Josh Huff, huffj@oregonstate.edu
Professor Schutfort
CS 325 – Winter 2018
February 4, 2018

Homework 4

---

1. Class Scheduling

Verbally describe an efficient greedy algorithm.

```
Start with the set of classes, C, sorted so that the start times, $S_x$, are in nondecreasing order.
Set count of lecture halls (set L) in use, L_count, to zero.
While C is not empty, performing the remaining steps in a loop:
    Pop the smallest value of S.
    For each hall from L[0] up to L[L_count]:
        If the last assigned’s finish time is less than or equals the classes’s start time, it is vacant:
            Assign the class to it.
            Move on to the next element in C.
    If no lecture halls were available, increment L_count, open a new hall and assign the class to it.
```

What is the running time of your algorithm?

This algorithm runs in Θ(n lg n) if the classes are not already sorted.
If the input is already sorted, the algorithm runs in linear time.

2. Road Trip

Verbally describe an efficient greedy algorithm.

```
Sort the distances in nondecreasing order.
Set the starting point to 0 (this will hold the index of the last hotel visited).
Set count of number of days of travel, num_days, to one.
Set the number of miles you can drive in a day, m, to some integral value
Performing the remaining steps in an infinite loop (which is terminated by the if statement below):
    If m equals or exceeds the last value in distances[], you are finished.
    Find the index, i, in distances that exceeds m.
    The hotel indexed at d[i - 1] is the one chosen for that night.
    Subtract the value at d[i - 1] from every distance indexed from d[i] to d[n]
    Set the hotel at d[i - 1] as the new starting point.
    Increment the num_days counter.
```

What is the running time of your algorithm?

This algorithm runs in Θ(n lg n) if the distances are not already sorted. Because the input (the set of distances from the starting point) is divided by some constant number of miles, the algorithm itself is in Θ(n).

3. Scheduling jobs with penalties

Verbally describe an efficient greedy algorithm.

Minimizing penalties is our priority. Our next focus is maximizing the number of jobs completed before the deadlines at that level are reached.

```
Sort the jobs in decreasing order by penalty.
Set the current penalty level, p_level, to the value at the first index.
For each job with the same penalty level:
    Add the job with the lowest deadline value to the job queue.
Set p_level to the next highest value and repeat the above loop
```

When all p_levels have been exhausted, the job queue will contain all jobs, sorted by deadlines within each penalty level. The first job will have the lowest deadline of the highest penalty, and the last job will have the highest deadline of the lowest penalty.

What is the running time of your algorithm?

This algorithm runs in $\Theta(n^2)$. The cost of sorting is dominated by the algorithm's need to iterate through the input in a nested loop of penalties and deadlines.

4. CLRS 16-1-2 Activity Selection Last-to-Start

Describe how this approach is a greedy algorithm

The only difference between the algorithms is the order the candidate solutions are considered. This approach is still a greedy algorithm because it makes the locally optimal choice with the expectation that doing so will create a smaller, similar subproblem and eventually produce a globally optimal result.

Prove that it yields an optimal solution

(Taken directly from CLRS – Theorem 16.1, with important distinctions in red font)

Let A_k be a maximum-size subset of mutually compatible activities in S_k, and let a_j be the activity in A_k with the latest start time.

If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k. If $a_j \neq a_m$, let the set $A'_k = A_k - \{A_j\} \cup \{a_m\}$ be A_k but substituting a_m for a_j.

The activities in A'_k are disjoint, which follows because the activities in A_k are disjoint, a_j is the last activity in A_k to start, and $f_m \leq s_j$.

Since $|A'_k| = |A_k|$, we conclude that A'_k is a maximum-size subset of mutually compatible activities of S_k , and it includes a_m.

5. Activity Selection Last-to-Start Implementation

Verbally describe an efficient greedy algorithm.

This algorithm functions similarly to first-to-start, except the input is sorted in reverse, and the finish times must be less than or equal to the start time of the last added activity to become part of the optimal solution.

Pseudocode

(Adapted from CLRS pseudocode for first-to-start, with important distinction in red font)

```
n = len(start_times)
solution[ ] = activities[0]
last_add = 0
for “current” in range from 1 to n:
    if finish_times[current] <= start_times[last_add]:
        solution += activities[current]
        last_add = current
return solution
```

What is the running time of your algorithm?

This algorithm runs in Θ(n). It need only consider each element in the input once.