CS 660, HOMEWORK 2 (DUE: SUNDAY SEPTEMBER 15, 11:59 PM)

INSTRUCTOR: HOA VU

Each question is worth 25 points. The extra credit question is worth 10 points.

When you are asked to design an algorithm, do the following: a) describe the algorithm, b) explain (or more rigorously prove) why it is correct, and c) provide the running time.

Unless specified otherwise, the problems are from the textbook by Jeff Erickson that we use.

For dynamic programming questions. Make sure a) you define the dynamic programming table, b) what is the recursive structure (how to fill an entry of the table), c) boundary cases & the filling order, d) how to get the answer from the table, and e) state the running time.

(1) **Question 1:** Problem 5a page 125.
(2) **Question 2:** Problem 42a page 148.
(3) **Question 3:** Consider the following game. A dealer produces a sequence of cards $s_1, \ldots, s_n$ face up where each card s_i has value v_i. Then two players, Alice and Bob, take turns picking a card from the sequence, but can only pick the first or the last card of the (remaining) sequence. Alice and Bob are each trying to maximize the total value of the cards they collect. Assume n is even and that Alice goes first.

    Give an $O(n^2)$ time algorithm to compute an optimal strategy for Alice. Given the initial sequence, your algorithm should precompute in $O(n^2)$ time some information. Then Alice should be able to make each move optimally in $O(1)$ time by looking up the precomputed information.

    Hint: Define $S[i, j]$ to be the maximum value that Alice can get if the remaining sequence is $s_i, s_{i+1}, \ldots, s_j$.
(4) **Question 4:** Problem 3 page 178.
(5) **Extra credit:** In the last homework, we try to solve the majority element problem in an array A of integers using Quick Select. Suppose there is no total ordering on the element of A (for example, A can be an array of fruits: $A[1] = apple, A[2] = banana, A[3] = durian, A[4] = apple$). Now, we can only check if $A[i] = A[j]$ but there is no ordering such as "$A[i] < A[j]$". Design an algorithm that outputs the majority element if there is one and outputs "no majority element" otherwise for this case. For simplicity, you can assume n is a power of 2. Hint: compare $A[1]$ to $A[2]$, if they are the same keep one copy in B, then do the same to $A[3]$ and $A[4]$, and so on. Then recurse on B. First show that if A has a majority element, that

element also has to be the majority element in B (the converse may not be true). Design an algorithm based on that observation. Show that it runs in $O(n)$ time.