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_i 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 $\Sigma\, p_i\, (x_i / w_i)$

    where $0 <= x_i <= w_i$ for each i and $\Sigma\, x_i <= W$

    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

```
Order S by key v_j = p_j / w_j          O(n) where n = |S|
Initialize array X of size |S| with all 0s.          O(n)
remW = W
while (remW > 0) {
   i = findMax(S);
   S = deleteMax(S,i)          O(log(n))
   X[i] = min(w_i , remW);
   remW = remW – X[i];
}          O(n)
```

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

Time Complexity – O(n*log(n))

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