

<http://www.cs.arizona.edu/classes/cs345/spring08/>

Homework #4

(70 points)

Due Date: May 6th, 2008, at the beginning of class

Write complete, legible answers to each of the following questions. Show your work, when appropriate, for possible partial credit. **This is not a group project; do your own work!** The TAs will go over the homework problems with you during the final exam review session. *If you need help, remember that the TAs and I have office hours for just this eventuality, and you may post general questions to the class newsgroup.*

On the due date, at the start of class, turn in your neatly written (or, better yet, electronically-formatted) answers and turn them in to the TAs. Solutions submitted after the first five minutes of class on the due date will not be accepted. Want to be safe and submit your homework early? Feel free to do so; you can give it to me or to either TA during our office hours. I strongly encourage you to retain a copy of your submitted answers, just in case you need to refer to them before you get your graded homework back.

- [5 points] In general terms, where can the smallest value be stored within a *max* heap? Assume that the smallest value is unique.
- [5 points] R-2.17
- [5 points] C-2.10
- [5 points] Answer C-2.19, assuming that T is a 2-3-4-5 tree.
- [5 points] C-2.32
- [10 points] Insert the sequence of keys (4, 15, 21, 44, 2, 17, 49, 11, 1), in the order given, into each of these structures. Show each structure after (i) the insertion of 21, (ii) of 17, and (iii) of 1.
 - (5) ... a 2-3 tree, using bottom-up splitting (Note: a 2-3 tree is not the same as a 2-3-4-5 tree.)
 - (5) ... a max heap
- [20 points] Consider a sequence s of 2D Cartesian points, where $s = P_1, P_2, \dots, P_n$, and where $P_k = (x_k, y_k)$. We would like an efficient way to determine which two points are closest.
 - (5) In pseudocode, provide a 'brute-force' algorithm that solves this problem in $O(n^2)$ time. Include your analysis that shows that your solution is, in fact, $O(n^2)$.
 - (10) Devise, and express in pseudo-code, a divide-and-conquer algorithm that solves this problem in $O(n \log_2 n)$ time. Again, explain how you know your solution is that efficient. *HINT*: Think about the recursive solution to the maximum subsequence sum problem from earlier in the semester. The code can be found in the T02n06.java example on the class web page.
 - (5) Can this problem be solved with a greedy algorithm? If so, state your algorithm in pseudo-code, and explain how you know it will always produce the best answer. If not, explain why a greedy approach won't work for this problem.
- [15 points] In class I used the classic change-making algorithm as an example of a greedy algorithm. This question refers to that example.
 - (5) Consider adding a 32-cent piece to the existing U.S. currency. Will the algorithm always generate optimal solutions with this addition? Explain your answer.
 - (10) I showed in class that adding a 12-cent piece to the U.S. currency collection causes the greedy change-making algorithm to fail to produce change using the fewest coins in all situations. In pseudocode, write a dynamic programming algorithm that always gives the correct optimal solution when 12-cent pieces are available, *without* just precomputing the whole table of possible change amounts. Briefly explain how your algorithm works. *HINT*: When making 16 cents change, with which coin(s) could you start?