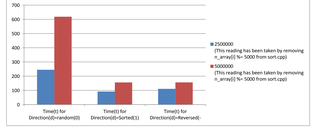
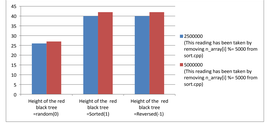
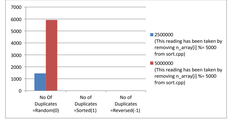
RED BLACK TREE REPORT

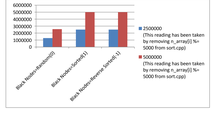
Input Size (n)	Time(t) for Direction(d)=random(0) (in ms)	Time(t) for Direction(d)=Sorted(1) (in ms)	Time(t) for Direction(d)=Reversed(-1) (in ms)	Height of the red black tree =random(0)	Height of the red black tree =Sorted(1)	Height of the red black tree =Reversed(-1)	No Of Duplicates =Random(0)	No of Duplicates =Sorted(1)	No of Duplicates =Reversed(-1)	Black Nodes=Random(0)	Black Nodes=Sorted(1)	Black Nodes=Reverse Sorted(-1)	Case1 insertion= Random(0)	Case1 insertion= Sorted(1)	Case1 insertion= Reverse(-1)	Case2 insertion= Random(0)	Case2 insertion= Sorted(1)	Case2 insertion= Reverse(-1)	Case3 insertion= Random(0)	Case3 insertion= Sorted(1)	Case3 insertion= Reverse(-1)	Left Rotation= Random(0)	Left Rotation= Sorted(1)	Left Rotation= Reverse(-1)	Right Rotation= Random(0)	Right Rotation= Sorted(1)	Right Rotation= Reverse(-1)
50000 (This reading has been taken by keeping n_array[i] %= 5000 from sort.cpp)	0.5	0.8	1.5	15	29	29	45000	0	0	2577	49981	49981	2571	49966	49966	1005	0	0	1977	49971	49971	1484	49971	0	1498	0	49971
100000 (This reading has been taken by keeping n_array[i] %= 5000 from sort.cpp)	0.6	2.1	2.4	15	31	31	95000	0	0	2551.5	99980	99980	2540	99964	99964	978	0	0	1954	99969	99969	1457	99969	0	1457	0	99969
250000 (This reading has been taken by keeping n_array[i] %= 5000 from sort.cpp)	0.5	4.7	5	15	33	33	245000	0	0	2560.4	249978	249978	2553	249961	249961	962	0	0	1941	249967	249967	1438	249967	0	1464	0	249967
500000 (This reading has been taken by keeping n_array[i] %= 5000 from sort.cpp)	0.5	15.4	18.2	15	35	35	495000	0	0	2575.8	499977	499977	2546	499959	499959	954	0	0	1951	499965	499965	1439	499965	0	1466	0	499959
1000000 (This reading has been taken by keeping n_array[i] %= 5000 from sort.cpp)	0.6	29.4	40.1	15	37	37	995000	0	0	2630.2	999976	999976	2569	999957	999957	967	0	0	1911	999963	999963	1445	999963	0	1433	0	999963

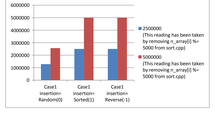
Input Size (n)	Time(t) for Direction(d)=random(0) (in ms)	Time(t) for Direction(d)=Sorted(1) (in ms)	Time(t) for Direction(d)=Reversed(-1) (in ms)	Height of the red black tree =random(0)	Height of the red black tree =Sorted(1)	Height of the red black tree =Reversed(-1)	No Of Duplicates =Random(0)	No of Duplicates =Sorted(1)	No of Duplicates =Reversed(-1)	Black Nodes=Random(0)	Black Nodes=Sorted(1)	Black Nodes=Reverse Sorted(-1)	Case1 insertion= Random(0)	Case1 insertion= Sorted(1)	Case1 insertion= Reverse(-1)	Case2 insertion= Random(0)	Case2 insertion= Sorted(1)	Case2 insertion= Reverse(-1)	Case3 insertion= Random(0)	Case3 insertion= Sorted(1)	Case3 insertion= Reverse(-1)	Left Rotation= Random(0)	Left Rotation= Sorted(1)	Left Rotation= Reverse(-1)	Right Rotation= Random(0)	Right Rotation= Sorted(1)	Right Rotation= Reverse(-1)
2500000 (This reading has been taken by removing n_array[i] %= 5000 from sort.cpp)	245.1	91.9	110.7	26	40	40	1450	0	0	1283113	2499972	2499972	1283100	2499952	2499952	484219	0	0	970725	2499960	2499960	727489	2499960	0	727455	0	2499960
5000000 (This reading has been taken by removing n_array[i] %= 5000 from sort.cpp)	618.5	155.8	156.2	27	42	42	5923	0	0	2564309	4999971	4999971	2564641	4999950	4999950	970213	0	0	1939947	4999958	4999958	1455858	4999958	0	1454302	0	4999958

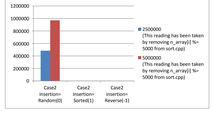


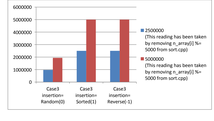


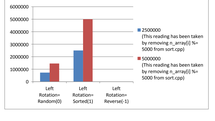


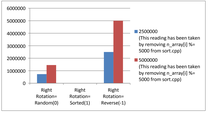












Analysis :

1) It can be inferred from the readings that, running time grows as the input size grows. The height of the tree grew as the input size grew.
2) No of duplicate nodes increases as the input size increases.
3) Experiment is conducted by removing modulo 5000 operations from sort.cpp for 2500000 and 5000000. Hence there are huge changes in the graph for those 2 values.
4) Running time for inorder tree traversal is theta(n) then, we can observe that, as input size increases, running time of inorder traversal increases.