HW8 Solutions

Problem 1

First read the first int of the first page, then the first int of the second page (to create a new TLB entry), and so on through all six pages (this is the inner loop below, with i=0). Then read the second int of the first page, the second int of the second page, and so on through all six pages. This access pattern is reflected in the code below:

```
uint64_t tlb_unfriendly() {
    int *a = page_alloc(6 * PAGE_SIZE);
    populate_array(a); // sets the integers in the array
    uint64_t sum = 0;

    /* YOUR CODE HERE: compute sum in the most TLB-unfriendly way possible */

    for (int i = 0; i < PAGESIZE/sizeof(int); i++)
        for (int j = 0;  j < 6; j++)
            sum += a[j*PAGESIZE/sizeof(int) + i];

    return sum;
}
```

Problem 2

Lay out the allocated memory so that the last legitimately allocated byte is on the last byte of the allocated page (this "wastes" the first part of a page). Mark the next virtual page (past the array) as "not in use" (this does not cost physical memory). At that point, memory references past the end of the array will generate page faults.