Question 5

It would appear after running multiple tests and taking averages that nested for-loops is the absolute worst algorithm by many orders of magnitude (10^3 to be exact for the 1024000 length list). Otherwise, it is not entirely which of the remaining three are the absolute best, although it appears binary search was the fastest in the most number of cases.

a) Report a table for the average running time for a single list intersection for each method and list sizes.

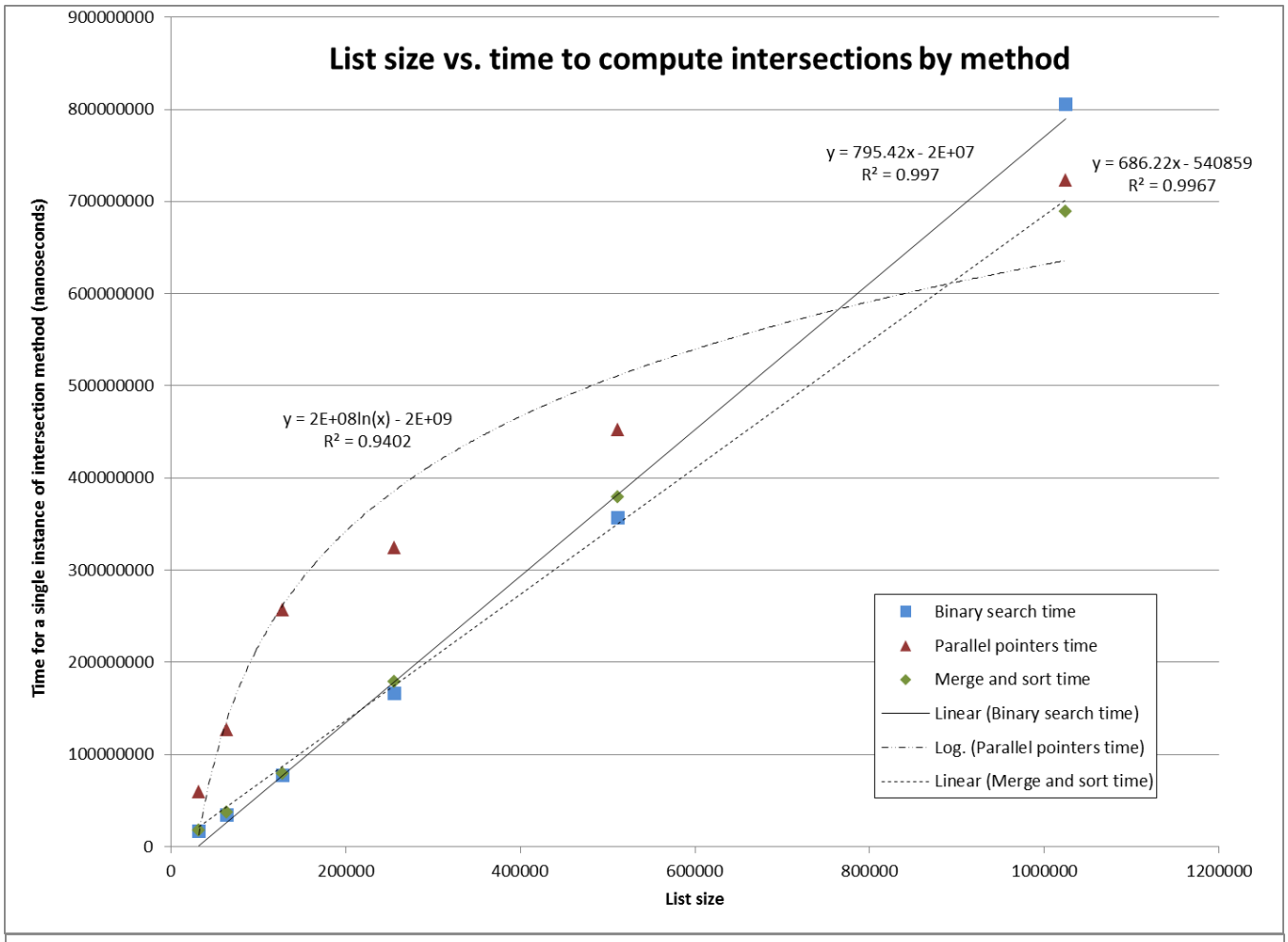
Time for a single instance of four intersection methods in nanoseconds*

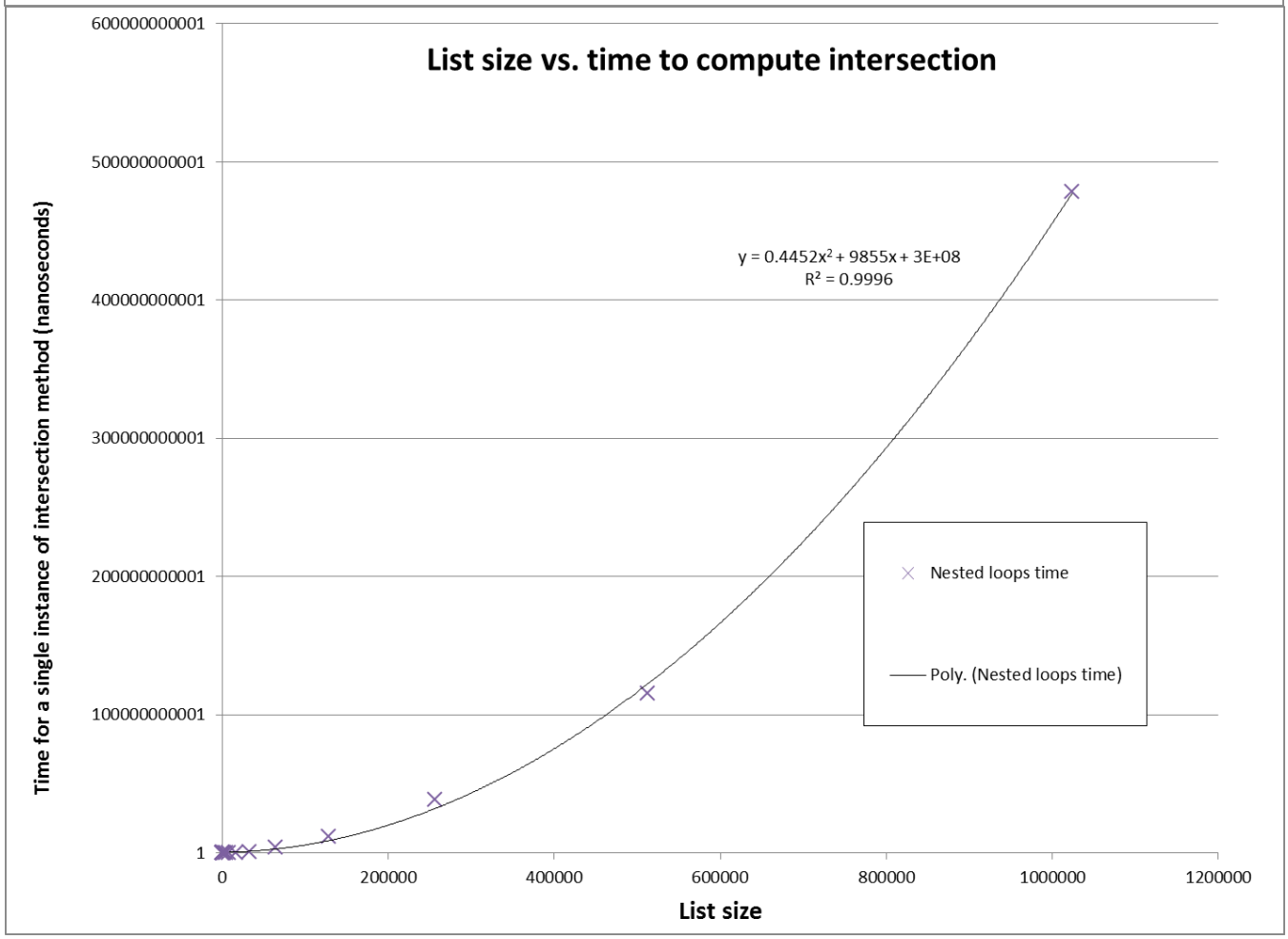
List Size	Nested loops time	Binary search time	Parallel pointers time	Merge and sort time
2	63,386	66,838	75,564	67,909
4	57,684	60,912	60,687	63,986
8	53,244	59,958	66,588	69,980
16	54,086	81,271	75,340	82,366
32	53,708	86,736	99,324	122,529
64	73,781	136,082	154,088	169,002
128	83,760	171,417	204,176	189,793
256	405,292	219,820	378,152	236,017
512	655,138	298,923	727,605	309,098
1000	1,071,761	469,936	1,685,199	452,612
2000	4,307,581	810,278	3,108,044	910,676
4000	15,829,276	1,705,711	6,728,251	1,929,063
8000	61,465,201	3,554,609	14,125,251	4,086,062
16000	256,870,244	7,550,665	26,053,265	8,355,625
32000	1,005,860,159	16,234,784	59,113,189	17,507,759
64000	3,704,548,021	34,494,532	127,259,364	37,077,923
128000	11,635,204,691	77,496,751	257,470,010	79,129,046
256000	38,684,336,816	165,628,576	324,033,878	178,828,216
512000	115,775,565,753	356,793,262	452,672,450	378,941,301
1024000	478,407,007,187	805,083,036	723,502,193	688,696,437

*see attached file for raw data

b) Based on your observations, try to determine how the running time of each method increases as a function of the size of the two lists.

I used excel to graph my results for n>=32000 and generated lines of best fit and their functions


List size vs. time to compute intersections by method
Time for a single instance of intersection method (nanoseconds)
List size
y = 795.42x - 2E+07
R² = 0.997
y = 686.22x - 540859
R² = 0.9967
y = 2E+08ln(x) - 2E+09
R² = 0.9402
Binary search time
Parallel pointers time
Merge and sort time
Linear (Binary search time)
Log. (Parallel pointers time)
Linear (Merge and sort time)
900000000
800000000
700000000
600000000
500000000
400000000
300000000
200000000
100000000
0
0
200000
400000
600000
800000
1000000
1200000


List size vs. time to compute intersection
Time for a single instance of intersection method (nanoseconds)
List size
y = 0.4452x² + 9855x + 3E+08
R² = 0.9996
Nested loops time
Poly. (Nested loops time)
600000000001
500000000001
400000000001
300000000001
200000000001
100000000001
1
0
200000
400000
600000
800000
1000000
1200000

It would appear that for the four methods, time to compute as a function of list size is as follows:

Nested Loops

$$T(n) = 0.4552n^2 + 9855n$$

Binary Search

$$T(n) = 795.42n$$

Parallel Pointers

$$T(n) = 2 \times 10^8 \log n - 2 \times 10^9$$

Merge and Sort

$$T(n) = 686.22n$$

c) Now, consider what happens if the two lists are of different sizes. For each of the four algorithms, report the running time for computing the intersection between a list L1 of 32000 students and a list L2 of 1024000 students, and the running time for computing the intersection between a list L1 of 1024000 students and a list of L2 of 32000 students. For each algorithm, explain (in 2-3 lines at most) why the running time changes or doesn't change.

The table of results are below:

List Size	**Nested loops time**	**Binary search time**	**Parallel pointers time**	**Merge and sort time**
L1<L2	25,774,630,042	144,600,129	310,702,244	401,844,397
L2<L1	22,509,476,209	357,136,717	335,978,549	405,983,488

There is no noticeable difference in time for the nested loops, parallel pointers, or merge and sort because in those cases switching the length of the lists (ie. switching m,n in m*n, m+n) does not impact the total number of comparisons the computer must do. However in binary search it matters since the length impacts the number of comparisons and a longer list takes more time to sort.

ATTACHMENT

List Size	List generating method	Nested Loops total	**Nested loops time**	Binary search total	**Binary search time**
2	7686	71072	**63386**	74524	**66838**
4	15033	72717	**57684**	75945	**60912**
8	24290	77534	**53244**	84248	**59958**
16	28188	82274	**54086**	109459	**81271**
32	37197	90905	**53708**	123933	**86736**
64	42599	116380	**73781**	178681	**136082**
128	51029	134789	**83760**	222446	**171417**
256	76663	481955	**405292**	296483	**219820**
512	132602	787740	**655138**	431525	**298923**
1000	235516	1307277	**1071761**	705452	**469936**
2000	442574	4750155	**4307581**	1252852	**810278**
4000	844909	16674185	**15829276**	2550620	**1705711**
8000	1661535	63126736	**61465201**	5216144	**3554609**
16000	3405325	260275569	**256870244**	10955990	**7550665**
32000	6887683	1012747842	**1005860159**	23122467	**16234784**
64000	14411454	3718959475	**3704548021**	48905986	**34494532**
128000	29475747	11664680438	**11635204691**	106972498	**77496751**
256000	66132195	38750469011	**38684336816**	231760771	**165628576**
512000	141375278	115916941031	**115775565753**	498168540	**356793262**
1024000	278185688	478685192875	**478407007187**	1083268724	**805083036**
L1<L2	191,172,728.00	25,965,802,770.00	**25,774,630,042.00**	335,772,857.00	**144,600,129.00**
L2<L1	195,007,345.00	22,704,483,554.00	**22,509,476,209.00**	552,144,062.00	**357,136,717.00**

List Size	Parallel pointers total	**Parallel pointers time**	Merge and sort total	**Merge and sort time**
2	83250	**75564**	75595	**67909**
4	75720	**60687**	79019	**63986**
8	90878	**66588**	94270	**69980**
16	103528	**75340**	110554	**82366**
32	136521	**99324**	159726	**122529**
64	196687	**154088**	211601	**169002**
128	255205	**204176**	240822	**189793**
256	454815	**378152**	312680	**236017**
512	860207	**727605**	441700	**309098**
1000	1920715	**1685199**	688128	**452612**
2000	3550618	**3108044**	1353250	**910676**
4000	7573160	**6728251**	2773972	**1929063**
8000	15786786	**14125251**	5747597	**4086062**
16000	29458590	**26053265**	11760950	**8355625**
32000	66000872	**59113189**	24395442	**17507759**
64000	141670818	**127259364**	51489377	**37077923**
128000	286945757	**257470010**	108604793	**79129046**
256000	390166073	**324033878**	244960411	**178828216**
512000	594047728	**452672450**	520316579	**378941301**
1024000	1001687881	**723502193**	966882125	**688696437**
L1<L2	501,874,972.00	**310,702,244.00**	593,017,125.00	**401,844,397.00**
L2<L1	530,985,894.00	**335,978,549.00**	600,990,833.00	**405,983,488.00**