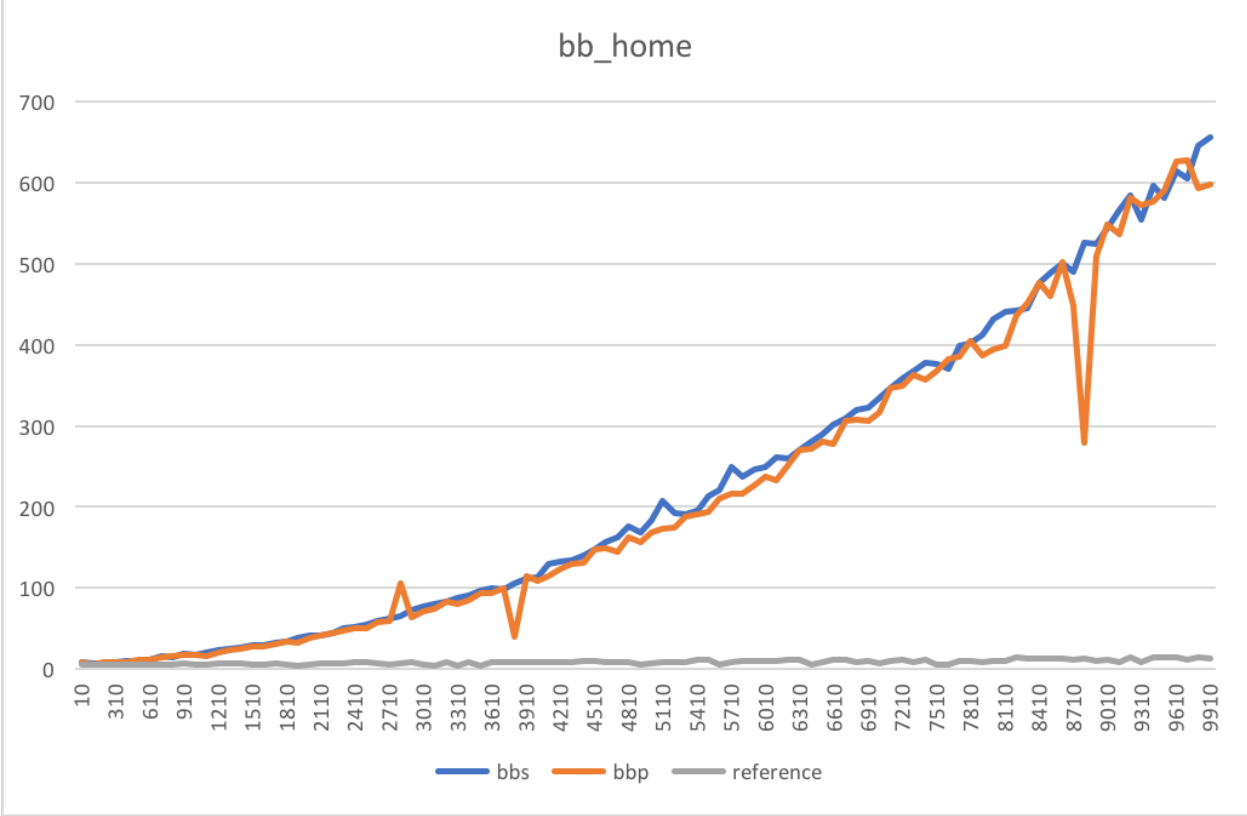
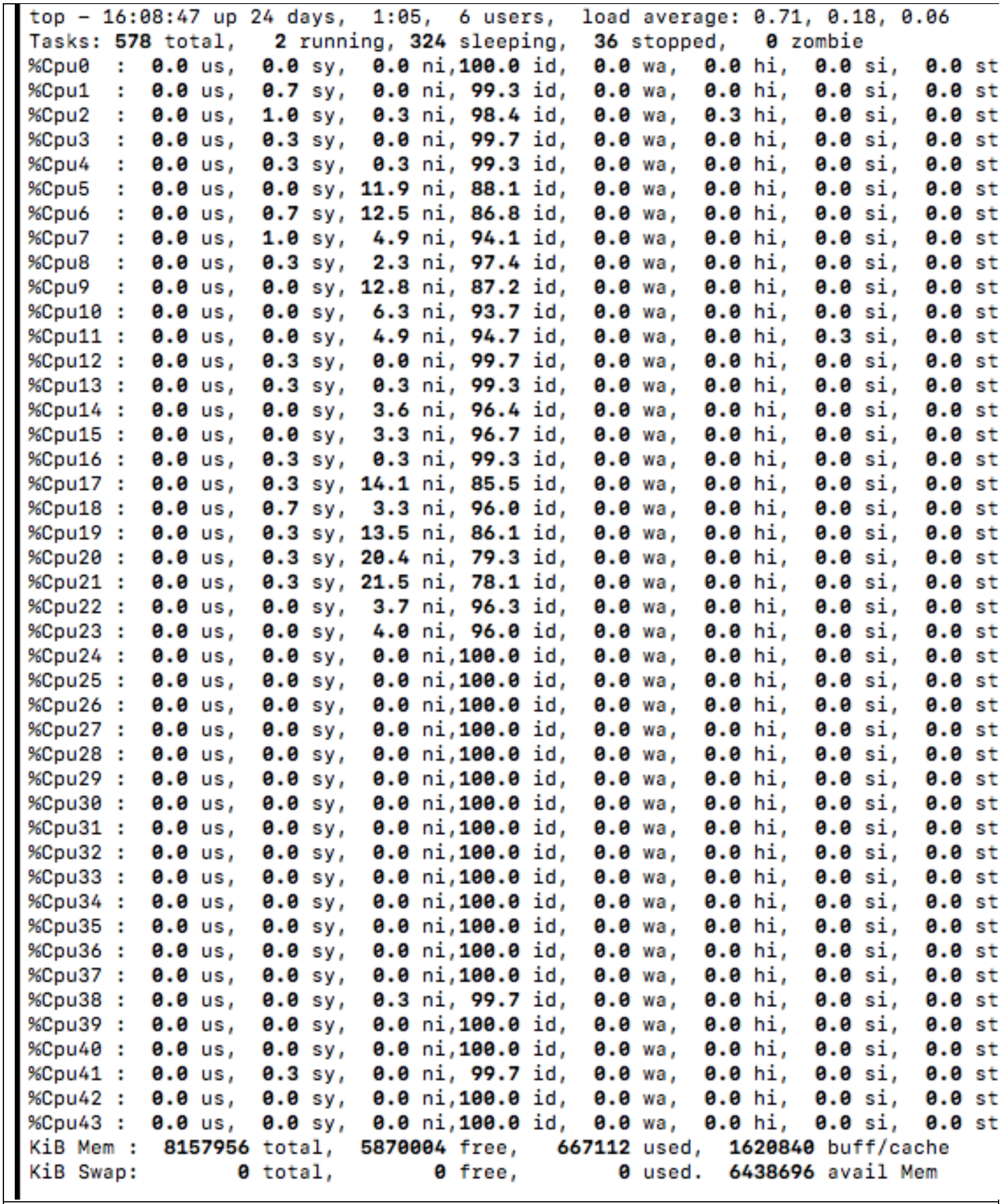
Project1
By Scott Fraundorf

I ran the tests of the algorithms using a shell script that timed the sort and wrote the results to csv files. Each trial started with an array size of 10 which was incremented by 100 on each iteration. The seed was consistently 10. Number of buckets was not carefully controlled, and that could affect the results with bucket bubble sort. In the graphs, all times recorded are in milliseconds.

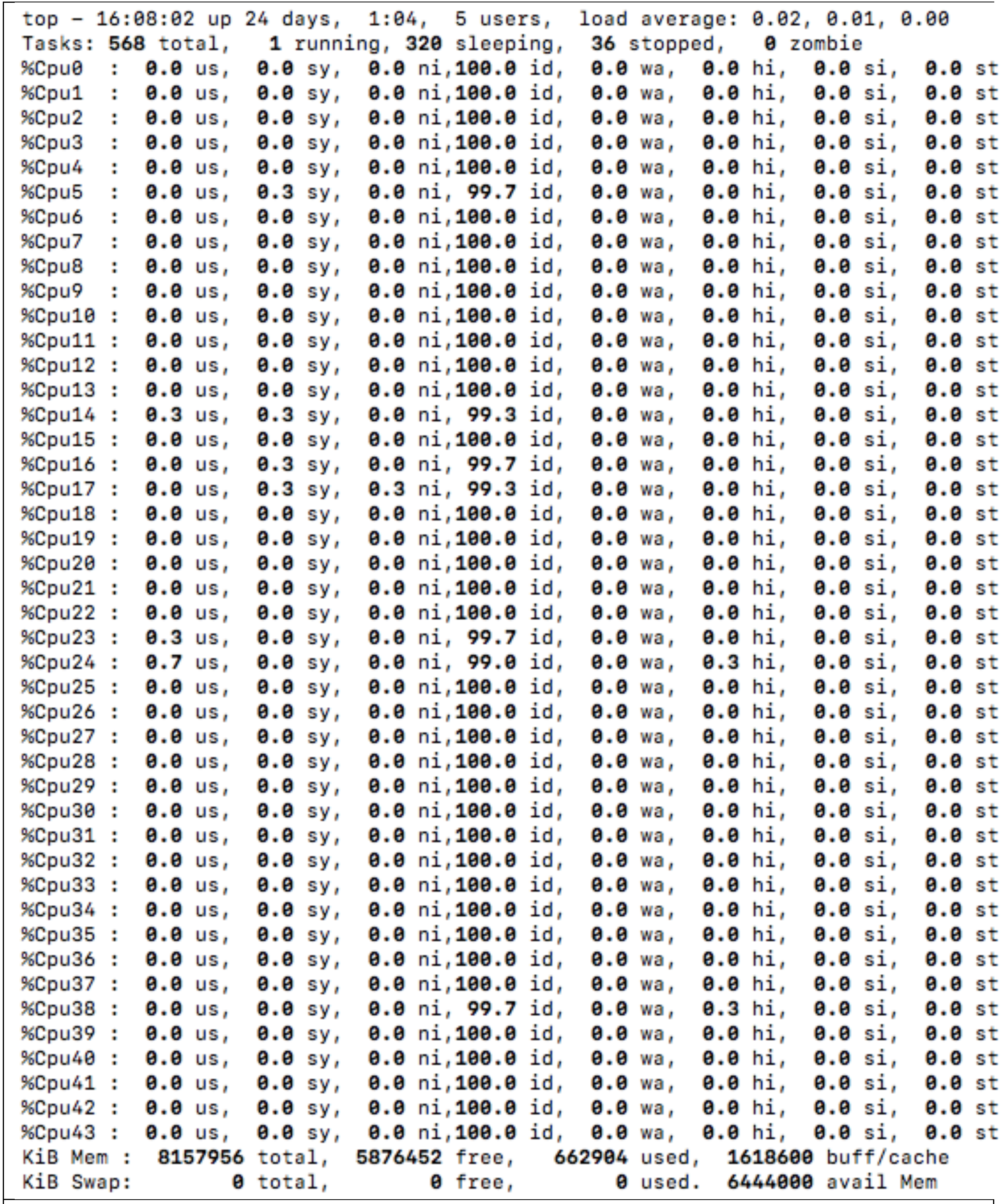


Section 1: Bucket bubble sort on home:
These results are expected because the performance of bubble sort. Bubble sort has a polynomial performance, and clearly the stl reference sort (grey) leaves it in the dust. For the most part the parallel and the sequential versions of the bucket bubble sort perform very similarly. As implemented, OpenMP threads (the two run on home) are not resulting in much of a performance advantage. It is noticeable that there are a few times where the parallel algorithm had a smaller time than the sequential algorithm. There is also one sort where the parallel performed significantly worse.



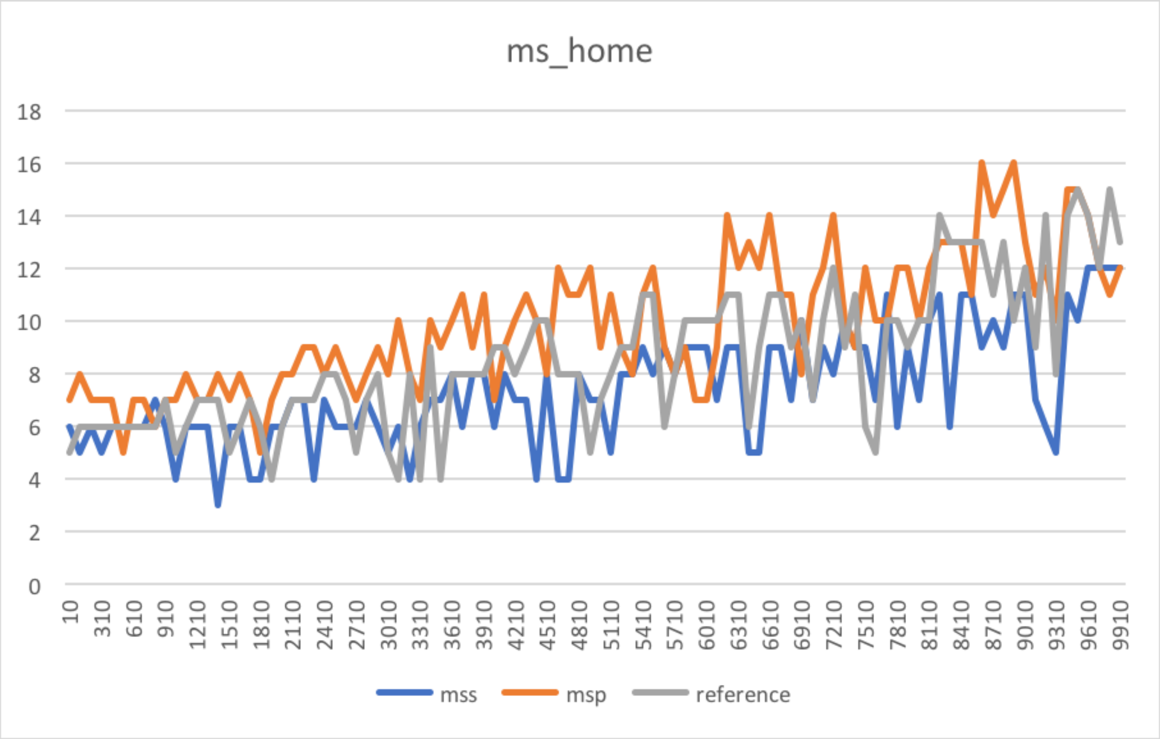
```
top - 16:08:47 up 24 days,  1:05,  6 users,  load average: 0.71, 0.18, 0.06
Tasks: 578 total,   2 running, 324 sleeping,  36 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.7 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  1.0 sy,  0.3 ni, 98.4 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.3 sy,  0.3 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy, 11.9 ni, 88.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.7 sy, 12.5 ni, 86.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  1.0 sy,  4.9 ni, 94.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.3 sy,  2.3 ni, 97.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy, 12.8 ni, 87.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  6.3 ni, 93.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  4.9 ni, 94.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu12 :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.3 sy,  0.3 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.0 us,  0.0 sy,  3.6 ni, 96.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.0 us,  0.0 sy,  3.3 ni, 96.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 :  0.0 us,  0.3 sy,  0.3 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 :  0.0 us,  0.3 sy, 14.1 ni, 85.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :  0.0 us,  0.7 sy,  3.3 ni, 96.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.3 sy, 13.5 ni, 86.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.0 us,  0.3 sy, 20.4 ni, 79.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  0.0 us,  0.3 sy, 21.5 ni, 78.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  0.0 us,  0.0 sy,  3.7 ni, 96.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  0.0 us,  0.0 sy,  4.0 ni, 96.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu24 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu25 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu26 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu27 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu28 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu29 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu30 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu31 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu32 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu33 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu34 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu35 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu36 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu37 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu38 :  0.0 us,  0.0 sy,  0.3 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu39 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu40 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu41 :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu42 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu43 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8157956 total,  5870004 free,   667112 used,  1620840 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  6438696 avail Mem
```

Top with tests running



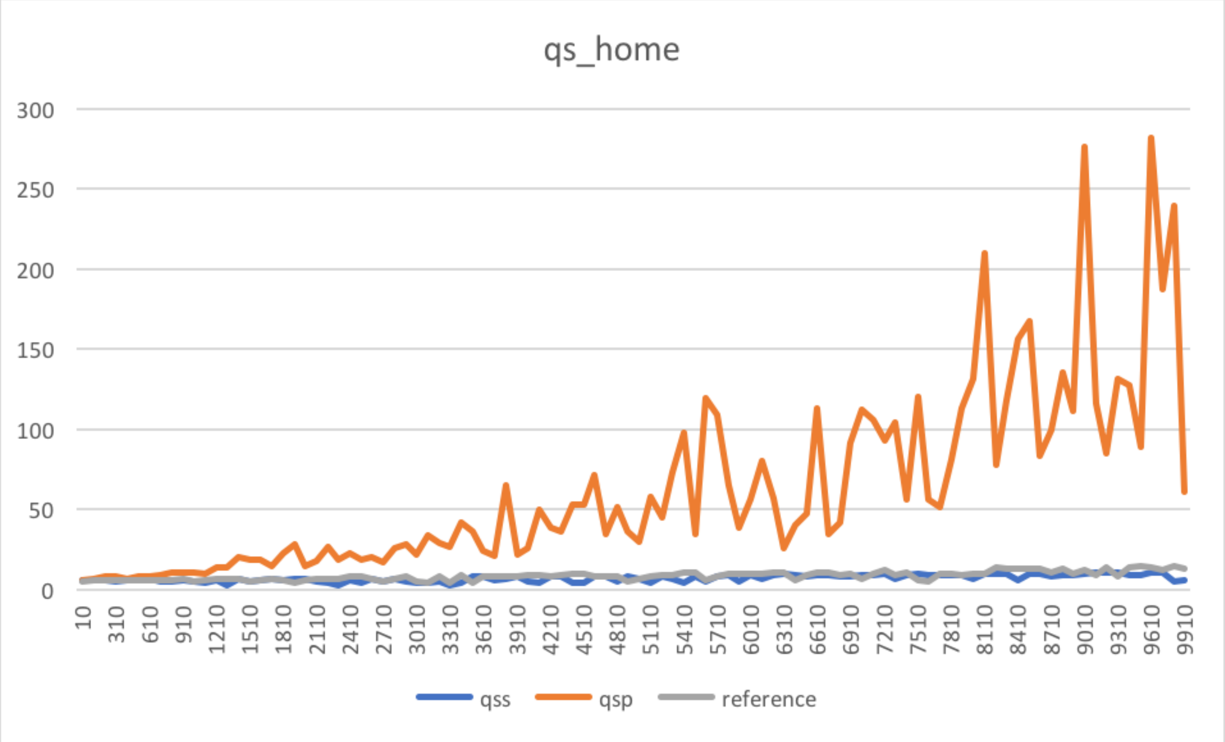
```
top - 16:08:02 up 24 days,  1:04,  5 users,  load average: 0.02, 0.01, 0.00
Tasks: 568 total,   1 running, 320 sleeping,  36 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 :  0.0 us,  0.3 sy,  0.3 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu24 :  0.7 us,  0.0 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
%Cpu25 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu26 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu27 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu28 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu29 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu30 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu31 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu32 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu33 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu34 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu35 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu36 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu37 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu38 :  0.0 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
%Cpu39 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu40 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu41 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu42 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu43 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8157956 total,  5876452 free,   662904 used,  1618600 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  6444000 avail Mem
```

Top without tests running.



Section 2: Merge sort on home:

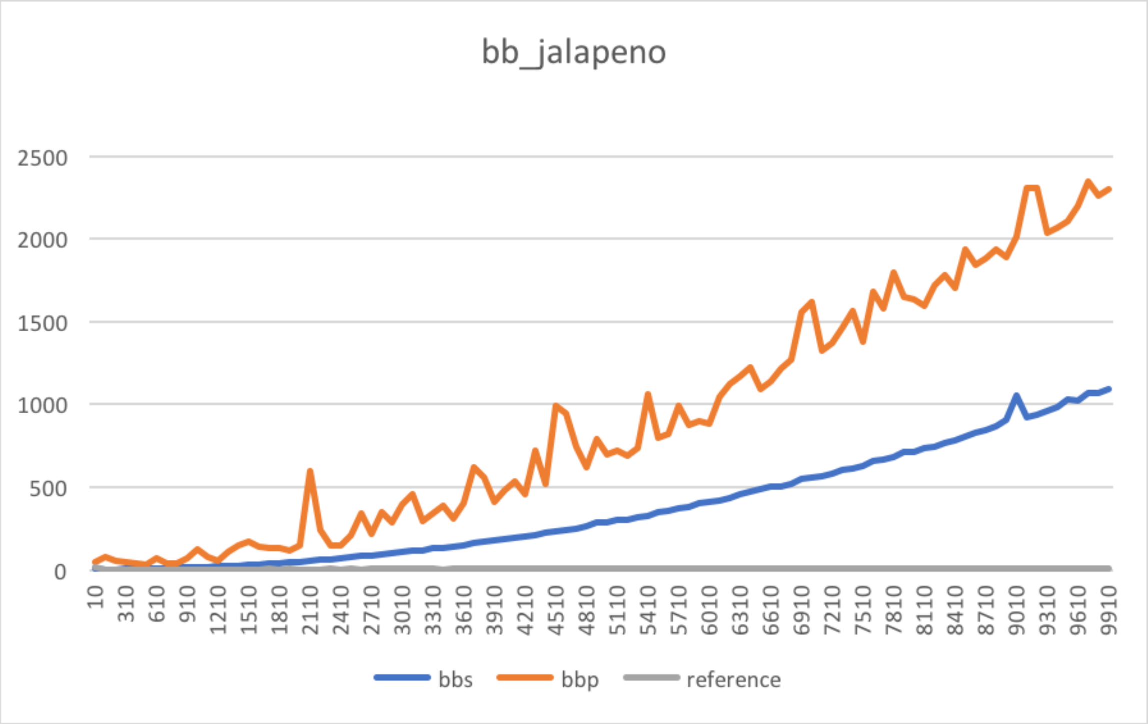
Clearly the three algorithms are peforming similarly. Merge sort has a similar asymptotic behavior to the reference stl sort, probably because the reference uses an nlogn algorithm. Again, the parallel version does not seem to have much an advantage. One reason for this, is that the merge function was implemented as a critical section. Attempts were made to make finer critical sections within the merge function, but these caused failure in the unit test, even if it sorted successfully some or most of the time.



Section 3: Quick sort on home:

Clearly in this case, the parallel version is performing much worse than either the sequential version or the reference stl sort. Below, you can see how the quick sort parallel version was implemented. The recursive part of the program is implemented as tasks. Perhaps this is the problem. The overhead of creating tasks within tasks outweighs any performance benefit.

```
void quick_sort(int *array, int p, int r)
{

        int q = 0;


        if(p < r)
        {

#pragma omp critical
                q = partition(array, p, r);

#pragma omp task
                {

                        quick_sort(array, p, q - 1);
                }//end parallel

#pragma omp task
                {
                        quick_sort(array, q + 1, r);
                }//end parallel

        }//end if
}//end function
```



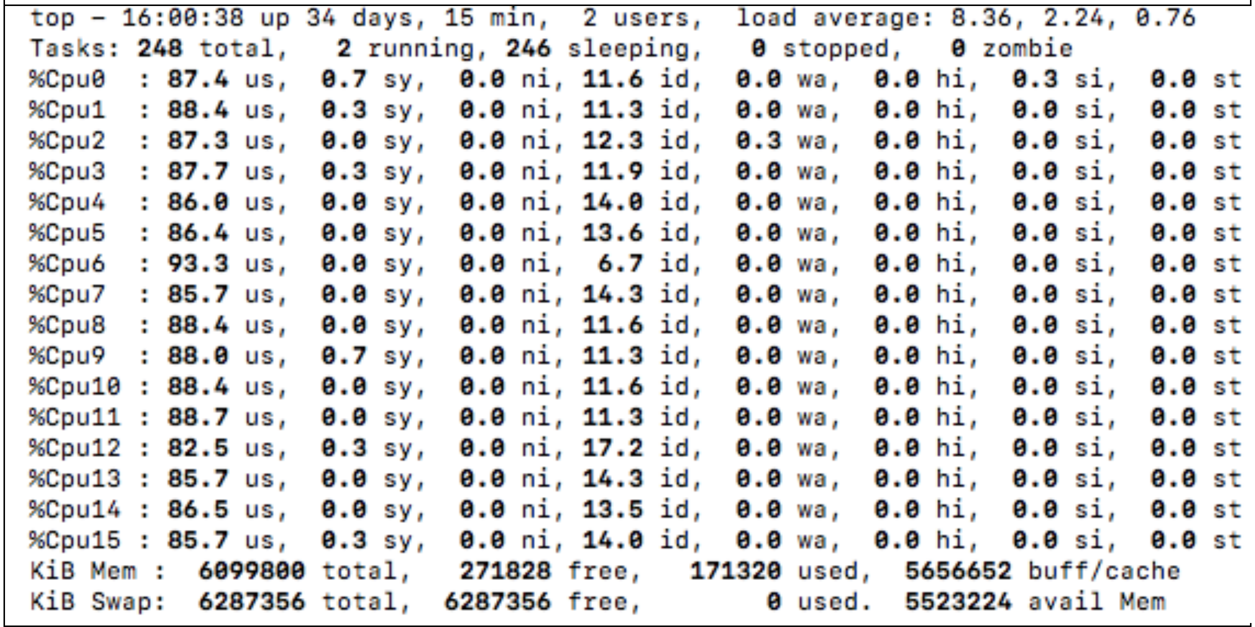
Section 4: Bucket bubble sort, merge sort, and quick sort on Jalapeno with Hyperthreading: On all of the tests on Jalapeno with max 16 threads and hyperthreading on, the performance was significantly worse in the multithreaded versions of the algorithms. The reason it must be performing worse, is because the overhead is increasing for starting the increased number of threads (i.e. 16 vs. 2). There isn't a corresponding benefit to the hyperthreading on Jalapeno. This isn't because the cores are not being used as can be seen from the image from top.

The problems in the way multithreading was implemented in quick sort, tasks within tasks, is increased here. The exponential graph with quick sort both with and without hyperthreading suggest that maybe the OpenMP is not limiting it to the specified number of threads. Maybe because of the tasks within tasks.

Below is a result of lscpu on the Jalapeno server.

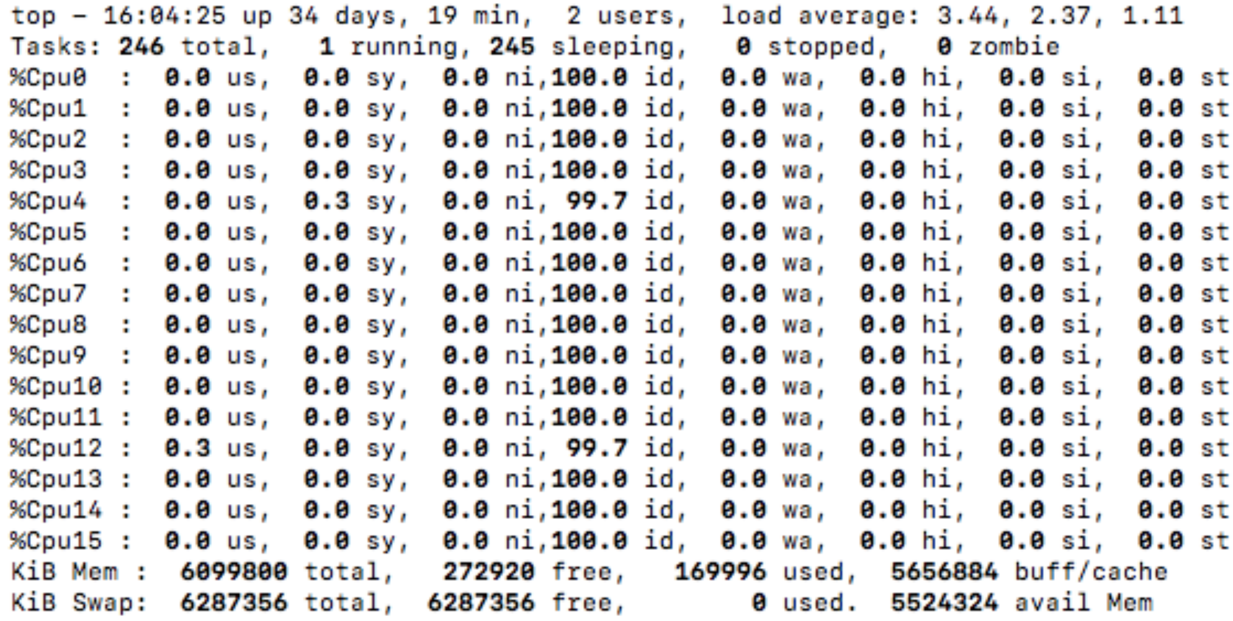
```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:   0-15
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             2
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Model name:            Intel(R) Xeon(R) CPU           E5620  @ 2.40GHz
Stepping:              2
```

```
CPU MHz:                2393.940
BogoMIPS:               4787.97
Virtualization:         VT-x
L1d cache:              32K
L1i cache:              32K
L2 cache:               256K
L3 cache:               12288K
NUMA node0 CPU(s):      0-15
```



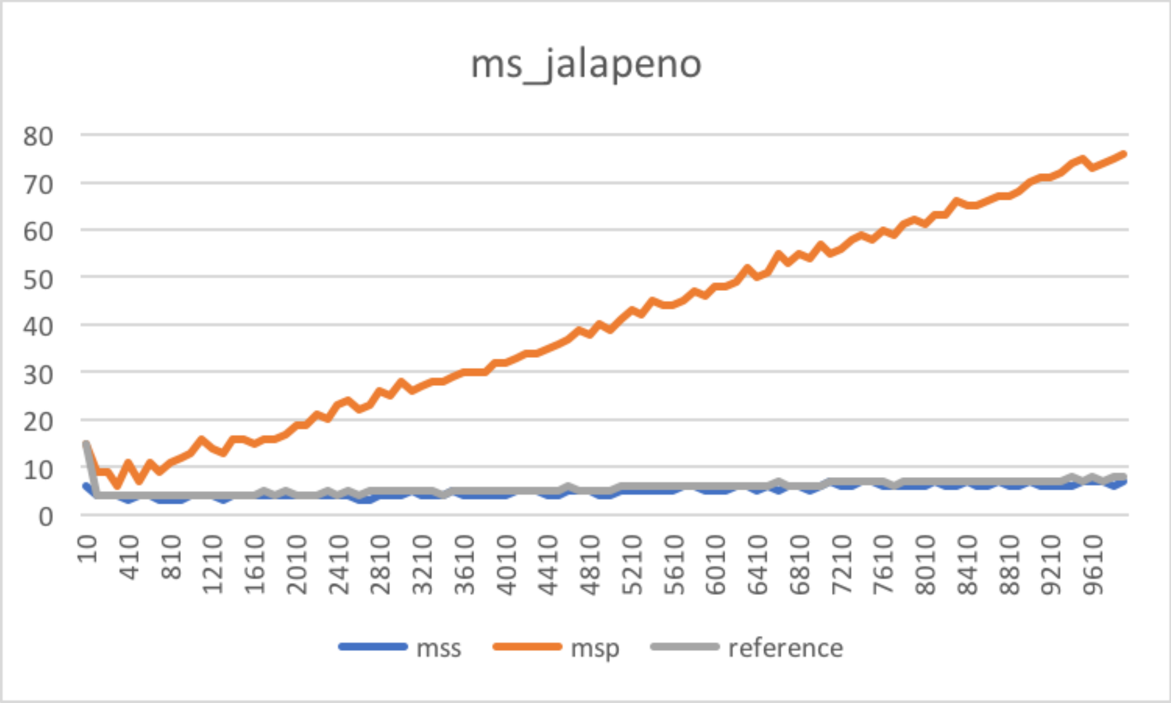
```
top - 16:00:38 up 34 days, 15 min,  2 users,  load average: 8.36, 2.24, 0.76
Tasks: 248 total,   2 running, 246 sleeping,   0 stopped,   0 zombie
%Cpu0  : 87.4 us,  0.7 sy,  0.0 ni, 11.6 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  : 88.4 us,  0.3 sy,  0.0 ni, 11.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 87.3 us,  0.0 sy,  0.0 ni, 12.3 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 87.7 us,  0.3 sy,  0.0 ni, 11.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 86.0 us,  0.0 sy,  0.0 ni, 14.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 86.4 us,  0.0 sy,  0.0 ni, 13.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 93.3 us,  0.0 sy,  0.0 ni,  6.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 85.7 us,  0.0 sy,  0.0 ni, 14.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  : 88.4 us,  0.0 sy,  0.0 ni, 11.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  : 88.0 us,  0.7 sy,  0.0 ni, 11.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 : 88.4 us,  0.0 sy,  0.0 ni, 11.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 : 88.7 us,  0.0 sy,  0.0 ni, 11.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 : 82.5 us,  0.3 sy,  0.0 ni, 17.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 : 85.7 us,  0.0 sy,  0.0 ni, 14.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 : 86.5 us,  0.0 sy,  0.0 ni, 13.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 : 85.7 us,  0.3 sy,  0.0 ni, 14.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  6099800 total,   271828 free,   171320 used,  5656652 buff/cache
KiB Swap:  6287356 total,  6287356 free,        0 used.  5523224 avail Mem
```

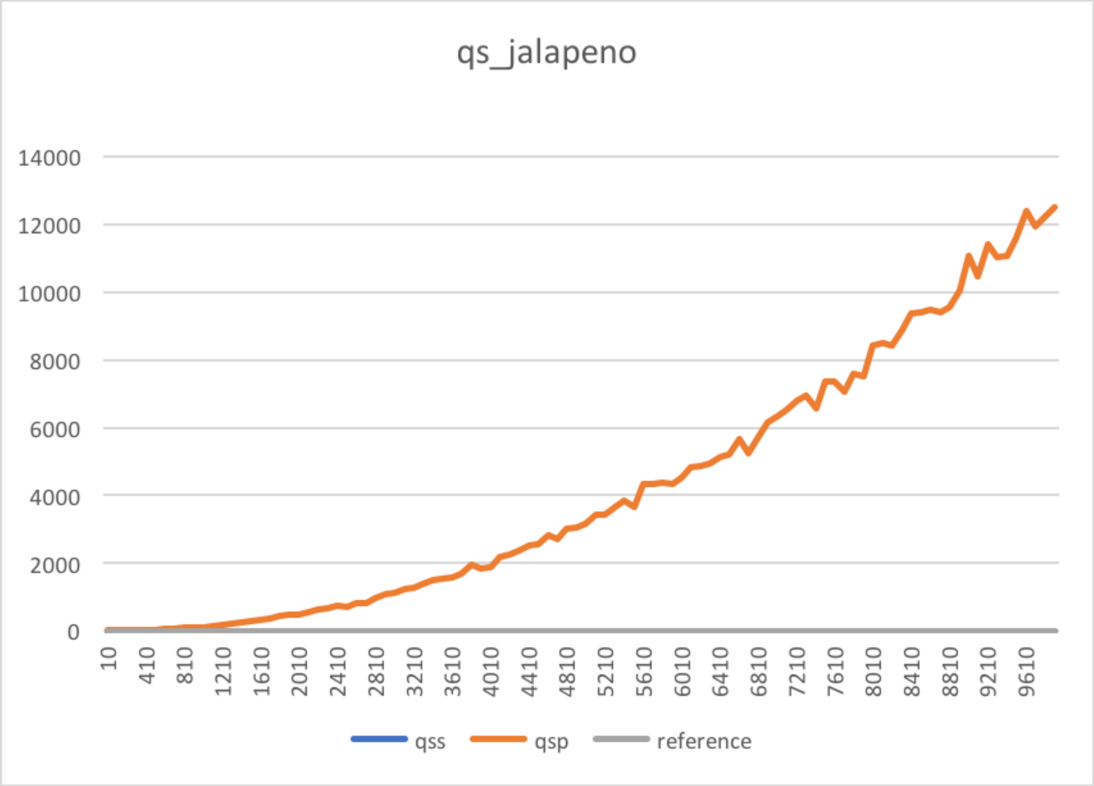
Top with tests running.



```
top - 16:04:25 up 34 days, 19 min,  2 users,  load average: 3.44, 2.37, 1.11
Tasks: 246 total,   1 running, 245 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  6099800 total,   272920 free,   169996 used,  5656884 buff/cache
KiB Swap:  6287356 total,  6287356 free,        0 used.  5524324 avail Mem
```

Top without tests running.



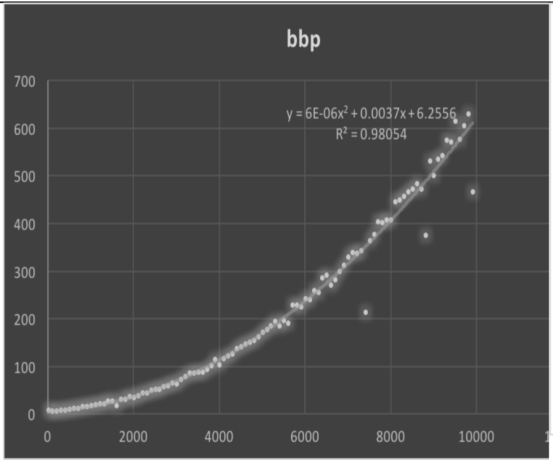


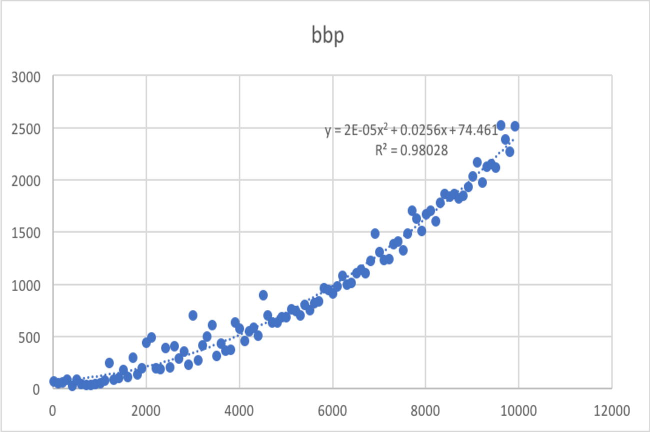
```
void unit_test_sort()
{
        for(int i = 10; i <= 100; i ++)
```

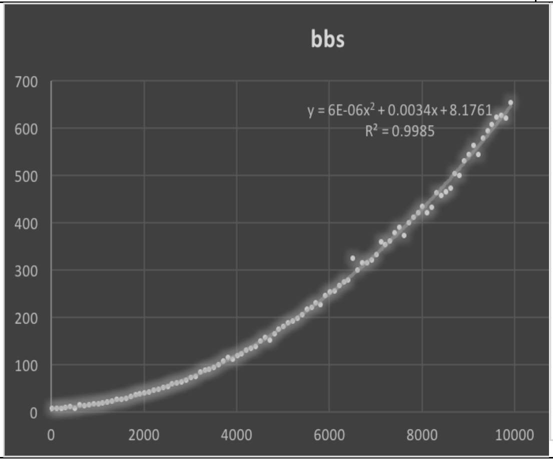
```
       {



              for(int j = 0; j <= 1000; j ++)
              {

                     int * array = randNumArray( i, j );

                     int array2[i];

                     for(int k = 0; k <= i - 1; k ++)
                     {

                            array2[k] = array[k];

                     }//end for k



#pragma omp parallel
                     mergesort(array, 0, i - 1);

              /* Verifying that all members of the unsorted array are in the sorted
array. (i.e. no missing
numbers) */

                     for(int k = 0; k <= i - 1; k ++)
                     {

                            bool number_in_sorted_array = false;

                            for(int z = 0; z <= i - 1; z ++)
                            {

                                   if(array2[k] == array[z])
                                   {

                                          number_in_sorted_array = true;

                                   }//end if


                            }//end for z

                            assert(number_in_sorted_array == true);

                     }//end for k


/* Verifying sort result in ascending order */

                     for(int k = 1; k <= i - 1; k ++)

                            assert(array[k] >= array[k - 1]);


                     }//end for k

              }//end for j
```

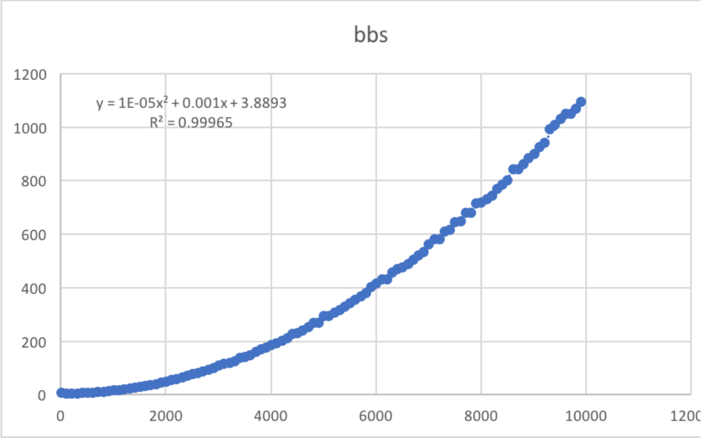
```
        }//end for i

}//end function
```

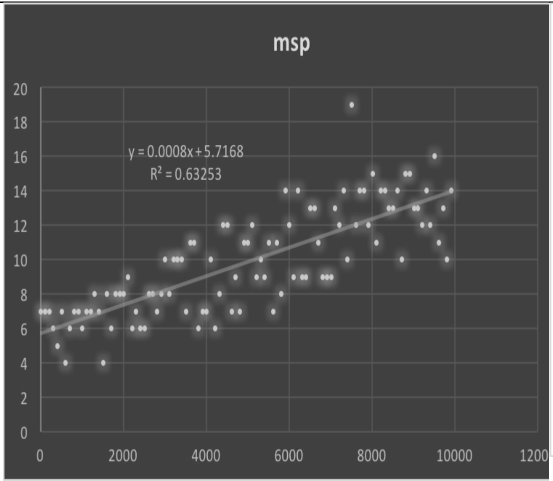
Unit test function, used to check if the function is working properly. Tests with array sizes of 10 to 100, and 1000 trials for each array size with a different seed. All algorithms passed the test.

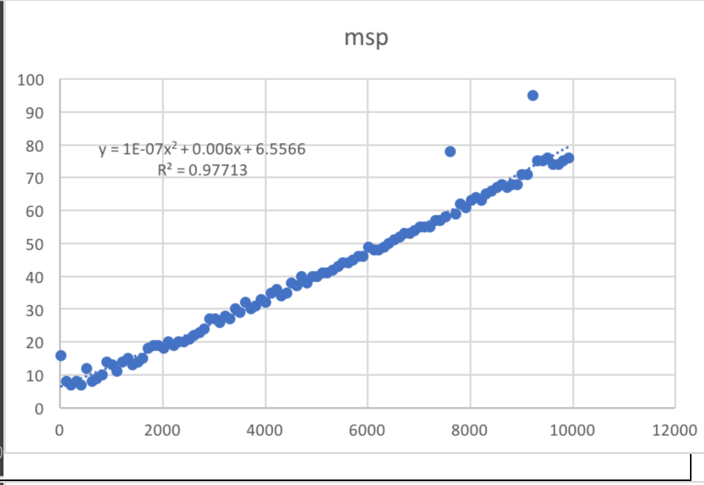


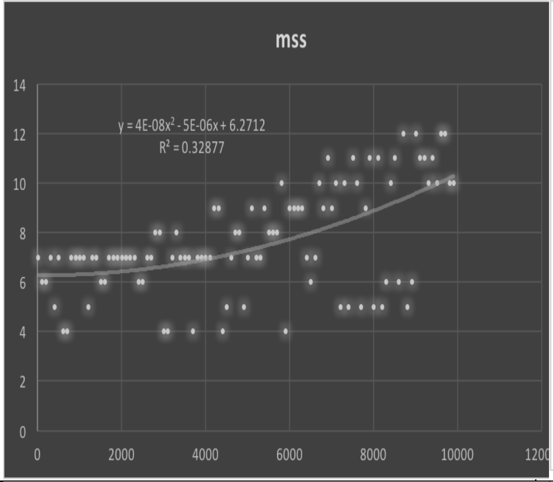


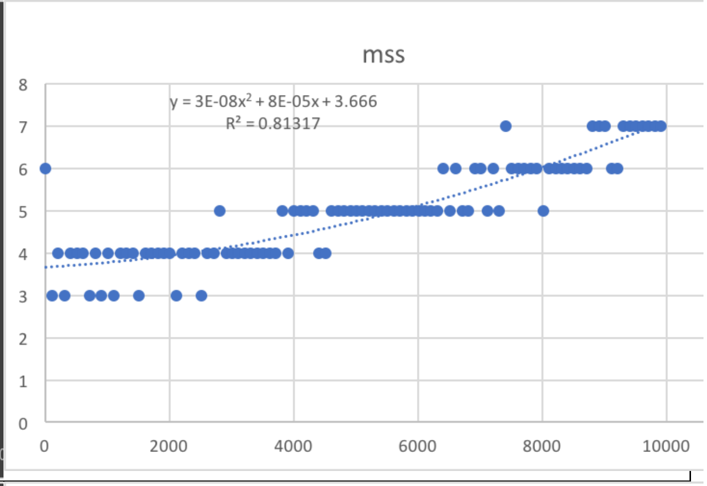


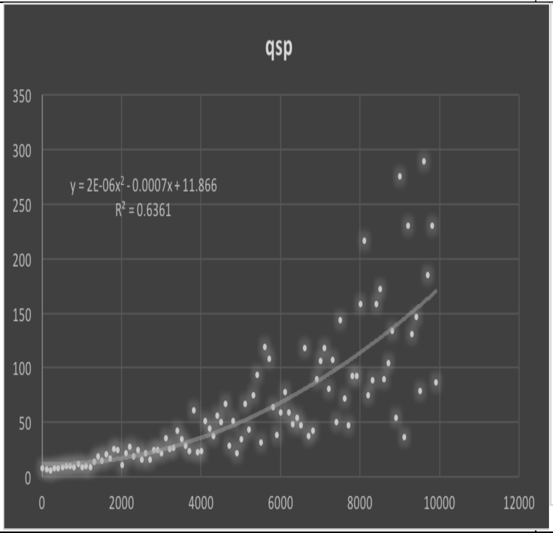


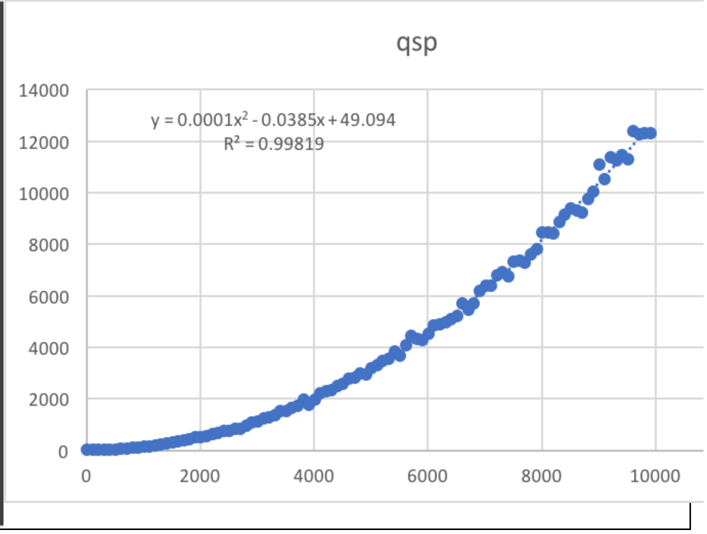

msp
y = 0.0008x + 5.7168
R² = 0.63253
20
18
16
14
12
10
8
6
4
2
0
0
2000
4000
6000
8000
10000
1200

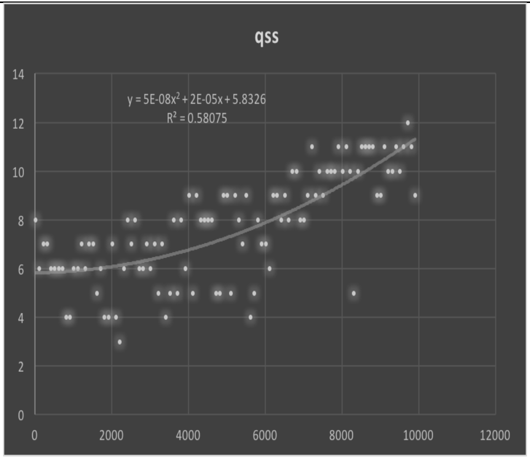
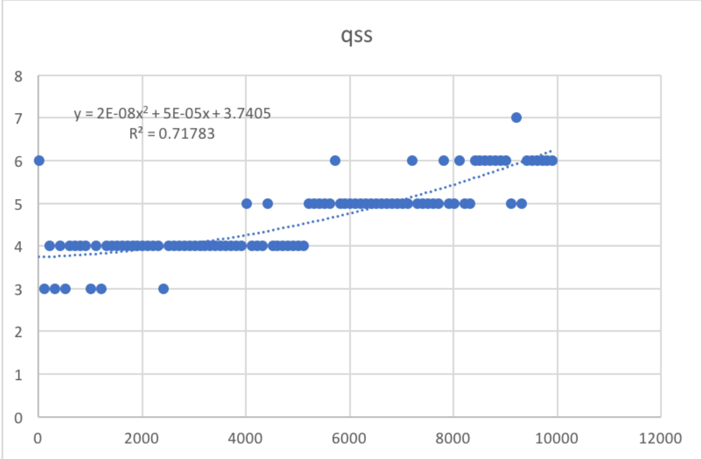
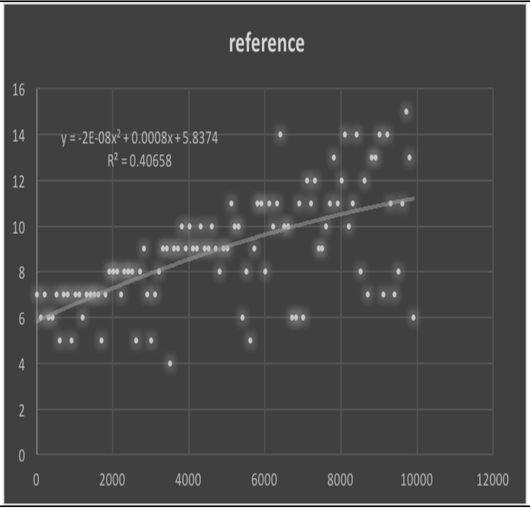
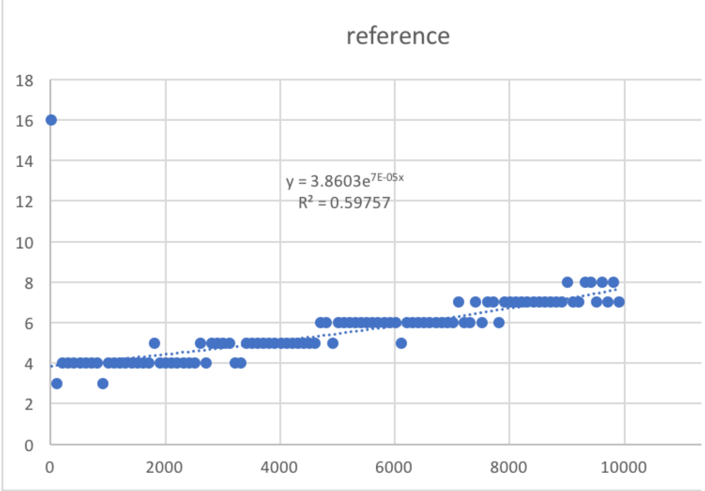

msp
y = 1E-07x² + 0.006x + 6.5566
R² = 0.97713
100
90
80
70
60
50
40
30
20
10
0
0
2000
4000
6000
8000
10000
12000


mss
y = 4E-08x² - 5E-06x + 6.2712
R² = 0.32877
14
12
10
8
6
4
2
0
0
2000
4000
6000
8000
10000
1200


mss
y = 3E-08x² + 8E-05x + 3.666
R² = 0.81317
8
7
6
5
4
3
2
1
0
0
2000
4000
6000
8000
10000


qsp
y = 2E-06x² - 0.0007x + 11.866
R² = 0.6361
350
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000


qsp
y = 0.0001x² - 0.0385x + 49.094
R² = 0.99819
14000
12000
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000

	
	
Run on home server with max 2 threads	Run on Jalapeno server with max 16 threads