1. Generate random addresses with the following arguments: -s 0 -n 10, -s 1 -n 10, and -s 2 -n 10. Change the policy from FIFO, to LRU, to OPT. Compute whether each access in said address traces are hits or misses.

./paging-policy.py -s 0 -n 10

ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 8 Miss [8]
Access: 7 Miss [8, 7]
Access: 4 Miss[8, 7, 4]
Access: 2 Miss[7, 4, 2]
Access: 5 Miss[4, 2, 5]
Access: 4 Hit[4, 2, 5]
Access: 7 Miss[2, 5, 7]
Access: 3 Miss[5, 7, 3]
Access: 4 Miss[7, 3, 4]
Access: 5 Miss[3, 4, 5]

./paging-policy.py -s 0 -n 10 -p LRU

ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of LRU, and a cache of size 3 pages, figure out whether each of the following page references hit or miss in the page cache.

Access: 8 Miss[8]
Access: 7 Miss[8, 7]
Access: 4 Miss[8, 7, 4]
Access: 2 Miss[2, 7, 4]
Access: 5 Miss[2, 5, 4]
Access: 4 Hit[2, 5, 4]
Access: 7 Miss[7, 5, 4]
Access: 3 Miss[7, 3, 4]
Access: 4 Hit[7, 3, 4]
Access: 5 Miss[5, 3, 4]

./paging-policy.py -s 0 -n 10 -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of OPT, and a cache of size 3 pages, figure out whether each of the following page references hit or miss in the page cache.

Access: 8 Miss[8]
Access: 7 Miss[8,7]
Access: 4 Miss[8, 7, 4]
Access: 2 Miss[2, 7, 4]
Access: 5 Miss[5, 7, 4]
Access: 4 Hit[5, 7, 4]
Access: 7 Hit[5, 7, 4]
Access: 3 Miss[5, 3, 4]
Access: 4 Hit[5, 3, 4]
Access: 5 Hit[5, 3, 4]

```
./paging-policy.py -s 3 -n 10
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 3
ARG notrace False

Access: 2  MISS FirstIn ->          [2] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 5  MISS FirstIn ->       [2, 5] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 3  MISS FirstIn ->    [2, 5, 3] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 6  MISS FirstIn ->    [5, 3, 6] <- Lastin  Replaced:2 [Hits:0 Misses:4]
Access: 6  HIT  FirstIn ->    [5, 3, 6] <- Lastin  Replaced:- [Hits:1 Misses:4]
Access: 0  MISS FirstIn ->    [3, 6, 0] <- Lastin  Replaced:5 [Hits:1 Misses:5]
Access: 0  HIT  FirstIn ->    [3, 6, 0] <- Lastin  Replaced:- [Hits:2 Misses:5]
Access: 8  MISS FirstIn ->    [6, 0, 8] <- Lastin  Replaced:3 [Hits:2 Misses:6]
Access: 2  MISS FirstIn ->    [0, 8, 2] <- Lastin  Replaced:6 [Hits:2 Misses:7]
Access: 2  HIT  FirstIn ->    [0, 8, 2] <- Lastin  Replaced:- [Hits:3 Misses:7]
```

```
./paging-policy.py -s 3 -n 10 -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 3
ARG notrace False

Access: 2  MISS LRU ->          [2] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 5  MISS LRU ->       [2, 5] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 3  MISS LRU ->    [2, 5, 3] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 6  MISS LRU ->    [5, 3, 6] <- MRU Replaced:2 [Hits:0 Misses:4]
Access: 6  HIT  LRU ->    [5, 3, 6] <- MRU Replaced:- [Hits:1 Misses:4]
Access: 0  MISS LRU ->    [3, 6, 0] <- MRU Replaced:5 [Hits:1 Misses:5]
Access: 0  HIT  LRU ->    [3, 6, 0] <- MRU Replaced:- [Hits:2 Misses:5]
Access: 8  MISS LRU ->    [6, 0, 8] <- MRU Replaced:3 [Hits:2 Misses:6]
Access: 2  MISS LRU ->    [0, 8, 2] <- MRU Replaced:6 [Hits:2 Misses:7]
```

Access: 2 HIT LRU -> [0, 8, 2] <- MRU Replaced:- [Hits:3 Misses:7]

FINALSTATS hits 3 misses 7 hitrate 30.00

./paging-policy.py -s 3 -n 10 -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 3
ARG notrace False

Access: 2 MISS Left -> [2] <- Right Replaced:- [Hits:0 Misses:1]
Access: 5 MISS Left -> [2, 5] <- Right Replaced:- [Hits:0 Misses:2]
Access: 3 MISS Left -> [2, 5, 3] <- Right Replaced:- [Hits:0 Misses:3]
Access: 6 MISS Left -> [2, 5, 6] <- Right Replaced:3 [Hits:0 Misses:4]
Access: 6 HIT Left -> [2, 5, 6] <- Right Replaced:- [Hits:1 Misses:4]
Access: 0 MISS Left -> [2, 5, 0] <- Right Replaced:6 [Hits:1 Misses:5]
Access: 0 HIT Left -> [2, 5, 0] <- Right Replaced:- [Hits:2 Misses:5]
Access: 8 MISS Left -> [2, 5, 8] <- Right Replaced:0 [Hits:2 Misses:6]
Access: 2 HIT Left -> [2, 5, 8] <- Right Replaced:- [Hits:3 Misses:6]
Access: 2 HIT Left -> [2, 5, 8] <- Right Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4 misses 6 hitrate 40.00

2. For a cache of size 5, generate worst-case address reference streams for each of the following policies: FIFO, LRU, and MRU (worst-case reference streams cause the most misses possible. For the worst case reference streams, how much bigger of a cache is needed to improve performance dramatically and approach OPT?

For FIFO:
./paging-policy.py -C 5 -a 1,2,3,4,5,6,1,2,3,4,5,6 -c
ARG addresses 1,2,3,4,5,6,1,2,3,4,5,6
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 5

ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 1 MISS FirstIn ->   [1] <- Lastin Replaced:- [Hits:0 Misses:1]
Access: 2 MISS FirstIn ->   [1, 2] <- Lastin Replaced:- [Hits:0 Misses:2]
Access: 3 MISS FirstIn ->   [1, 2, 3] <- Lastin Replaced:- [Hits:0 Misses:3]
Access: 4 MISS FirstIn -> [1, 2, 3, 4] <- Lastin Replaced:- [Hits:0 Misses:4]
Access: 5 MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin Replaced:- [Hits:0 Misses:5]
Access: 6 MISS FirstIn -> [2, 3, 4, 5, 6] <- Lastin Replaced:1 [Hits:0 Misses:6]
Access: 1 MISS FirstIn -> [3, 4, 5, 6, 1] <- Lastin Replaced:2 [Hits:0 Misses:7]
Access: 2 MISS FirstIn -> [4, 5, 6, 1, 2] <- Lastin Replaced:3 [Hits:0 Misses:8]
Access: 3 MISS FirstIn -> [5, 6, 1, 2, 3] <- Lastin Replaced:4 [Hits:0 Misses:9]
Access: 4 MISS FirstIn -> [6, 1, 2, 3, 4] <- Lastin Replaced:5 [Hits:0 Misses:10]
Access: 5 MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin Replaced:6 [Hits:0 Misses:11]
Access: 6 MISS FirstIn -> [2, 3, 4, 5, 6] <- Lastin Replaced:1 [Hits:0 Misses:12]

FINALSTATS hits 0   misses 12   hitrate 0.00

./paging-policy.py -C 5 -p LRU -a 1,2,3,4,5,6,1,2,3,4,5,6 -c
ARG addresses 1,2,3,4,5,6,1,2,3,4,5,6
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 5
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 1 MISS LRU ->   [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 2 MISS LRU ->   [1, 2] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 3 MISS LRU ->   [1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 4 MISS LRU -> [1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 5 MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 6 MISS LRU -> [2, 3, 4, 5, 6] <- MRU Replaced:1 [Hits:0 Misses:6]
Access: 1 MISS LRU -> [3, 4, 5, 6, 1] <- MRU Replaced:2 [Hits:0 Misses:7]
Access: 2 MISS LRU -> [4, 5, 6, 1, 2] <- MRU Replaced:3 [Hits:0 Misses:8]

Access: 3 MISS LRU -> [5, 6, 1, 2, 3] <- MRU Replaced:4 [Hits:0 Misses:9]
Access: 4 MISS LRU -> [6, 1, 2, 3, 4] <- MRU Replaced:5 [Hits:0 Misses:10]
Access: 5 MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:11]
Access: 6 MISS LRU -> [2, 3, 4, 5, 6] <- MRU Replaced:1 [Hits:0 Misses:12]

FINALSTATS hits 0   misses 12   hitrate 0.00

./paging-policy.py -C 5 -p MRU -a 1,2,3,4,5,6,5,6,5,6,5,6 -c
ARG addresses 1,2,3,4,5,6,5,6,5,6,5,6
ARG addressfile
ARG numaddrs 10
ARG policy MRU
ARG clockbits 2
ARG cachesize 5
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 1 MISS LRU ->   [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 2 MISS LRU ->   [1, 2] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 3 MISS LRU ->   [1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 4 MISS LRU -> [1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 5 MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 6 MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:6]
Access: 5 MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:7]
Access: 6 MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:8]
Access: 5 MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:9]
Access: 6 MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:10]
Access: 5 MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:11]
Access: 6 MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:12]

FINALSTATS hits 0   misses 12   hitrate 0.00

By increase the cache size to 6 it will be equal to OPT for each different policy

3. Generate a random trace (use python or perl). How would you expect the different policies to perform on such a trace?

```
nickliu@Nicks-Mac chapter22:$ ./paging-policy.py -s 100
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 100
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 1  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
```

I expect the hit rate is : OPT >= LRU >= CLOCK >= FIFO >= RAND

4. Now generate a trace with some locality. How can you generate such a trace? How does LRU perform on it? How much better than RAND is LRU? How does CLOCK do? How about CLOCK with different numbers of clock bits?

./paging-policy.py -C 3 -a 1,2,3,4,5,1,2,3,2,3,1,3,2,1 -c -p LRU
ARG addresses 1,2,3,4,5,1,2,3,2,3,1,3,2,1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

FINALSTATS hits 6   misses 8   hitrate 42.86

./paging-policy.py -C 3 -a 1,2,3,4,5,1,2,3,2,3,1,3,2,1 -c -p RAND
ARG addresses 1,2,3,4,5,1,2,3,2,3,1,3,2,1
ARG addressfile
ARG numaddrs 10
ARG policy RAND
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10

FINALSTATS hits 4   misses 10   hitrate 28.57

./paging-policy.py -C 3 -a 1,2,3,4,5,1,2,3,2,3,1,3,2,1 -c -N -p CLOCK -b 1
ARG addresses 1,2,3,4,5,1,2,3,2,3,1,3,2,1
ARG addressfile
ARG numaddrs 10
ARG policy CLOCK
ARG clockbits 1
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace True

FINALSTATS hits 7   misses 7   hitrate 50.00

./paging-policy.py -C 3 -a 1,2,3,4,5,1,2,3,2,3,1,3,2,1 -c -N -p CLOCK -b 2
ARG addresses 1,2,3,4,5,1,2,3,2,3,1,3,2,1

ARG addressfile
ARG numaddrs 10
ARG policy CLOCK
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace True

FINALSTATS hits 7   misses 7   hitrate 50.00

Both LRU and clock bit 1 and 2 doing really good, but RAND has the worst performance.

5. Use a program like valgrind to instrument a real application and generate a virtual page reference stream. For example, running valgrind --tool=lackey --trace-mem=yes ls will output a nearly-complete reference trace of every instruction and data reference made by the program ls. To make this useful for the simulator above, you'll have to first transform each virtual memory reference into a virtual page-number reference (done by masking off the offset and shifting the resulting bits downward). How big of a cache is needed for your application trace in order to satisfy a large fraction of requests? Plot a graph of its working set as the size of the cache increases.

```
sudo apt install libhugetlbfs-bin
nick@ubuntu:~/CS5600_hw/week7/chapter22$ getconf PAGESIZE
4096
```

The page size for the OS is 4096(4KB), need 13bits(2^13=4096) to represent the offset
One of the instruction memory address is 0x1fff0003b0 need 40bits, then all VPN should be first (40-13) = 27 bits. No matter how may level of page director, the 27bits find a page. Thus, we need to shift 27 bits

Cache size need be 5 to satisfy a large fraction of requests.