Assignment 10

csci2200, Algorithms

Honor code: *Work on this assignment alone or with one partner. Between different teams, collaboration is at level 1 [verbal collaboration only]. There are lots of resources online, such as animations, visualizations, practice problems, videos, and solutions— which you are encouraged to explore to deepen your understanding. However, you must be careful not to search for the specific problems in the assignment with the intent of getting hints for the solution. Searching for the assignment problems on the internet violates academic honesty for this class.*

---

1. **Finding the singleton:** You are given a sorted array of numbers where every value except one appears exactly twice, and one value appears only once. Design an efficient algorithm for finding which value appears only once.

   Note: A general solution should not assume anything about the numbers in the array; specifically, they may not be in a small range, and may not be consecutive.

   Example: Here are some example inputs to the problem:

   $$1, 1, 2, 2, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8$$

   $$10, 10, 17, 17, 18, 18, 19, 19, 21, 21, 23$$

   $$1, 3, 3, 5, 5, 7, 7, 8, 8, 9, 9, 10, 10$$

   We expect: pseudocode, a clear (and brief) English description of what the algorithm is doing and why it is correct; running time analysis.

2. **Art gallery guarding:** In the *art gallery guarding* problem we are given a line L that represents a long hallway in an art gallery. We are also given a set $X = \{x_0, x_1, x_2, ..., x_{n-1}\}$ of real numbers that specify the positions pf paintings in this hallway; assume that each painting is a point. Suppose that a single guard can protect all the paintings within a distance at most 1 of his or her position, on both sides.

   Design an algorithm for finding a placement of guards that uses the minimum number of guards to guard all the paintings in X. You can assume that the positions of the paintings are sorted. Briefly argue why your algorithm is correct and analyze its running time.

We expect: The algorithm as pseudocode, a brief explanation, analysis and justification of correctness i.e. why does it always assure the minimum nb of guards? Remember that to show that a greedy algorithms is correct it is sufficient to show that there exists an optimal solution that includes the first greedy choice.)

3. **Unbounded knapsack:** We have n items, each with a value and a positive weight; Item i has value v_i and weight w_i. We have a knapsack that holds maximum weight W. In this problem you'll consider a variation of the $0-1$ Knapsack problem, where we have *infinite copies* of each item. Describe an algorithm that, given $W, \{w_1, ..., w_n\}$ and $\{v_1, ..., v_n\}$, finds the maximal value that can be loaded into the knapsack.

   The sequence of steps below will guide you towards the solution.

   (a) A friend proposes the following greedy algorithm: start with the item with the largest cost-per-pound, and pick as many copies as fit in the backpack. Repeat.

   Show your friend this is not correct by coming up with a counter-example.

   We expect: a small example, the greedy solution, and the optimal solution

   (b) Define your sub-problem and state clearly what its argument(s) represent and what it returns.

   Hint: With the classical knapsack, once you take an item, you cannot take it again, so you have fewer items to choose from. So the subproblem we use has to keep track of both the knapsack size as well as which items are allowed in the knapsack. We used $knapsack(w, i)$ to be the maximum value we can get for a knapsack of weight w considering items 1 through i. For the unbounded knapsack, we have infinite supplies of each item. So we don't need to keep track of which items we already included (only of the size of the knapsack).

   (c) Argue optimal sub-structure:

   Hint: Suppose the optimal solution contains item i. Then the remaining part of the optimal solution must be an optimal solution for .... .

   (d) Come up with a recursive definition for the subproblem you chose in (b)

   Hint: Consider all the choices, and pick the best.

   (e) Develop a DP solution — either top-down or bottom up. What is the running time of your algorithm ?