ECE 420 Assignment 2
Jiannan Lu 157761

1. schedule(static,2):
   First thread : iteration 1 2 5 6…9997 9998
   Second thread: iteration 3 4 7 8…9999
   schedule(guided):
   First thread : iteration 1…5000
   Second thread: iteration 5001…9999

2. The OpenMP may better initiate parallel threads at the outer loop and start threads within conditional statements

```
# pragma omp parallel num threads(thread count) \ default(none)
shared(a, n) private(i, tmp)
for (phase = 0; phase < n; phase++) {
    if (phase % 2 == 0)
# pragma omp for
        for (i = 1; i < n; i += 2) {
            if (a[i1] > a[i]) {
                tmp = a[i1];
                a[i1] = a[i];
                a[i] = tmp;
            }
        }
    else
# pragma omp for
        for (i = 1; i < n1; i += 2) {
            if (a[i] > a[i+1]) {
                tmp = a[i+1];
                a[i+1] = a[i];
                a[i] = tmp;
            }
        }
}
```

3.

```
int temp, largest;
largest = 0;
#pragma omp parallel private(temp){
    temp = 0;
#pragma omp for
    for ( int i = 0; i < 1000; i++ ) {
        if (data[i] > temp)
            temp = data[i];
```

```
    }
    if (temp > largest) {
#pragma critical
        if (temp > largest) {
            largest = temp;
        }
    }
}
```

4. The implementation of the first is right, the one of the second is wrong, race condition occurs on v2[i] operation. Fix it pretty much the same concept as q3.

```
// Parallelize by columns
# pragma omp parallel default(none) shared(v2,v1,matrix,tam)
private(i,j)
{
    double temp[tam] = {};
    for (i = 0; i < tam; i++) {
# pragma omp for
        for (j = 0; j < tam; j++)
            temp[i] += matrix[i][j] * v1[j];
    }
# pragma omp critical
    for (i = 0; i < tam; i++)
        v2[i] += temp[i];
}
```

5.
0: 2 4
Inner: 4
1: 2 6
Inner: 6
count me. (the same output for 10 times in total)

6. 1)

```
int fib(int n) {
    int temp1,temp2;
    if (n<2)
        return n;
    else {
#pragma omp parallel sections firstprivate(n) shared(temp1,temp2)
        {
#pragma omp section
            temp1 = fib(n-1);
#pragma omp section
```

```
            temp2 = fib(n-2);
        }
        return temp1 + temp2;
    }
}
```

2)

```
int fib_tasks(int n) {
    int temp1, temp2;
    if (n<2)
        return n;
    else {
#pragma omp task shared(temp1) firstprivate(n)
        temp1 = fib_tasks(n-1);
#pragma omp task shared(temp2) firstprivate(n)
        temp2 = fib_tasks(n-2);
#pragma omp taskwait
        return temp1 + temp2;
    }
}

int main(int argc, char *argv[]) {
    int result;
    printf("Doing sequential fibonacci:\n");
    result = fib_tasks(5);
    printf("Result is %d\n", result);
    return 0;
}
```

3)

In program 1, there will be 3 threads (including thread origin) executing fib(5), 5 threads executing fib(4), 9 threads executing fib(3), 15 threads executing fib(2), therefore, 15 in total. In program 2, 3 threads throughout.