0; i < num; i++) {

            System.out.print("\nFor item " + (i + 1) + ", enter: ");

            System.out.print("\nWeight: ");

            weight = in.nextFloat();

            System.out.print("\nProfit: ");

            profit = in.nextFloat();

            arr[i] = new Item(weight, profit);

        }

        System.out.print("\nEnter the size of knapsack: ");

        size = in.nextFloat();

        EXPT\_05\_Knapsack obj = new EXPT\_05\_Knapsack();

        long start = System.currentTimeMillis();

        System.out.print("\nCase I (Maximum Profit): ");

        obj.Merge\_Sort\_Profit(arr, 0, arr.length - 1);

        obj.Max\_Profit(arr, size);

        System.out.print("\nCase II (Minimum Weight): ");

        obj.Merge\_Sort\_Weight(arr, 0, arr.length - 1);

        obj.Max\_Profit(arr, size);

        System.out.print("\nCase III (Profit By Weight): ");

        obj.Merge\_Sort\_Profit\_Weight(arr, 0, arr.length - 1);

        obj.Max\_Profit(arr, size);

        long end = System.currentTimeMillis();

        long time = end - start;

        System.out.print("\nTime taken is " + time + " milliseconds.");

    }

}

class Item {

    float wt, pf, pw, wei\_cons;

    Item(float wt, float pf) {

        this.wt = wt;

        this.pf = pf;

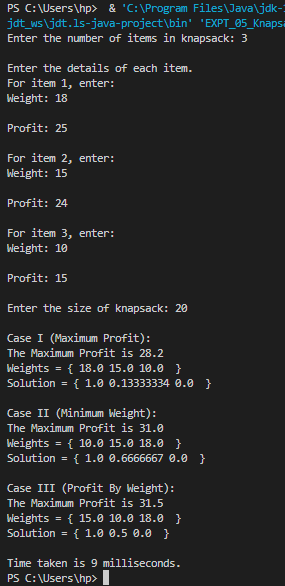
        pw = pf / wt;

    }

}

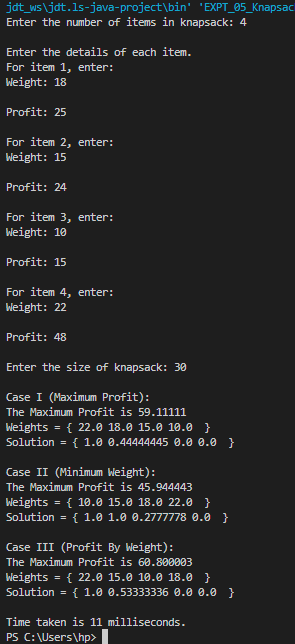
Case 1: When the number of items is 3

Time taken is 9 milliseconds.



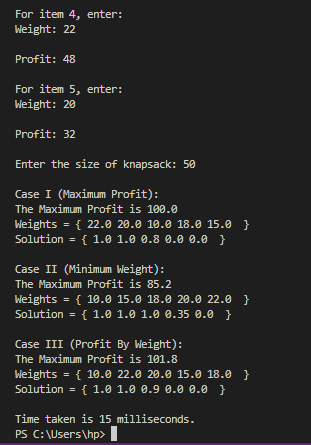
Case 2: When the number of items is 4

Time taken is 11 milliseconds.



Case 3: When the number of items is 5

Time taken is 15 milliseconds.



Case 4: When the number of items is 6

Time taken is 13 milliseconds.

