Implementation:

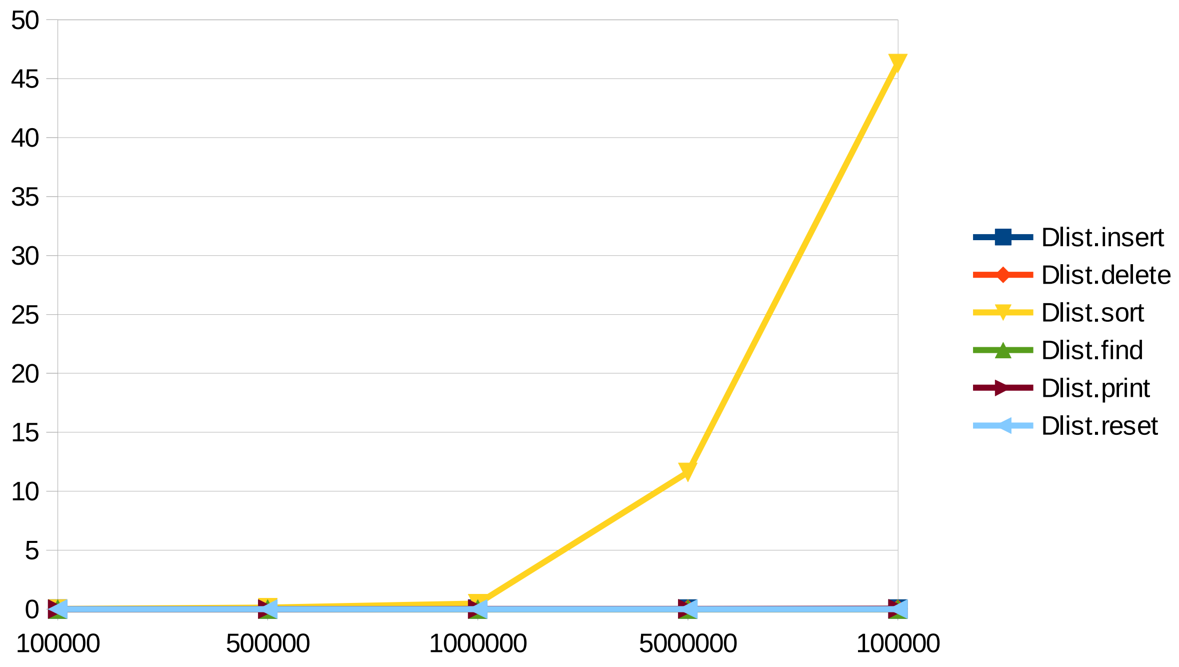
For Dlist and Array , I did exactly the same as instructed .

For BST ,I defined class BSTree and class BSTreeNode,similar to Dlist.Every BSTreeNode has

BSTreeNode* _left pointing to its left child , _right pointing to its right child and _parent pointing to its parent . If a node have no _left , _right or _parent(if it's root), the pointer points to itself.There

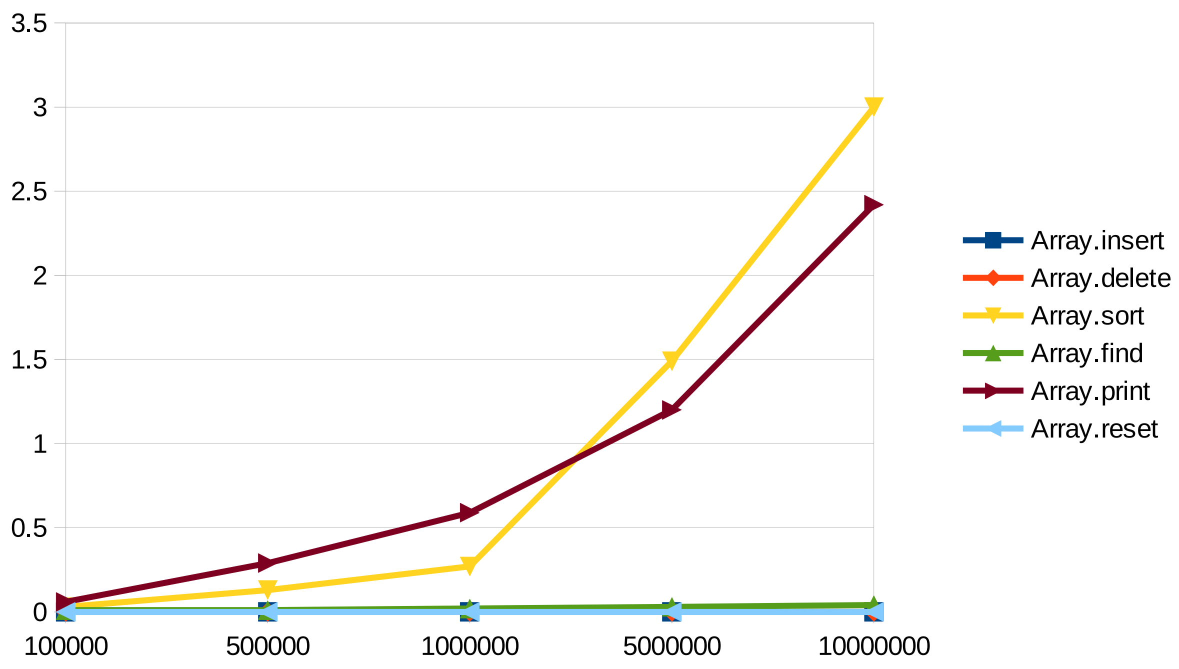
also two private fuctions findbig and findsmall in BSTreeNode,which finds the next and previous node .As for BSTree,there are three BSTreeNode* _root pointing to the root, _first pointing to the smallest element and dummy node _end pointing to the next node of the largest element.The member functions in BSTree is nothing else than the mustexist functions.

Experiment:



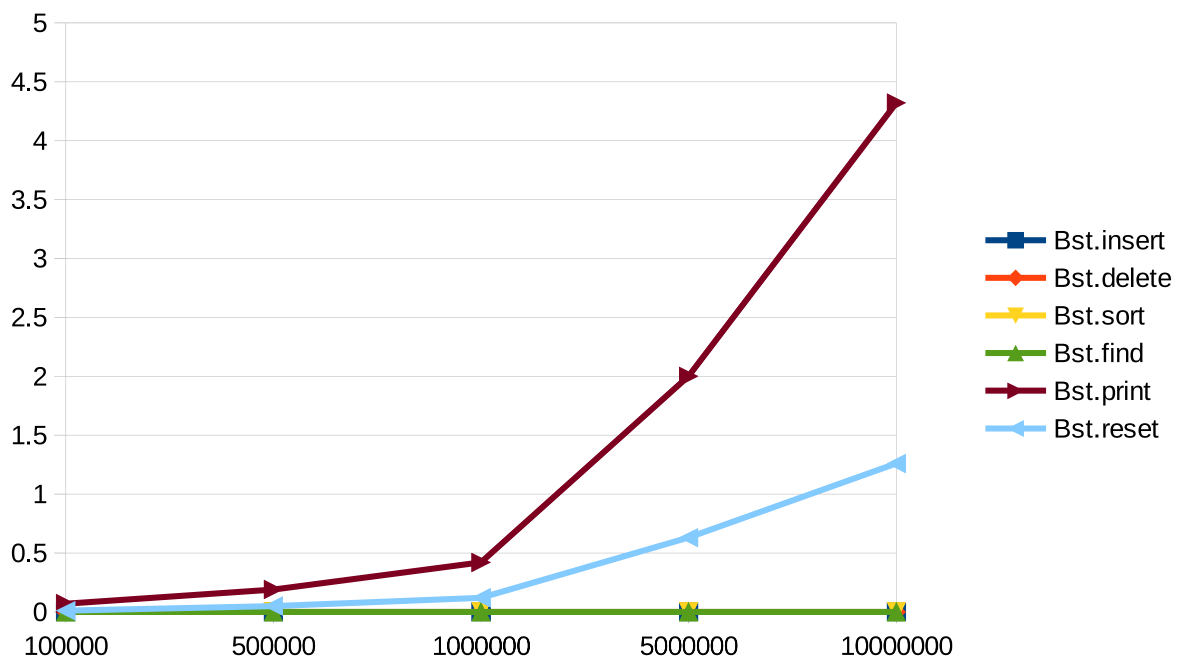
	100000	500000	1000000	5000000	100000
Dlist.insert	0	0	0	0	0
Dlist.delete	0	0	0	0	0
Dlist.sort	0.01	0.13	0.47	11.6	46.28
Dlist.find	0	0	0	0	0
Dlist.print	0	0	0.01	0.02	0.03
Dlist.reset	0	0	0	0	0

For Dlist, insert one node and delete one node is O(1).Sorting is about O(n^2),because I implemented Insertion sort.Intrestingly is that find one element , print , reset is so fast for all the cases.It seems like O(1) but they're all theoretically O(n).



	100000	500000	1000000	5000000	10000000
Array.insert	0	0	0	0	0
Array.delete	0	0	0	0	0
Array.sort	0.03	0.13	0.27	1.49	3
Array.find	0.01	0.01	0.02	0.03	0.04
Array.print	0.06	0.29	0.59	1.2	2.42
Array.reset	0	0	0	0	0

For Array, insert one node and delete one node is O(1),because we didn’t maintain the input order.Sorting is about O(n),implemented by std::sort().Print is about O(n),because each element costs O(1) .Reset is O(1),because only reset _size to 0.Intrestingly is that find one element is so fast for all the cases,but it’s theoretically O(n).So I did another test,I sorted the array before I do query. I queried “aaaab” and “zzzzz”,the result is 0 and 0.04,in the case of 10000000 inputs.The result is that searching through the array is so fast that we couldn’t tell the difference.



	100000	500000	1000000	5000000	10000000
Bst.insert	0	0	0	0	0
Bst.delete	0	0	0	0	0
Bst.sort	0	0	0	0	0
Bst.find	0	0	0	0	0
Bst.print	0.07	0.19	0.42	2	4.32
Bst.reset	0.01	0.05	0.12	0.63	1.26

For BST, insert one node, delete one node and find one node is so fast,probably O(lg n),also theoretically O(lg n).Sorting is no need because BST is already sorted.Print is about O(n),I think it’s because finding next element in BST is usually O(1) .Reset is about O(n) because I applied pop_front,which is O(1),in reset.