**<http://cse.iitkgp.ac.in/~abhij/course/theory/Algo1/Autumn13/lab/A3.html>**

**Dynamic-Programming Algorithms**

In this assignment, you solve [Mr. Sched's second problem](http://cse.iitkgp.ac.in/~abhij/course/theory/Algo1/Autumn13/lab/A2.html). You are given *n* intervals. We have seem that the greedy strategy "earliest finish time first" helped us pick the maximum number of mutually non-overlapping intervals. Now, your task is to pick a set of mutually non-overlapping intervals in such a manner that the total length of the chosen intervals is as large as possible. If the intervals stand for processes to be scheduled on a processor, this maximizes the utilization of the processor.

Approach the problem in three different ways.

1. First, implement the greedy strategy "earliest finish time first", and output the utilization that this gives. This should run in O(*n* log *n*) time (including the sorting phase).
2. Then, implement the greedy strategy "earliest start time first", and output the utilization that this gives. The first two parts may be a copy from your second assignment. Do not copy from other people's (including my) codes. This function too should run in O(*n* log *n*) time (including sorting).
3. Finally, try a dynamic-programming strategy which is guaranteed to give an optimal solution. Formulate the problem recursively as follows. Suppose that the intervals *I*1, *I*2, ..., *In* are sorted with respect to their left end points (start times). This sorted list is already available from Part 2. Now, consider two possibilities: include *I*1, and do not include *I*1 in your list of scheduled intervals. If you include *I*1, then find out the first compatible interval after *I*1 in the sorted list (provided that such an interval exists). Let this interval be *Ik*. You then recursively find an optimal schedule among the intervals *Ik* through *In*. Also add the utilization produced by the inclusion of *I*1. On the other hand, if *I*1 is not included, recursively compute an optimal schedule from the intervals *I*2 through *In*. Take the larger of these two as the final optimal solution.

A straightforward recursive implementation of this idea will lead to an exponential explosion in the number of subproblems solved repeatedly. You therefore need to convert the recursive algorithm to a dynamic-programming algorithm. Identify the bottom, solve small subproblems (and store the results in an array), and gradually solve larger problems using the solutions already computed, until you solve the problem for the entire collection *I*1, *I*2, ..., *In*. Your dynamic-programming implementation must run in O(*n* log *n*) time (including or excluding sorting).

In all the three parts, you should also print the (sub)optimal schedules that you get. In Part 3, you may use an extra array in order to facilitate the printing of the optimal schedule.

**Sample Output**

The following sample run illustrates that the greedy algorithms may fail to give optimal solutions for Mr. Sched's second problem. Indeed, a smaller number of intervals may lead to better utilization. Doing a lot of things does not necessarily mean that you keep yourself busy for the maximum amount of time.

n = 20

[ 485, 488] [ 960,1048] [ 83, 161] [ 213, 297] [ 757, 814] [ 388, 410]

[ 658, 721] [ 105, 183] [ 119, 196] [ 497, 556] [ 229, 269] [ 82, 97]

[ 221, 265] [ 315, 408] [ 678, 681] [ 264, 280] [ 404, 430] [ 202, 242]

[ 954,1023] [ 374, 387]

+++ Schedule 1: Earliest finish time first

[ 82, 97] [ 83, 161] [ 105, 183] [ 119, 196] [ 202, 242] [ 221, 265]

[ 229, 269] [ 264, 280] [ 213, 297] [ 374, 387] [ 315, 408] [ 388, 410]

[ 404, 430] [ 485, 488] [ 497, 556] [ 678, 681] [ 658, 721] [ 757, 814]

[ 954,1023] [ 960,1048]

--- Total number of intervals selected = 11

[ 82, 97] [ 105, 183] [ 202, 242] [ 264, 280] [ 374, 387] [ 388, 410]

[ 485, 488] [ 497, 556] [ 678, 681] [ 757, 814] [ 954,1023]

Total utilization = 375

+++ Schedule 2: Earliest start time first

[ 82, 97] [ 83, 161] [ 105, 183] [ 119, 196] [ 202, 242] [ 213, 297]

[ 221, 265] [ 229, 269] [ 264, 280] [ 315, 408] [ 374, 387] [ 388, 410]

[ 404, 430] [ 485, 488] [ 497, 556] [ 658, 721] [ 678, 681] [ 757, 814]

[ 954,1023] [ 960,1048]

--- Total number of intervals selected = 10

[ 82, 97] [ 105, 183] [ 202, 242] [ 264, 280] [ 315, 408] [ 485, 488]

[ 497, 556] [ 658, 721] [ 757, 814] [ 954,1023]

Total utilization = 493

+++ Schedule 3: Dynamic programming

The intervals are already sorted with respect to start time

Total utilization = 540 (computed by the d-p approach)

--- Total number of intervals selected = 9

[ 82, 97] [ 105, 183] [ 213, 297] [ 315, 408] [ 485, 488] [ 497, 556]

[ 658, 721] [ 757, 814] [ 960,1048]

Total utilization = 540 (computed from the optimal schedule)