(Sol) HW05 Greedy or Dynamic Programming

Duplicate, edit and export as a pdf to submit on Canvas. No late submission is accepted.

1. (2 points) Show that the dynamic programming algorithm for the knapsack problem we discussed in class finds an optimal solution using the proof by induction.

   Given: n items, array v: $v[i]$ is the value of item i, array w: $w[i]$ is the weight of item i, and the weight limit W.

   $$OPT[i,w] = \begin{cases} 0 & i = 0 \\ OPT[i-1,w] & w[i] > w \\ max(OPT[i-1,w], OPT[i-1, w-w[i]] + v[i]) & otherwise \end{cases}$$

   To prove: $OPT[i,w]$ returns the maximum sum of values possible from packing a subset of i given items under the weight limit w when $i \geq 0$. ← (S)

   Base case: when $n = 0$, the maximum sum of values is 0, so (S) is true when $n = 0$.

   Assumption: when $i < n$, we assume that (S) is true. ← (A)

   Inductive case: when $i = n$, there are two possible cases: 1 $w[i] > w$ and $w[i] \leq w$.
   Case 1: If $w[i] > w$, then the item i cannot be packed for the given weight limit so the maximum sum of values possible comes from packing a subset of $i-1$ items. Thus, $OPT[i,w] = OPT[i-1,w]$.
   Case 2: if $w[i] \leq w$, then again there are two cases: the optimal solution includes the item i, or not.
   If the optimal solution includes the item i, then the maximum sum of values would be the maximum sum of values from items $1, 2, \cdots, i-1$ under the weight limit of $w - w[i]$ plus $v[i]$. By (A), the maximum sum of values from items $1, 2, \cdots, i-1$ under the weight limit of $w - w[i]$ is stored in $OPT[i-1, w-w[i]]$, so if the optimal solution includes the item n, then $OPT[i-1, w-w[i]] + v[i]$ is the maximum sum of values from i given items under the weight limit w.
   If the optimal solution does not include the item i, then the maximum sum of values from i items would be equal to the max sum of values from $i-1$ items. By (A), the max sum of values from $i-1$ items is stored in $OPT[i-1,w]$, thus $OPT[i,w] = OPT[i-1,w]$ is the max value of sum from i items.

In all cases, $OPT[i, w]$ stores the max sum of values from i items under the weight limit w.

2. (5 points) Leetcode 253. Given an array of meeting time intervals consisting of start and end times $[[s_1, e_1], [s_2, e_2], \cdots, [s_n, e_n]] (s_i < e_i)$ for all i, find the minimum number of conference rooms required.) Note: (0,8) and (8,10) do not conflict at 8. Submit your code to GitHub.
    a. (1 point) Is your solution a Greedy algorithm or Dynamic Programming? Explain your answer.
    b. (2 points) What is the time complexity of your code in GitHub, in terms of n? Explain your answer.
    c. (2 points) What is the space complexity of your code in GitHub, in terms of n? Explain your answer.
3. (8 points) We have n students to be partitioned to two teams for a friendly game match. Student i's game skill is already quantified as $s[i] > 0$. Everyone must belong to exactly one group (say, either A or B). In addition, you want to minimize the difference between the sum of $s[i]$'s of each team.

    For instance, suppose $n = 4$ and $s = [1, 2, 2, 3]$. Students 1 and 4 can be in group A and 2 and 3 can be in group B. The difference would be |(1 + 3) − (2 + 2)| = 0 (this is optimal).

    Another example: suppose $n = 5$ and $s = [1, 2, 2, 3, 3]$. Students 1, 2, and 5 can be in group A (sum = 1+2+3 = 6) and 3 and 4 can be in group B (sum=2+3=5). The difference would be 1 (this is optimal).

    Yet another example: suppose $n = 4$ and $s = [1, 2, 4, 100]$. The optimal solution is to have student 4 alone in one group, and the rest in the other group – the difference would be 100 − (1 + 2 + 4) = 93.

    Assume $2 \leq n \leq 1000$ and $1 \leq s[i] \leq 4,000$.
    a. (2 points) Alice tried to solve this problem using a greedy algorithm. She considered the students by their skill levels in a descending order. i.e. from the end of the array. She compared the differences between placing each student in group A vs. in group B, and put the student in the group that results in a smaller difference. For example, suppose $n = 4$ and $s = [1, 2, 2, 3]$. The groups are decided as following:
    $[3], []$

$[3], [2]$
$[3], [2, 2]$ because $[3, 2][2]$ has a greater difference.
$[3, 1], [2, 2]$(this is optimal)

Write the pseudocode of this greedy algorithm.

```
public static List partition(int[] s) {
        Arrays.sort(s);
        ArrayList a = new ArrayList();
        ArrayList b = new ArrayList();
        a.add(s[s.length-1]);
        int sumA = s[s.length-1];
        int sumB = 0;
        for (int i = s.length-2; i>=0; i--) {
            int d1 = Math.abs(sumA+s[i]-sumB);
            int d2 = Math.abs(sumA-s[i]-sumB);
            if (d1 > d2) {
                b.add(s[i]);
                sumB += s[i];
            } else {
                a.add(s[i]);
                sumA += s[i];
            }
        }
        return a;
    }
```

b. (2 points) Analyze the time complexity of the greedy algorithm above.
Answer: sorting takes $O(n \log n)$. The for-loop runs $n - 1$ times, and each iteration takes a constant amount of work, so the for-loop takes $O(n)$. Overall, it takes $O(n \log n)$.

c. (2 points) Solve this problem using dynamic programming. Write the pseudocode.
Answer: For every student, we have two choices: include them in A or not include them in A (=include them in B). Consider the example of $n = 4$ and $s = [1, 2, 2, 3]$. To minimize the difference, we can try to make the sum from group A to be as close as possible to 4. If we put student 4 to group A, then the group A's sum becomes 3, so the target for the rest of $s = [1, 2, 2]$ changes to 1. If we put student 4 to group B, then the target for group A's sum stays as 4 for the rest of $s = [1, 2, 2]$.
Without losing generality, we can assume the sum of students' skill values in A is less than or equal to that in B. The optimal case is when the sum from A is as close as possible to the half of total of all skill values. OPT returns the sum from A in the optimal case.

```
static List partition(int[] s) {
        int target = 0;
        for (int i: s) {
            target += i;
        }
        target = target/2;

        int[][] OPT = new int[s.length+1][target+1];
        Arrays.fill(OPT[0], 0);
        for (int i=1; i<=s.length; i++) {
            for (int j=0; j<=target; j++)
            if (s[i-1] > j) {
                OPT[i][j] = OPT[i-1][j];
            } else {
                OPT[i][j] = Math.max(OPT[i-1][j-s[i-1]]+s[i-1], OPT[i-1][j]);
            }
        }
        return getSet(s, OPT);
    }

static List getSet(int s[], int[][] OPT) {
        ArrayList a = new ArrayList();
        int i = OPT.length - 1;
        int j = OPT[0].length - 1;
        while (j>=0 && i > 0) {
            if (OPT[i][j]!=OPT[i-1][j]) {
                a.add(s[i-1]);
                j -= s[i-1];
            }
            i = i-1;
        }
        return a;
    }
```

d. (2 points) Analyze the time complexity of the dynamic programming solution.

   Answer: Each element takes the max of two values, so the time complexity of filling one element in OPT is $\Theta(1)$. Let n be the number of students and $total$ to be the sum of all students' skill values. The size of OPT is $(n+1) * (total/2+1)$, so the time complexity is $O(n * total)$.

e. (Extra credit: 2 points) Show that the dynamic programming solution finds an optimal solution using proof by induction.

f. (Extra credit: 2 points) Find a counterexample to show that the greedy algorithm doesn't find an optimal solution
   Answer: {9, 8, 8, 7, 7, 6, 4, 4, 3}. Greedy algorithm splits them into {9, 7, 7, 4} and {8, 8, 6, 4, 3} (diff 2) while the optimal solution is {9, 8, 8, 3} and {7, 7, 6, 4, 4} (diff 0).

Note: Extra credits are capped at 2. You can get 1 points from each, 2 points from (e), or any other combination.