Programming Assignment 1
Student ID: 0660004

Name: 王順興

A. The idea of your program

Serial version:

```
num_city: number of cities to connect
2opt_swap(start, end);

for (i = 1; i < num_city - 1; ++i)
        for (j = i + 1; j < num_city; ++j)
                2opt_swap(i, j);
```

Can be changed to:

```
start from (1) to (num_city - 1)
        2opt_swap with *depth* (2) to (num_city - i)

for (i = 1; i < num_city - 1; ++i)
        for (depth = 1; depth < num_city - i; ++depth)
                2opt_swap(i, i + depth);
```

depth is the length between the 2opt_swap indices

Idea: Split *depth* across threads

```
maximum_depth = num_city - 1
depth_for_each_thread = maximum_depth / num_thread

for (depth = thread_depth_start[n]; depth < thread_depth_end[n]; ++depth)
        for (i = 1; i < num_city - depth; ++i)
                2opt_swap(i, i + depth);

2opt_swap(start, end):
        _read_lock
                create new_route
        _unlock
        assert(distance(new_route) < distance(current_route))
        _write_lock
                current_route = new_route
        _unlcok
end
```

Note:

- A race condition may occur when a thread pass through assertion, but not yet change the `current_route`; a better `current_route` may be overwritten, so the assertion have to be done again after applying `_write_lock`
- The aforementioned race condition also prevents partial update of the `current_route`, so a new array(new_route) is created every operation.
- `distance(current_route)` can be cached.
- **Distance calculation can be optimized by calculate only the update segment of the route. (**Tests marked as Dist. Calc. optimized**)**
- There are also two ways to split the *depth*

```
Example 9 depth, 3 threads

Chunk:
Thread 1: 1/2/3 depth
Thread 2: 4/5/6 depth
Thread 3: 7/8/9 depth

Balanced:
Thread 1: 1/4/7 depth
Thread 2: 2/5/8 depth
Thread 3: 3/6/9 depth
```

B. Performance (run-time) analysis with 1, 2, 4, 8, 16 core(s)

# of 2opt_swap called in 10 mins				
	Before distance calculation is optimized		Dist. Calc. optimized	
Th	**Chunk**	**Balanced**	**Chunk.opt**	**Balanced.opt**
1	24856	25119	66276	63110
2	46971	49701	80953	129261
4	76105	85136	96509	261605
8	167108	201120	85412	541035
16	239296	405542	71206	857152

C. Record your distance every 30 seconds with **test11**

	Before distance calculation is optimized		Dist. Calc. optimized	
Time	**Chunk**	**Balanced**	**Chunk.opt**	**Balanced.opt**
0m 30s	5236217548888	5236767376441	5239234504229	5233894202422
1m 00s	5231959394787	5233243836456	5238367721581	5228312584841
1m 30s	5227752503221	5229821611984	5237506418954	5223445466794
2m 00s	5223639454509	5226335277762	5236697033023	5218777773759
2m 30s	5219355854175	5222886988838	5235881056704	5214000570217
3m 00s	5214973671001	5219493109615	5235048453474	5209375645966
3m 30s	5210569949525	5215917028459	5234220435439	5204544189800
4m 00s	5206257153593	5212576696538	5233415665273	5199606540301
4m 30s	5201869454315	5209312681293	5232583414345	5194835256033
5m 00s	5197506692526	5206066426828	5231731245305	5190038327006
5m 30s	5193214637330	5202621568215	5230898861426	5185116719534
6m 00s	5188657856652	5199072196711	5230066163885	5180219477646
6m 30s	5184220766738	5195754086165	5229208916041	5175506414432
7m 00s	5179929194095	5192451415552	5228305331581	5170612659967
7m 30s	5175611197172	5189208714037	5227440837385	5165726046679
8m 00s	5171157164455	5185829449116	5226608781609	5160854768811
8m 30s	5166698519634	5182298344208	5225715564559	5156041475717
9m 00s	5162187441111	5178791170286	5224870726305	5151151922781
9m 30s	5157769845727	5175618917479	5224035406388	5146242747622
10m 00s	5153487487696	5172455395479	5223038216906	5141353455597

D. Discussion

The number of race condition happening between rd_lock and wr_lock scales with thread count:

% of 2opt_call with race condition				
	before distance calculation is optimized		Dist. Calc. optimized	
Th	**Chunk**	**Balanced**	**Chunk.opt**	**Balanced.opt**
1	0	0	0	0
2	2.38	4.79	7.79	12.00
4	9.17	7.76	17.55	19.55
8	22.55	10.21	28.09	23.59
16	31.63	12.82	33.78	25.04

The race condition caused the whole work done in that 2opt_swap() call to be wasted.

The reason for Balanced algo. to scale well and performed better with distance calculation optimization is the shorter average length of updated route comparing to Chunk algo. :

Avg. length of updated route				
	before distance calculation is optimized		Dist. Calc. optimized	
Th	**Chunk**	**Balanced**	**Chunk.opt**	**Balanced.opt**
1	0	0	0	0
2	367913.65	2.58	51417.19	2.53
4	423225.60	3.35	167824.67	3.60
8	460001.93	5.16	326708.33	5.68
16	470879.20	9.28	419019.19	9.76

Different job distribution lead to different allocation of resource for CPU. Balanced algo., on average, spend more time working on shorter route, so the distance calculation optimization is actually helpful.