Report

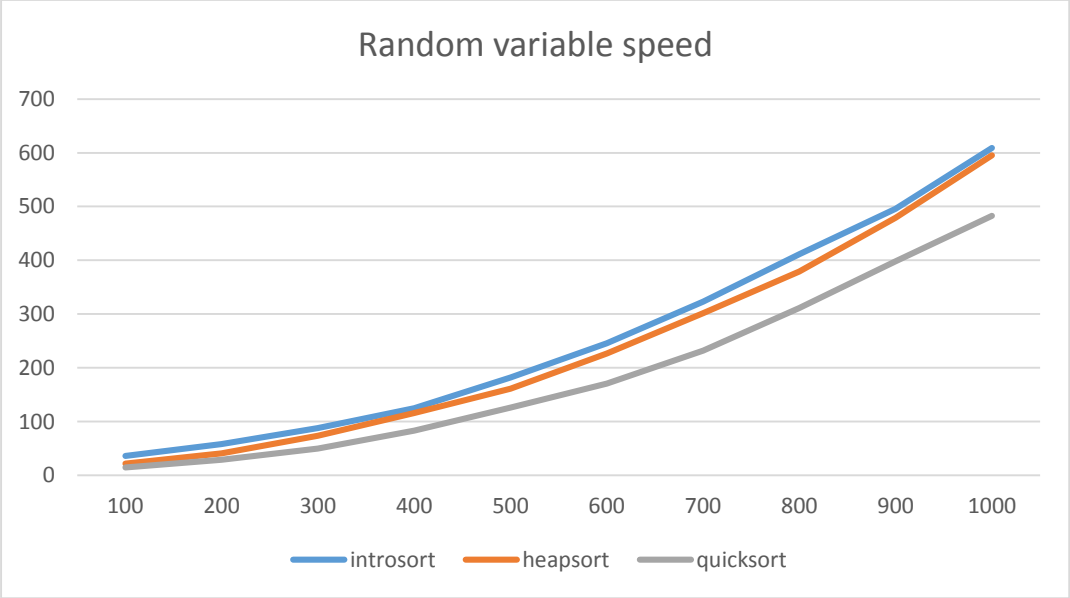
Qingyuan Zhang

Introsort is a method that combine quicksort and heapsort together. Quicksort run fast in general case but in worst case when the variable is unbalanced it cannot avoid the O(n) complexity. Heapsort, on the other hand, run slightly slower but it will ensure O(nlogn) for all case. It begins with running quicksort and use heapsort when the depth exceeds "2 ·floor(log2n)" This combines the advantage of both sorting method.

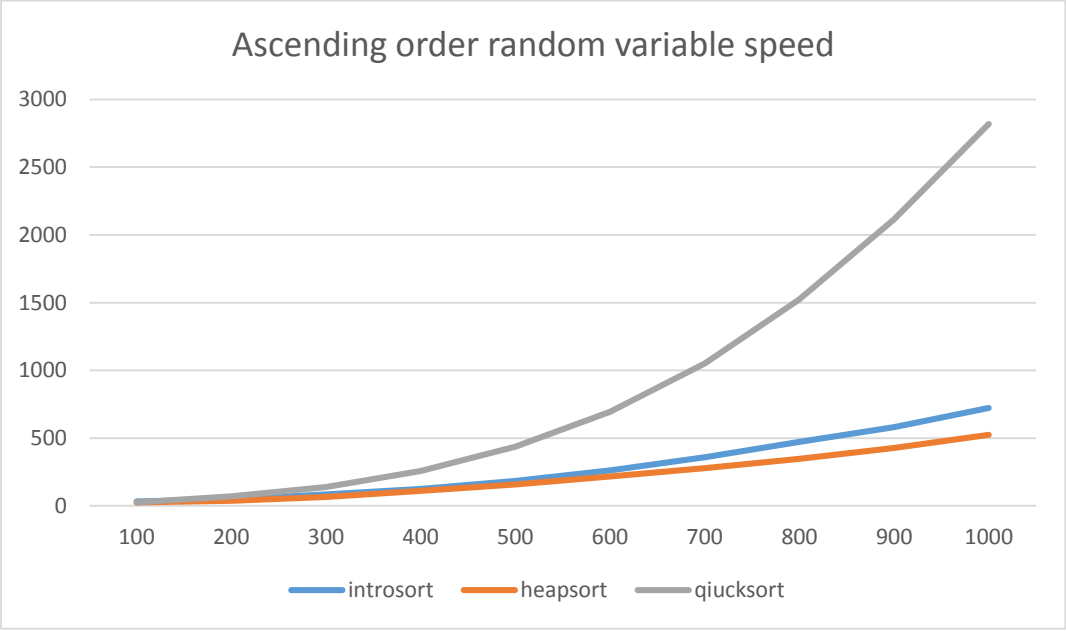
My introsort class extends the Abstractsort class, so I created a constructor for introsort class with T variable array as parameter. Then I wrote a method called introSortLoop. The introSortLoop method is a loop that keep running when beginIndex < endIndex and there is an if statement that decide whether it use quicksort or heapsort. The method has a parameter called depth_limit which equals to 2*(log2n). It is compute by the method I wrote called floor_log which use the math.floor and math.log. Every time introSortLoop method runs quicksort the depth_limit will subtract by one. When the depth_limit is equal to 0 it will start to run heapsort. For the quicksort and heapsort methods, I use the same methods offered by Professor Choi. However, Professor Choi's heapsort method have to sort the whole array, so I make a copy of the partition that needs to be heapsort and run heapsort with the copy array. Then I copy the result back to the original array. This method implement the sort that start with quicksort and switch to heapsort when depth is greater than "2 ·floor(log2n)".

For the speed test, I create lists of 100, 200, .., 1000 random comparable keys, keys in ascending order, and keys in descending order. The result is in the following tables:

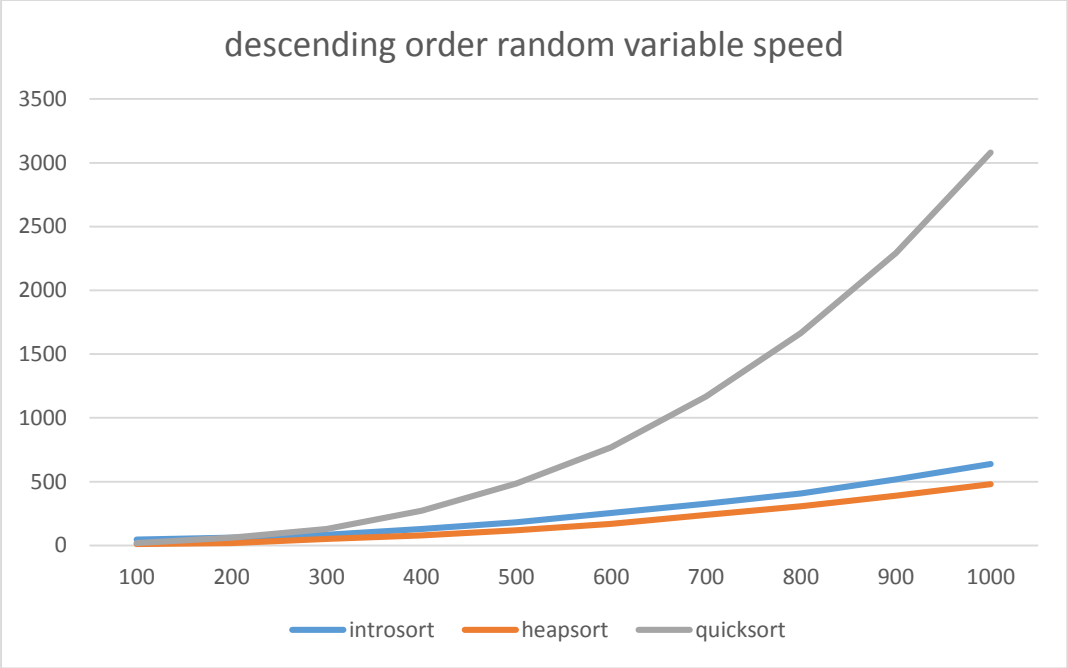
100	36	22	15
200	58	41	29
300	88	74	50
400	125	116	83
500	182	161	126
600	246	227	171
700	323	302	232
800	411	379	311
900	496	479	398
1000	609	595	483



100	33	26	25
200	46	38	70
300	84	65	140
400	125	111	256
500	183	157	437
600	262	216	695
700	358	278	1051
800	472	346	1524
900	580	428	2115
1000	723	523	2818



100	45	10	18
200	61	18	61
300	84	52	129
400	128	79	271
500	181	119	486
600	255	170	768
700	327	238	1168
800	408	308	1665
900	518	389	2293
1000	637	481	3080



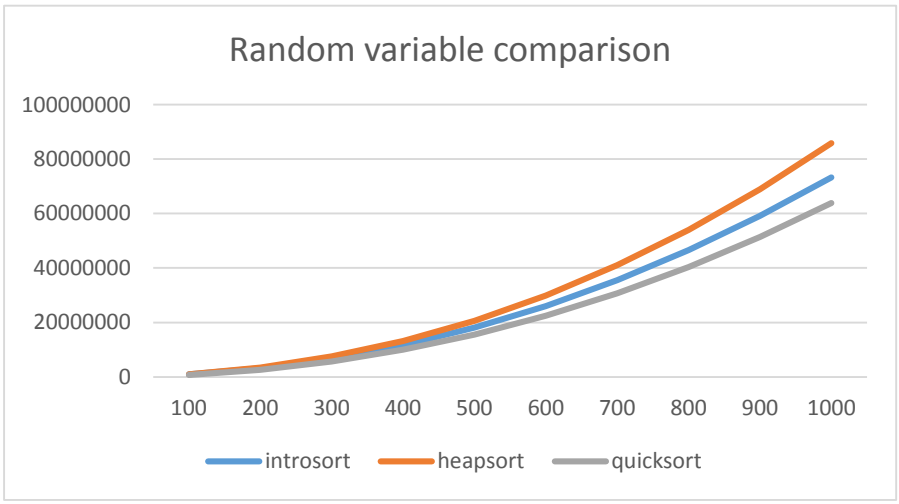
For random variable without order, quick sort have the best performance. Introsort and heapsort have almost the same run speed. However for the ascending and descending order variables, quicksort have the worst performance, since the variable is so unbalanced that quicksort has its worst case which has the O(n) complexity. In ascending and descending order variable test, heapsort has the best performance and introsort is slightly slow than heapsort but much better then quicksort.

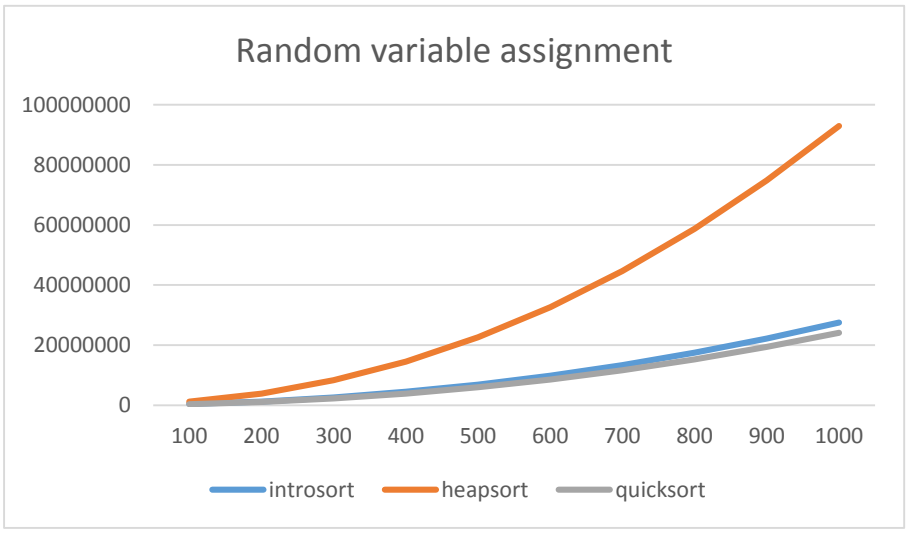
As for the comparison and assignment counts, I also I create lists of 100, 200, .., 1000 random comparable keys, keys in ascending order, and keys in descending order. The result is in the following tables:

In random variable without order case, quicksort, introsort, and heapsort have a slightly difference in comparison. However heapsort needs more assignment than introsort and quicksort. So basiclly, for random variable without order, the run speed of the three sort

method is quicksort > introsort > heapsort.

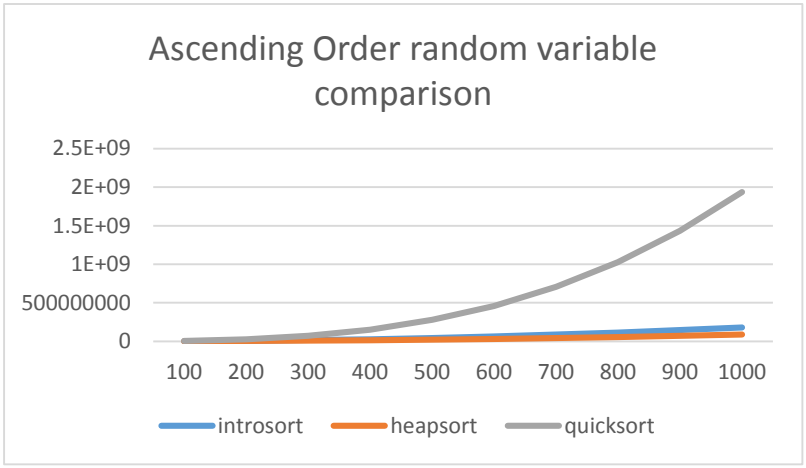
100	959497	1027645	795946	376824	1158042	317534
200	3162542	3480995	2663372	1220174	3872868	1041248
300	6723925	7505941	5702328	2566090	8288968	2206092
400	11695369	13210802	9989911	4436418	14515266	3833886
500	18090458	20637473	15547011	6852326	22594422	5944102
600	26032835	29885676	22432945	9822116	32625494	8548998
700	35507170	41001828	30683221	13357074	44656694	11658844
800	46551178	54011462	40343376	17471578	58710294	15283800
900	59135742	68936036	51378273	22172776	74809176	19431960
1000	73306371	85789558	63841691	27468524	92967974	24111818

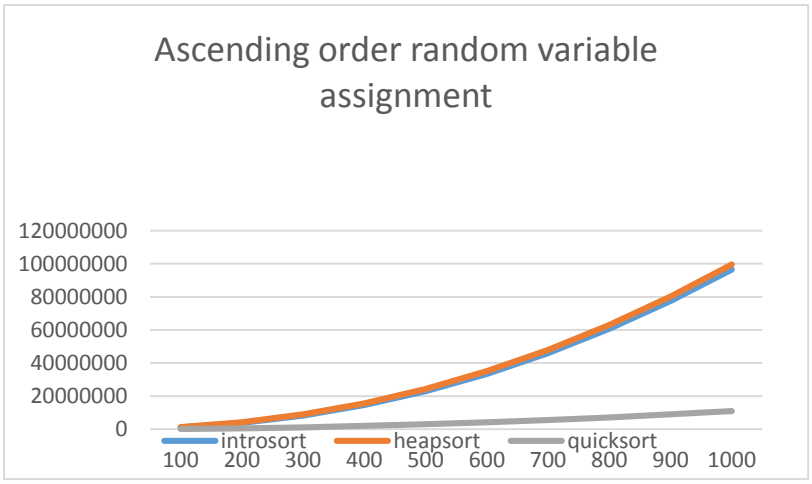




In random variable with ascending order case, introsort is basically run same as heapsort. As a result, introsort, and heapsort have a slightly difference in comparison. While, quicksort needs much more comparison. As for assignment, introsort and heapsort needs more assignment than quicksort. So for random variable with ascending order, the run speed of the three sort method is heapsort > introsort > quicksort.

100	2060014	1082263	5196959	1091658	1266040	198000
200	7135843	3659489	25593882	3803804	4212428	596000
300	15762701	7880317	71191276	8276534	8984264	1194000
400	27747828	13858211	1.52E+08	14685666	15689642	1992000
500	43090492	21595313	2.78E+08	23030026	24334136	2990000
600	63070147	31272886	4.59E+08	33418520	35095106	4188000
700	86814587	42915615	7.06E+08	45949920	47999692	5586000
800	1.14E+08	56523868	1.03E+09	60624876	63046284	7184000
900	1.46E+08	72098371	1.43E+09	77443630	80238378	8982000
1000	1.81E+08	89638995	1.94E+09	96406744	99573356	10980000





In random variable with descending order case, the result is almost same as ascending order. introsort also basically run same as heapsort. As a result, introsort, and heapsort have a slightly difference in comparison. While, quicksort needs much more comparison since the variable is unbalanced and it needs lots of recursion. As for assignment, introsort and heapsort needs more assignment than quicksort. So for

random variable with desicending order, the run speed of the three sort method is also heapsort > introsort > quicksort.

100	1955079	943658	5098536	899136	1019852	198000
200	6774345	3220590	25297102	3143334	3425366	596000
300	15011120	6984475	70595617	6898418	7406378	1194000
400	26407642	12336508	1.51E+08	12290220	13049588	1992000
500	41056051	19342002	2.76E+08	19390180	20393392	2990000
600	60147776	28060165	4.57E+08	28245722	29525918	4188000
700	82800485	38560411	7.03E+08	38896544	40505902	5586000
800	1.09E+08	50841101	1.02E+09	51474940	53342876	7184000
900	1.39E+08	65013133	1.43E+09	65914426	68153620	8982000
1000	1.73E+08	80987761	1.93E+09	82250598	84781794	10980000