Name ______________________ Student No. ______________________

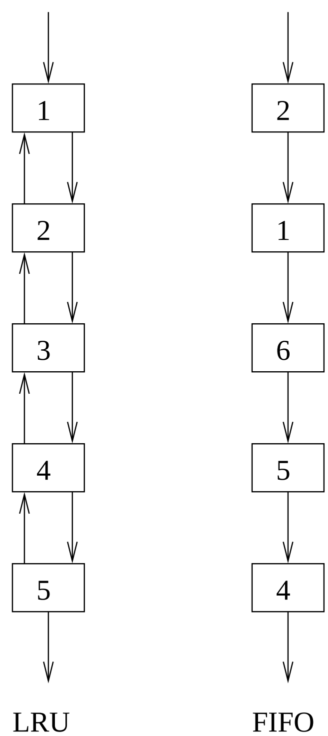
No aids allowed. Answer all questions on test paper. Use backs of sheets for scratch work.

Total Marks: 30

[10] 1. Consider the following sequence of requests:

$$1, 2, 3, 4, 5, 4, 6, 6, 2, 1, 5, 4, 3, 2, 1$$

Fill in the following two lists, with the contents of the cache, <u>after</u> the entire sequence of requests has been processed. The left doubly-linked list corresponds to the LRU page replacing algorithm; the right list corresponds to the FIFO page replacing algorithm.



How many faults were incurred with LRU? 8. How many with FIFO? 8.

2. Let k, i be positive integers, and let $\sigma_{k,i}$ be a sequence of requests consisting of i segments of the form: $1, 2, 3, \ldots, k, k+1$. For example, if $k = 2$ and $i = 4$, then:

$$\sigma_{2,4} = \underbrace{1, 2, 3}_{1}, \underbrace{1, 2, 3}_{2}, \underbrace{1, 2, 3}_{3}, \underbrace{1, 2, 3}_{4}$$

Assume that we have a cache of size k.

[3] (a) How many page faults does the FIFO page-replacing algorithm incur on $\sigma_{k,i}$? Explain briefly.

On the first segment the cache is empty, so the first k requests incur k faults. Then, $k+1$ incurs a fault, so 1 is evicted. On the second segment, we also incur $k+1$ faults, since we are always trailing by 1 the evictions. Thus, FIFO incurs $i \cdot (k+1)$ many faults on $\sigma_{k,i}$.

[3] (b) How many page faults does the OPT page-replacing algorithm incur on $\sigma_{k,i}$? Explain briefly.

On the first segment, the cache is empty, so OPT incurs k faults on the first k requests. Then, when $k+1$ comes along, OPT evicts k since it has the LFD (Longest-Forward-Distance). Thus, in the second segment, we do not incur faults on the first $k-1$ requests, then we incur a fault on k, so we evict $k-1$, and we do not incur a fault on $k+1$. Thus, on all the segments after the first one, OPT incurs just one fault, so on $\sigma_{k,i}$, OPT incurs $k+1+i-1 = k+i$ faults in total.

[4] (c) Is it possible for FIFO to be 2-competitive? Explain.

No, it is not possible for FIFO to be 2-competitive. From part (a) and (b), we see that:

$$\text{FIFO}(\sigma_{k,i}) = i \cdot (k+1)$$
$$\text{OPT}(\sigma_{k,i}) = k + i$$

Therefore,

$$\text{FIFO}(\sigma_{k,i}) = \left(\frac{i \cdot (k+1)}{k+i}\right) \cdot \text{OPT}(\sigma_{k,i})$$

If FIFO is 2-competitive, then it follows that there is an α, that can depend only on k, such that:

$$\text{FIFO}(\sigma) \leq 2 \cdot \text{OPT}(\sigma) + \alpha_k \quad (1)$$

But, for any fixed k, we can choose i big enough, so that (1) will not work with $\sigma = \sigma_{k,i}$.

[10] 3. The Linux OS disk paging policy implements a "Least-Frequently-Used" (LFU) replacing policy. That is, a page is evicted from the cache if it is the least frequently used page in the cache.

Give an *explicit* implementation of this algorithm. (Hint: associate an age counter with each page in the cache; show how to update the ages on a hit and on a fault.)

Let k be the size of the cache. Every slot i in the cache has an age counter associated with it, call it a_i. Suppose that a page p is requested. If p is not in the cache, then replace a page whose age counter has value $\min\{a_1, a_2, \ldots, a_k\}$, let the age of the slot containing p be 1, and decrese the age counters of all the other pages in the cache.

If p is in the cache, then increase the age counter of p by one, and decrease the age counters of all the other pages in the cache by 1, except if an age is zero, in which case it remains zero.

NOTE: it is necessary to decrease the ages of those pages in the cache that have not been requested. Otherwise, the paging wouldn't work correctly. To see this, suppose we process a sequence of requests, and for a long time we only request pages which are already in the cache, so all their ages increase to 15. And now, suppose that we request two new pages, p, q, as follows:

$$p, q, p, q, p, q, p, q, p, q, \ldots$$

Each time we evict p or q, and we never evict any of the pages which are already in the cache, even though the frequency of p, q grows, because the ages of p, q are always at most 1.

NOTE: it is not possible to implement this algorithm *implicitly* with a doubl- linked list, because there is no way to decrease the frequency of all the pages in a doubly-linked list.