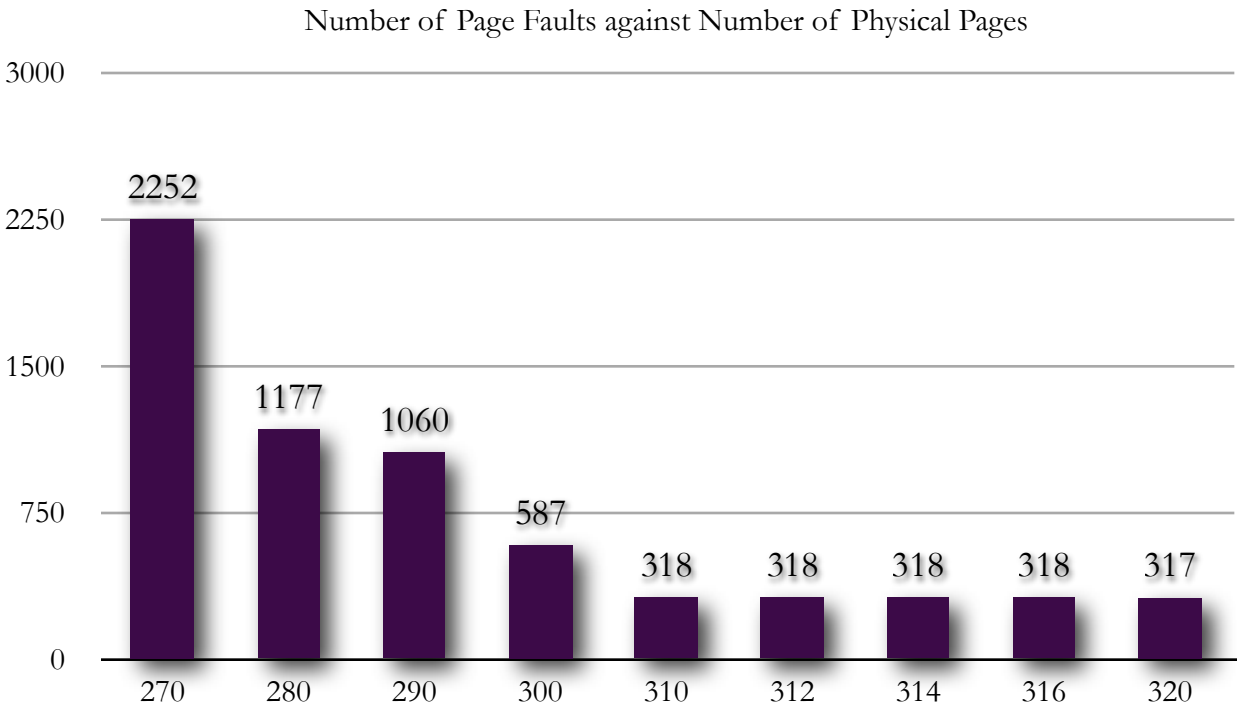
Part 1:
Run the bzip benchmark with the default 1-level page table, with 320, 316, 314, 312, 310, 300, 290, 280 and 270 physical pages, and plot the number of page faults as a function of the number of physical pages available.



Part 2:
Extend the simulator to get an estimate of the running time of a given trace, based on the following costs:

i. 100 cycles if the page table entry is present and valid

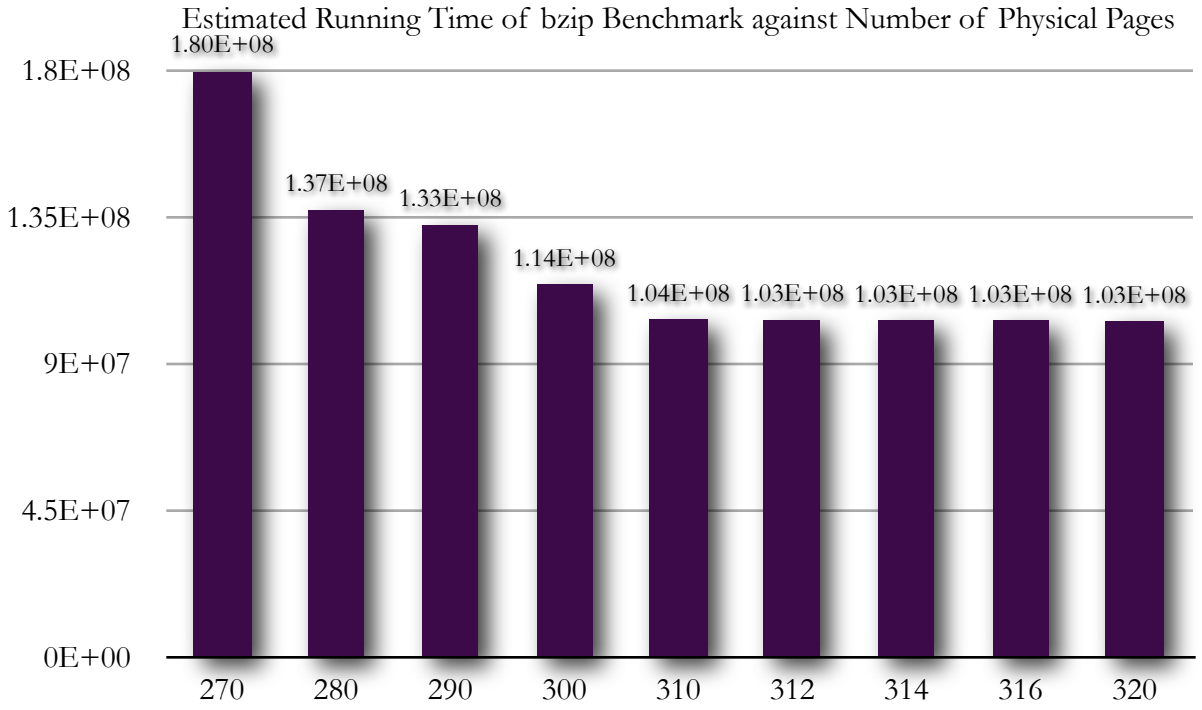
ii. 10000 cycles if the page table entry is valid but not present, and a free page is immediately available in the physical memory manager.

iii. if the VM system needs to scan through and examine page table entries to find a victim page to evict, each access to examine a page table entry costs 100 cycles, plus a fixed cost of 10000 cycles from (ii) above.

iv. if the VM system decides to evict a dirty page, it will need to write the dirty contents of the page to the disk, incurring a 50000 cycle penalty, plus 100 cycles per each page table entry access from (iii) above to find the dirty victim, plus a fixed cost of 10000 cycles from (ii) above.

Assume that all non-memory related instructions (which do not even appear in our trace files) cost 0 cycles.

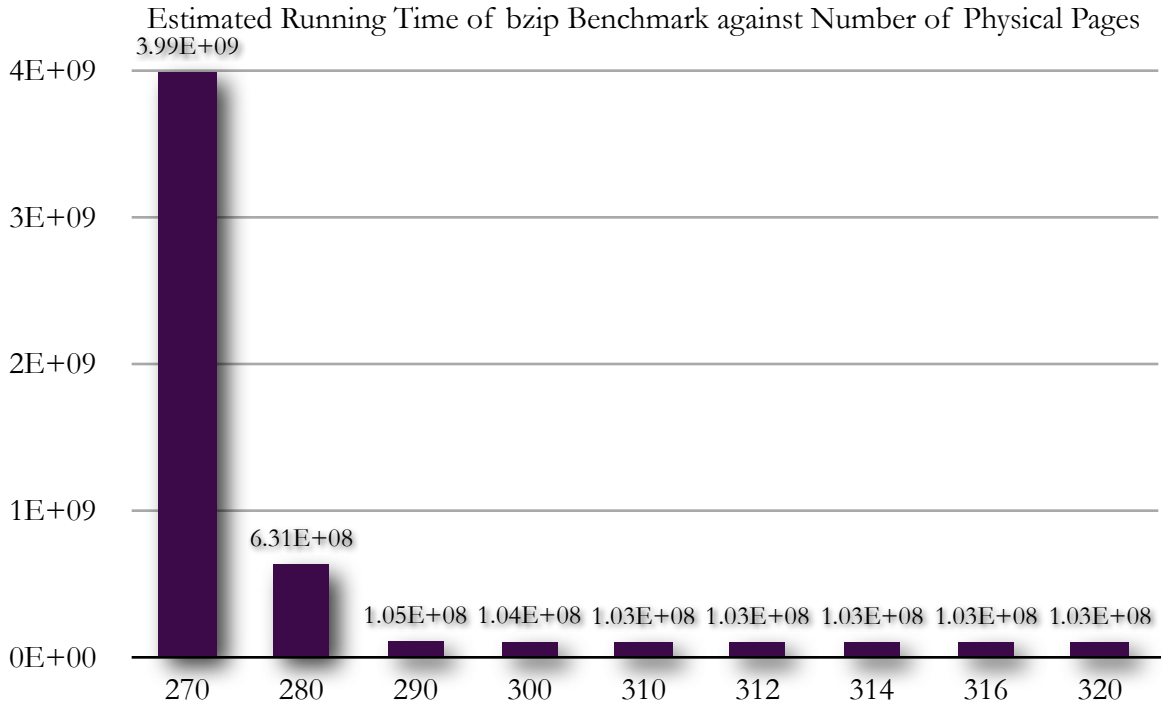
Run the bzip benchmark with the default 1-level page table, with 320, 316, 314, 312, 310, 300, 290, 280 and 270 physical pages, and plot the estimated running time of the bzip benchmark as a function of the number of physical pages available.



Part 3:
As you might have noticed, the eviction algorithm used in the current simulator is not at all sophisticated. Modify it to approximate LRU replacement. Specifically, use the "access" bits in each page table entry to find a non-dirty physical page that has not been accessed in the last ~10000 instructions that is mapped to the lowest virtual address, and evict it. Where such a page cannot be located, fall back on a dirty physical page that has not been accessed in the last ~10000 instructions that is mapped to the lowest virtual address. Where such a page cannot be located, default to the original behavior of the code.

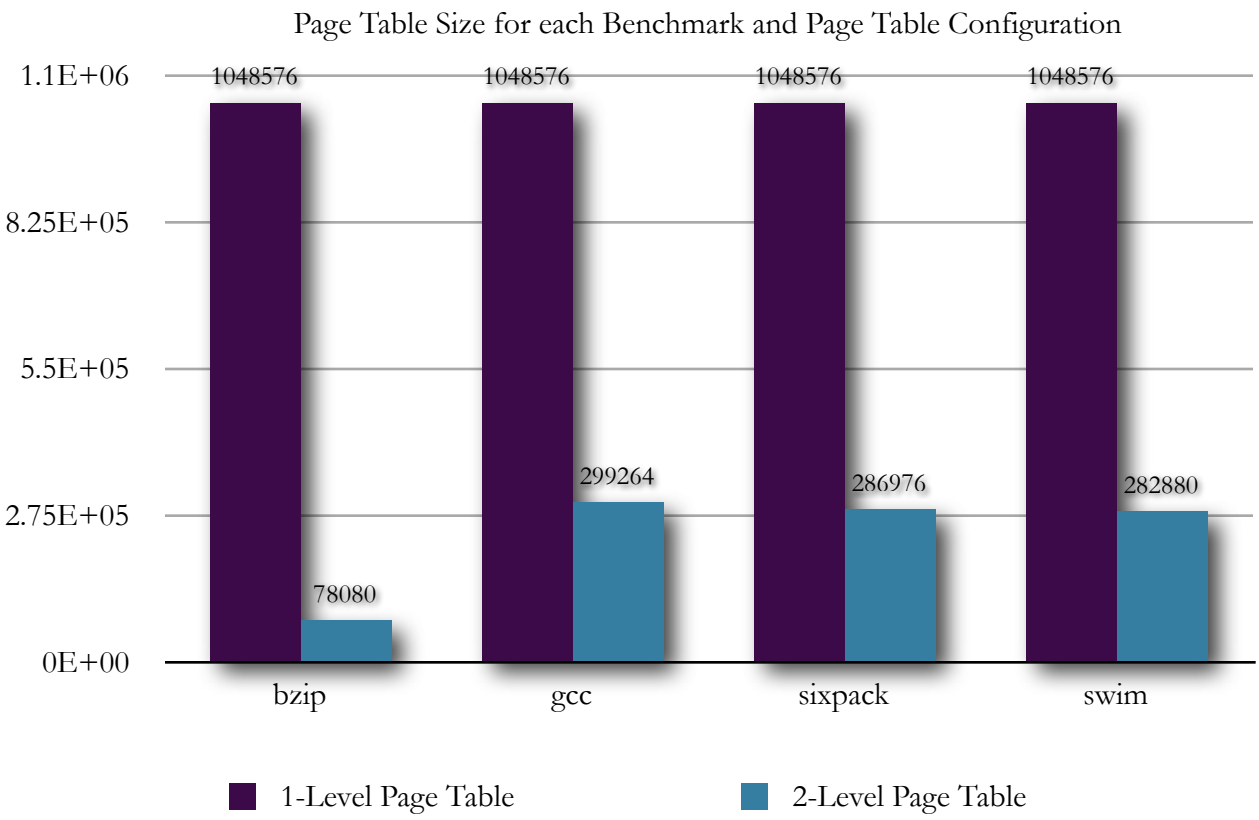
Given what the hardware provides, it is difficult to figure out which pages have and have not been touched in the last 10K instructions. For this assignment, you should run a periodic OS task every 10K instructions that clears the access bits, and, on eviction, pick a page table entry whose access bit is 0.

Plot the running time of the bzip benchmark again, this time with this new eviction algorithm.



Part 4:
Change the 1-level page table implementation to a 2-level page table. Use 8 bits for the first level, 12 bits for the second, and 12 bits for the page offset. Plot, as a bar chart and for each of the benchmarks, the size of the old 1-level page table and the size of the new 2-level page table. TO do this, you'll want to copy the 1level code to a new directory named 2level, and modify it there.

Benchmark	1-Level Page Table	2-Level Page Table
bzip	1048576	78080
gcc	1048576	299264
sixpack	1048576	286976
swim	1048576	282880


Page Table Size for each Benchmark and Page Table Configuration
1.1E+06
8.25E+05
5.5E+05
2.75E+05
0E+00
1048576
1048576
1048576
1048576
78080
299264
286976
282880
bzip
gcc
sixpack
swim
1-Level Page Table
2-Level Page Table