Problem 1: Road Trip: Suppose you are going on a road trip with friends. Unfortunately, your headlights are broken, so you can only drive in the daytime. Therefore, on any given day you can drive no more than d miles. You have a map with n different hotels and the distances from your start point to each hotel x1< x2< ... < xn. Your final destination is the last hotel.

a)

* Description

In order to minimize the number of days, the farthest hotel within the distance ‘d’ from the starting point(start point, or previous hotel) has to be visited for each day.

* Pseudo Code

1. Build heap for the hotels
2. Find the farthest hotel within distance ‘d’ from the starting point in the heap.
3. Set the starting point as the hotel found in step I.
4. Iterate step II ~ III until the last hotel is found in Step II.

b)

Running time for searching the hotel from the heap is , and building heap is .

* Best Case:

(The destination is distance ‘d’ away from the start point, )

* Average Case:

(running time: )

* Worst Case:

(Each hotel is distance ‘d’ away from each other, )

Problem 2:

* Explanation

This approach selects the latest starting time of the activities every time. After selecting the latest starting, this approach selects the next latest starting time of activity, and so on. Since this approach is keep selecting the best case in the aspect of last activity to start, this approach is a greedy algorithm.

* Proof

Let be an activity in (non-empty subset) with the latest starting time.

Let be a maximum-size subset of mutually compatible activities in , and let be the activity in with the latest start time. If , we are done, since we have shown that is in some maximum-size subset of mutually compatible activities of . If , let the set be but substituting for . The activities in are disjoint, which follows because the activities in are disjoint, is the last activity in to start, and . Since , we conclude that is a maximum-size subset of mutually compatible activities of , and it includes

Problem 3:

* Explanation

The program will first sort the activity timeline in descending order. By using greedy algorithm, the program will select the biggest start time, and then pop that activity from the array. The program will still iterate this process until there is no activity which fits in the time frame. Through these processes, the program finds maximum number of activities and the list from the activity array.

* Pseudo Code

1. Sort the activity start time with descending order (Merge Sort)
2. Put the first element of activity to the bag
3. M = 2 ~n(number of activities), I = 1
4. If start[M] is greater than or equal end[I], then put Mth element of activity to the bag, I <- M
5. Go to step IV until M equals to n(amount of activities)

* Sorting

I used merge sort to sort the starting time of activity in descending order. Theoretical running time of merge sort is .

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* Greedy algorithm: Last activity to Start

Last-to-start algorithm iterates until the variable m reaches n-1 so that theoretical running time of the algorithm is .

텍스트이(가) 표시된 사진

자동 생성된 설명

* Entire running time

The entire running time of the algorithm is which is equivalent to . Therefore, the theoretical entire running time of Activity Selection Last-to-Start is .