Assignment 2

**Section 1: Pen-and-paper Exercises**

1. Consider the interval scheduling problem discussed in class. Design a greedy algorithm to solve the problem, and prove that the greedy solution is the optimal solution.

**Important: In all of the assignments of this course, when you are asked to give an**

**algorithm for a problem, you are (unless otherwise indicated) expected to**

**(i) describe the idea behind your algorithm in English (10 points);**

**(ii) provide pseudocode (10 points);**

**(iii) analyze its running time (10 points).**

**Regarding requirement (iii): Unless otherwise specified, show the steps of your analysis**

**and present your result using big-O.**

**For this particular problem, we ask you for a proof (10 points).**

Answer:

Idea:

Input: n lectures and their starting time and ending time {si, fi}

1. Sort lectures by ending time in ascending order, such that f1<=f2<=f3…<=fn.
2. Start with the first lecture, add lecture 1 into the classroom. Set lecture 1 as current lecture.
3. Remove all the lectures that are incompatible with the current lecture.
4. Add the lecture that is compatible with the current lecture and has the earliest ending time. Set the lecture added as the current lecture.
5. Repeat 3 and 4 until there are no lectures left.

Pseudocode:

Sort L={lectures}by fi in ascending order // Quicksort nlogn

Add lecture 1 into the classroom //1

Remove lecture 1 from L //1

Current = lecture 1 //1

While (L is not empty )

Remove all the lectures in L that are incompatible with current lecture // 1\*at most n-1

Add the first lecture that is compatible with current lecture into room, set it as current // 1\*at most n-1

End while

Running time: O(nlogn)

Proof:





1. Consider the segment covering problem discussed in class. Design a greedy algorithm to solve the problem, and prove that the greedy solution is the optimal solution.

Answer:

Idea:

Let s1 = [l1; ri] be a segment that covers the start point, 0, and has the largest r value among all segments covering 0. We then choose a segment s2 = [l2; r2] that covers r1 and has the largest r value among all segments. We continue this process until the entire segment [0;M].

Pseudocode:

Sort segments by left end point in ascending order // Quicksort O(nlogn)

currentpoint = 0; //1

while(currentpoint < M)

find all the segments that cover currentpoint //1\*at most n

among all the segments that cover currentpoint, pick the one that has the largest right end point, output this segment, let currentpoint = its right end point //1\*at most n

endwhile

Running time: O(nlogn)

Proof:

In order to prove the correctness of the greedy algorithm, we make the following claim: There exists an optimal solution that include segment s1 above. If this claim is true, we can pick s1 in an optimal solution, modify the input space from [0;M] to [r1;M], and try to solve the problem. The resulting solution would then be exactly same as the result obtained by our greedy algorithm. So we only need to prove this claim.Let O be an optimal solution that does not include s1, and let sk = [lk, rk] be a segment in O that covers point 0. We then note that rk <= r1 by the choice of s1 in our greedy solution.Let O' be defined as O' = O - {sk} + {s1}, which clearly covers the entire segment [0,M] and |O| = |O'|, hence, an optimal solution. The claim is thus proved.

**Section 2: Java Implementation**

1. Implement the Interval Scheduling Problem in Java.

Note:

Find a file called IntervalScheduling.java in assignment 2 folder.

Complete the method of greedyscheduling().

Test your method in the main method provided following the comments.

**Important: In all of the assignments of this course, when you are asked to implement an**

**algorithm for a problem, your code will be evaluated based on:**

**5 points - Execution**

**Each file must run without error or warning on valid input described in the main method provided.**

**5 points - Within Code Documentation**

**Is the code documented for obvious understanding of the use, preconditions, and postconditions of each function?**

**20 points - Correctness**

**Is the algorithm implemented correctly? Does your method pass the test?**

1. Implement the Coin Changing Problem in Java.

Note:

Find a file called Coinchange.java in assignment 2 folder.

Complete the method of greedycoinchange().

Test your method in the main method provided following the comments.

1. Implement the Fractional Knapsack Problem in Java.

Note:

Find a file called Fractionalknapsack.java in assignment 2 folder.

Complete the method of greedyfractionalknapsack().

Test your method in the main method provided following the comments

**TURN-IN CHECKLIST:**

1. **Answers to Section 1 (.doc/.txt/.pdf), and to Section 2 (all your source Code (.java files)). Remember to include your name, the date, and the course number in comments near the beginning of your code/report.**
2. **Create a folder and name it 'FirstName\_LastName\_assignment\_2'. In the newly created folder copy and paste your files (.doc/.txt/.java files). Then compress the folder, and push it to iLearn.**