Practical No: 03

Name: Mohit Deepak Agrawal
Section: A2
Batch: B2
Roll No: 33

Part A:

Aim: Perform Fractional Knapsack for the given scenario

Problem Definition: Suppose you are a transport dealer and want to load a truck with different types of boxes. Assume there are 50 types of boxes (Box-1 to Box-50), which weigh different and that the truck has a maximum capacity (truckSize). Each box has a profit value associated with it. It is the commission that the transporter will receive after transporting the box. You can choose any box to put on the truck as long as the number of boxes does not exceed truckSize. You can load partial boxes.

Tasks-1:

A. Load the truck using different methods: **Minimum weight, Maximum profit, Profit/weight ratio.** Compute the total profit using each method and infer the best performing method.

B. Compute the time required in each method.

Given Data:

- Capacity of truck 850 Kgs
- Weight in kg for each box:

[7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42, 9, 0, 42, 47, 52, 32, 26, 48, 55, 6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3, 86, 66, 31, 65, 0, 79, 20, 65, 52, 13]

- Profit in Rs for each box:

[360, 83, 59, 130, 431, 67, 230, 52, 93, 125, 670, 892, 600, 38, 48, 147, 78, 256, 63, 17, 120, 164, 432, 35, 92, 110, 22, 42, 50, 323, 514, 28, 87, 73, 78, 15, 26, 78, 210, 36, 85, 189, 274, 43, 33, 10, 19, 389, 276, 312]

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
typedef struct {
 int profit;
 int weight;
} Item;
int compareByRatio(const void* a, const void* b) {
 Item* item1 = (Item*)a;
 Item* item2 = (Item*)b;
double r1 = (item1->weight == 0) ? (double)item1->profit :
(double)item1->profit / item1->weight;
 double r2 = (item2->weight == 0) ? (double)item2->profit :
(double)item2->profit / item2->weight;
 return (r2 > r1) - (r2 < r1);
}
int compareByProfit(const void* a, const void* b) {
 Item* item1 = (Item*)a;
 Item* item2 = (Item*)b;
 return item2->profit - item1->profit;
}
int compareByWeight(const void* a, const void* b) {
 Item* item1 = (Item*)a;
 Item* item2 = (Item*)b;
 return item1->weight - item2->weight;
}
double knapsack(Item* items, int n, int capacity) {
 double totalValue = 0.0;
 int remaining = capacity;
 for (int i = 0; i < n; i++) {
 if (items[i].weight == 0) {
 totalValue += items[i].profit;
 continue;
 }
 if (items[i].weight <= remaining) {
 totalValue += items[i].profit;
 remaining -= items[i].weight;
 } else {
 totalValue += ((double)items[i].profit / items[i].weight) *
remaining;
 break;
```

```c
    }
  }
  return totalValue;
}
void result(const char* label, Item* originalItems, int n, int capacity,
int (*cmp)(const void*, const void*)) {
  Item* tempItems = (Item*)malloc(n * sizeof(Item));
  for (int i = 0; i < n; i++) tempItems[i] = originalItems[i];
  clock_t start = clock();
  qsort(tempItems, n, sizeof(Item), cmp);
  double profit = knapsack(tempItems, n, capacity);
  clock_t end = clock();
  double time_taken = (double)(end - start) / CLOCKS_PER_SEC;
  printf("%s: %.2f (Time: %.6f sec)\n", label, profit, time_taken);
  free(tempItems);
}
int main() {
int profit[] = {360, 83, 59, 130, 431, 67, 230, 52, 93, 125, 670, 892,
600, 38, 48, 147, 78, 256, 63, 17, 120,
 164, 432, 35, 92, 110, 22, 42, 50, 323, 514, 28, 87, 73, 78,
15, 26, 78, 210, 36, 85, 189, 274,
 43, 33, 10, 19, 389, 276, 312};
 int wt[] = {7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15,
42, 9, 0, 42, 47, 52, 32, 26, 48, 55,
 6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3, 86, 66, 31, 65, 0,
79, 20, 65, 52, 13};
 int capacity = 850;
 int n = sizeof(profit) / sizeof(profit[0]);
 Item* items = (Item*)malloc(n * sizeof(Item));
 for (int i = 0; i < n; i++) {
 items[i].profit = profit[i];
 items[i].weight = wt[i];
 }
 result("Profit for Profit/Weight ratio", items, n, capacity,
compareByRatio);
 result("Profit for Maximum profit", items, n, capacity,
compareByProfit);
 result("Profit for Minimum weight", items, n, capacity,
compareByWeight);
 free(items);
 return 0;
}
```

Output:-->


```
Minimum Weight: 6231.68 (Time: 0.000008 sec)
Maximum Profit: 7073.79 (Time: 0.000004 sec)
Profit/Weight Ratio: 7525.29 (Time: 0.000004 sec)


=== Code Execution Successful ===
```

TASK-2: SUBMISSION ON LEETCODE

https://leetcode.com/problems/maximum-units-on-a-truck/description/

➔ Do it yourself on leetcode, pass the test cases. Paste the submission screenshot with pass % in submission word file.
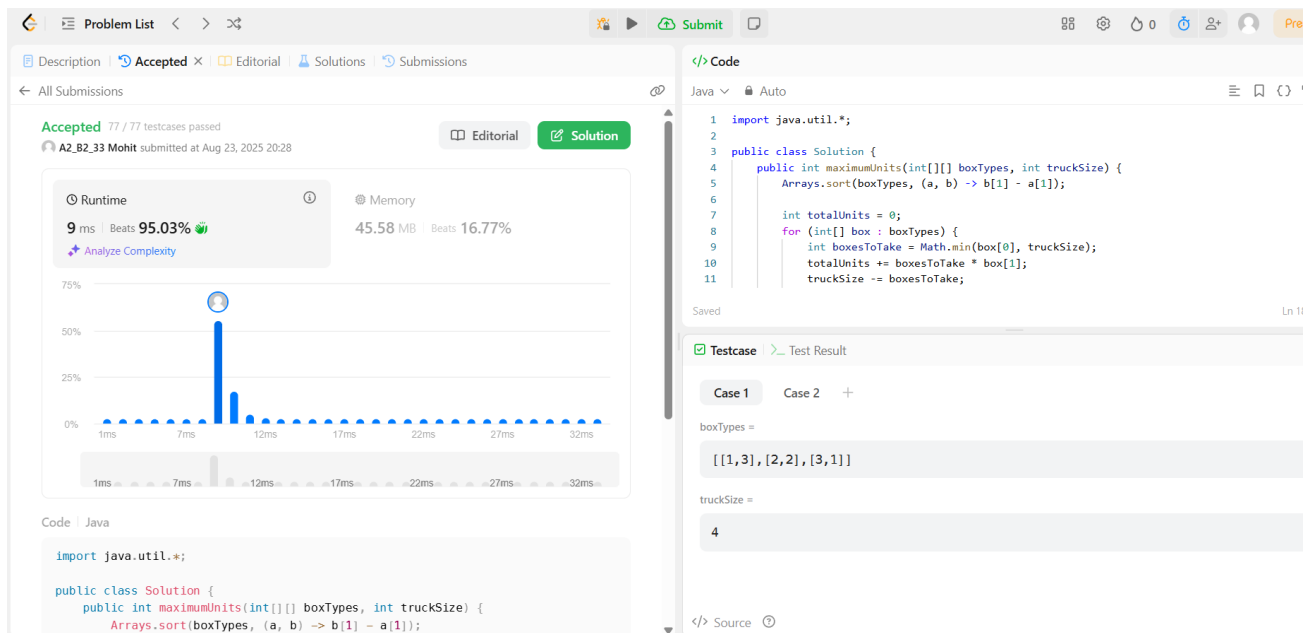
Part B:

Aim: Implement activity selection algorithm for the given scenario.

Problem Definition: In the single-machine scheduling problem, we are given a set of n activities Ai. Each job i has a starting time si, deadline di and profit pi. At any time instant, we can do only one job. Doing a job i earns a profit pi. Generate a solution to select the largest set of mutually compatible jobs and calculate the total profit generated by the machine. The greedy algorithm for single-machine scheduling selects the job using activity selection algorithm.

Start time (si)	Finish time (di)	Activity name(Ai)	Profit
1	4	A1	10
3	5	A2	15
0	6	A3	14
5	7	A4	12
3	9	A5	20
5	9	A6	30
6	10	A7	32
8	11	A8	28
8	12	A9	30
2	14	A10	40
12	16	A11	45

CODE:
```
#include <stdio.h>
#include <string.h>

int main() {
    int n = 11;
    int s[] = {1,3,0,5,3,5,6,8,8,2,12};
    int f[] = {4,5,6,7,9,9,10,11,12,14,16};
    int p[] = {10,15,14,12,20,30,32,28,30,40,45};
    char* a[] = {"A1","A2","A3","A4","A5","A6","A7","A8","A9","A10","A11"};

    int totalProfit = 0;
    int lastFinish = f[0];

    printf("The selected activities: ");
    printf("%s ", a[0]);
    totalProfit += p[0];

    for(int i = 1; i < n; i++) {
        if(s[i] >= lastFinish) {
            printf("%s ", a[i]);
            totalProfit += p[i];
            lastFinish = f[i];
```

```
        }
    }

    printf("\nTotal Profit = %d\n", totalProfit);
    return 0;
}
```

Output

The selected activities: A1 A4 A8 A11
Total Profit = 95

=== Code Execution Successful ===