Fall 21

Experimental Analysis

CIS 3501- PROGRAMMING HW3

HAIDAR, KARIM 79772246

1-Merge Sort

Theoretical study:

Merge(B,C,A):

Preconditions: List b and List C are sorted

Post conditions: List A has the elements of list B and C in sorted order.

Merge sort(A):

Preconditions: List A

Postconditions: sorted List A

Randomizer(A,p) used in all algorithms:

Preconditions: Sorted List A

Postconditions: list A randomized with a certain percentage n*p

Function	Time	space
Merge	O(n)	O(n)
Merge Sort	O(nlogn)	O(n)
randomizer	O(n*p)	O(1)

Experimental study:

Fig1.1: A table showing the average number of comparisons of 25 instance for each (n,p) pair.

#Comp as function of n&p		number of inputs 10	20	50	200	500	1000
randomness	0	39	99	315	1663	4931	10863
	0.1	42.32	109.36	364.8	2043.12	6202.16	13911.88
	0.2	42.96	114	383.16	2125.52	6382.36	14249.8
	0.5	46.2	119.24	399	2198.28	6522.96	14564.96
	1	46.28	121.84	403.36	2213.64	6561.52	14626.36

Fig1.1

Fig1.2: A graph showing the number of comparisons done for each (n,p) pair

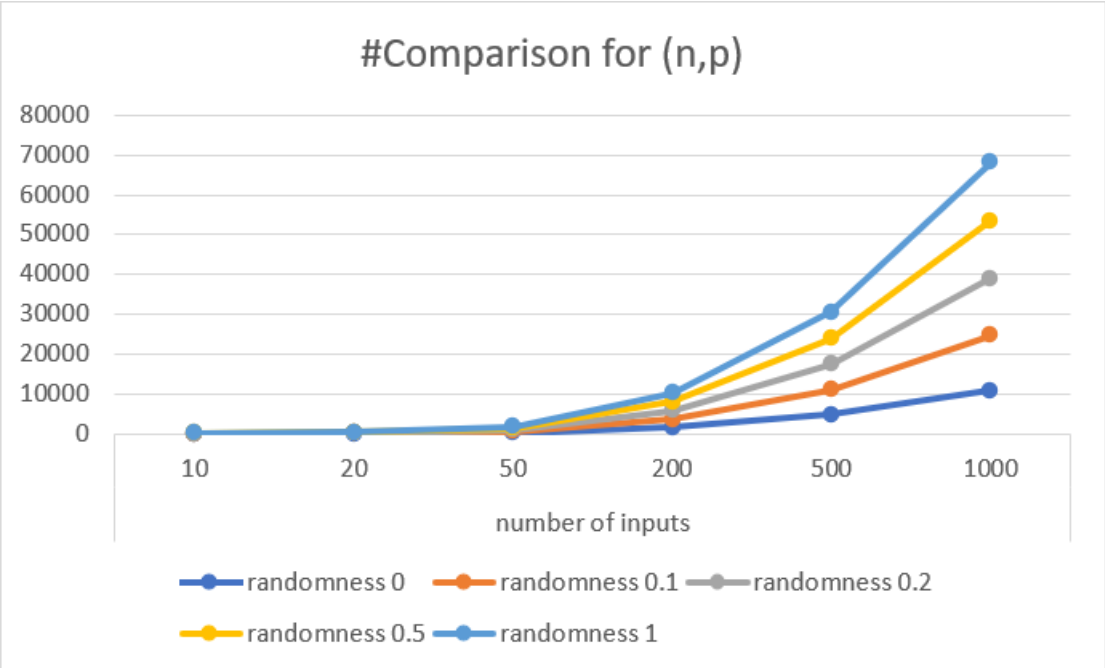

Fig 1.2

Fig 1.3: A table showing the average cpu time in nanoseconds taken by 25 instance for each (n,p) pair.

Cputime as function of n&p		number of inputs 10	20	50	200	500	1000
randomness	0	66688	32268	97292	847128	1627572	2892028
	0.1	30680	34036	102192	601424	1483100	2953008
	0.2	17848	29968	104096	578536	1438652	2947460
	0.5	16192	39672	114488	530968	1695988	3301568
	1	18964	32512	141432	561736	1425488	3350092

Fig 1.3

Fig1.4: A graph showing the cpu time taken for each (n,p) pair

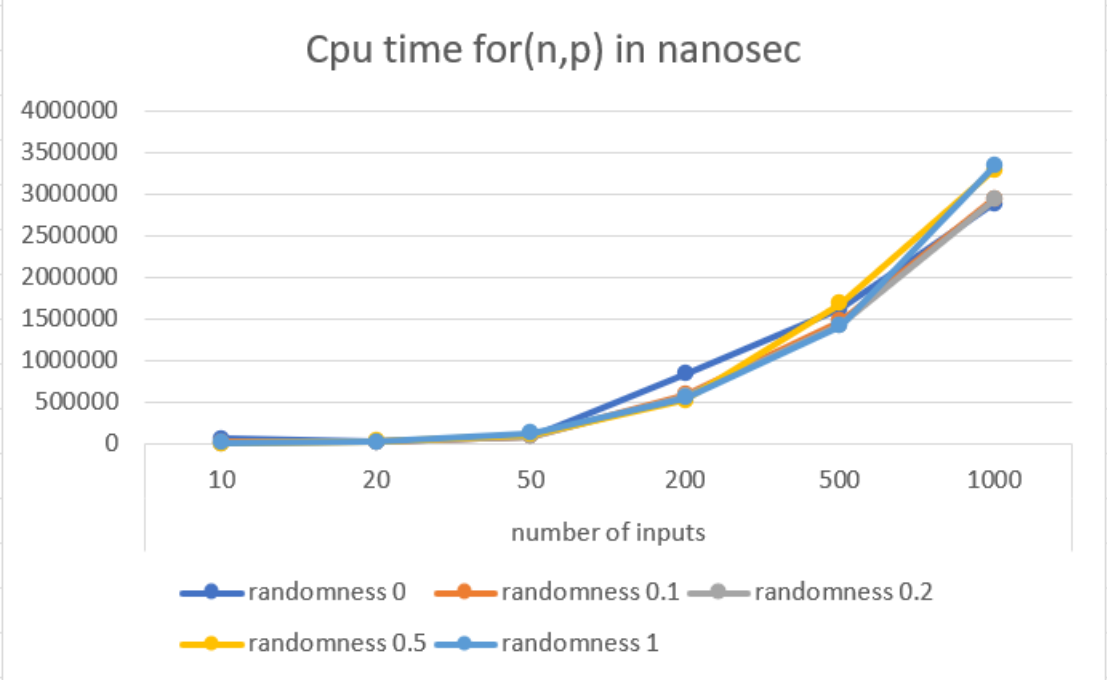

Fig 1.4

Fig1.5: A table showing the 95% CI for number of comparisons for each (n,p) pair.

95% ci for #comp		number of inputs 10	20	50	200	500	1000
randomness	0	0	0	0	0	0	0
	0.1	1.74	4.18	10.44	22.61	44.26	78.31
	0.2	2.15	2.7	8.12	13.87	28.21	42.4
	0.5	1.17	2.56	4.75	9.6	14.83	20.74
	1	1.33	2.28	3.94	5.55	11.37	12.28

Fig 1.5

Fig1.6: A bar graph showing the 95% CI for number of comparisons for each (n,p) pair.

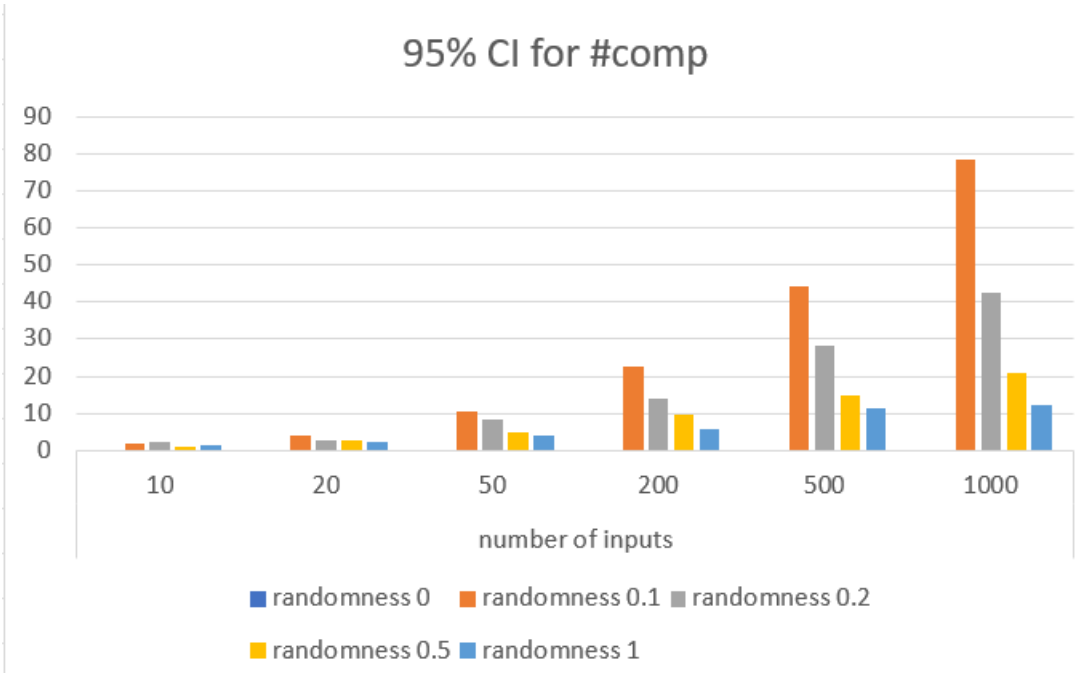

Fig 1.6

Fig1.7: A table showing the 95% CI for cpu time taken for each (n,p) pair.

95% CI for Cputime		number of inputs					
		10	20	50	200	500	1000
randomness	0	10722.56	73391.29	6440.05	14686.67	99914	816183.78
	0.1	4544.45	4044.96	3711.36	9072.83	334023.7	110089.36
	0.2	6126.31	2321.07	1759.16	31411.9	371491.45	61873.62
	0.5	4228.84	2978.71	3215.66	12182.95	379006.05	43509.08
	1	6079.89	48123.99	5295.8	66957.73	67677.83	76770.53

Fig 1.7

Fig1.8: A bar graph showing the 95% CI for cpu time taken for each (n,p) pair.

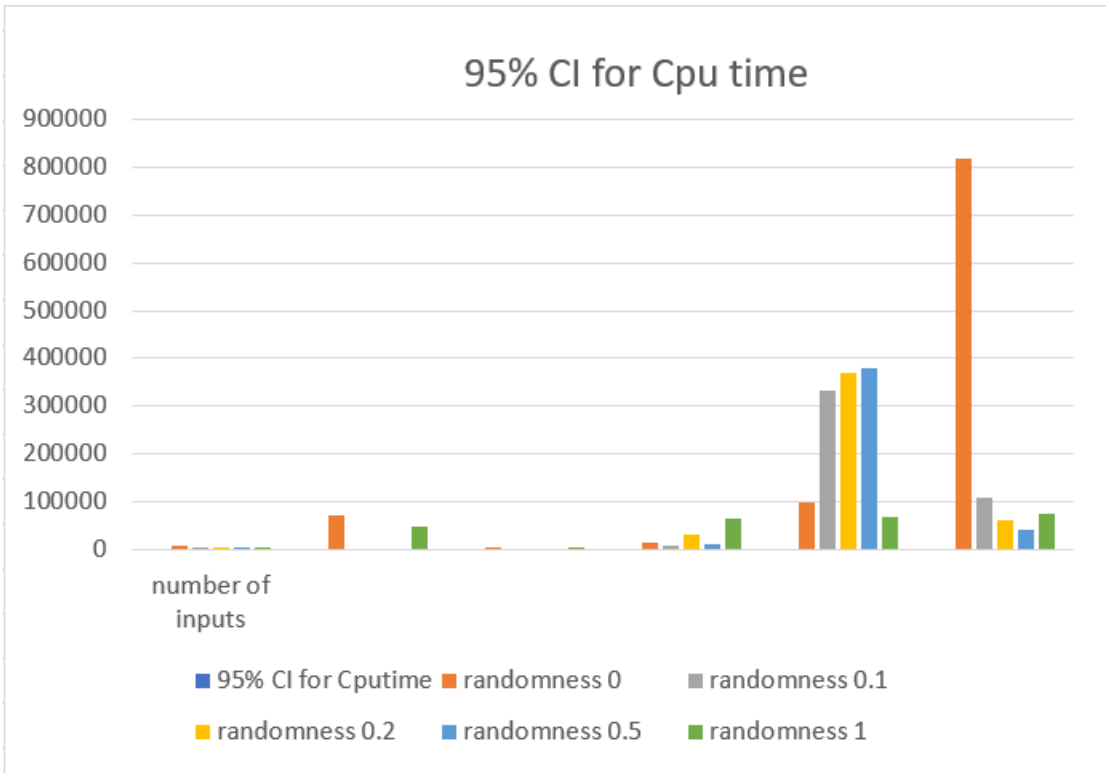
95% CI for Cpu time
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
number of inputs
95% CI for Cputime
randomness 0
randomness 0.1
randomness 0.2
randomness 0.5
randomness 1

Fig 1.8

2-Quick Sort:

Theoretical study:

Quick sort(A,Ran):

Preconditions: List A, Boolean Ran

Postconditions: sorted List A

2.1 Deterministic quick sort Ran: false

Function	Time	space
QuickSort	O(n^2)	O(n)

Experimental study:

Fig2.1.1: A table showing the average number of comparisons of 25 instance for each (n,p) pair.

		number of inputs						
		10	20	50	100	200	500	1000
randomness	0	136	476	2696	10396	40796	251996	1003996
	0.1	107.44	343.64	1473.08	3725	9320.6	28045	67430
	0.2	96.52	271.88	1090.12	2663.04	6515.2	20473.68	43818.36
	0.5	89.16	226.8	781.92	1893.76	4606.08	13989.16	29784.72
	1	82.2	210.36	708.08	1674.08	3926.6	12008.44	26518.6

Fig2.1.1

Fig2.1.2: A graph showing the number of comparisons done for each (n,p) pair

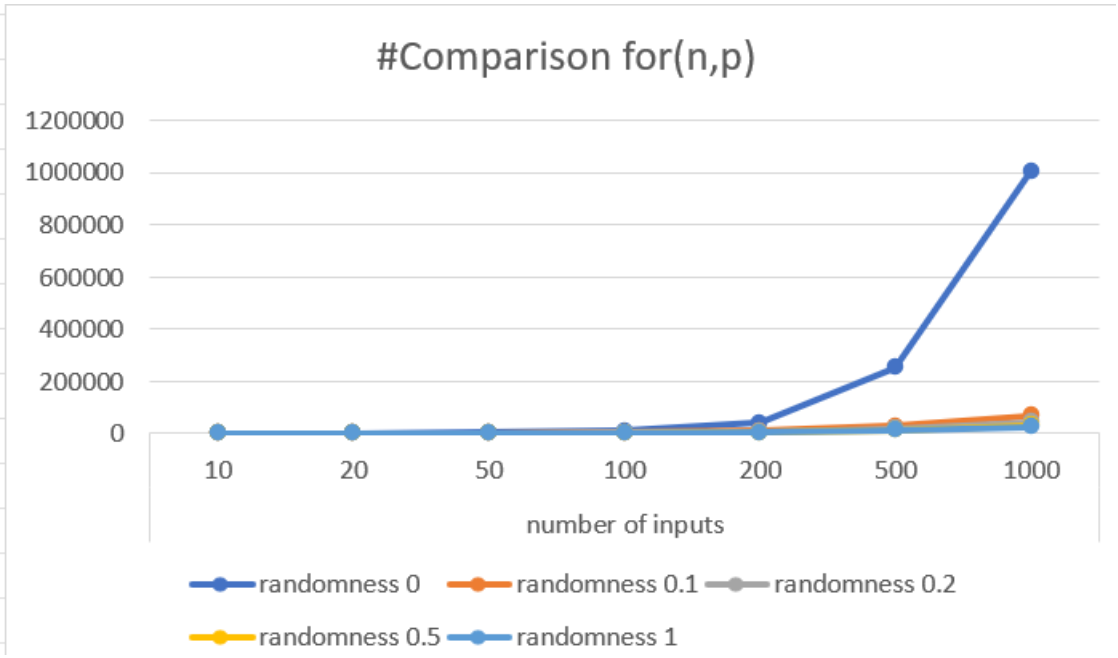

Fig 2.1.2

Fig2.1.3: A table showing the average cpu time taken by 25 instance for each (n,p) pair.

		number of inputs						
		10	20	50	100	200	500	1000
randomness	0	101540	66240	442084	1207344	6024396	50261904	200691088
	0.1	27968	48020	166940	408100	1158424	3542348	6884000
	0.2	13536	32596	151992	287736	861540	2495436	3893144
	0.5	11976	35916	80100	259908	536132	1689440	2579596
	1	48020	34528	72120	180076	680880	1364928	2336780

Fig 2.1.3

Fig2.1.4: A graph showing the cpu time taken by 25 instance for each (n,p) pair.

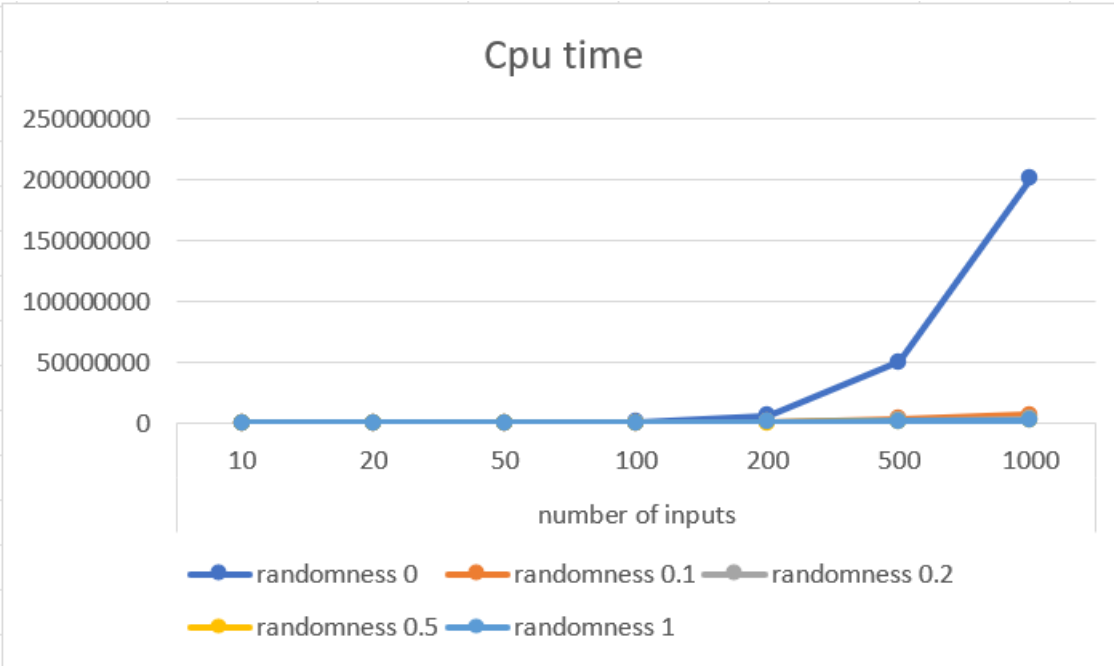

Fig 2.1.4

Fig2.1.5: A table showing the 95% CI for number of comparisons for each (n,p) pair.

		number of inputs						
		10	20	50	100	200	500	1000
randomness	0	0	0.5	0.78	1.59	1.87	3.04	4.35
	0.1	13.6	46.85	248.47	727.99	1929.96	4540.59	11189.55
	0.2	11	42.78	183.96	334.79	1295.94	3673.67	7996.41
	0.5	8.7	27.89	64.62	256.88	532.6	1329.01	2104.93
	1	7.89	19.23	80.59	110.97	234.48	604.05	1178.06

Fig 2.1.5

Fig2.1.6: A bar graph showing the 95% CI for number of comparisons for each (n,p) pair.

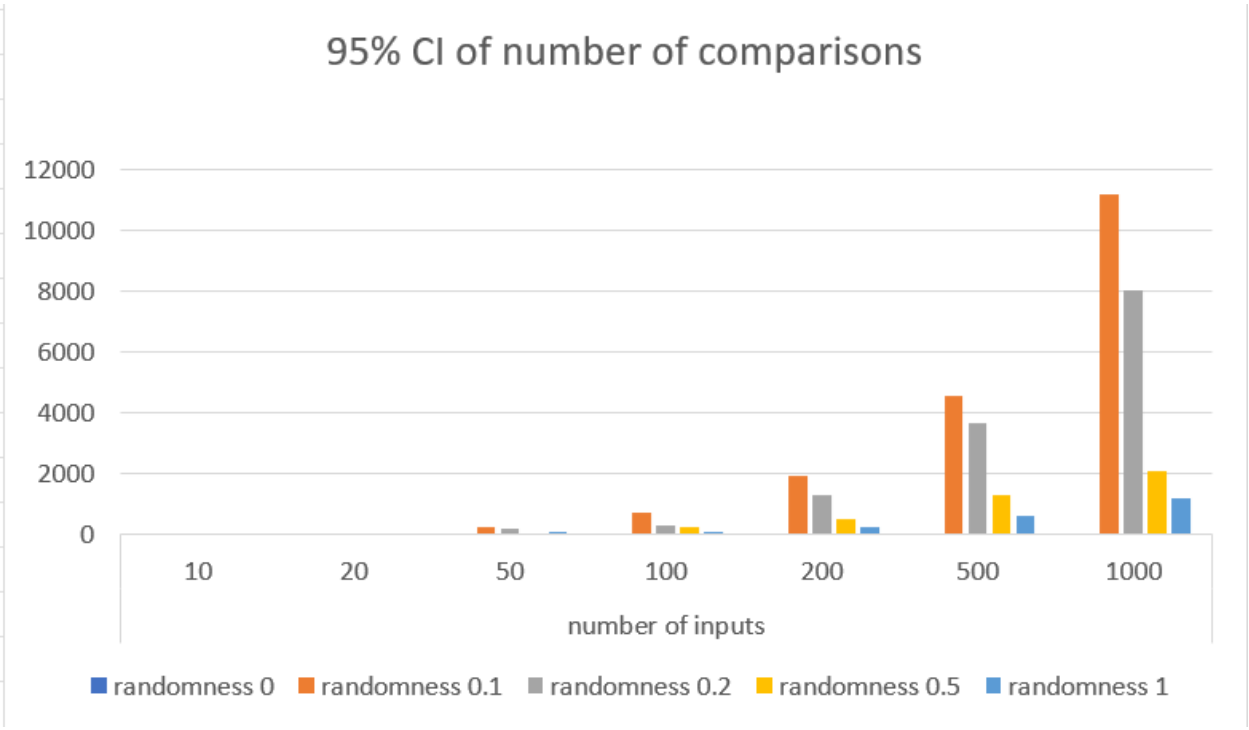

Fig 2.1.6

Fig2.1.7: A table showing the 95% CI for cpu time taken for each (n,p) pair.

		number of inputs						
		10	20	50	100	200	500	1000
randomness	0	101540	66240	442084	1207344	6024396	50261904	200691088
	0.1	27968	48020	166940	408100	1158424	3542348	6884000
	0.2	13536	32596	151992	287736	861540	2495436	3893144
	0.5	11976	35916	80100	259908	536132	1689440	2579596
	1	48020	34528	72120	180076	680880	1364928	2336780

Fig 2.1.7

Fig2.1.8: A bar graph showing the 95% CI for cpu time taken for each (n,p) pair.

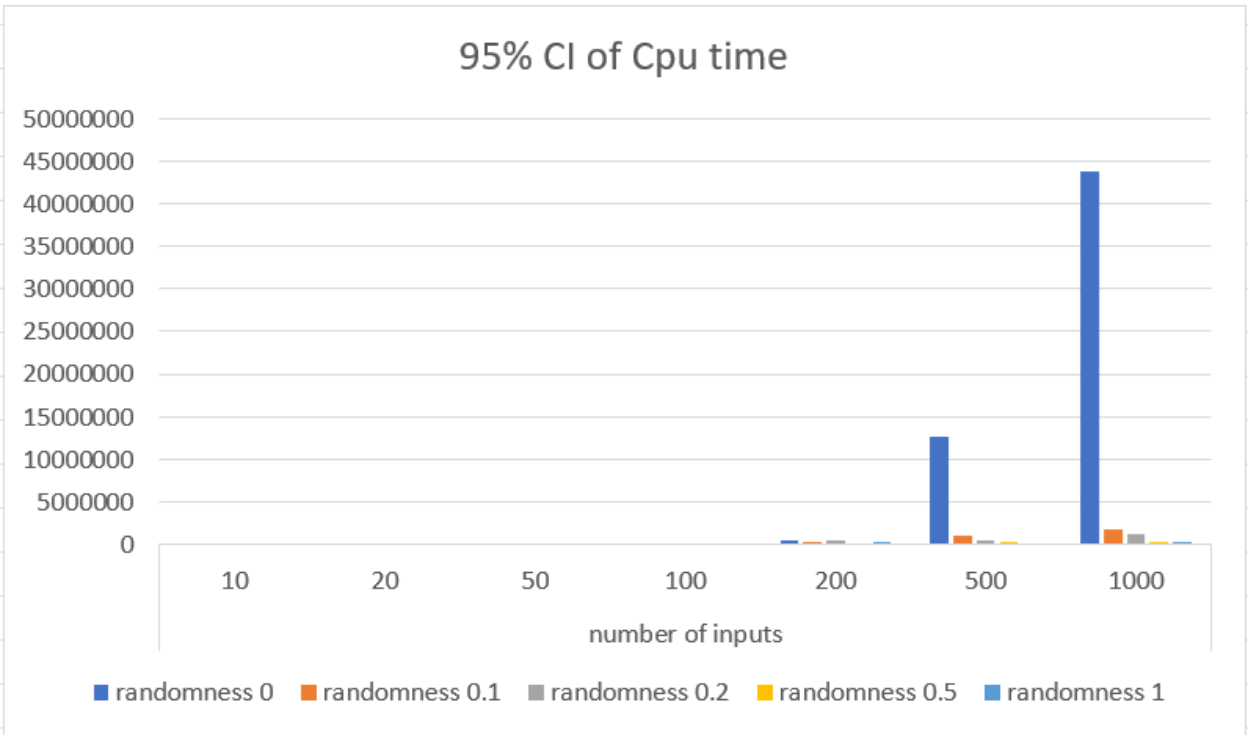

Fig 2.1.8

2.2 Randomized quick sort Ran:true

Theoretical study:

Function	Time	space
QuickSort	O(n^2) expected running time O(nlogn)	O(n)

Experimental study:

Fig2.2.1: A table showing the average number of comparisons of 25 instance for each (n,p) pair.

		number of inputs						
		10	20	50	100	200	500	1000
randomness	0	81	218.64	683.8	1653.64	3684.88	11392.76	25327.32
	0.1	76.8	202.44	683.24	1587.68	3700.72	11298.6	25184.52
	0.2	82.92	203.52	696.12	1637.84	3700.12	11409.96	25406.56
	0.5	79.76	202.84	672.84	1638	3844.28	11188.44	25361
	1	79.6	206.04	688.48	1670.12	3823.32	11317.52	25544.84

Fig2.2.1

Fig2.2.2: A graph showing the number of comparisons done for each (n,p) pair

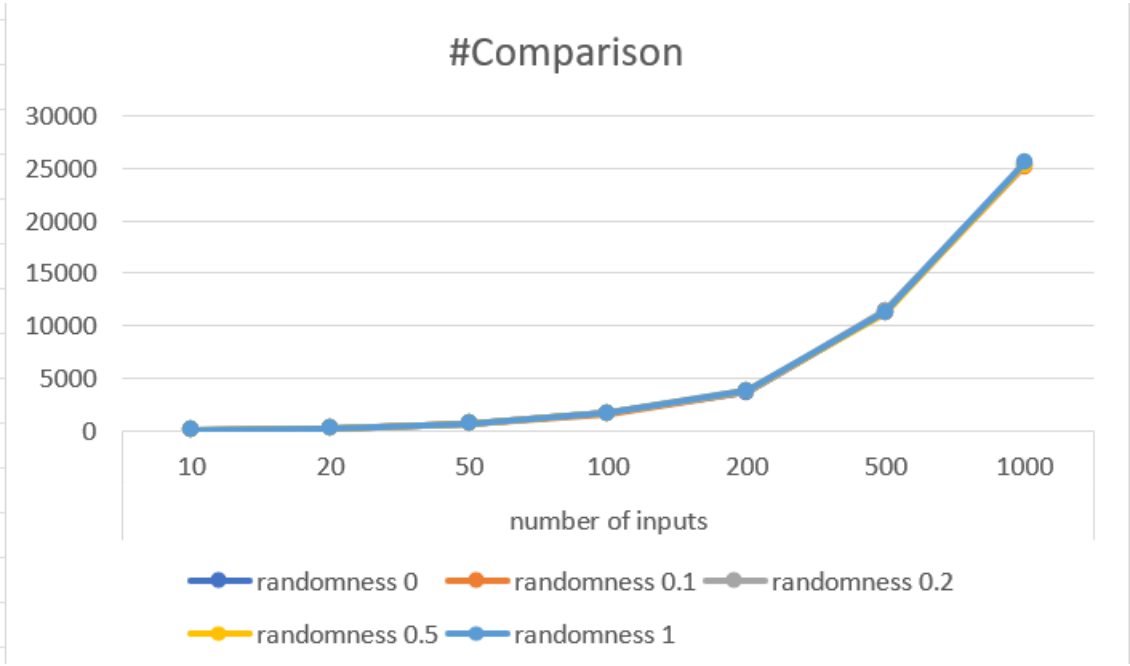

Fig 2.2.2

Fig2.2.3: A table showing the average cpu time taken by 25 instance for each (n,p) pair.

		number of inputs						
		10	20	50	100	200	500	1000
randomness	0	71200	32396	96760	213992	454592	1782912	7795064
	0.1	27932	37808	95720	188672	506884	1728836	8392992
	0.2	12208	33240	107912	305412	466568	1609164	7379228
	0.5	14324	29636	88336	241716	593872	1560596	5519080
	1	13220	32012	93648	175556	585304	2086528	3631148

Fig 2.2.3

Fig2.2.4: A graph showing the cpu time taken by 25 instance for each (n,p) pair.

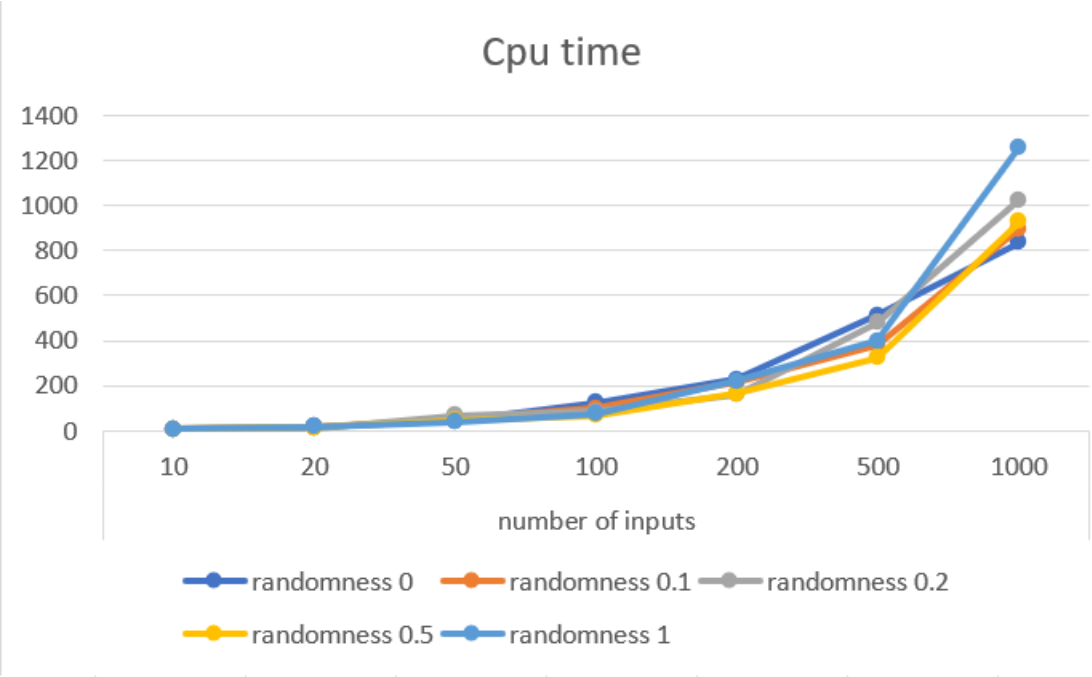

Fig 2.2.4

Fig2.2.5: A table showing the 95% CI for number of comparisons for each (n,p) pair.

		number of inputs						
		10	20	50	100	200	500	1000
randomness	0	7.01	17.9	41.29	124.1	226.29	512.73	836.32
	0.1	6.85	13.26	53.28	97.77	214.08	379.84	896.86
	0.2	5.37	11.44	68.49	83.59	161.54	480.08	1025.18
	0.5	7.37	13.33	46.39	66.21	164.82	323.4	928.67
	1	6.66	17.05	37.55	74.1	222.19	399.02	1261.9

Fig 2.2.5

Fig2.2.6: A bar graph showing the 95% CI for number of comparisons for each (n,p) pair.

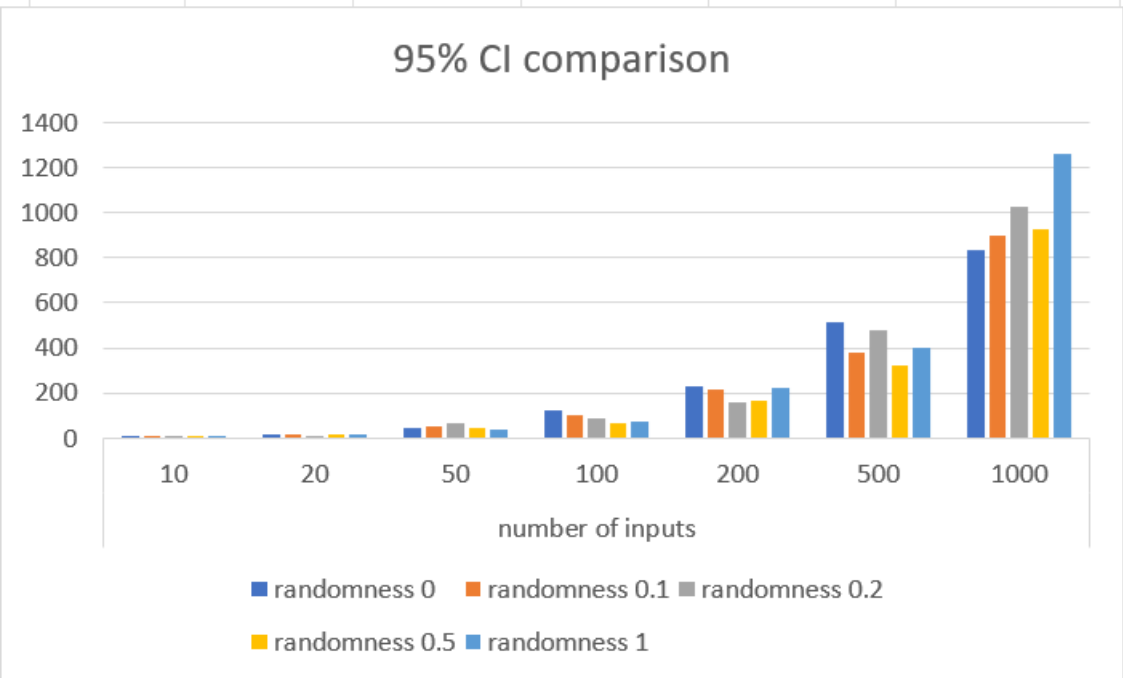

Fig 2.2.6

Fig2.2.7: A table showing the 95% CI for cpu time taken for each (n,p) pair.

		number of inputs						
		10	20	50	100	200	500	1000
randomness	0	22793.67	5332.11	16758	27980.47	47630.66	252725.55	1115112.94
	0.1	11003.59	5791.05	23731.77	29480.9	46117.47	245905.62	647273.57
	0.2	6623.82	6629.67	13329.82	182095.6	50325.09	36300.38	554879.02
	0.5	2747.06	9680.68	11733.04	27596.19	105611.82	185273.83	1512323
	1	3787.31	20238.94	8305.01	12398.07	232408.3	590119.46	417831.71

Fig 2.2.7

Fig2.2.8: A bar graph showing the 95% CI for cpu time taken for each (n,p) pair.

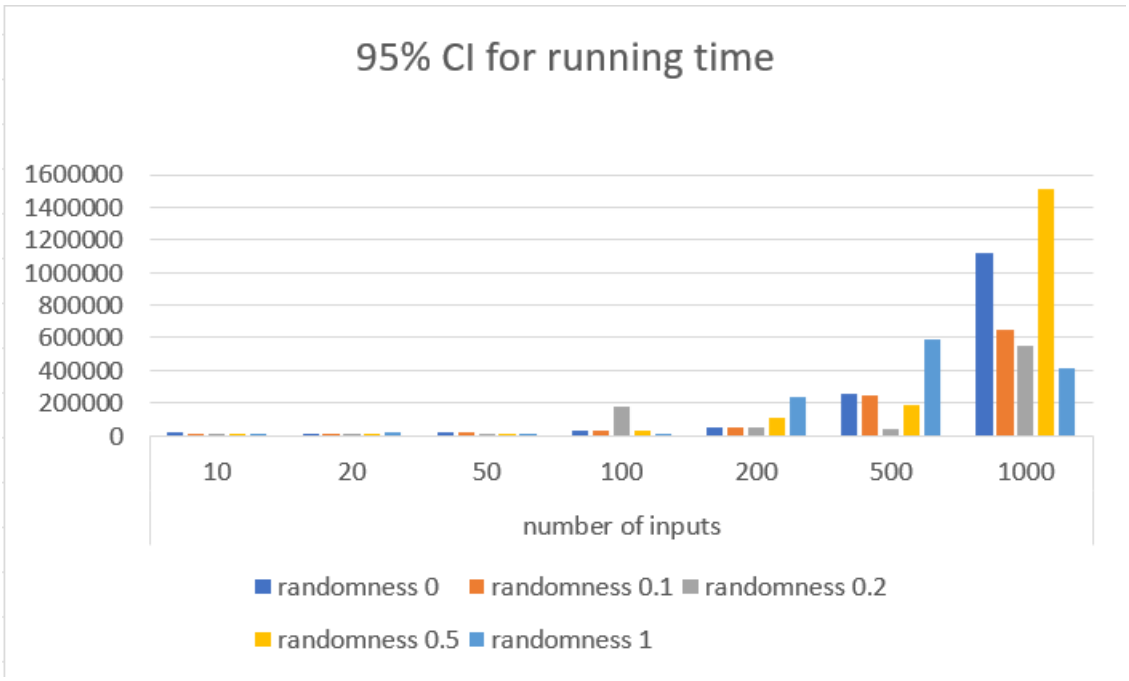

Fig 2.2.8

3-Conclusion:

Merge Sort: stable in terms of time, the experiments showed a graph similar to that of nlogn.

Deterministic quick sort: horrible for randomness zero as we are always choosing the first element which is the smallest element in the array.

Randomized quick sort: turned out as expected (nlogn) except for some points.

All three algorithms followed the lower bound we set (nlogn) for the number of comparisons, as none of the graphs turned out to have a faster growth than nlogn.