1. (25 points) **(Offline Paging Problem)** In class, we discussed the paging problem, and presented a randomized online algorithm for it. We now solve the offline setting of the problem: here, we are given a collection of n pages, and k cache slots which are initially empty. Then we are also given a sequence of requests $p_1, p_2, \ldots, p_T$ ahead of time. The goal is figure out an eviction policy so that the number of *cache misses* is minimized, i.e., any time a page is requested and it is not in the cache at the time it was requested, it is a cache miss, and the page has to be brought into cache, while evicting some other page which already exists in cache. As mentioned, the goal is to figure out the optimal sequence of evictions to minimize the number of cache misses.

   (a) (5 points) Consider the FIFO policy: when a page is requested which is not in the current cache, evict the page which has been around in the cache for the longest (i.e., it arrived first among all pages in cache). Do you expect fewer cache misses when the cache gets larger and larger? Answer this by analyzing the number of cache misses for the following input sequence of page requests $3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4$ in two cases: (i) the cache has 3 slots, and (ii) the cache has 4 slots.

   Solution:

   Proof. $1 + 1 = 2$. □

   (b) (5 points) Also, how bad can FIFO be in terms of the performance? Is it an optimal algorithm? As a function of k, find a simple example where FIFO doesn't do well compared to the optimal eviction sequence.

   Solution:

   Proof. $1 + 1 = 2$. □

   (c) (10 points) Finally come up with the optimal eviction algorithm if the entire sequence is known, by relating it to a problem we studied in the mid-sem.

   Solution:

   Proof. $1 + 1 = 2$. □

2. (10 points) **(Doubling Trick in Algorithms: Do we need Promises?)** In class, we designed an online routing algorithm with the following guarantees: for a request sequence σ of sources and destinations, let L_σ be the minimum congestion (i.e., max load on any edge) with which all the requests can be routed. Then, we have an algorithm A (which if it *knows the value* of L_σ), finds a routing where every edge has congestion at most $CL_\sigma \log m$ for some constant C. Now we get around the assumption that the algorithm needs to know L_σ, by running the algorithm in phases: initially, in the first phase, start with a guess of $\hat{L}_\sigma = 1$ and run algorithm A; whenever the congestion in the *current phase* exceeds $C\hat{L}_\sigma \log m$, double the estimate of $\hat{L}_\sigma$, and start the next phase.

Show that, at any stage, the congestion of this algorithm is $O(\log m)L_\sigma$, where L_σ is the optimal congestion for the set of requests which have currently arrived until now.

Solution:

Proof. $1 + 1 = 2$. □