6

Greedy algorithms

* A greedy algo always makes the choice that looks best at the moment.

* These makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

* The greedy algos do not always yield optimal solutions, but for many problems they do.

Activity selection problem

* Suppose we have a set $S = \{1, 2, 3, \ldots\}$ of n proposed activities that wish to use a resource such as a lecture hall, which can be used by only one activity at a time.

* Each activity i has a start time s_i and a finish time f_i where $s_i \leq f_i$.

* If selected, activity i takes place during half open time interval $[s_i, f_i)$

* Activities i and j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap ie if $s_i \geq f_j$ or $s_j \geq f_i$

* Activity selection problem is to select a maximum size set of mutually compatible activities.

We assume that input activities to this problem are in sorted order of increasing finishing time. ie if $f_1 \leq f_2 \leq f_3 \cdots \leq f_n$.

```
Greedy-Activity-Selector (s, f)
n ← length [s]
A ← {1}
j ← 1
for i ← 2 to n
    do if si ≥ fj
       then A ← A ∪ {i}
            j ← i
return A
```

Set A collects selected activities
j specifies most recent addition to A
Initially j is 1, ie A contains 1st activity.

Condition is that its start time s_i is not earlier than finish time f_j of activity most recently added to A.

Running time if not sorted by finish time = $O(n \lg n)$

Running time if sorted by finish time = $\Theta(n)$

Proving greedy algo correct

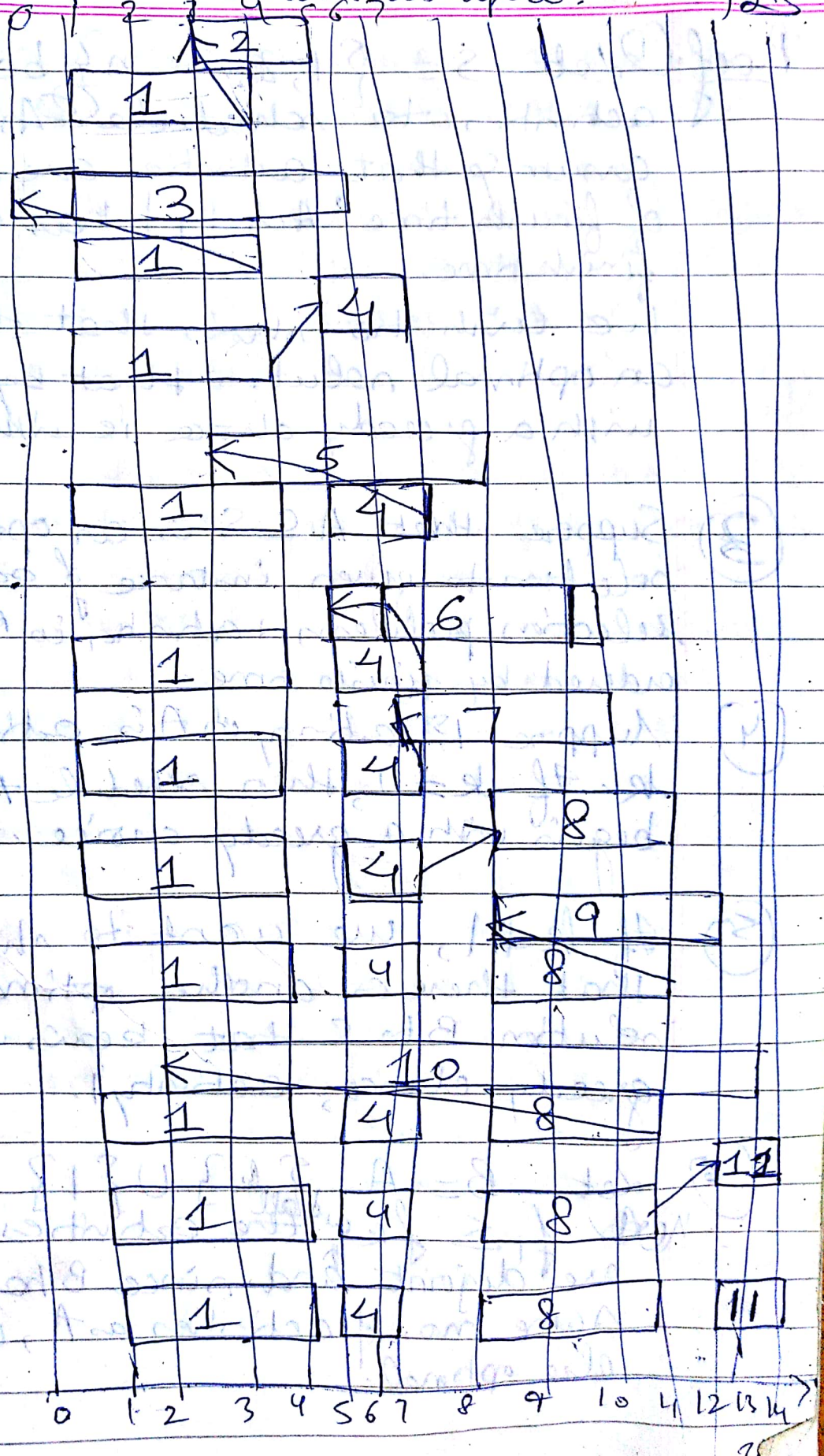
Greedy algo do not always produce optimal solution. However, Greedy-activity-selector always finds an optimal solution to an instance of activity selection problem.

Theorem :-

Algorithm Greedy-Activity-Selector produces solution of maximum size for activity selection problem.

Each row corresponds to an iteration of for loop. Activity is rejected if s_i occurs before finishing time f_j (arrow is left). Activity Accepted if arrow is right.

i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14



Proof: (1) Let $S = \{1, 2, \ldots n\}$ be set of activities to schedule (2) As we are assuming that activities are in order of finish time. Activity 1 has earliest finish time.
We wish to show that there is an optimal solution that begins with a greedy choice i.e with activity 1.

(3) Suppose that $A \subseteq S$ is an optimal solution to given instance of activity selection problem + activities in A are ordered by finish time.

(4) Suppose 1st activity of A is activity k. If $k = 1$, then schedule A begins with a greedy choice.

(5) If $k \neq 1$, we want to show that there is another optimal solution B to S that begins with greedy choice, activity 1.

(6) Let $B = A - \{k\} \cup \{1\}$
Note $f_1 \leq f_k$ and the activities in B are disjoint and since B has the same no. of activities as A, it is also optimal.

Thus B is an optimal solution for S that contains greedy choice of activity 1.

If A is an optimal solution to original problem S, then $A' = A - \{1\}$ is an optimal solution to activity selection problem $S' = \{i \in S : s_i \geq f_1\}$

so, when greedy choice of activity 1 is made the problem reduces to finding an optimal solution for activity-selection problem over those activities in S that are compatible with activity 1.

eg. If k ≠ 1

eg. $A = \{2, 8, 14\}$

Let $B = \{1, 8, 14\}$

$f_1 \leq f_2$

For greedy strategy to be applicable problem must have:-

Elements of greedy strategy

Two ingredients that are exhibited by most problems that lend themselves to greedy strategy are:-)

(i) Greedy choice property:- ie a globally optimal solution can be derived as by making a locally optimal (greedy) choice.

Diff b/w DP + greedy

(a) But in dynamic programming we make a choice at each step + choice depends on solutions to subproblems. And in greedy we make choice that looks best at the moment (it does not depend on solution to subproblems)

(b) DP is bottom up technique
Greedy is top-up strategy where one greedy choice is made after another, iteratively reducing each given problem to smaller one.

common to D.P + greedy.

(2) Optimal substructure: if optimal sol^n to problem contains within it optimal solutions to subproblems.
problem has optimal substructure

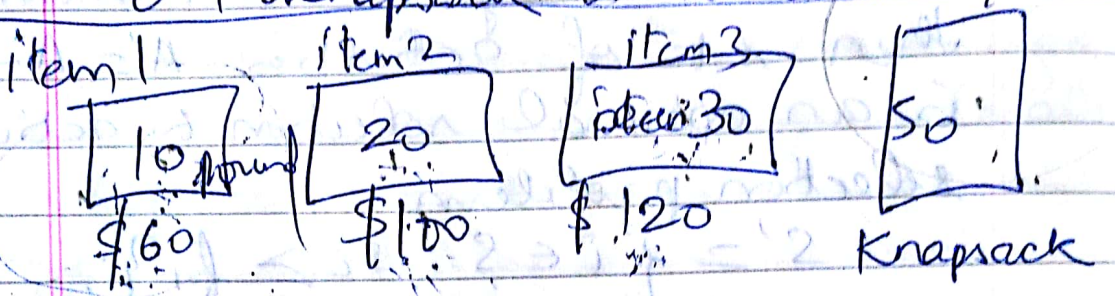
eg. activity selection problem: if optimal solution A begins with activity 1, then set of activities $A' = A - \{1\}$ is an optimal solution to activity selection problem
$S' = \{i \in S : s_i \geq f_1\}$.

Greedy vs D.P.

0-1 knapsack problem:- A thief robbing a store finds n items, ith item is worth v_i dollars + weighs w_i pounds where v_i and w_i are integers. He can carry atmost W pounds. Thief can take either an item or not (not fractional amount).

In fractional knapsack problem, setup is same but thief can take fractions of items, rather than having to make binary choice for each item.

Both knapsack problems exhibit optimal-substructure property. But fractional knapsack problem is solvable by greedy strategy whereas 0-1 knapsack is not.
0-1 knapsack is solvable by D.P.

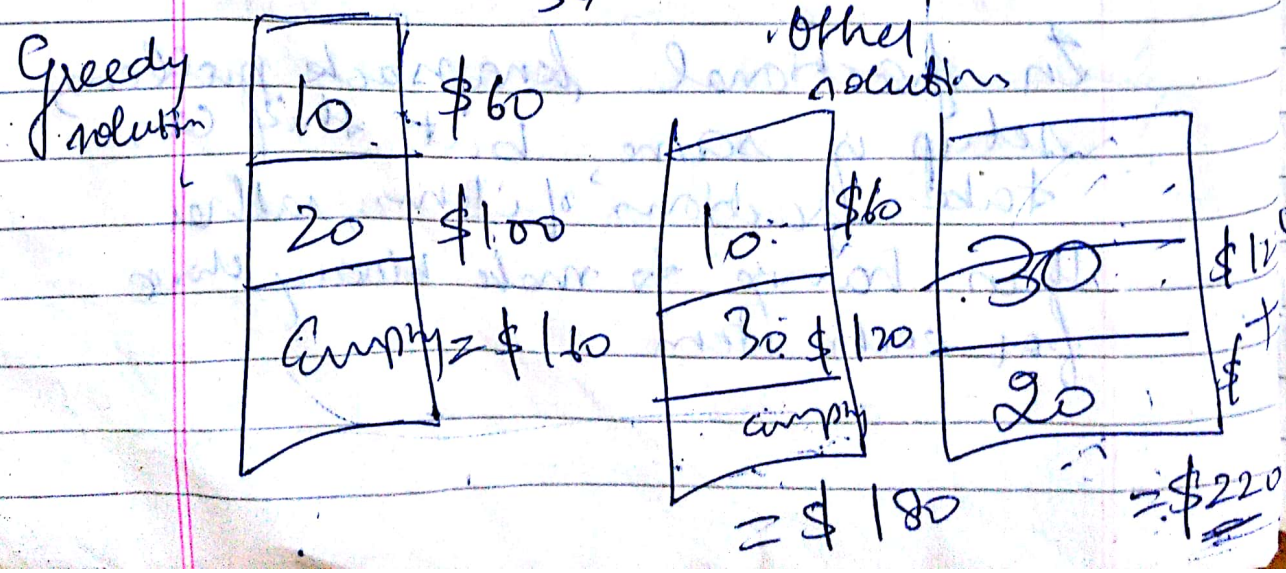


0-1 Knapsack problem
We calculate vi/wi per item

Item 1 = $\frac{60}{10}$ = 6 dollar/pound

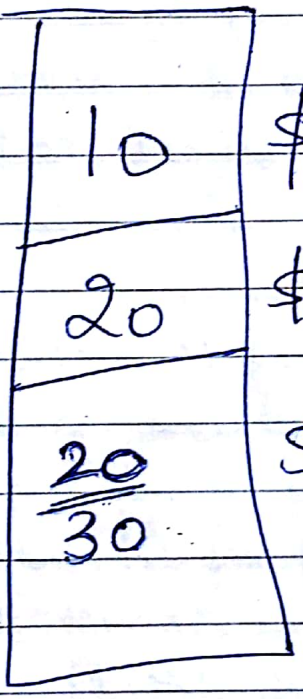
Item 2 = $\frac{100}{20}$ = 5 dollar/pound

Item 3 = $\frac{120}{30}$ = 4 dollar/pound



Greedy takes item 1 and 2, but optimal takes 2 and 3 leaving 1, no 0-1 knapsack cannot be solved using greedy.

Fractional Knapsack



$240

$(1, 1, 2/3)$

$10 + 20 + \frac{2}{3} \times 30$

$= 30 + 20$

$= 50$

$\frac{2}{3} \times \$240 = 80$

Task scheduling problem

- It is problem of optimally scheduling unit time tasks on a single processor, where each task has a deadline + a penalty that must be paid if deadline is missed.

- A unit time task is a job such as program to be run on a computer, that requires exactly one unit of time to complete.

- Given a finite set S of unit time tasks, a schedule for S is a permutation specifying order in which these tasks are to be performed. (1st task begins at time 0 + finishes at time 1, 2nd task at time 1 + 4 4 + 2 + so on)

- The problem has foll^{n} inputs:-
  (1) A set $S = \{1, 2, \ldots n\}$ of n unit time tasks
  (2) A set of n integer deadlines $d_1, d_2, \ldots d_n$ s.t. $1 \le d_i \le n$ + task i is supposed to finish by time d_i
  (3) A set of n nonnegative weights or penalties $w_1, w_2, \ldots w_n$ s.t. w_i is incurred if task i is not finished by time d_i + no penalty is incurred if a task finishes by its deadline.

- We need to find a schedule for S that minimizes total penalty incurred for missed deadlines.

- A task is late in schedule if it finishes after its deadline, otherwise task is early.

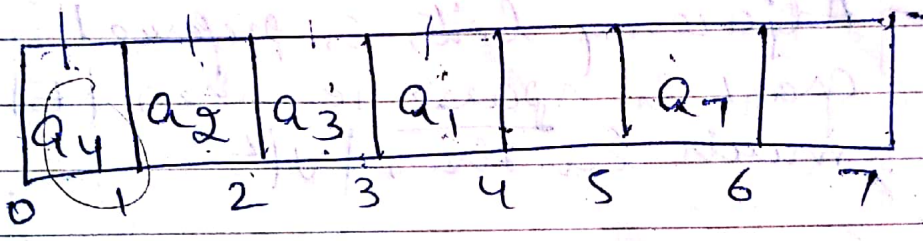
- A schedule is put into early first form in which early tasks precede late tasks.

- An arbitrary schedule can always be put into canonical form in which early tasks precede late tasks and early tasks are scheduled in order of nondecreasing deadlines.

- A set A of tasks is independent if there exists a schedule for these tasks such that no tasks are late.

	Task						
eg.	1	2	3	4	5	6	7
deadline d_i	4	2	4	3	1	4	6
penalty w_i	70	60	50	40	30	20	10

(1) Arrange tasks in ↓ order of penalties and start filling deadlines



Total penalty $w_5 + w_6$
$\Rightarrow 30 + 20 = 50$

(2) Sort tasks in order of deadlines in ↑ing order.

	a_4	a_2	a_3	a_1	a_7
d_i	3	2	4	4	6

↳ sorting

	a_2	a_4	a_3	a_1	a_7
d_i	2	3	4	4	6

Final optimal schedule is
$(a_2, a_4, a_1, a_3, a_7, a_5, a_6)$
or
$(2, 4, 1, 3, 7, 5, 6)$
Total Penalty $w_5 + w_6 = 50$