Greedy Algorithms

Design and Analysis of Algorithms

zeshan.khan@nu.edu.pk

Optimization Problems

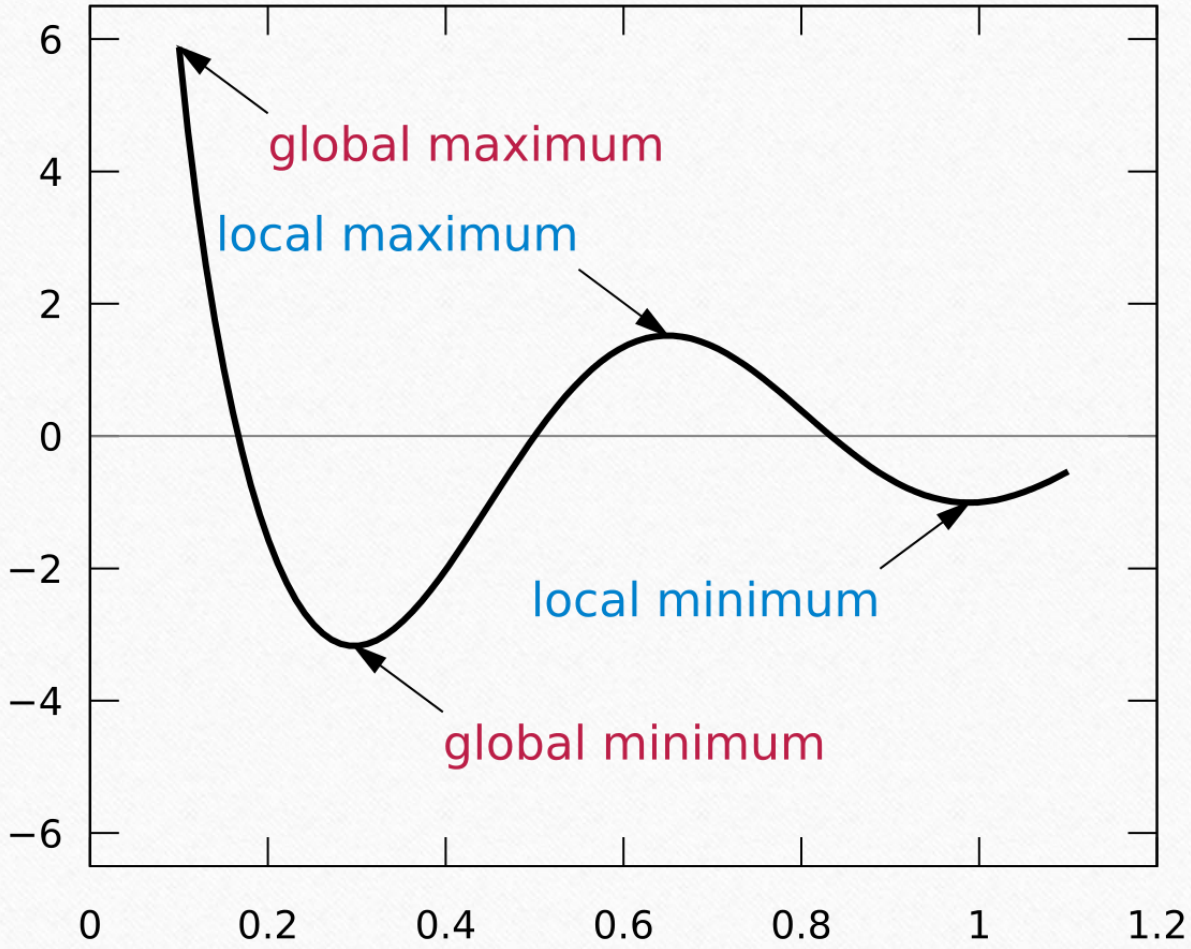
- For most optimization problems you want to find, not just ***a*** solution, but the ***best*** solution.
- A ***greedy algorithm*** sometimes works well for optimization problems. It works in phases. At each phase:
  - You take the best you can get right now, without regard for future consequences.
  - You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum.

Greedy Approach

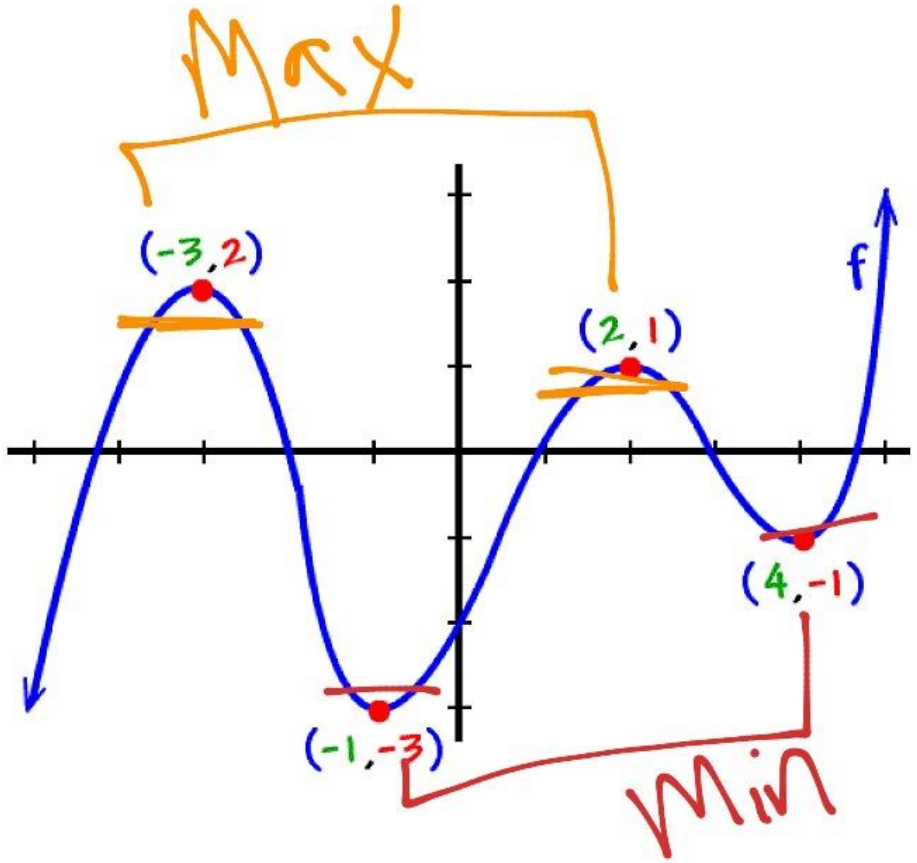
- Like dynamic programming, used to solve optimization problems.
- Problems exhibit optimal substructure (like DP).
- Problems also exhibit the **greedy-choice** property.
  - When we have a choice to make, make the one that looks best *right now*.
  - Make a **locally optimal choice** in hope of getting a **globally optimal solution**

- Does **greedy-choice** property lead us to optimize solution?
-

- **Greedy algorithms** mostly (but not **always**) fail to find the globally **optimal solution** because they usually **do** not operate exhaustively on all the data.


global maximum
local maximum
local minimum
global minimum
6
4
2
0
−2
−4
−6
0
0.2
0.4
0.6
0.8
1
1.2

Maximums and minimums

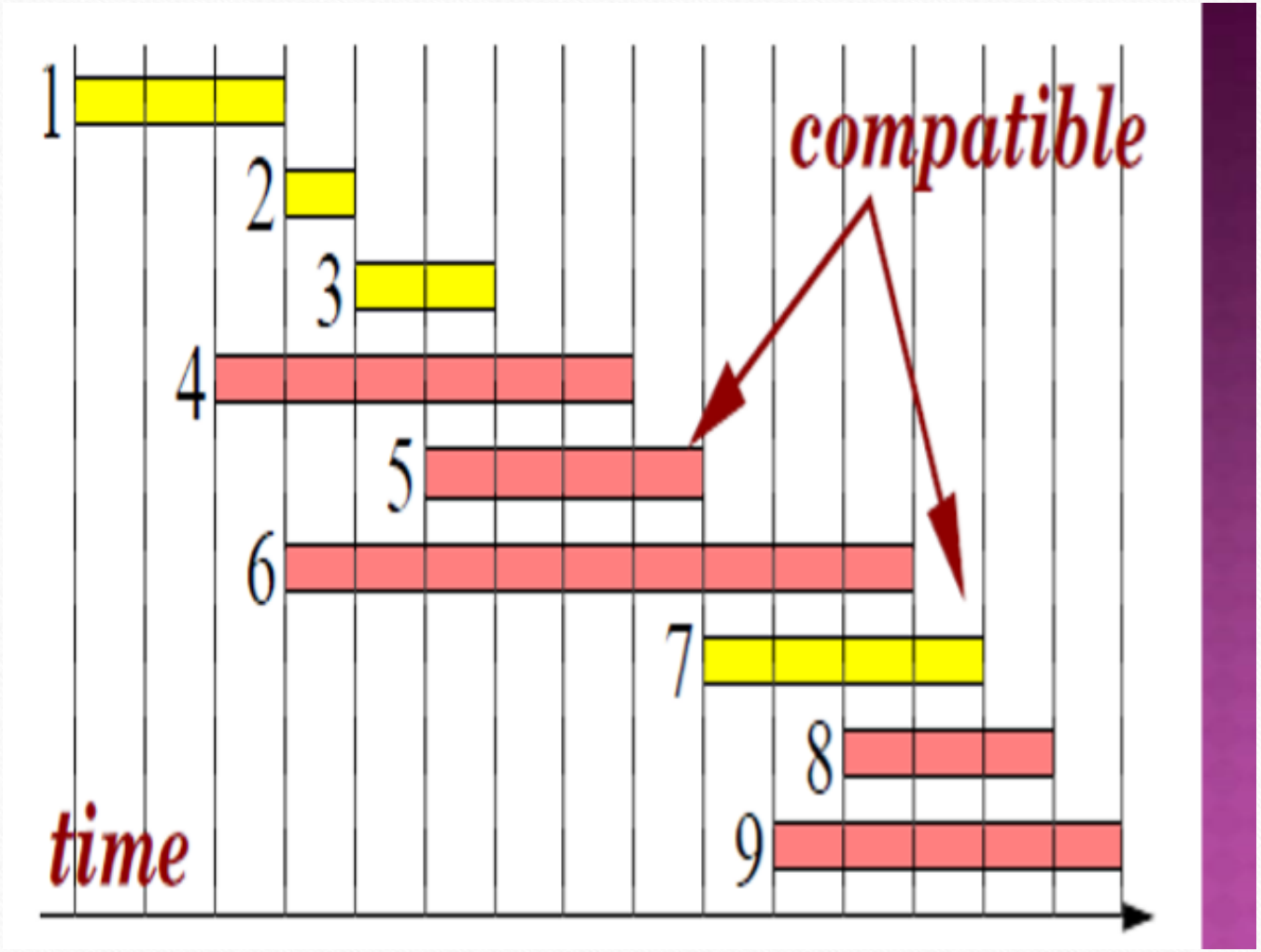


Activity Selection

- **Problem**: Given n activities with their start and finish time.
- **Goal**: select maximum activities performed by a single machine

Compatible Activities

- compatible$(a_{s,f}, b_{s,f})$ iff $b_s \geq a_f$ or $a_s \geq b_f$


1
2
3
4
5
6
7
8
9
compatible
time

Activities

Activity	1	2	3	4	5	6	7	8	9	10	11
Start	1	3	0	5	3	5	6	8	8	2	12
Finish	4	5	6	7	9	9	10	11	12	14	16

$$Res = \{a_1, a_4, a_8, a_9\}$$
$$Optimal = \{a_3, a_7, a_{11}\}$$
$$Optimal = \{a_3, a_9, a_{11}\}$$

Solution

Solution

Greedy Approach -

1) Sort the activities according to their finishing time
2) Select the first activity from the sorted array and print it.
3) Do following for remaining activities in the sorted array.
   -If the start time of this activity is greater than or equal to the finish time of previously selected activity then select this activity and print it.

Solution

Greedy Approach -

1) Sort the activities according to their finishing time

Activity	A1	A2	A3	A4	A5	A6
Start	0	3	1	5	5	8
Finish	6	4	2	9	7	9

Solution

Greedy Approach -

1) Sort the activities according to their finishing time

Activity	A1	A2	A3	A4	A5	A6
Start	0	3	1	5	5	8
Finish	6	4	2	9	7	9

Sorted

Activity	A3	A2	A1	A5	A6	A4
Start	1	3	0	5	8	5
Finish	2	4	6	7	9	9

Greedy Approach -

3) If the start time of next activity is greater than or equal to the finish time of previously selected activity then select this activity and print it.

Activity	A3	A2	A1	A5	A6	A4
Start	1	3	0	5	8	5
Finish	2	4	6	7	9	9

Answer = A3 → A2 → A5 → A6 **Total** = 4 activities

```
void printMaxActivities(int s[], int f[], int n)
{
    int i, j;

    printf ("Following activities are selected n");

    // The first activity always gets selected
    i = 0;
    printf("%d ", i);

    // Consider rest of the activities
    for (j = 1; j < n; j++)
    {
      // If this activity has start time greater than or
      // equal to the finish time of previously selected
      // activity, then select it
      if (s[j] >= f[i])
      {
          printf ("%d ", j);
          i = j;
      }
    }
}
```

Complexity

Time Complexity :

For sorted input : O(n)
For unsorted input : O(nlogn)

n=number of activities

Example: Counting Money

- Suppose you want to count out a certain amount of money, using the fewest possible bills and coins
- A greedy algorithm to do this would be:
  At each step, take the largest possible bill or coin that does not overshoot
  - Example: To make $6.39, you can choose:
    - a $5 bill
    - a $1 bill, to make $6
    - a 25¢ coin, to make $6.25
    - A 10¢ coin, to make $6.35
    - four 1¢ coins, to make $6.39
- For US money, the greedy algorithm always gives the optimum solution

Greedy Approach Failure

- In some (fictional) monetary system, "krons" come in 1 kron, 7 kron, and 10 kron coins
- Using a greedy algorithm to count out 15 krons, you would get
  - A 10 kron piece
  - Five 1 kron pieces, for a total of 15 krons
  - This requires six coins
- A better solution would be to use two 7 kron pieces and one 1 kron piece
  - This only requires three coins
- The greedy algorithm results in a solution, but not in an optimal solution