```c
/*
   iobasic.c

   Demonstrates the effect of the memory hierarchy and the limitations
   of the RAM model.

   The RAM model assumes that all memory accesses take the same and
   defines

   complexity = CPU complexity =  nb. CPU operations

   When working with large data the cache and IO behaviour (nb. cache
   misses/page faults) are relevant and may actually be dominant.
   Theoretically this is modeled by

   cache complexity = nb blocks transferred between cache and main
   memory

   IO complexity = nb blocks transferred between main memory and disk
*/
#include <stdio.h>
#include <time.h>
#include <assert.h>
#include <stdlib.h>

#define PRINT if(0)

int main(int argc, char** argv) {

  clock_t  t1, t2;
  if (argc !=2)  {
    printf("usage: %s [n]\n", argv[0]);
    exit(1);
  }
  long n = atol(argv[1]);
  printf("n=%ld (%.1f G) ints which require  %.1f GB RAM\n",
         n,
         (float)n/(1<<30),
         (float)n*sizeof(int)/(1<<30));


  //allocate  and initialize
  t1 = clock();
  printf("%30s", "allocate and initialize a: ");
```

```c
  int* a = (int*)malloc(n * sizeof(int));
  assert(a);
  for (long i=0; i<n; i++) {
    a[i]=i;
  }
  t2 = clock();
  printf("time elapsed %.3f seconds\n", (double)(t2-t1)/CLOCKS_PER_SEC);


  //SEQUENTIAL ACCESS ARRAY A
  int sum = 0;
  t1 = clock();
  printf("%30s", "sequential access: ");
  printf("\n");
  for (long i=0; i<n; i++)  {
    sum =(sum + a[i])%2;

    //print % progress
    PRINT {if (i% (n/100) ==0) printf("%d %% ", (int) (i/(n/100)));
fflush(stdout);}
  }
  t2 = clock();
  printf("\n%30s time elapsed %.3f seconds\n", " ", (double)(t2-
t1)/CLOCKS_PER_SEC);


  //RANDOM ACCESS ARRAY A
  t1 = clock();
  printf("%30s", "random access: ");
  printf("\n");
  for (long i=0; i<n; i++) {
    //generate a random index in 0..n
    long  k = random() % n;
    sum =(sum + a[i])%2;

    //print % progress
    PRINT  { if (i% (n/100) ==0) printf("%d %% ", (int)
(i/(n/100)));fflush(stdout);}
  }
  t2 = clock();
  printf("\n%30s time elapsed %.3f seconds\n"," ",  (double)(t2-
t1)/CLOCKS_PER_SEC);

  return 0;
}
```