This homework is due Thursday, October 29, at the start of class. The questions are drawn from the material in the lectures and Chapters 16 and 17 of the text on greedy algorithms and amortized analysis.

The homework is worth a total of 100 points. When questions with several parts do not specify the points for each part, each part has equal weight.

For questions that ask you to design a greedy algorithm, prove that your algorithm is correct making use of a lemma of the following form:

**Lemma** If a partial solution $P$ is contained in an optimal solution, then the greedy augmentation of $P$ is still contained in an optimal solution.

Prove the lemma using an exchange-style argument, where you transform the optimal solution to contain the augmentation and argue that this transformation does not worsen the solution value.

Remember to write on just one side of a page, do not use scrap paper, put your answers in the correct order, and staple your pages together. If you can’t solve a problem, state this, and write only what you know to be correct. Neatness and conciseness count.

(1) **(Continuous knapsack)** (10 points) Prove that the greedy algorithm for the Continuous Knapsack Problem from Section 16.2 of the text is correct, using a greedy augmentation lemma of the form stated above.

(2) **(Room scheduling)** (20 points) Suppose you have $n$ classes that you want to schedule in rooms. Each class has a fixed time interval at which it is offered, and classes whose times overlap cannot be scheduled in the same room. There are enough rooms to schedule all the classes.

Design a greedy algorithm to find an assignment of classes to rooms that minimizes the total number of rooms used, in $O(n \log n)$ time. Prove that your algorithm finds an optimal assignment using a lemma of the required form.

(3) **(Minimizing average completion-time)** (20 points) Suppose you are given a collection of $n$ tasks that need to be scheduled. With each task, you are given its duration. Specifically, task $i$ takes $t_i$ units of time to execute, and can be started at any time. At any moment, only one task can be scheduled.

The problem is to determine how to schedule the tasks so as to minimize their average completion-time. More precisely, if $c_i$ is the time at which task $i$ completes in a particular schedule, the average completion-time for the schedule is $\frac{1}{n} \sum_{1 \leq i \leq n} c_i$.

(a) (20 points) Design an efficient greedy algorithm that, given the task durations $t_1, t_2, \ldots, t_n$, finds a schedule that minimizes the average completion-time, assuming that once a task is started it must be run to completion.

Analyze the running time of your algorithm, and prove that your algorithm is correct using a lemma of the required form.

(b) (bonus) (10 points) Suppose with each task we also have a release time $r_i$, and that a task may not be started before its release time. Furthermore, tasks may be preempted, in that a scheduled task can be interrupted and later resumed, and this can happen repeatedly.

Design an algorithm that finds a schedule that minimizes the average completion-time in this new situation. Analyze its running time and prove that it is correct.
(4) **Simulating a queue using stacks** (10 points) Show how to implement the queue data structure by using two stacks, so that the amortized time for queue operations in the stack-based implementation matches their worst case time in a standard queue implementation. More specifically, show how to implement the operations

- Put\( (x, Q) \), which adds element \( x \) to the rear of queue \( Q \), and
- Get\( (Q) \), which removes the element \( x \) on the front of queue \( Q \) and returns \( x \),

so that both operations run in \( O(1) \) amortized time. Use the potential function method for your analysis.

(5) **Deleting the larger half** (10 points) Design a data structure that supports the following two operations on a set \( S \) of integers:

- Insert\( (x, S) \), which inserts element \( x \) into set \( S \), and
- DeleteLargerHalf\( (S) \), which deletes the largest \( \lceil |S|/2 \rceil \) elements from \( S \).

Show how to implement this data structure so both operations take \( O(1) \) amortized time.
(Note: You may use the accounting method or the potential method for your analysis.)

(6) **Amortized search trees** (30 points) For a binary search tree \( T \) and a node \( x \) in \( T \), let

- \( s(x) \) be the size of the subtree rooted at \( x \),
- \( \ell(x) \) be the left child of \( x \), and
- \( r(x) \) be the right child of \( x \).

Given a constant \( \alpha \), where \( \frac{1}{2} \leq \alpha < 1 \), a binary tree \( T \) is said to be \( \alpha \)-balanced if at every node \( x \) of \( T \), both of the following hold:

\[
\begin{align*}
    s\left(\ell(x)\right) & \leq \alpha s(x), \\
    s\left(r(x)\right) & \leq \alpha s(x).
\end{align*}
\]

(a) (5 points) Show that an arbitrary \( n \)-node tree can be made \( \frac{1}{2} \)-balanced in \( \Theta(n) \) time using \( \Theta(n) \) space.

(b) (5 points) Show that performing a Find operation in an \( n \)-node \( \alpha \)-balanced binary search tree takes \( O(\log n) \) worstcase time.

(c) (10 points) Consider the following amortized approach for supporting the Insert and Delete operations on a search tree. Suppose Insert and Delete are implemented in the standard way in an ordinary search tree that is not balanced, except that now after an Insert or Delete, the tree is rebalanced in the following way. After the Insert or Delete, find the highest node \( x \) in the tree that is not \( \alpha \)-balanced, and rebuild the subtree rooted at \( x \) so it becomes \( \frac{1}{2} \)-balanced, using your solution to Part (a). We call this task of rebuilding the subtree at \( x \) in this way, rebalancing the tree.

Prove that rebalancing an \( n \)-node \( \alpha \)-balanced tree, where \( \alpha > \frac{1}{2} \), takes \( O(1) \) amortized time.

To prove this, use the potential method with the following potential function \( \Phi(T) \).

For a node \( x \) in \( T \), let

\[
d(x) := \left| s(\ell(x)) - s(r(x)) \right|.
\]
Then
\[ \Phi(T) := \frac{1}{2\alpha - 1} \sum_{x \in T} d(x). \]

(d) (10 points) Using your answer to Part (c), show that an Insert or a Delete on an \( n \)-node \( \alpha \)-balanced tree, where \( \alpha > \frac{1}{2} \), takes \( O(\log n) \) amortized time.

Note that Problem (3)(b) is a bonus question. It is not required, and its points are not added to regular points.