

UNCLASSIFIED

AD

AD-E403 685

Technical Report ARWSE-TR-14024

ORDERED VERSUS UNORDERED MAP FOR PRIMITIVE DATA TYPES

Tom Nealis

September 2015



U.S. ARMY ARMAMENT RESEARCH, DEVELOPMENT AND
ENGINEERING CENTER

Weapons and Software Engineering Center

Picatinny Arsenal, New Jersey

Approved for public release; distribution is unlimited.

UNCLASSIFIED

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

The citation in this report of the names of commercial firms or commercially available products or services does not constitute official endorsement by or approval of the U.S. Government.

Destroy this report when no longer needed by any method that will prevent disclosure of its contents or reconstruction of the document. Do not return to the originator.

UNCLASSIFIED

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-01-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden to Department of Defense, Washington Headquarters Services Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) September 2015		2. REPORT TYPE Final		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE ORDERED VERSUS UNORDERED MAP FOR PRIMITIVE DATA TYPES				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHORS Tom Nealis				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army ARDEC, WSEC Fire Control Systems & Technology Directorate (RDAR-WSF-M) Picatinny Arsenal, NJ 07806-5000				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army ARDEC, ESIC Knowledge & Process Management (RDAR-EIK) Picatinny Arsenal, NJ 07806-5000				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) Technical Report ARWSE-TR-14024	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT A map in programming is an associative container consisting of some key value mapped to some element. C++ provides two types of map containers within the standard template library, the std::map and the std::unordered_map classes. As the name implies, the containers main functional difference is that the elements in the std::map are ordered by the key, and the std::unordered_map are not ordered based on their key. The std::unordered_map elements are placed into "buckets" based on a hash value computed for their key. This report will concentrate on the performance difference of these two containers using a primitive data type for the key.					
15. SUBJECT TERMS std::map std::unordered_map					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT SAR	18. NUMBER OF PAGES 13	19a. NAME OF RESPONSIBLE PERSON Tom Nealis
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) (973) 724-8048

CONTENTS

	Page
Introduction	1
Methodology	1
Conclusions	7
Distribution List	9

INTRODUCTION

A map in computer science programming is an associative container consisting of a key value mapped to some element. The C++ programming language provides two types of map containers within the standard template library, the `std::map` and the `std::unordered_map` classes. The main difference between these two containers is the way in which the elements are stored. In the `std::unordered_map`, the elements are not stored in order by the key. The `std::unordered_map` elements are placed into “buckets” based on a hash value computed for their key.

The underlying structure of a `std::map` is typically a binary search tree. The `std::map` is thought to be generally slower than unordered maps but certainly have their use if ordered access is necessary. The `std::unordered_map` is stored in a hash table. This allows for faster access to elements based on a hash computation done on the key value. This computed value is then used to look up the location of the element.

METHODOLOGY

In order to test the performance of these containers for comparison, a small program was written to measure and then log the execution time of certain tasks involving the containers. The main concentration was on some of the more common tasks such as inserting, finding, erasing, and traversal. The following is the program written to accomplish this:

```
int _tmain(int argc, _TCHAR* argv[])
{
    LARGE_INTEGER frequency;
    QueryPerformanceFrequency(&frequency);

    LARGE_INTEGER starting_time, ending_time, elapsed_microseconds;

    srand(static_cast<unsigned int>(time(nullptr)));

    std::ofstream a_file("outfile.txt");

    std::map<unsigned int, unsigned int> ordered_map;
    std::unordered_map<unsigned int, unsigned int> unordered_map;

    std::vector<unsigned int> my_ints;
    std::vector<unsigned int> find_ints;
    auto intergers = 0u;
    for(auto j = 0u; j < 20; ++j)
    {
        intergers += 100000;

        //clear all vectors and arrays
        my_ints.clear();
        find_ints.clear();
        ordered_map.clear();
        unordered_map.clear();

        //populate vector
        for(auto i = 0u; i < intergers; ++i)
            my_ints.push_back(rand() % intergers);
    }
}
```

UNCLASSIFIED

```
//add 10 elements to find
for(auto i = 0; i < 10; ++i)
    find_ints.push_back(rand() % intergers);

//populate ordered map
QueryPerformanceCounter(&starting_time);

for(const auto &j : my_ints)
    ordered_map.insert(std::pair<unsigned int, unsigned int>(j, j));

QueryPerformanceCounter(&ending_time);
elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

auto time_elapsed_ord_insert = static_cast<double>((elapsed_microseconds.QuadPart *
1000000.0) / frequency.QuadPart);
//printf("Ordered Map Insert took: %4.2f microseconds\n", time_elapsed_ord_insert);

//populate unordered map
QueryPerformanceCounter(&starting_time);

for(const auto &j : my_ints)
    unordered_map.insert(std::pair<unsigned int, unsigned int>(j, j));

QueryPerformanceCounter(&ending_time);
elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

auto time_elapsed_unordered_insert = static_cast<double>((elapsed_microseconds.QuadPart *
1000000.0) / frequency.QuadPart);
//printf("Unordered Map Insert took: %4.2f microseconds\n", time_elapsed_unordered_insert);

//find some ints in ordered map
QueryPerformanceCounter(&starting_time);

for(const auto &j : find_ints)
    auto it = ordered_map.find(j);

QueryPerformanceCounter(&ending_time);
elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

auto time_elapsed_ordered_find = static_cast<double>((elapsed_microseconds.QuadPart *
1000000.0) / frequency.QuadPart);
//printf("Ordered Map Find took: %4.2f microseconds\n", time_elapsed_ordered_find);

//find the same ints in unordered map
QueryPerformanceCounter(&starting_time);

for(const auto &j : find_ints)
    auto it = unordered_map.find(j);

QueryPerformanceCounter(&ending_time);
elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;
```

UNCLASSIFIED

```
    auto time_elapsed_unordered_find = static_cast<double>((elapsed_microseconds.QuadPart *
1000000.0) / frequency.QuadPart);
    //printf("Unordered Map Find took: %4.2f microseconds\n", time_elapsed_unordered_find);

//erase some ints in ordered map
QueryPerformanceCounter(&starting_time);

for(const auto &j : find_ints)
    auto it = ordered_map.erase(j);

QueryPerformanceCounter(&ending_time);
elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

    auto te_ordered_delete = static_cast<double>((elapsed_microseconds.QuadPart * 1000000.0) /
frequency.QuadPart);

//erase the same ints in unordered map
QueryPerformanceCounter(&starting_time);

for(const auto &j : find_ints)
    auto it = unordered_map.erase(j);

QueryPerformanceCounter(&ending_time);
elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

    auto te_unordered_delete = static_cast<double>((elapsed_microseconds.QuadPart * 1000000.0) /
frequency.QuadPart);

//range ordered map
QueryPerformanceCounter(&starting_time);

    auto it = ordered_map.begin();
    for(const auto &j : ordered_map)
        ;

    QueryPerformanceCounter(&ending_time);
    elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

    auto te_ordered_range = static_cast<double>((elapsed_microseconds.QuadPart * 1000000.0) /
frequency.QuadPart);

//range unordered map
QueryPerformanceCounter(&starting_time);

for(const auto &j : unordered_map)
    ;

QueryPerformanceCounter(&ending_time);
elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

    auto te_unordered_range = static_cast<double>((elapsed_microseconds.QuadPart * 1000000.0) /
frequency.QuadPart);
```

```

a_file << intergers << ",";
a_file << time_elapsed_ord_insert << ",";
a_file << time_elapsed_unordered_insert << ",";
a_file << time_elapsed_ordered_find << ",";
a_file << time_elapsed_unordered_find << ",";
a_file << te_ordered_delete << ",";
a_file << te_unordered_delete << ",";
a_file << te_ordered_range << ",";
a_file << te_unordered_range << "\r\n";//"\r\n";

printf("Integers: %d \tOrdered Insert: %4.2f \tUnordered Insert: %4.2f\r\n", intergers,
time_elapsed_ord_insert, time_elapsed_unordered_insert);
printf("Integers: %d \tOrdered Find : %4.2f \t Unordered Find : %4.2f\r\n", intergers,
time_elapsed_ordered_find, time_elapsed_unordered_find);
printf("Integers: %d \tOrdered Erase : %4.2f \t Unordered Erase : %4.2f\r\n", intergers,
te_ordered_delete, te_unordered_delete);
printf("Integers: %d \tOrdered Range : %4.2f \t Unordered Range : %4.2f\r\n", intergers,
te_ordered_range, te_unordered_range);
}

a_file.close();

printf("All done!\n");

//this stops the program in order to see data;
//getchar();

return 0;
}

```

This program first populates two standard vectors with random numbers. The first one stores integers that will be inserted into the containers. The second one consists of 10 random numbers used for finding values in the maps. The next step was to measure how long it took to insert all the values into the containers. Then, the time it took to find all 10 values was recorded, and then the time it took for the removal of all 10 of those values was recorded. Lastly, the traversal of the entire map was timed. Figures 1 through 4 show the measurements that were logged.

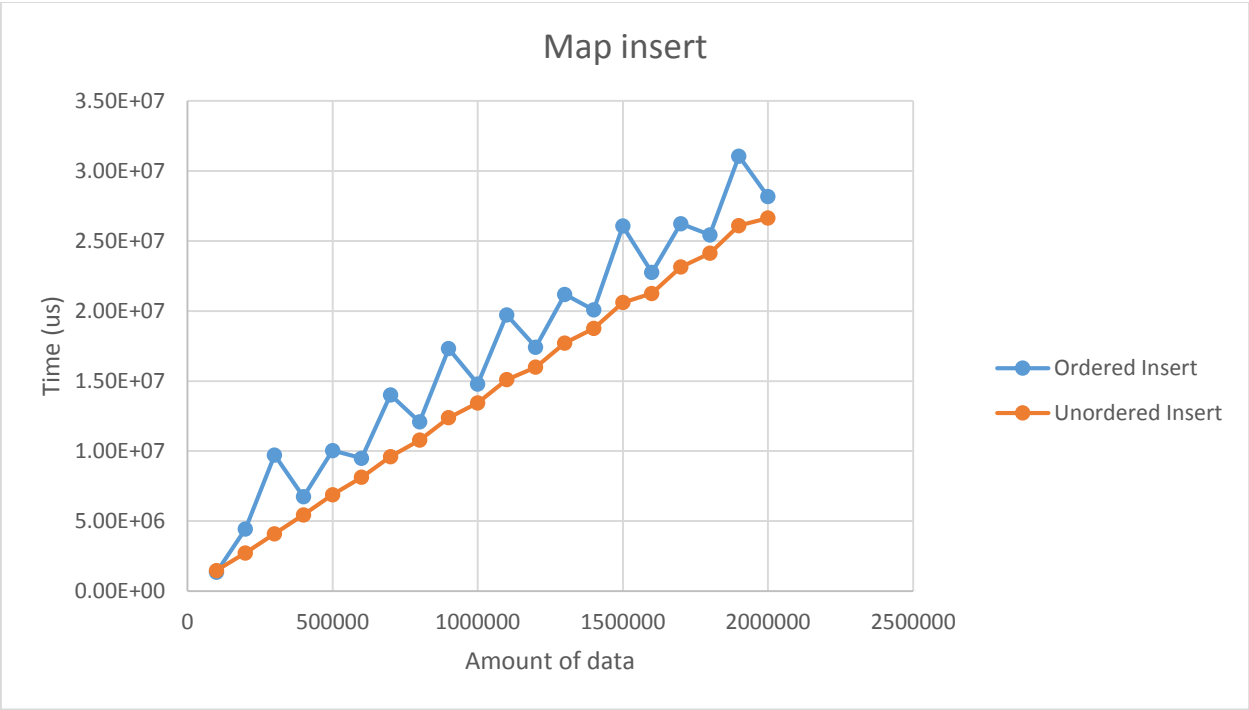


Figure 1
Map insert

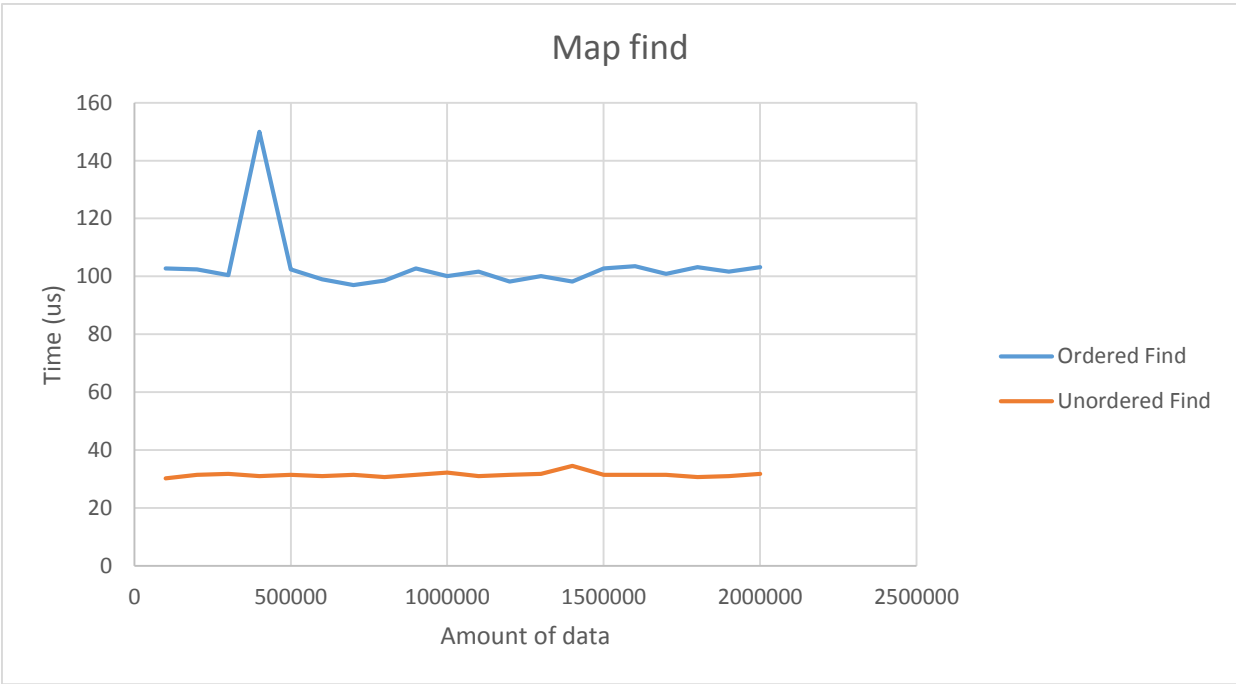


Figure 2
Map find

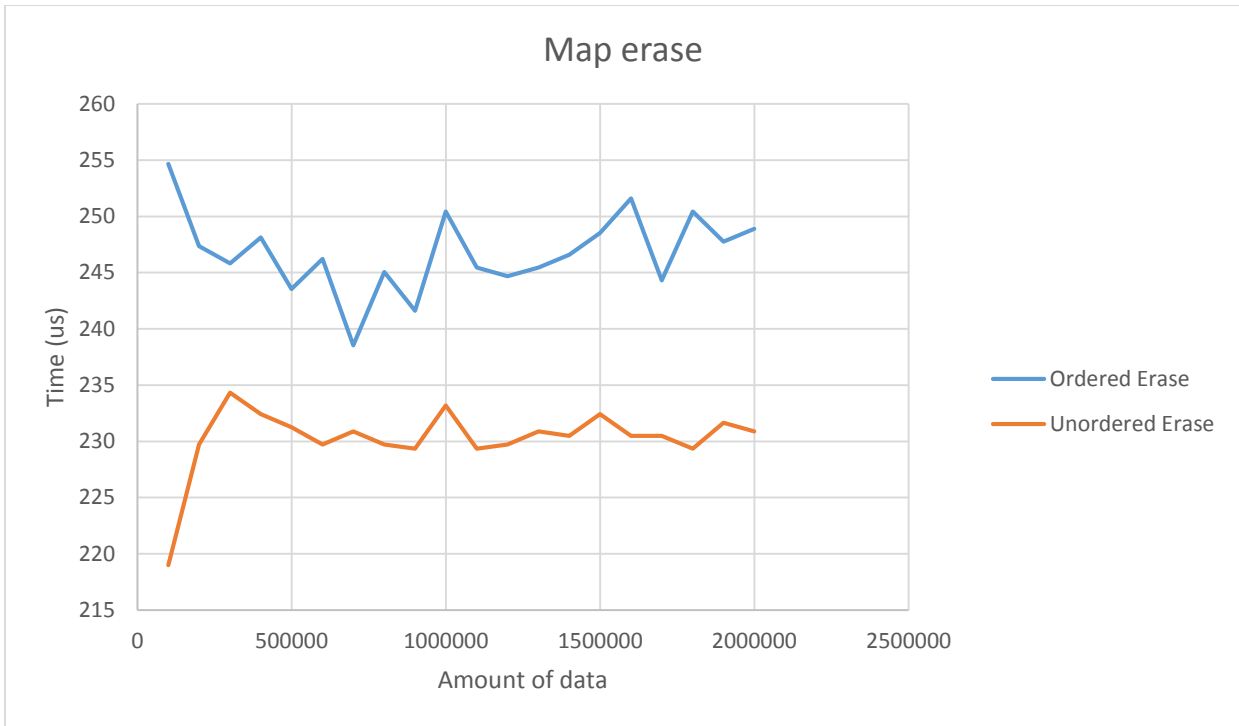


Figure 3
Map erase

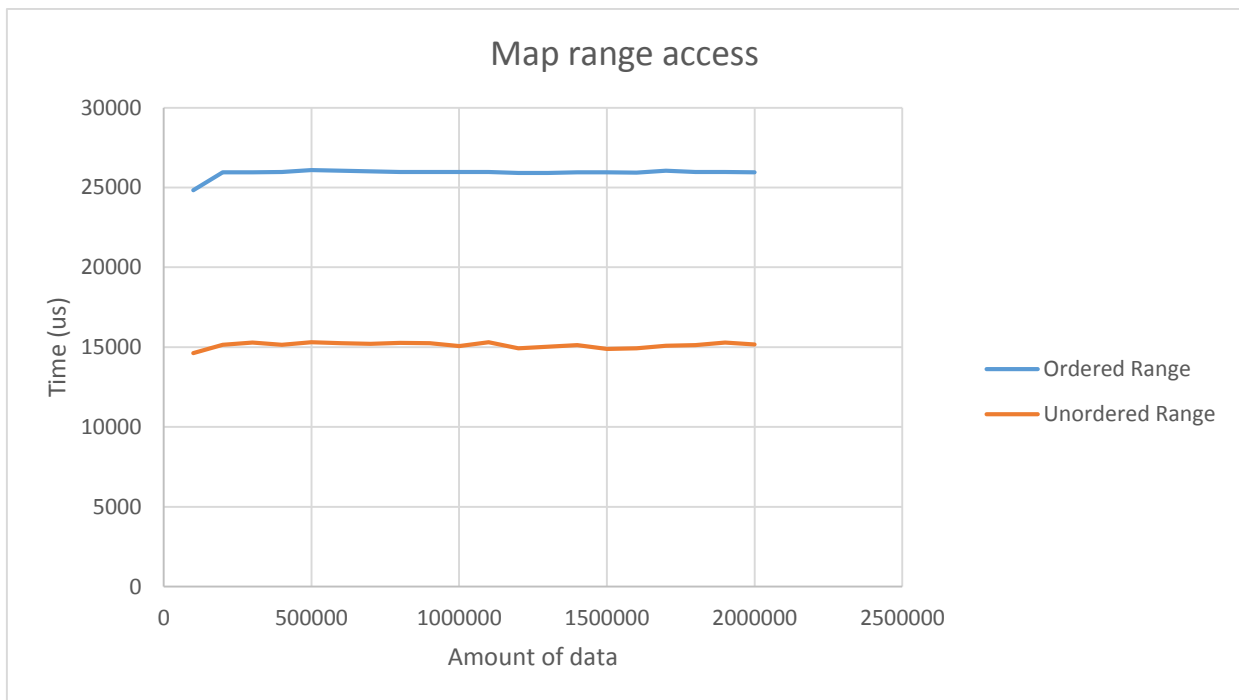


Figure 4
Map range access

CONCLUSIONS

The `std::unordered_map` outperformed the `std::map` in all categories measured. If it is not necessary to have the elements sorted, using the `std::unordered_map` class is suggested. Using the `std::unordered_map` class will make your program faster and more efficient.

UNCLASSIFIED

DISTRIBUTION LIST

U.S. Army ARDEC
ATTN: RDAR-EIK
RDAR-WSF-M, T. Nealis
Picatinny Arsenal, NJ 07806-5000

Defense Technical Information Center (DTIC)
ATTN: Accessions Division
8725 John J. Kingman Road, Ste 0944
Fort Belvoir, VA 22060-6218

GIDEP Operations Center
P.O. Box 8000
Corona, CA 91718-8000
gidep@gidep.org

REVIEW AND APPROVAL OF ARDEC TECHNICAL REPORTS

Ordered Vs. Unordered Map for Primitive Data Types _____
 Title _____ Date received by LCSD _____

Thomas M. Nealis _____
 Author/Project Engineer _____ Report number (to be assigned by LCSD) _____

X8048 31 RDAR-WSF-M

Extension Building Author's/Project Engineers Office
 (Division, Laboratory, Symbol)

PART 1. Must be signed before the report can be edited.

- a. The draft copy of this report has been reviewed for technical accuracy and is approved for editing.
- b. Use Distribution Statement A, X, B, C, D, E, F or X for the reason checked on the continuation of this form. Reason: Operational Use
 1. If Statement A is selected, the report will be released to the National Technical Information Service (NTIS) for sale to the general public. Only unclassified reports whose distribution is not limited or controlled in any way are released to NTIS.
 2. If Statement B, C, D, E, F, or X is selected, the report will be released to the Defense Technical Information Center (DTIC) which will limit distribution according to the conditions indicated in the statement.
- c. The distribution list for this report has been reviewed for accuracy and completeness.

Patricia Alameda
 Division Chief

9/10/15
 (Date)

PART 2. To be signed either when draft report is submitted or after review of reproduction copy.

This report is approved for publication.

Patricia Alameda
 Division Chief

9/10/15
 (Date)

Andrew Pskowski
 RDAR-CIS

10/1/15
 (Date)

LCSD 49 supersedes SMCAR Form 49, 20 Dec 06