Problem Set #3: CMPSC 463

1. (20 points) You have a large collection of images, image$_1$, image$_2$, …, image$_n$, that you wish to store on 100GB USB flash drives. You are given an array `size`, where `size[i]` is the size of the image$_i$ in GB. You may reorder the images as you write them to the drives. You are using a greedy strategy that fills one drive at a time, writing the largest unwritten image to the current drive. When there are no more images that will fit on the current drive, you start filling the next one.

   If the goal is to use as few flash drives as possible, is this algorithm correct? You must provide a justification for your answer to receive credit.

   No it is not correct.

   Suppose that size = [90, 50, 42, 8, 6, 4].

   The given algorithm would use three drives: putting the 90GB and 8 GB files on the first drive; the 50 GB, 42 GB and 6 GB files on the second drive; and the 4 GB file on the last drive.

   This is not optimal because on could use 2 drives by putting the 90 GB, 6 GB, and 4 GB files on one drive and the 50 GB, 42 GB, and 8 GB files on the second.

2. (40 points) You have a large collection of images, image_1, image_2, …, image_n, that you wish to store on 100GB USB flash drives. You are given an array `size`, where `size[i]` is the size of the image_i in GB. The goal is to use as few flash drives as possible, while storing the images in order. In other words, image_1 through image_i must be stored on drive 1, image_{i+1} through image_j on drive 2, image_{j+1} through image_k on drive 3, etc.

   a. Provide an efficient and correct algorithm to allocate images to drives.

      Idea: We write the images to the drives in order. We write out as many as will fit on the current drive, and then we move to the next drive.

```
writeDrives(size[1..n])
    drive_ndx = 1
    space = 0
    image_ndx = 1
    while image_ndx <= n
        if space + size[image_ndx] > 100 then
            drive_ndx = drive_ndx + 1
            space = 0
        end if
        space = space + size[image_ndx]
        write image image_ndx to drive drive_ndx
        image_ndx = image_ndx + 1
    end while
end writeDrives
```

   b. What is the time complexity of your algorithm?

      All the lines, with the exception of the loop run in $\theta(1)$ time. The while loop executes exactly n times. Thus:

$$T_w(n) = \theta(1) + n\,\theta(1) = \theta(n)$$

c. Prove that your algorithm is correct.

First note that our algorithm terminates (based on our running time analysis), and thus we need only to prove partial correctness.

Using an exchange argument:

Assume that the algorithm is incorrect and it produces a suboptimal assignment X, while the optimal assignment is X*. In other words X* uses fewer drives than X.

We will transform X* into X using the following approach:

> Let *i* be the index of the first image written to a different drive in X and X*. If the image is written to drive *j* in X*, it must be written to *j*-1 in X, because our algorithm would not have written it to drive *j*+1 if it fit on drive *j*. We can move image$_i$ to drive *j*-1 in X*, and now the two solutions agree on the first i images.
>
> We repeat this step until X* has been transformed into X. Since all transformations moved images to lower numbered drives, X cannot use more drives than X*.

We arrive our assumption that our algorithm was incorrect leads us to a contradiction, and thus our algorithm is correct.

3. (40 points) In the game *Angry Dogs*, you launch dogs at cars that are arranged in a line. You get one point for every car within 100 feet where the dog lands. Suppose that you are given an array `car[1..n]` with car locations (given in feet).

   a. Write an efficient and correct algorithm that minimizes the number of dogs launched while achieving the maximum score.

   Idea: We will sort the cars, and then place the dogs sequentially, locating each dog at as large a position as possible while covering the uncovered car with the smallest location value.

```
angryDogs(car[1..n])
    sort the array car in ascending order
    lastDog = -infinity
    for i = 1 to n
        if car[i] > lastDog + 100 then
            Target car[i] + 100 with next dog
            lastDog = car[i] + 100
        end if
    end for
 end AngryDogs
```

   b. What is the time complexity of your algorithm?

   We can use a θ(*n* log *n*) algorithm to sort.

   The remaining lines, with the exception of the loop run in θ(1) time.

   The loop executes *n* times.

   Thus:

$$T_w(n) = \theta(n \log n) + n\,\theta(1) = \theta(n \log n + n) = \theta(n \log n)$$

c. Prove that your algorithm is correct.

We will describe the placement of cars and dogs on a number line. Our running time analysis shows that the algorithm terminates, so we need only to show partial correctness.

Using a greedy-stays-ahead approach:

Assume that our algorithm is incorrect.

Let $A =< a_1, a_2, \dots, a_i >$ be the locations of the dog landings specified by our algorithm in sorted order, and $O =< o_1, o_2, \dots, o_j >$ be the optimal locations of dog landings, also in sorted order. Note that by our assumption $i > j$.

Let $f(S, k)$ = number of points earned by the first k dogs in a solution *S*, ordered as above. We will show that our algorithm stays ahead of the optimal solution, by induction on the number of dogs.

Basis Step: Clearly $f(A, 1) \geq f(O, 1)$ because the location of the dog 1. It is placed as far to the right as possible while still covering the left-most car, and thus covers as many of the initial cars as possible.

Induction Hypothesis: Assume that $f(A, k) \geq f(O, k)$.

Inductive Step: Show that $f(A, k+1) \geq f(O, k+1)$.

Let's say that $f(A, k)$ cover up to car *m.* By our inductive hypothesis, no other solution covers past car *m* with the first *k* dogs. Our algorithm places the *k*+1th dog as far to the right as possible while still covering car *m*+1. We therefore cover as many additional cars past car m as possible, and, thus, $f(A, k+1) \geq f(O, k+1)$.

Having shown that greedy stays ahead, we now prove optimality:

$f(O, j) = n$ Since all valid solutions cover all cars

$f(A, j) \geq f(O, j)$, by our greedy-stays-ahead argument

$=> f(A, j) = n$

Thus our algorithm would have covered all *n* cars using $j < i$ dogs. This is a contradiction because our algorithm would not have terminated and not allocated additional dogs beyond the *j*th dog.

Thus, we conclude that our assumption was wrong, and our algorithm is therefore correct.