**Problem Set #3: CMPSC 463**

1. (20 points) You have a large collection of images, image1, image2, …, 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, the specified algorithm is not correct for the given problem.**

**Why:**

**Assume 6 images where:**

**Image 1 = 95 GB**

**Image 2 = 94 GB**

**Image 3 = 4 GB**

**Image 4 = 3 GB**

**Image 5 = 2 GB**

**Image 6 = 2 GB**

**Now if I go by the greedy approach, I will place 95 GB in my first flash drive, then look for the next highest image which can be fit in the remaining space.**

**In this case, I have 5 GB left and I fit 4 GB, which is the next highest image that will fit and place that in. Now when I move on to the next flash drive, I fit image 2, image 4, and image 5, i.e. 94 + 3 + 2 = 99 GB.**

**For my final image, I could require a new flash drive and, in this case, I’d need 3 flash drives, however, the optimal solution is 2 since:**

**flash drive 1 = 95 + 3 + 2 and flash drive 2 = 94 + 4 + 2**

**Bottom line, the greedy algorithm wouldn’t hold for the given problem.**

1. (40 points) You have a large collection of images, image1, image2, …, 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, image1 through image*i* must be stored on drive 1, imagei+1 through imagej on drive 2, imagej+1 through imagek on drive 3, etc.
   1. Provide an efficient and correct algorithm to allocate images to drives.
   2. What is the time complexity of your algorithm? Use asymptotic notation. Justify your answer.
   3. Prove that your algorithm is correct.
2. **Algorithm:**

**I have n images with me, each of size[i]. I now use a greedy approach because the images have to be stored in a contiguous manner, I start storing from the first image, until I hit the image size which exceeds the current flash drive and move it on the next flash drive.**

**Psuedocode:**

**Let size[] contain n images’ size**

**cur\_size = 0;**

**flash\_disks = 0;**

**for(i = 0; i < n; i++){**

**if(cur\_size + size[i] > 100){**

**flash\_disks++;**

**cur\_size = size[i];**

**}**

**else cur\_size += size[i];**

**}**

**flash\_disks++; // last disk containing images**

1. **Time Complexity: O(n), n is the number of images**
2. **Proof of Correctness:**

**The main reason why my greedy approach is the most optimal solution is because the images which I have to store should be stored contiguously. If not, then the greedy approach wouldn’t work (As seen in the first problem).**

**If the images are to be stored in contiguous fashion, then by going by my greed approach, I am optimizing the data stored in each flash drive. If I don’t store the ith image in the flash drive, then the next flash drive would have:**

**100 – size[i] of space to work with, which is obviously less optimal. Therefore, the greedy approach is the most optimal approach.**

1. (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).
   1. Write an efficient and correct algorithm that minimizes the number of dogs launched while achieving the maximum score.
   2. What is the time complexity of your algorithm? Use asymptotic notation. Justify your answer.
   3. Prove that your algorithm is correct.
2. **I’m going to use a greedy approach to this problem. I first sort the cars in ascending order and place the first dog 100 feet from the first car. Now, this would span + 100 from the placement of the dog, so I now throw the next dog at 100 feet from the next car which is not covered.**

**Pseudo code:**

**Let car[] contain n cars**

**First sort the cars in increasing order then remove duplicates(optional)**

**dog\_pos = car[0] + 100;**

**dogs = 1;**

**for (i = 2; i < n; i++){**

**if(abs(car[i] – dog\_pos) > 100){**

**dogs++;**

**dog\_pos = car[i] + 100;**

**}**

**}**

**return dogs;**

1. **Time Complexity: If cars are not sorted: O(n log n) if cars are sorted: O(n)**
2. **The solution is optimal because:**
3. **The dog ranges don’t overlap: The greedy solution is clearly non-overlapping and any solution which overlaps will be less efficient than the greedy one.**
4. **By placing the dogs at 100 feet + the last covered dog, we are efficiently solving sub-problems for each new dog placed.**

**Therefore, the solution is optimal.**