**Design PS16Q2**

The design implements the greedy algorithm to solve the fractional knapsack problem.

Steps –

1. Read the input file completely line by line and prepare an array of lists for food items
2. Each list in the array stored the food item name, weight and calories as given below

Each food item - [‘F1’, 5, 22]

List of all food items: Array/List of each item list

Items: [['F1', 5, 22], ['F2', 3, 50], ['F3', 8, 18], ['F4', 2, 60], ['F5', 1, 35], ['F6', 2, 22]]

1. The design makes use of greedy algorithm where we try to maximize the profit (calories). We calculate the calorie/weight for each food item and directly put then in an array as follows.

Unsorted calorie\_to\_weight ratio: [[-1, 0.0], [0, 4.4], [1, 16.67], [2, 2.25], [3, 30.0], [4, 35.0], [5, 11.0]]

As we note above index 0 in the above array is unused and dummy entry as we will apply max heap on this array representation of the default tree. Index 0 is left free as we want to use i = root, 2i = left child and 2i + 1 = right child of each node in the tree

Also note that index of the food item in the Item list is kept together with the calorie to weight ratio to match against the food item it belongs to.

1. We then use heap sort which at run times keep doing max\_heapify making the root element as the maximum. Element.
2. During heapsort after each heapify operation we then swap the root node (max) value with the last item in the tree and then again call heapify which is recursive in nature
3. Every swap of the max elemnt at root with last element also keeps reducing the size of the tree (array) with max element being removed
4. Result is that we get the final array in place sorted in increasing order as below –

Sorted array: [[-1, 0.0], [2, 2.25], [0, 4.4], [5, 11.0], [1, 16.67], [3, 30.0], [4, 35.0]]

We see the most profitable item is in the last index of the array and all items have food list item index as well – [index, ratio]

1. Next, we apply the greedy algorithm to pick items completely from biggest index of the above sorted array as long as the bag weight is greater.
2. As soon as the bag weight is not left sufficient to pick the next item, we calculate the fraction needed using target\_wt\_left / item\_weight and add that ratio of item weight
3. Profit is calculated in the same loop as we go through the above sorted array adding weight to knapsack.

# Important functions:

Heap sort is implemented using the functions –

1. def max\_heapify(arr, n, i)

Where array is the array of the elements to be sorted, n is input size, i is the root node index of the subtree to heapify

Operations:

* Checks the left and right chile values with the root node value to determine the maxima
* If root value itself is maximum then function is over
* If root is not maxima, we swap it with the child with maximum value
* Recursively call max\_heapify with root of subtree as the index of the child having maxima

This function atmost with have to check the complete height of the tree that is log n. Hence the complexity of the max\_heapify is O(log n)

1. def heapsort(arr, n)

This function has two sub problems to solve. Once to build the heap initially. This will first find the last parent node and then recursively call max\_heapify

def heapSort(arr, n):

# start with the rightmost node's parent and loop till index 1

# index 0 is unused to enable root = i, left = 2i and right = 2i + 1

for i in range(int(n/2), 0, -1):

max\_heapify(arr, n, i)

The above will have to loop through all the nodes (indexes) as n->infinity and hence the loop provides complexity of O(n). max\_heapify() which is recursive has complexity of O(log n). So the total complexity of this sub problem is O(n).O(log n) = O(n log n)

Second subproblem then swaps the root element with last node and then redoes the heapify for the new tree recursively as sown below –

# now swap the root which will be max with last element

# and reheapify after removing the last element from the

# array size. Again loop till index 1 as index 0 is unused

for i in range (n - 1, 0, -1):

max = arr[1]

arr[1] = arr[i]

arr[i] = max

max\_heapify(arr, i, 1)

This will then again run O(n-1).O(log n) = O (n log n). So over all heapsort complexity is

O(n log n)

1. After heap sort the input ‘arr’ has the ordered list of items in the ratio of their profit value. We simply loop through this array from last index to 1 (0th index is dummy to help in building heap)

Here we worst case have to loop over all ‘n’ food items. And then probably check the last item in fraction that we might have to add.

for i in range(item\_cnt, 0, -1):

index = calorie\_to\_weight[i][0]

if item\_list[index][1] > target\_wt:

break;

output[index] = 1

target\_wt -= item\_list[index][1]

calories += item\_list[index][2] \* item\_list[index][1]

Loop complexity is O(n) and the complexity of adding fractional item is O(1).

if i > 0: # some items not yet covered

index = calorie\_to\_weight[i][0]

output[index] = target\_wt / item\_list[index][1]

calories += (output[index] \* item\_list[index][2] \* item\_list[index][1])

So total complexity of adding is O(n) + O(1) = O(n)

Profit is calculated within the above loop and last fractional operation itself.

Total complexity of the complete algorithm is summation of all subproblems –

heapsort + knapsack filling =

O(n log n) + O(n) = **O(n log n)**

# Param Validation

A detailed input validation routine is written to validate the input file format and all parameters are correct. Any error is captured in the prompts **promptsPS16Q2.txt**

Also, all debug output if any is captured in the promptsPS16Q2.txt file as well. All stdout prints are disabled.

Prompts file was not needed for this question as per the professor Sheetal. It is added only for stdout error prints while debugging and can be ignored. There is no stdout prints when program is executed and output file is generated with final output.

# Output

Output is captured in **outputPS16Q2.txt** file. Sample input file is **inputPS16Q2.txt**

# Logging

Added a logger implementation which logs into the prompts and output file. It has capability to simultaneously print in console or in the specified files. By default console logging is commented.