Test Plan For MP3

Student: Yulin Xiao
CUID: C16278133
Student UserName: Yulinx

1.Pre-TEST Plan:(See MP2)

2.Prep. TEST Plan:(without the geninput.c)(Page1-6)

3.Post-TEST Plan(with the geninput.c)(Page7-)

1.About the 5 kinds of sort, I will first test every sort type by using the inputx. txt about every kind of source data, like 5 3 6 7 8 4 2 1 9.

So my test files include:

(1) input1.txt: general test for all of 5 types for 9 records
(2) input2.txt: general test for all of 5 types for 2 records
(3) input3.txt: general test for all of 5 types for 1 records
(4) input4.txt: general test for all of 5 types for 0 records

To illustrate this:(see the inputx.txt and the Outputx.tx):

(Tips: How? Using command: valgrind --leak-check=full ./lab3 5 <inputx.txt>outputx.txt to see the memory condition and open the .txt file to check if the expected Output is consistent)

Input1.txt:	**Output1.txt(Expected)**
ADDTAIL 5	Lab3 list size is 5. Possible commands:
ADDTAIL 3	MP3: ADDTAIL priority
ADDTAIL 6	SORT sort# direction
ADDTAIL 7	PRINTMP3
ADDTAIL 8	
ADDTAIL 4	List: INSERT
ADDTAIL 2	FIND id
ADDTAIL 1	REMOVE id
ADDTAIL 9	UPDATE id state
PRINTMP3	SCHEDULE id priority
SORT 1 1	DETERMINE
PRINTMP3	CLEAN
SORT 1 2	REVERSE
PRINTMP3	PRINT
SORT 2 1	Queue : ADDTAIL; RMHEAD; PRINTHEAD; PRINTQ
PRINTMP3	: STATS; QUIT
SORT 2 2	
PRINTMP3	
SORT 3 1	List has 9 records
PRINTMP3	0: Task ID: 0 priority = 5 state = QUEUED
SORT 3 2	1: Task ID: 0 priority = 3 state = QUEUED
PRINTMP3	2: Task ID: 0 priority = 6 state = QUEUED
SORT 4 1	3: Task ID: 0 priority = 7 state = QUEUED
PRINTMP3	4: Task ID: 0 priority = 8 state = QUEUED
SORT 4 2	5: Task ID: 0 priority = 4 state = QUEUED
PRINTMP3	6: Task ID: 0 priority = 2 state = QUEUED
SORT 5 1	7: Task ID: 0 priority = 1 state = QUEUED
PRINTMP3	8: Task ID: 0 priority = 9 state = QUEUED
SORT 5 2	List has 9 records

PRINTMP3 QUIT	0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 8 state = QUEUED 2: Task ID: 0 priority = 7 state = QUEUED 3: Task ID: 0 priority = 6 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 4 state = QUEUED 6: Task ID: 0 priority = 3 state = QUEUED 7: Task ID: 0 priority = 2 state = QUEUED 8: Task ID: 0 priority = 1 state = QUEUED List has 9 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 2 state = QUEUED 2: Task ID: 0 priority = 3 state = QUEUED 3: Task ID: 0 priority = 4 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 6 state = QUEUED 6: Task ID: 0 priority = 7 state = QUEUED 7: Task ID: 0 priority = 8 state = QUEUED 8: Task ID: 0 priority = 9 state = QUEUED List has 9 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 8 state = QUEUED 2: Task ID: 0 priority = 7 state = QUEUED 3: Task ID: 0 priority = 6 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 4 state = QUEUED 6: Task ID: 0 priority = 3 state = QUEUED 7: Task ID: 0 priority = 2 state = QUEUED 8: Task ID: 0 priority = 1 state = QUEUED List has 9 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 2 state = QUEUED 2: Task ID: 0 priority = 3 state = QUEUED 3: Task ID: 0 priority = 4 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 6 state = QUEUED 6: Task ID: 0 priority = 7 state = QUEUED 7: Task ID: 0 priority = 8 state = QUEUED 8: Task ID: 0 priority = 9 state = QUEUED List has 9 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 8 state = QUEUED 2: Task ID: 0 priority = 7 state = QUEUED 3: Task ID: 0 priority = 6 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 4 state = QUEUED 6: Task ID: 0 priority = 3 state = QUEUED 7: Task ID: 0 priority = 2 state = QUEUED 8: Task ID: 0 priority = 1 state = QUEUED List has 9 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 2 state = QUEUED 2: Task ID: 0 priority = 3 state = QUEUED 3: Task ID: 0 priority = 4 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 6 state = QUEUED 6: Task ID: 0 priority = 7 state = QUEUED

	7: Task ID: 0 priority = 8 state = QUEUED 8: Task ID: 0 priority = 9 state = QUEUED List has 9 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 8 state = QUEUED 2: Task ID: 0 priority = 7 state = QUEUED 3: Task ID: 0 priority = 6 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 4 state = QUEUED 6: Task ID: 0 priority = 3 state = QUEUED 7: Task ID: 0 priority = 2 state = QUEUED 8: Task ID: 0 priority = 1 state = QUEUED List has 9 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 2 state = QUEUED 2: Task ID: 0 priority = 3 state = QUEUED 3: Task ID: 0 priority = 4 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 6 state = QUEUED 6: Task ID: 0 priority = 7 state = QUEUED 7: Task ID: 0 priority = 8 state = QUEUED 8: Task ID: 0 priority = 9 state = QUEUED List has 9 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 8 state = QUEUED 2: Task ID: 0 priority = 7 state = QUEUED 3: Task ID: 0 priority = 6 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 4 state = QUEUED 6: Task ID: 0 priority = 3 state = QUEUED 7: Task ID: 0 priority = 2 state = QUEUED 8: Task ID: 0 priority = 1 state = QUEUED List has 9 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 2 state = QUEUED 2: Task ID: 0 priority = 3 state = QUEUED 3: Task ID: 0 priority = 4 state = QUEUED 4: Task ID: 0 priority = 5 state = QUEUED 5: Task ID: 0 priority = 6 state = QUEUED 6: Task ID: 0 priority = 7 state = QUEUED 7: Task ID: 0 priority = 8 state = QUEUED 8: Task ID: 0 priority = 9 state = QUEUED Goodbye

Input2.txt	**Output2.txt(Expected)**
ADDTAIL 1 ADDTAIL 9 PRINTMP3 SORT 1 1 PRINTMP3 SORT 1 2 PRINTMP3 SORT 2 1 PRINTMP3 SORT 2 2 PRINTMP3 SORT 3 1 PRINTMP3 SORT 3 2 PRINTMP3 SORT 4 1 PRINTMP3 SORT 4 2 PRINTMP3 SORT 5 1 PRINTMP3 SORT 5 2 PRINTMP3 QUIT	Lab3 list size is 5. Possible commands: MP3: ADDTAIL priority SORT sort# direction PRINTMP3 List: INSERT FIND id REMOVE id UPDATE id state SCHEDULE id priority DETERMINE CLEAN REVERSE PRINT Queue : ADDTAIL; RMHEAD; PRINTHEAD; PRINTQ : STATS; QUIT List has 2 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 9 state = QUEUED List has 2 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 1 state = QUEUED List has 2 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 9 state = QUEUED List has 2 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 1 state = QUEUED List has 2 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 9 state = QUEUED List has 2 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 1 state = QUEUED List has 2 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 9 state = QUEUED List has 2 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 1 state = QUEUED List has 2 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 9 state = QUEUED List has 2 records 0: Task ID: 0 priority = 9 state = QUEUED 1: Task ID: 0 priority = 1 state = QUEUED List has 2 records 0: Task ID: 0 priority = 1 state = QUEUED 1: Task ID: 0 priority = 9 state = QUEUED Goodbye

Input3.txt:	**Output3.txt(Expected)**
ADDTAIL 9	Lab3 list size is 5. Possible commands:
PRINTMP3	MP3: ADDTAIL priority
SORT 1 1	SORT sort# direction
PRINTMP3	PRINTMP3
SORT 1 2	
PRINTMP3	List: INSERT
SORT 2 1	FIND id
PRINTMP3	REMOVE id
SORT 2 2	UPDATE id state
PRINTMP3	SCHEDULE id priority
SORT 3 1	DETERMINE
PRINTMP3	CLEAN
SORT 3 2	REVERSE
PRINTMP3	PRINT
SORT 4 1	Queue : ADDTAIL; RMHEAD; PRINTHEAD; PRINTQ
PRINTMP3	: STATS; QUIT
SORT 4 2	
PRINTMP3	
SORT 5 1	List has 1 record.
PRINTMP3	0: Task ID: 0 priority = 9 state = QUEUED
SORT 5 2	List has 1 record.
PRINTMP3	0: Task ID: 0 priority = 9 state = QUEUED
QUIT	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	List has 1 record.
	0: Task ID: 0 priority = 9 state = QUEUED
	Goodbye

Input4.txt	**Output4.txt(Expected)**
PRINTMP3 SORT 1 1 PRINTMP3 SORT 1 2 PRINTMP3 SORT 2 1 PRINTMP3 SORT 2 2 PRINTMP3 SORT 3 1 PRINTMP3 SORT 3 2 PRINTMP3 SORT 4 1 PRINTMP3 SORT 4 2 PRINTMP3 SORT 5 1 PRINTMP3 SORT 5 2 PRINTMP3 QUIT	Lab3 list size is 5. Possible commands: MP3: ADDTAIL priority SORT sort# direction PRINTMP3 List: INSERT FIND id REMOVE id UPDATE id state SCHEDULE id priority DETERMINE CLEAN REVERSE PRINT Queue : ADDTAIL; RMHEAD; PRINTHEAD; PRINTQ : STATS; QUIT List empty List empty List empty List empty List empty List empty List empty List empty List empty List empty List empty Goodbye

3.Post-TEST Plan(With geninput.c)

3. We include the timer program to generate a number of record to test the performance of the algorithms: uncomment the time function in task_list_sort:

```
void task_list_sort(list_t ** list_ptr, int sort_type, int sort_order)
{
        clock_t start, end;
        double elapse_time;    // time in milliseconds
        int initial_size = list_size(*list_ptr);
        start = clock();

        list_sort(list_ptr, sort_type, sort_order);

        end = clock();
        elapse_time =    1000.0 * ((double) (end - start)) / CLOCKS_PER_SEC;
        assert(list_size(*list_ptr) == initial_size);
        printf("Sorting: n = %d time = %f type = %d order = %d\n", initial_size, elapse_time,
  sort_type, sort_order);
}
```

I will test each of the five sorting algorithms and each of the three list types (random, ascending, descending) with at least five different list sizes. Sort in ascending order and create graphs to illustrate my results. In particular, for the tests involving random list types, I shall include in my graph the result for at least one list size that requires more than one second to sort as determined using the C function clock. Remember that I am not timing the sort in isolation (i.e., other applications are running, competing for 3 CPU time and cache space). Finally, I shall run each experiment multiple times and average the results for each data point in my graph.

In my Test Log document, in addition to reporting my data using graphs, I will describe:
1). When lists that are initially random the differences in running time for the sorting algorithms. Do my iterative and recursive selection sort algorithms show dramatic differences in running times or are they similar? Why does the merge sort algorithm show a dramatic improvement in run time?
2). If a list is already in ascending or descending order, some sort algorithms are very fast while others still have to perform a similar number of comparisons as when the list is not sorted. I will describe which algorithm(s) show extremely fast performance if the list is already sorted, and explain why.

Example of the table are put HERE:(only ascending is required)

list_bubble_sort:

List_size:	Gen: Random(ms)	Gen: Ascending(ms)	Gen.Descending(ms)
1000	12.352	0.008	7.428
5000	272.675	0.044	213.456
10000	1112.266	0.108	817.156
15000	2591.752	0.143	1841.928
20000	4693.864	0.188	3280.223

R: Most time consuming A: Most efficient D:2/3 of the Random time: O(n^2)

list_insertion_sort:

List_size:	Gen: Random(ms)	Gen: Ascending(ms)	Gen: Descending(ms)
1000	3.629	3.052	0.048
5000	80.067	93.698	0.237
10000	327.050	328.752	0.441
15000	820.437	748.695	0.661
20000	1506.490	1305.741	1.096

R: 2nd Most time consuming A: 2nd Most efficient D:extremely efficient: O(n^2)

list_recursive_selection_sort:

List_size:	Gen: Random(ms)	Gen: Ascending(ms)	Gen: Descending(ms)
1000	3.649	3.066	2.851
5000	100.196	98.743	96.358
10000	372.531	342.199	321.814
15000	823.049	749.479	726.662
20000	1360.831	1340.099	1288.195

R:3rd time consuming A: average, D average

list_selection_sort:

List_size:	Gen: Random(ms)	Gen: Ascending(ms)	Gen: Descending(ms)
1000	3.353	3.088	2.991
5000	103.122	98.743	96.358
10000	371.648	340.199	323.517
15000	825.123	750.321	730.662
20000	1345.831	1287.454	1294.132

R:4th time-consuming, A:average, D:average

list_merge_sort:

List_size:	Gen: Random(ms)	Gen: Ascending(ms)	Gen: Descending(ms)
1000	1.021	1.040	0.58
5000	6.175	5.768	5.650
10000	18.360	15.021	17.732
15000	26.972	23.867	23.336
20000	38.316	30.147	29.814

Efficient in all domains!