PROBLEM SET 4

Zhiqiang Xie 77892769

Problem 1:

- The `arrival` lock protect the variable `count` to be increased by only one thread at the same time.
- Once one thread get the `arrival` lock, increase `count` and:
    - If not all threads are arrival, the `arrival` lock will be released, and the thread will try to access the `departure` lock.
    - If the thread is the last one to arrive, the `departure` lock will be released.
- Now the second phase starts:
    - The `departure` lock protect the variable `count` to be decreased by only one thread at the same time.
    - Similarly, after all threads leave, the `arrival` lock will be released and then the threads can access the `arrival` lock. The barrier will be back to the previous state.
- The two locks protect the increasement and decreasement respectively.

Problem 2

```
void barrier(){
    set_lock(lock);
    count++;
    if (count==N){
        count = 0;
        signal(lock,all_arrival);
    }else{
        wait(lock,all_arrival);
    }
    unset_lock(lock);
}
```

The blocked threads will be waked up when condition variable `all_arrival` turns to be true. Here we just need one lock to protect the `count` and condition variable `all_arrival`.

Problem 3

1. It would be suitable. As for large matrix multiplication, massive operations (multiply and summation) are parallelizable, it would be benefit from the large throughput of GPU.
2. Since $32 * 32$ is not that small, it's still a proper size to deploy CUDA threads to speed up in the similar way.
3. & 4.

It's not suitable to do binary search in GPU regardless of the size of array. Since the computation complexity of binary search is $O(\log n)$, it's quiet efficient to do it sequentially. Every time it pick and compare two elements, and then decide the way (branch and branch) to go. It's somehow hard to do it efficiently in parallel way.

Problem 4

- Compared to CPU threads and the IO latency, there's almost no cost to switch and schedule the GPU threads since they're extremely light-weighted.
- The SIMD architecture could take the advantage of light threads to perform massive and fast calculation in less clock cycle.
- If we increase the amount of work done in each CUDA thread, it would increase the **overhead** of switching and scheduling cost.