1. For timing, you'll need to use a timer (e.g., gettimeofday()). How precise is such a timer? How long does an operation have to take in order for you to time it precisely? (this will help determine how many times, in a loop, you'll have to repeat a page access in order to time it successfully)

```
파일(F)  편집(E)  보기(V)  검색(S)  터미널(T)  도움말(H)
 1 #include<stdio.h>
 2 #include<time.h>
 3 #include<sys/time.h>
 4
 5
 6 int main() {
 7
 8
 9     struct timeval tv;
10
11
12     gettimeofday(&tv, NULL);
13     printf("[TIME] %ld:%ld\n", tv.tv_sec, tv.tv_usec);
14
15
16
17     return 0;
18 }
```

It is code that I make to use a timer.

```
lasol@dasol-VirtualBox:~$ ./time
[TIME] 1555314384:796534
lasol@dasol-VirtualBox:~$ ./time
[TIME] 1555314394:696228
lasol@dasol-VirtualBox:~$ ./time
[TIME] 1555314406:91716
lasol@dasol-VirtualBox:~$ ./time
[TIME] 1555314414:892330
lasol@dasol-VirtualBox:~$ ./time
[TIME] 1555314424:786415
lasol@dasol-VirtualBox:~$ ./time
[TIME] 1555314434:803953
lasol@dasol-VirtualBox:~$ ./time
[TIME] 1555314444:918596
lasol@dasol-VirtualBox:~$ ./time
[TIME] 1555314454:924539
lasol@dasol-VirtualBox:~$
```

I checked the time in 10-second increments. It was confirmed to appear up to microseconds. In other words, it is quite accurate.

2. Write the program, called tlb.c, that can roughly measure the cost of accessing each page. Inputs to the program should be: the number of pages to touch and the number of trials.

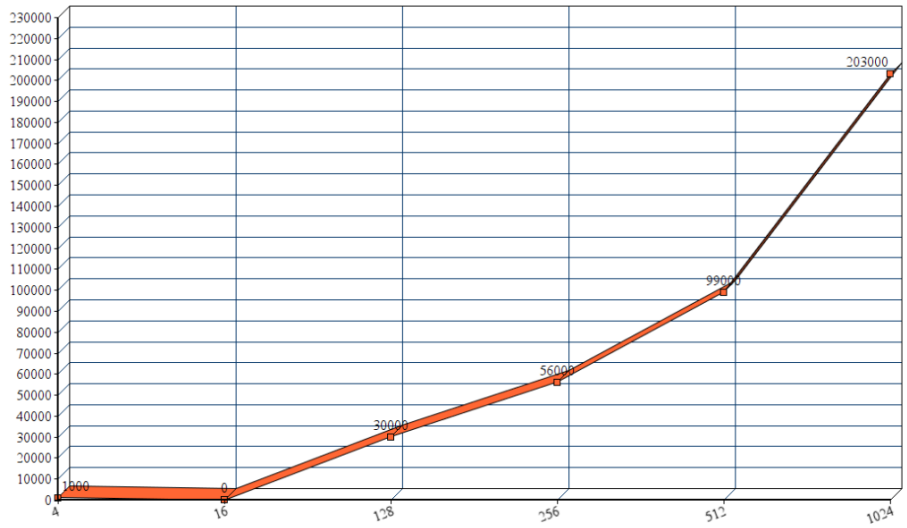
```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/time.h>

int main() {

   struct timeval t_Start;
   struct timeval t_End;
   double i, t;
   int page;
   int *array_;
   int jump = 128 / sizeof(int);

   array_ =(int*) malloc(sizeof(int)*10000);
   printf("put page number: ");
   scanf("%d", &page);

   gettimeofday(&t_Start, NULL);

   for (int j=0; j<page*jump; j += jump){
         array_[j] += 1;
      }


   gettimeofday(&t_End, NULL);
   t = ((t_End.tv_usec - t_Start.tv_usec) * 1000);
   printf("%lf\n", t);

   return 0;
}
```

This is my code to measure the cost of accessing each page. I used gettimeofday() function to get time cost. Also, I set page size as 128 because I think 4 is too small. In this loop, one integer per page of the array is updated until the end page. More increases jump in cost because of number of page, the first-level TLB is bigger. I can see TLB hits and misses can affect performance.

```
dasol@dasol-VirtualBox:~$ gcc -o tlb1 tlb1.c
dasol@dasol-VirtualBox:~$ ./tlb1
put page number: 4
1000.000000
dasol@dasol-VirtualBox:~$ ./tlb1
put page number: 16
0.000000
dasol@dasol-VirtualBox:~$ ./tlb1
put page number: 128
30000.000000
dasol@dasol-VirtualBox:~$ ./tlb1
put page number: 256
56000.000000
dasol@dasol-VirtualBox:~$ ./tlb1
put page number: 512
99000.000000
dasol@dasol-VirtualBox:~$ ./tlb1
put page number: 1024
203000.000000
dasol@dasol-VirtualBox:~$
```

This is result. Because I set page size as 128, there is no TLB miss in first two trials. So, first two trials (case 4, 16) have very short time and case 16 is even shorter then case 4. However, page number is increased, their time also become increased. The bigger the speed, the steeper the width is. I can see the width is steeper since the beginning of the miss.

4. Next, graph the results, making a graph that looks similar to the one above. Use a good tool like ploticus or even zplot. Visualization usually makes the data much easier to digest; why do you think that is?



This is my graph. I can see the first two trials (case 4, 16) is similar because they hit TLB but the width is steeper since the beginning of the TLB miss.

5. One thing to watch out for is compiler optimization. Compilers do all sorts of clever things, including removing loops which increment values that no other part of the program subsequently uses. How can you ensure the compiler does not remove the main loop above from your TLB size estimator?

Variables declared volatile always access memory when used. 'Volatile' means that the compiler is always told to access the memory because the value can change at any time. The compiler optimizes the code, delete repeats and allocates 10 to value.

So, you must declare it volatile so that you always have access to memory.