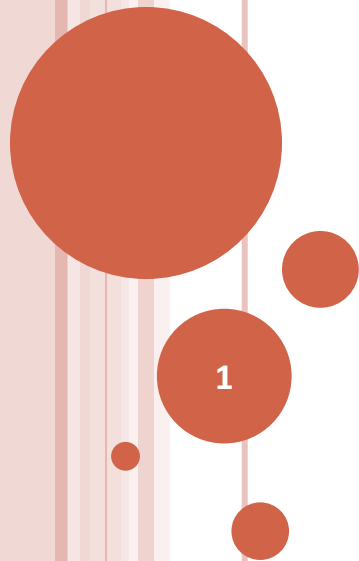


CS F364

Design & Analysis of Algorithms

ALGORITHM DESIGN TECHNIQUES - GREEDY

Greedy Algorithms - Example: (Fractional) KnapSack



PROBLEM – (FRACTIONAL) KNAPSACK

- A thief wants to rob the grocery store:
 - Has a sack with max. capacity by weight: W kg.
 - Each item j (in store) is labeled with
 - Package Size : w_j kg and Price (of the package) : Rs. p_j
- Assumption:
 - Any item can be taken in fractional quantity
 - All values (w_j , p_j , and W) are positive.
- Feasible Solution:
 - Fill the sack with maximum value (by price)
- Goal:
 - Maximize $\sum p_i (x_i / w_i)$
where $0 \leq x_i \leq w_i$ for each i and $\sum x_i \leq W$
if x_i is the amount taken of item i

KNAPSACK – GREEDY ALGORITHM

- Algorithm KnapSack(S, W)
- // S Set of items; W capacity

Sort S by key $v_j = p_j / w_j$

Initialize array X of size |S| with all 0s.

remW = W

while (remW > 0) {

 i = findMax(S);

 S = deleteMax(S,i)

 X[i] = min(w_i , remW);

 remW = remW – X[i];

}

output X

KNAPSACK – GREEDY CHOICE

- Knapsack satisfies Greedy Choice property:
 - Suppose there are items j and k such that
 - $x_k < w_k$, $x_j > 0$, and $v_k < v_j$
 - Let
 - $y = \min (w_k - x_k, x_j)$
 - Then
 - replace an amount y of item j , with same amount of item k
 - and increase the value without increasing the weight!

KNAPSACK – GREEDY ALGORITHM – TIME COMPLEXITY

○ Algorithm KnapSack(S, W) //S - list of items; W - capacity

1. Order S by key $v_j = p_j / w_j$ ← **$O(n)$ where $n = |S|$**
2. Initialize array X of size $|S|$ with all 0s. ← **$O(n)$**
3. $remW = W$
4. while ($remW > 0$) {
5. $i = \text{findMax}(S);$
6. $S = \text{deleteMax}(S, i)$ ← **$O(\log(n))$**
7. $X[i] = \min(w_i, remW);$
8. $remW = remW - X[i];$
9. } ← **$O(n)$**

Assuming a Heap is used for storing keys ordered by unit price

Time Complexity – $O(n \cdot \log(n))$

Question: Will there be an impact on performance if a sorted array is used for S instead of a heap?