Problem II. By Ahmedul Kabir and Prof. Ruiz

a. Pseudocode for Offline Caching problem

```
FurthestInFuture(<r1, r2, … rn>, k):
    for i = 1 to n
        if ri is in cache:
           then print "Cache Hit"
        else
            print "Cache Miss"
            if cache is not full
                Add ri to cache
            else (if cache is full)
                // Find element in cache with furthest distance
                distfurthest = 0
                for j = 1 to k
                    p = i + 1 // p is an index on the list of requests
                    Keep increasing p until rp == cache[j]
                    if p reaches end of sequence without finding rp
                        then
                            distj = infinity
                            furthest = j
                            break from the inner for loop
                    else
                        distj = p - i
                    if distj > distfurthest
                        then
                            furthest = j
                            distfurthest = distj

                Evict Cache[furthest]
                Add ri to cache
```

Kabir's Python implementation of this pseudo-code is available at:
http://web.cs.wpi.edu/~cs2223/b13/HW/HW5/Solutions

Running time of the algorithm

The outer loop runs n times, and the inner loop k times. For each iteration of the inner loop, we may have to traverse (n – i) items at worst. So the running time is

$k\sum_{i=1}^{n}(n-i)$ = $\frac{kn(n-1)}{2}$ = O(kn^2).

b. **The off-line caching problem exhibits optimal substructure.**

A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems.

Let $R = \{r_1, r_2, \dots, r_n\}$ be a list of requests, and k the cache of size. Let also $E = \{e_1, e_2, \dots, e_n\}$ be a list of evictions, where each e_i is either "cache hit", if r_i is in the cache at time i, or the name of the cache element to be evicted at time i because of a cache miss occurred at that time. Let's assume that E is optimal for R and k. That is, E contains the minimum possible number of evictions (= cache misses) for the list of requests R and a cache of size k.

In order to show that the off-line caching problem exhibits optimal substructure , we need to show that any subsequence of $E, E_i = \{e_i, e_{i+1}, \dots, e_n\}$ is optimal for the sublist of requests $R_i = \{r_i, r_{i+1}, \dots, r_n\}$ of R starting with the cache content at time i. It suffices to show that this is true for i equal to the time when the first cache miss occurred, and then apply the same argument for other cache misses afterwards. So let $i, 1 \leq i \leq n$, be the first time when a cache miss occurred. Assume by way of contradiction that the solution $E_i = \{e_i, e_{i+1}, \dots, e_n\}$ is NOT optimal for the sublist of requests $R_i = \{r_i, r_{i+1}, \dots, r_n\}$ starting with the cache content at time i right when the cache miss was detected. Hence, there must be another solution $O = \{o_i, o_{i+1}, \dots, o_n\}$ which incurs in fewer cache misses on $R_i = \{r_i, r_{i+1}, \dots, r_n\}$ than E_i, starting with the same cache content that E_i does at time i. In that case, the list of evictions $New = \{e_1, e_2, \dots, e_{i-1,} o_i, o_{i+1}, \dots, o_n\}$ is a solution for R with fewer cache misses than E. This is a contradiction, as E is optimal for R. Hence, as claimed, E_i is optimal for R_i.

c. The furthest-in-future strategy produces the minimum possible number of cache misses. See a proof of this fact in the slides of Chapter 4 of Jon Kleinberg's and Éva Tardos' Algorithm Design textbook. Available at Prof. Kevin Wayne's Algorithms Course Webpage at Princeton University. See Greedy Algorithms Part I slides 34-43 at http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf .