**Assignment – 9**

**Q1. Implementation of Fractional Knapsack Problem.**

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct {  
 int id;  
 float profit, weight, percentage;  
} Product;  
  
int comparator(const void\*, const void\*);  
float fractional\_knapsack(Product\*, int, float);  
  
int main() {  
 int size, i;  
 float target;  
 printf("Enter the total number of products: ");  
 scanf("%d", &size);  
 Product\* sack = (Product\*)malloc(size \* sizeof(Product));  
  
 if (!sack) {  
 printf("Memory was not allocated\n");  
 return 1;  
 }  
 for (i = 0; i < size; i++) {  
 printf("Enter the ID, profit and weight of %dth product: ", i + 1);  
 scanf("%d%f%f", &sack[i].id, &sack[i].profit, &sack[i].weight);  
 sack[i].percentage = 0;  
 }  
 printf("Enter the targeted weight to achieve: ");  
 scanf("%f", &target);  
 float max = fractional\_knapsack(sack, size, target);  
 printf("\nThe ratio of products to take to maximize profit are:\n");  
  
 for (i = 0; i < size; i++) {  
 printf("ID: %d\t", sack[i].id);  
 printf("Percentage: %.2f%%\n", sack[i].percentage);  
 }  
 printf("Maximum profit is: %.2f\n", max);  
 free(sack);  
 return 0;  
}  
  
int comparator(const void\* a, const void\* b) {  
 Product lhs = \*(Product\*)a, rhs = \*(Product\*)b;  
 float lhs\_ratio = lhs.weight != 0 ? lhs.profit / lhs.weight : 0, rhs\_ratio = rhs.weight != 0 ? rhs.profit / rhs.weight : 0;  
  
 if (lhs\_ratio < rhs\_ratio) {  
 return 1;  
 }  
 if (lhs\_ratio > rhs\_ratio) {  
 return -1;  
 }  
 return 0;  
}  
  
float fractional\_knapsack(Product\* sack, int size, float target) {  
 int i = 0;  
 float res = 0;  
 qsort(sack, size, sizeof(Product), comparator);  
  
 while (i < size && sack[i].weight <= target) {  
 target -= sack[i].weight;  
 sack[i].percentage = 100;  
 res += sack[i++].profit;  
 }  
 if (i < size && sack[i].weight > 0) {  
 sack[i].percentage = target / sack[i].weight \* 100;  
 res += sack[i].profit \* sack[i].percentage;  
 }  
 return res;  
}

**Output**:

Enter the total number of elements available: 3  
Enter the ID, profit and weight of 1th product: 1 25 18  
Enter the ID, profit and weight of 2th product: 2 24 15  
Enter the ID, profit and weight of 3th product: 3 15 10  
Enter the targeted weight to achieve: 20  
  
The ratio of products to take to maximize profit are:  
ID: 2 Percentage: 100.00%  
ID: 3 Percentage: 50.00%  
ID: 1 Percentage: 0.00%  
Maximum profit is: 774.00

**Q2. Implementation of Activity Selection Problem.**

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct {  
 int id, start, finish;  
} Activity;  
  
int comparator(const void\*, const void\*);  
int activity\_selection(Activity\*, int, Activity\*);  
  
int main() {  
 int size, i;  
 printf("Enter the total number of activities: ");  
 scanf("%d", &size);  
 Activity \*activities = (Activity\*)malloc(size \* sizeof(Activity)), \*selected = (Activity\*)malloc(size \* sizeof(Activity));  
  
 if (!activities || !selected) {  
 printf("Memory was not allocated\n");  
 exit(0);  
 }  
 for (i = 0; i < size; i++) {  
 printf("Enter the ID, starting time and finishing time of %dth activity: ", i + 1);  
 scanf("%d%d", &activities[i].start, &activities[i].finish);  
 activities[i].id = i + 1;  
 }  
 int selects = activity\_selection(activities, size, selected);  
 printf("\nAll the selected activities are:\n");  
  
 for (i = 0; i < selects; i++) {  
 printf("ID: %d\t", selected[i].id);  
 printf("Start Time: %d\t", selected[i].start);  
 printf("Finish Time: %d\n", selected[i].finish);  
 }  
 free(activities);  
 free(selected);  
}

int comparator(const void\* a, const void\* b) {  
 Activity lhs = \*(Activity\*)a, rhs = \*(Activity\*)b;  
  
 if (lhs.finish < rhs.finish) {  
 return -1;  
 }  
 if (lhs.finish > rhs.finish) {  
 return 1;  
 }  
 return 0;  
}  
  
int activity\_selection(Activity\* activities, int size, Activity\* selected) {  
 qsort(activities, size, sizeof(Activity), comparator);  
 int i, k = 0;  
 selected[0] = activities[0];  
  
 for (i = 1; i < size; i++) {  
 if (activities[i].start >= selected[k].finish) {  
 selected[++k] = activities[i];  
 }  
 }  
 return k + 1;  
}

**Output**:

Enter the total number of activities: 11  
Enter the ID, starting time and finishing time of 1th activity: 1 4  
Enter the ID, starting time and finishing time of 2th activity: 3 5  
Enter the ID, starting time and finishing time of 3th activity: 0 6  
Enter the ID, starting time and finishing time of 4th activity: 5 7  
Enter the ID, starting time and finishing time of 5th activity: 3 9  
Enter the ID, starting time and finishing time of 6th activity: 5 9  
Enter the ID, starting time and finishing time of 7th activity: 6 10  
Enter the ID, starting time and finishing time of 8th activity: 8 11  
Enter the ID, starting time and finishing time of 9th activity: 8 12  
Enter the ID, starting time and finishing time of 10th activity: 2 14  
Enter the ID, starting time and finishing time of 11th activity: 12 16  
  
All the selected activities are:  
ID: 1 Starting Time: 1 Finishing Time: 4  
ID: 4 Starting Time: 5 Finishing Time: 7  
ID: 8 Starting Time: 8 Finishing Time: 11  
ID: 11 Starting Time: 12 Finishing Time: 16