QUADOTS MEASUREMENTS

Souren Papazian (ssp2155)
Surashree Kulkarni (ssk2197)
Srilakshmi Chintala (sc3772)

INTRODUCTION

Quadots is a 2D Simulation library that allows the user to create points in 2D space and assign behavior to them. The library should be used for any application that depends on changing point behavior based on the behavior of its closest neighbors. Quadots uses Quad tree as the data structure to hold the points created by the user. The main focus of this document is the *getNearestNeighbour*() function. We measure the time taken to run the simulation by varying the following features:

- Density - Number of points added to the simulation by the user. In our measurements, they range from 10 to 10000.
- MAX_OBJECTS - This is the number of points each node in a quad tree can hold before it splits. MAX_OBJECTS = {x} means that a quad tree node will split into 4 quadrants if more than x objects are inserted into it.

We also look at the Brute Force approach that performs NxN checks to find points closest to a point. We compare it's performance against Quadots.

From the measurements, we get observations and make a conclusion.

NOTE: All measurements have been done on a 2.7GHz dual-core Intel Core i5 machine.

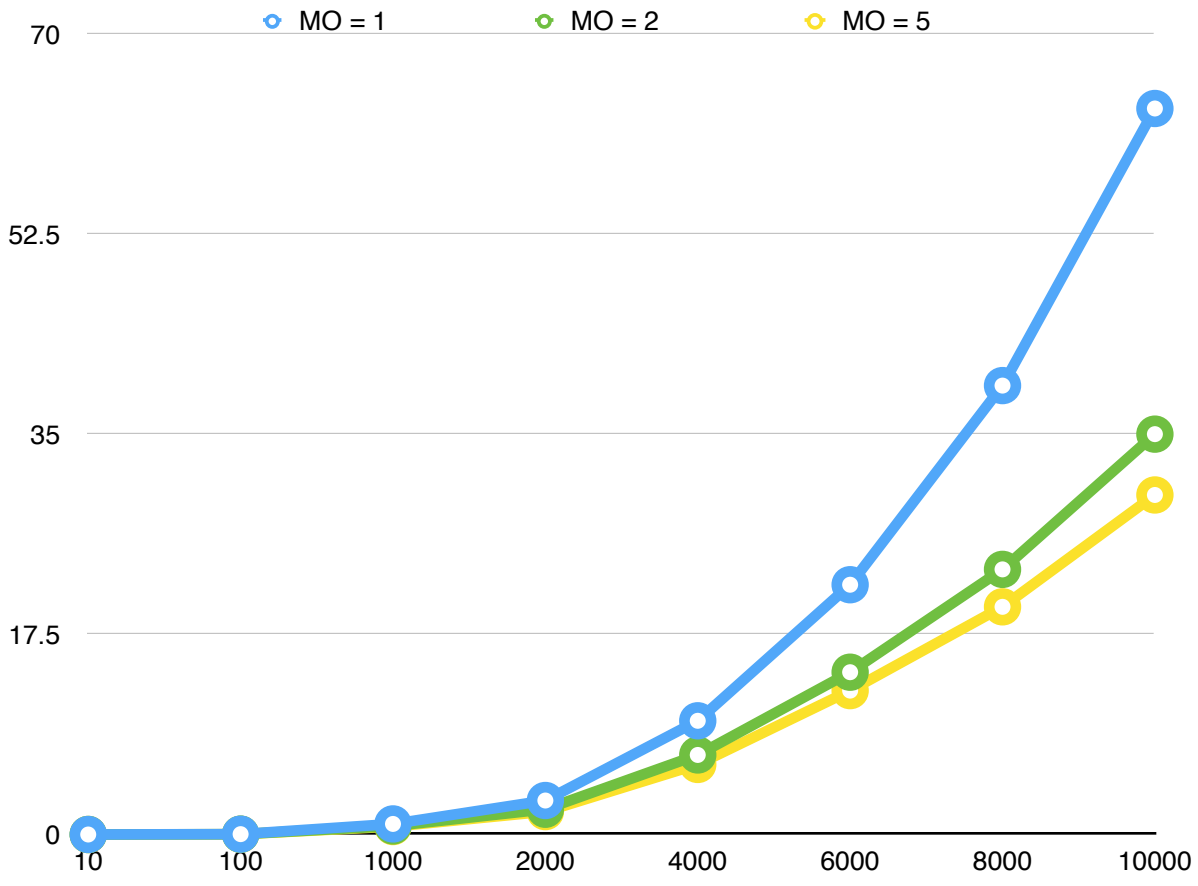
MEASURING NEAREST NEIGHBORS FUNCTION

A) USING QUADTREES

1. Range (Distance of nearest neighbor from a point) = 50, Run Steps = 50

MO (MAX_OBJECTS) = {1,2,5}

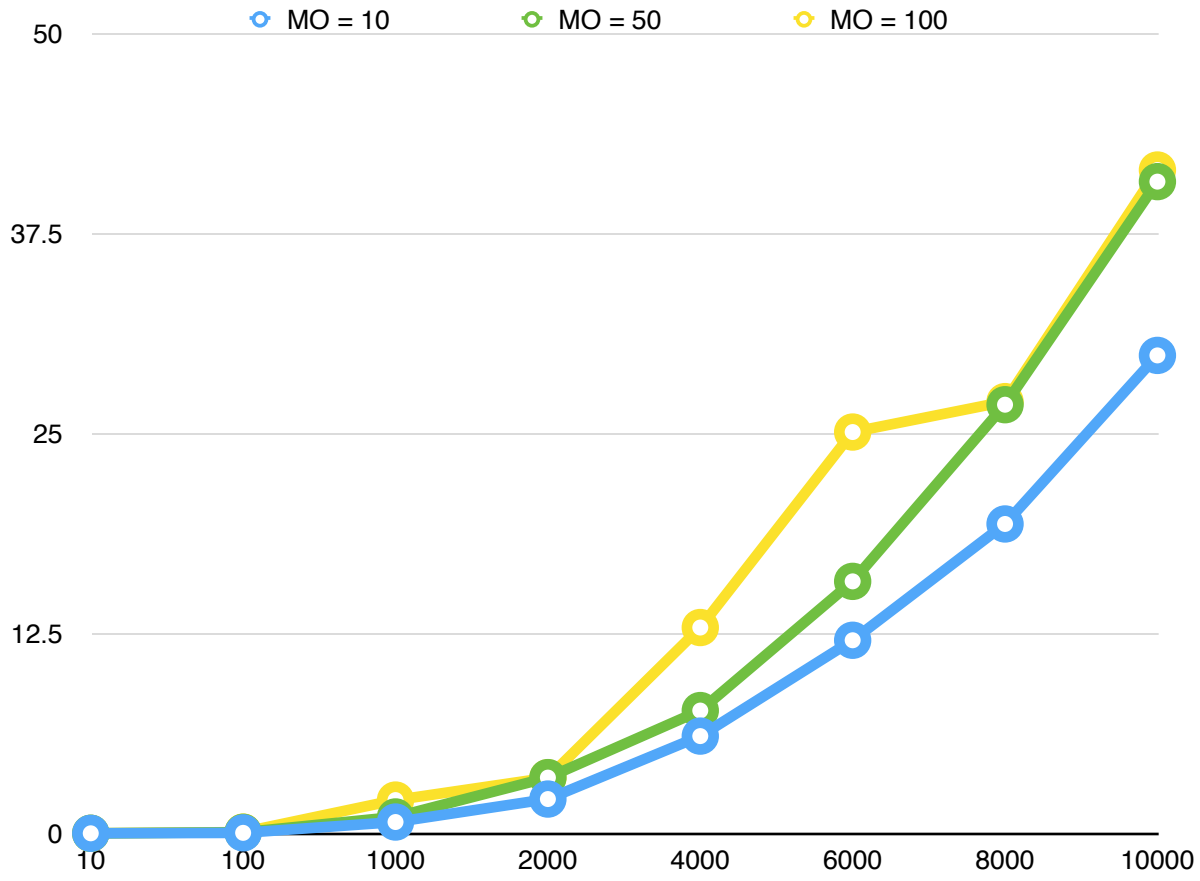
MO	10	100	1000	2000	4000	6000	8000	10000
1	0.00232	0.033657	0.927885	2.97609	9.95166	21.8577	39.282	63.5282
2	0.001183	0.028265	0.749425	2.2113	6.96298	14.2031	23.2009	35.0338
5	0.0031651	0.027077	0.698004	2.006	6.13335	12.593	19.9242	29.7115



2. Range = 50, Run Steps = 50

MO (MAX_OBJECTS) = {10,50,100}

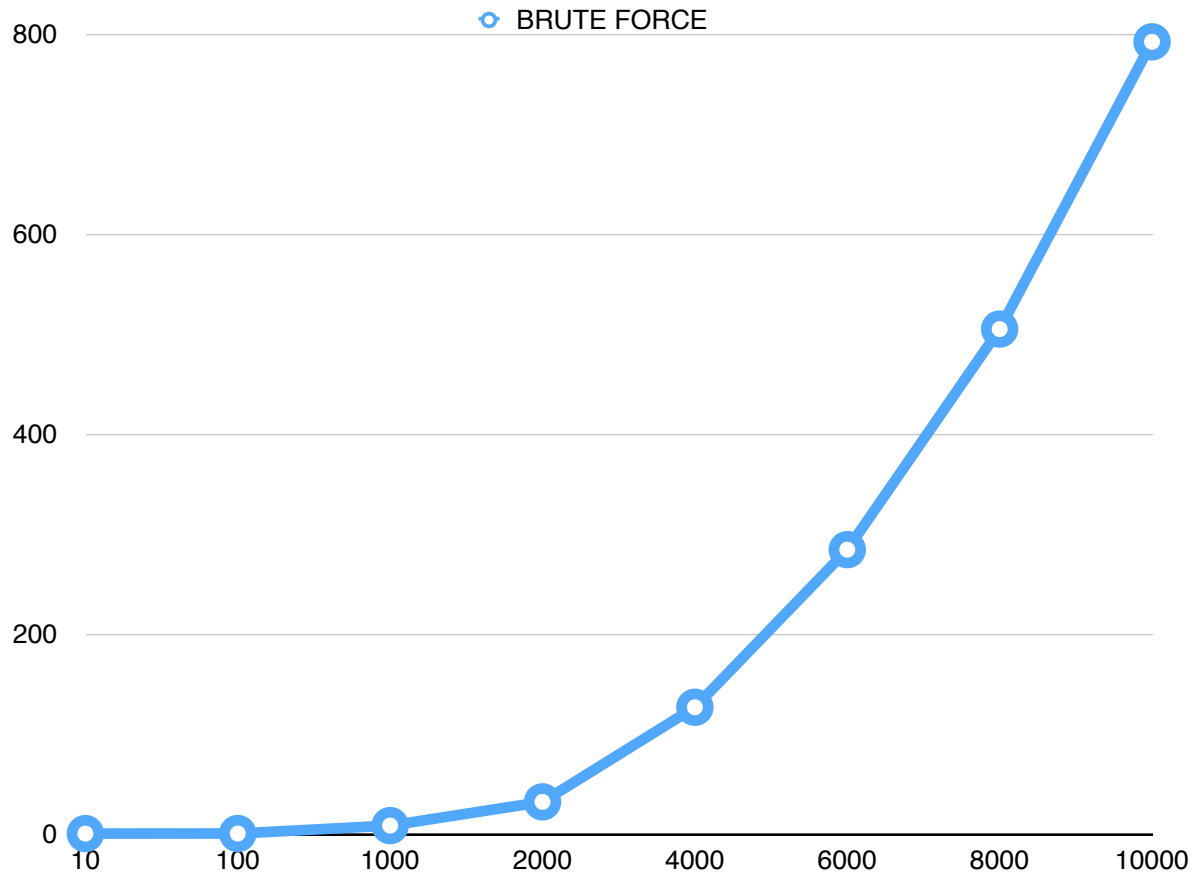
MO	10	100	1000	2000	4000	6000	8000	10000
10	0.001302	0.030299	0.70659	2.14257	6.07703	12.0665	19.3351	29.8704
50	0.001302	0.06456	1.02358	3.48238	7.67288	15.7477	26.789	40.7312
100	0.001307	0.075789	2.07863	3.47315	12.8625	25.0805	26.9823	41.4569



B) USING BRUTE FORCE

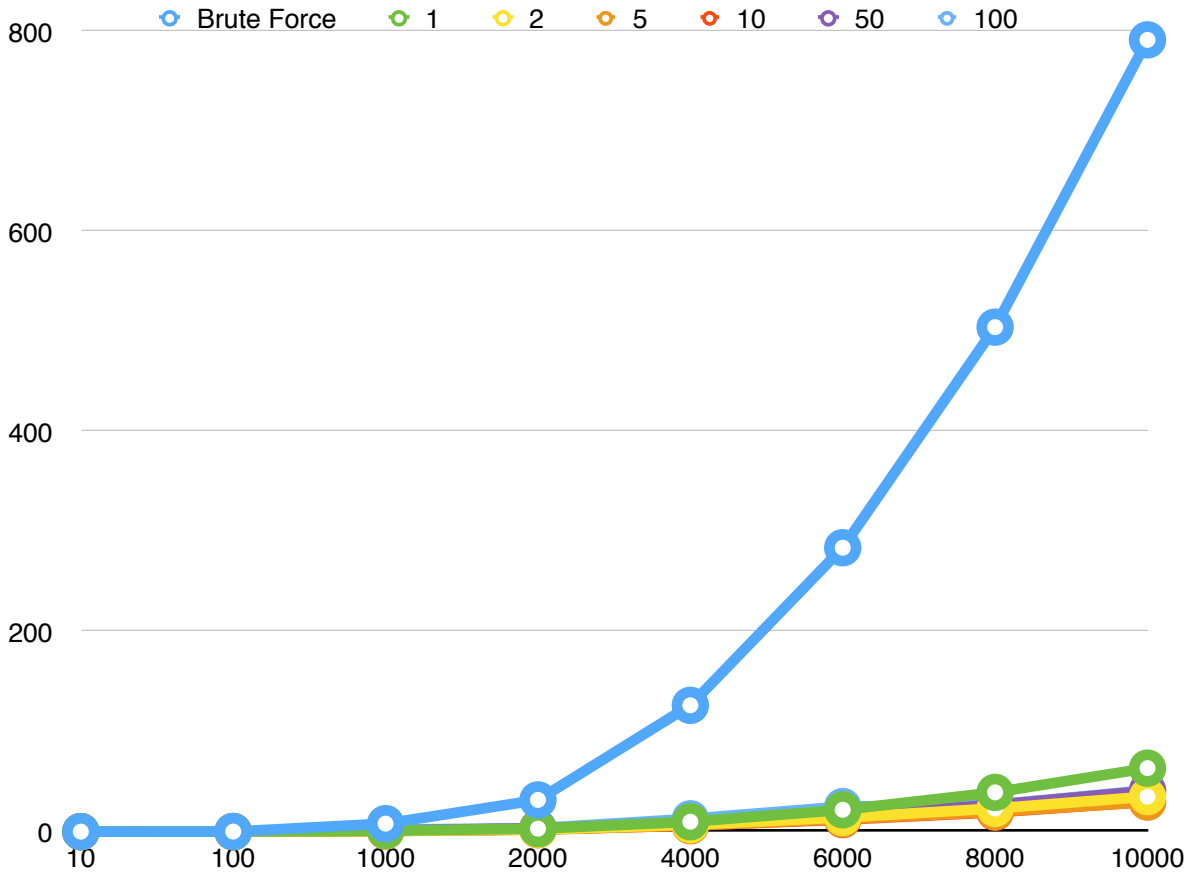
Range = 50, Run Steps = 50

10	100	1000	2000	4000	6000	8000	10000
0.001082	0.083717	8.02135	31.725	126.333	283.941	504.526	791.782



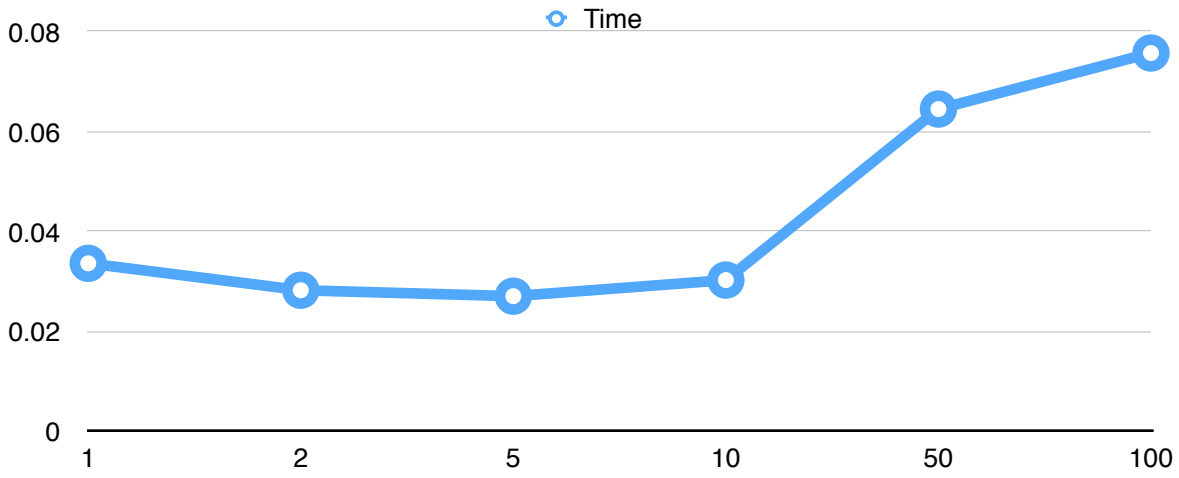
QUADTREES VS. BRUTE FORCE

MO	10	100	1000	2000	4000	6000	8000	10000
BF	0.001082	0.083717	8.02135	31.725	126.333	283.941	504.526	791.782
1	0.002321	0.033657	0.927885	2.97609	9.95166	21.8577	39.282	63.5282
2	0.001183	0.028265	0.749425	2.2113	6.96298	14.2031	23.2009	35.0338
5	0.003165	0.027077	0.698004	2.006	6.13335	12.593	19.9242	29.7115
10	0.001302	0.030299	0.70659	2.14257	6.07703	12.0665	19.3351	29.8704
50	0.001302	0.064561	1.02358	3.48238	7.67288	15.7477	26.789	40.7312
100	0.001307	0.075789	2.07863	3.47315	12.8625	25.0805	26.9823	41.4569

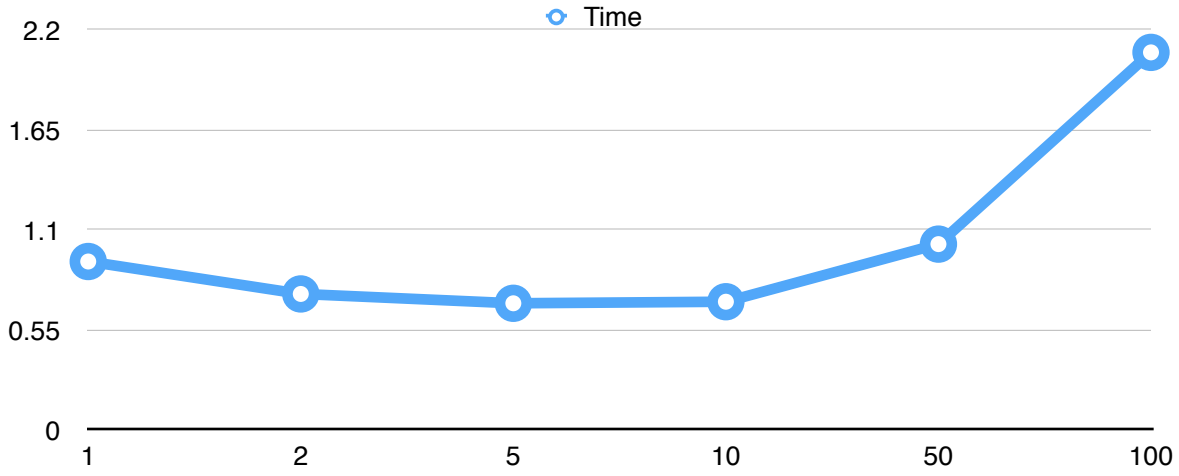


MAX_OBJECTS ANALYSIS

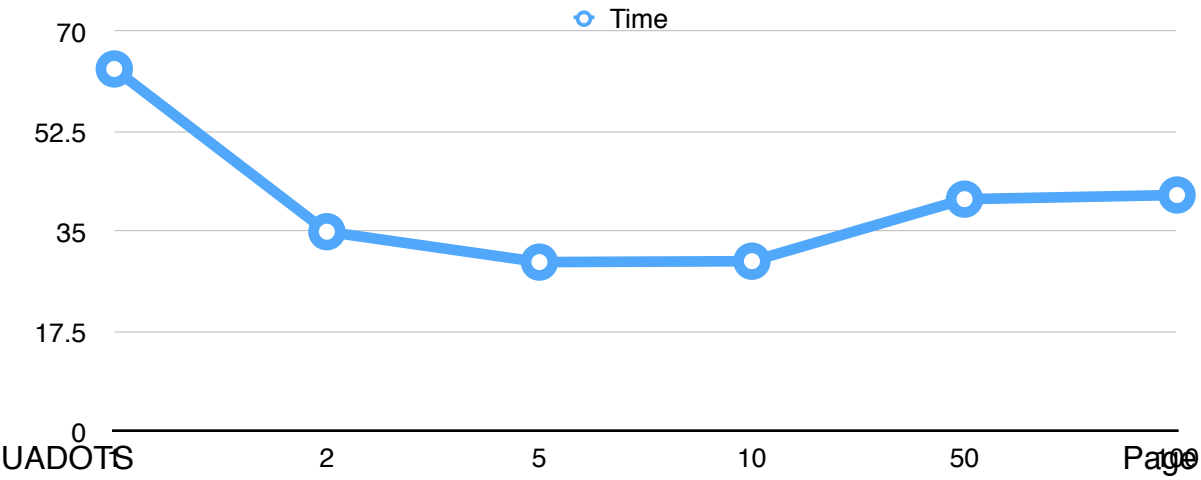
C) 1. # of points in simulation = 100



2. # of points in simulation = 1000



3. # of points in simulation = 10000



OBSERVATIONS

A1. - We plot the performance of quad trees by varying density of the simulation from very low to very high. We keep the MAX_OBJECTS low (1,2,5).

i) The nearest neighbor function using quad trees requires about the same time when the density of points in the simulation is low (< 1000), regardless of the maximum objects that can be contained in a node of a quad tree.
ii) However, there's a noticeable difference when the density increases, especially at No. of points = 10000. Time taken when MO = {2,5} is almost 50% of that taken when MO = 1.
iii) MO = 5 performs better than MO = 2 for density = 10000 by almost 6 s.

A2. - We plot the performance of quad trees by varying density of the simulation from very low to very high. We keep the MAX_OBJECTS high (10,50,100).

i) Again, the function behaves similarly in terms of time for very low density (<100) when MO = {10,50,100}.
ii) As density increases, performance becomes worse with increasing MO.
iii) At much higher density(<= 10000), MO = {50,100} takes almost 30% more time than MO = {10}
iv) MO = {5} and MO = {10} perform almost similarly, with the latter being slightly worse than the former.
v) If MO = density, performance is worse than for MO < density.

B/C - We check how the Brute Force approach performs against quad trees. We use the measurements from part A1 and A2.
i) Brute force performs as well as quad trees for very low density (< 100).
ii) As density increases, the time taken by Brute Force increases exponentially.
iii) For density = 1000, quad trees are 8x faster.
iv) For density = 10000, quad trees are almost 25x faster.

C - We try to find the optimal number of MAX_OBJECTS by keeping the density constant and changing MAX_OBJECTS.

i) For low densities, MO = {5,10} does marginally better than others, but for moderate to high densities, MO = {5,10} performs best.
ii) MO = 1 performs poorly in case of high density.
iii) MO = {50,100} shows unpredictable behavior for high densities, but still remains worse than MO = {5,10}.

CONCLUSION

- Quad trees are clearly faster than brute force because of the number of comparisons performed per time frame.
- If MO for a quad tree is greater than or equal to density, it reduces to brute force.
- As MO for a quad tree increases, its performance becomes noticeably worse. This tells us that there may be an optimum value of MO for the best performance. From our measurements, we conclude that MO = 5 to10 works best. We expected optimum value of MO to change with the density of the simulation, however, we found that MO = {5,10} always does better than all others.
- Brute force and quad trees give similar performance times for low densities, but the former's time grows exponentially, and is very high when density is >1000.