Greedy Algorithms

A greedy algorithm always makes the choice that looks best at this moment.

Practice problems:

Instructions:

1. Do not adopt unfair means. **10 marks will be deducted from the final marks for adopting unfair means**.
2. No more than 40% marks for uncompilable codes.

Fractional knapsack

```
Function Fractional-Knapsack (W, v[n], w[n])
1.  Order item-list by v_i/w_i descending
2.  while w > 0 and as long as there are items remaining
3.      pick item with maximum v_i/w_i
4.      x_i ← min (1, w/w_i)
5.      remove item i from list
6.      w ← w – x_i w_i
w: the amount of space remaining in the knapsack (initially w = W)
```

Activity Selection Problem

Greedy algorithms for this problem:

- Sort by start time
- Sort by finish time (this one gives optimal answer)
- Sort by interval

The following two pseudocodes assume that the activities are sorted according to finish time.

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

Activity selection in different scenario: Version 1:

Suppose, you are a coder. You write code for customers. You charge all customers the same amount of money irrespective of how many hours it takes to write the code. Your customers let you know when he will come to you and how many hours it will take to write the code for him in the beforehand (the previous day). You can write code for one customer at a time.

Write a code to maximize your income tomorrow.

Sample input	Sample output
4 'a', 2, 8 'b', 3, 4 'd', 8, 1 'c', 7, 1	'b' 'c' 'd'

Huffman Encoding

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

Huffman Decoding

Greedy Coin Change

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer. Write a greedy algorithm to make change consisting of quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent).

Maximize your payment

Suppose, you are temporarily working as a photoshop editor. You work **10 hours a day** from 12:00 PM to 10 PM. You have a unique set of customers; each customer provides you three information for the next day tasks. **One is**, what he is going to pay you (**payment**); **two**, **how many hours** will be needed to do that task; and **three**, within which time you must complete this task (**last time**). If you fail to do the tasks within that time, then that customer does not give the task. You cannot serve multiple customers at a time. **Write the following greedy algorithm to choose the tasks to maximize your payment.** [You can do a task partially at one time and finish it at another time.]

Algorithm:

1. Sort the tasks by descending order of *payment per hour*.
2. Choose the task with the maximum *payment per hour* and do it in the last possible hours. Make that hour occupied.
3. Choose the next maximum paid (*payment per hour*) task, find the last possible time that is not occupied, do it at that time and make that hour occupied. If no such time exists, you cannot do this task.
4. Move to the next maximum profitable task (based on *payment per hour*) and repeat step 3.

Sample Input	Sample Output
Customer | Last time (PM) | Time needed (hour) | Payment (Tk) a | 4 | 2 | 2000 b | 1 | 1 | 1000 c | 1 | 1 | 4000 d | 2 | 2 | 3000	c, a 6000
Customer | Last time (PM) | Time needed (hour) | Payment (Tk) a | 2 | 1 | 1000 b | 1 | 1 | 190 c | 2 | 1 | 270 d | 1 | 1 | 250 e | 3 | 1 | 150	c, a, e 1420
Customer | Last time (PM) | Time needed (hour) | Payment (Tk) a | 2 | 1 | 1000 b | 4 | 1 | 1000 c | 4 | 2 | 500	