```
'''
Demonstration of simple page replacement algorithms

FIFO
LRU
Optimal

Usage

    python paging.py [number of page frames]
'''

#for obtaining command line parameters
import sys

'''
Simple FIFO page replacement algorithm
'''
def FIFO(size, pages):
    SIZE = size
    count = 0
    memory = []
    faults = 0
    fifoIndex = 0

    for page in pages:
        if memory.count(page) == 0 and count < SIZE:
                                          # not present, but no replacement is necessary - page fault
                                          memory.append(page)
```

```
                      count += 1
                      faults += 1
    elif memory.count(page) == 0 and count == SIZE:
                      # memory is full - replace FIFO page - page fault
                      memory[fifoIndex] = page
                      fifoIndex = (fifoIndex + 1) % SIZE
                      faults += 1
    elif memory.count(page) > 0:
                      # present - do nothing
                      pass

    #print ' faults =',faults,'inserting',page,'memory = ',memory

  return faults

'''
LRU algorithm
'''
def LRU(size, pages):
  SIZE = size
  count = 0
  memory = []
  faults = 0

  for page in pages:
    if memory.count(page) == 0 and count < SIZE:
      # not present, but no replacement is necessary - page fault
      memory.append(page)
      count += 1
```

```
            faults += 1
        elif memory.count(page) == 0 and count == SIZE:
            # memory is full - replace LRU page - page fault
            # page at index 0 is LRU page
            memory.pop(0)
            memory.append(page)
            faults += 1
        elif memory.count(page) > 0:
            # page is present - reorder stack
            memory.remove(page)
            memory.append(page)

    return faults

'''
Optimal algorithm
This algorithm works by replacing the page that will not be used
for the longest period of time.
'''
def OPT(size,pages):
    SIZE = size
    count = 0
    memory = []
    faults = 0
    x = 0

    for page in pages:
        if memory.count(page) == 0 and count < SIZE:
            # not present, but no replacement is necessary - page fault
            memory.append(page)
            count += 1
            faults += 1
        elif memory.count(page) == 0 and count == SIZE:
            # memory is full - replace the page using optimal algorithm
            # iterate through the existing pages and determine where
            # they will be used again in the future.
            # Evict the page that will not be used for the longest period of time.
            future = -1
            for i in memory:
                # if this page won't be used again, evict it
                if pages[x:].count(i) == 0:
                    evictedPage = i
                    break
                else:
                    index = pages[x:].index(i)
                    if index > future:
                        future = index
                        evictedPage = i

            # evictedPage is the page to evict
            p = memory.index(evictedPage)
            memory.remove(evictedPage)
            memory.insert(p,page)
            faults += 1
        elif memory.count(page) > 0:
            # page is present - do nothing
            pass
```

```
        # adjust the index so it now indicates the next page in the string
        x += 1

    return faults



def main():
    pages = (7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1)

    size = int(sys.argv[1])

    print 'FIFO', FIFO(size,pages), 'page faults.'

    print 'LRU', LRU(size,pages), 'page faults.'

    print 'OPT', OPT(size,pages), 'page faults.'

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print 'Usage: python paging.py [number of pages]'
    else:
                main()
```