#Exercise 2 Dynamic data structures

Uladzimir Ivashka, 150281

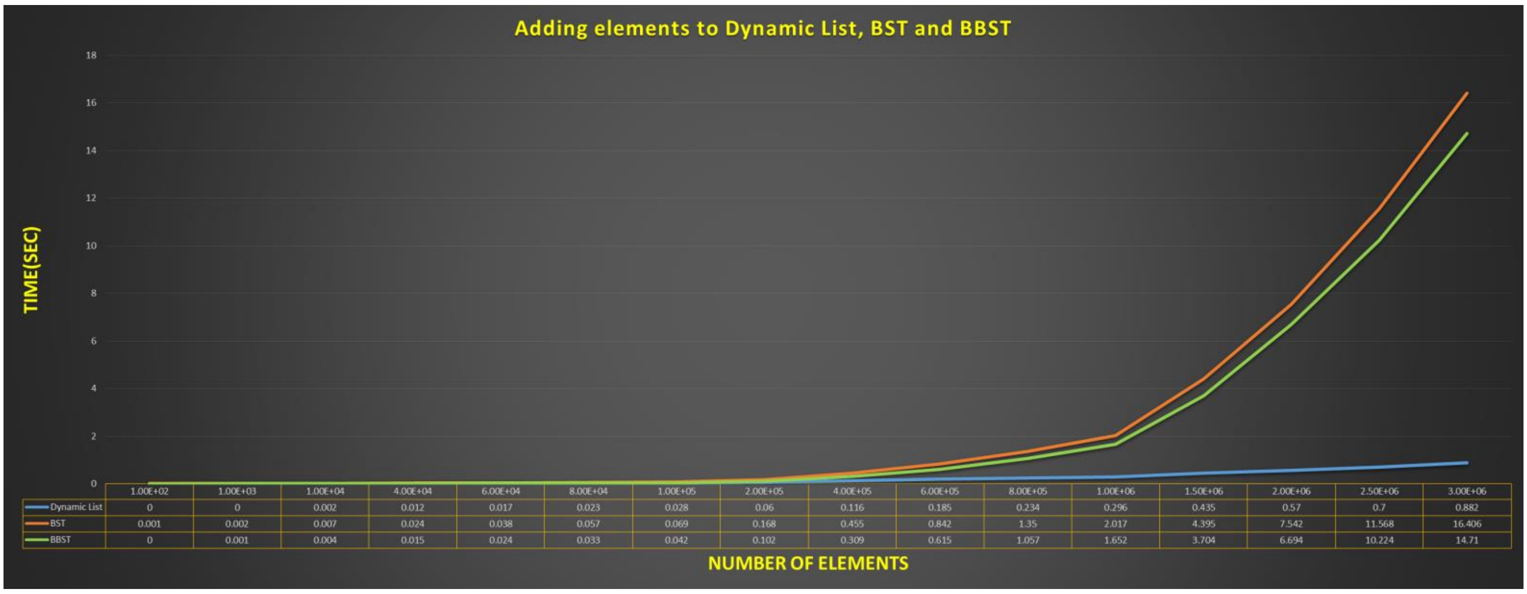
Sofya Aksenyuk, 150284

Remark:

Given results of all the below experiments are average of 10 runs.

The initial size of data set is 9 million of elements.

I. Dependence of the insertion time of three data structures: Dynamic Ordered List, BST, BBST on the number of evaluated elements.



	1.00E+02	1.00E+03	1.00E+04	4.00E+04	6.00E+04	8.00E+04	1.00E+05	2.00E+05	4.00E+05	6.00E+05	8.00E+05	1.00E+06	1.50E+06	2.00E+06	2.50E+06	3.00E+06
Dynamic List	0	0	0.002	0.012	0.017	0.023	0.028	0.06	0.116	0.185	0.234	0.296	0.435	0.57	0.7	0.882
BST	0.001	0.002	0.007	0.024	0.038	0.057	0.069	0.168	0.455	0.842	1.35	2.017	4.395	7.542	11.568	16.406
BBST	0	0.001	0.004	0.015	0.024	0.033	0.042	0.102	0.309	0.615	1.057	1.652	3.704	6.694	10.224	14.71

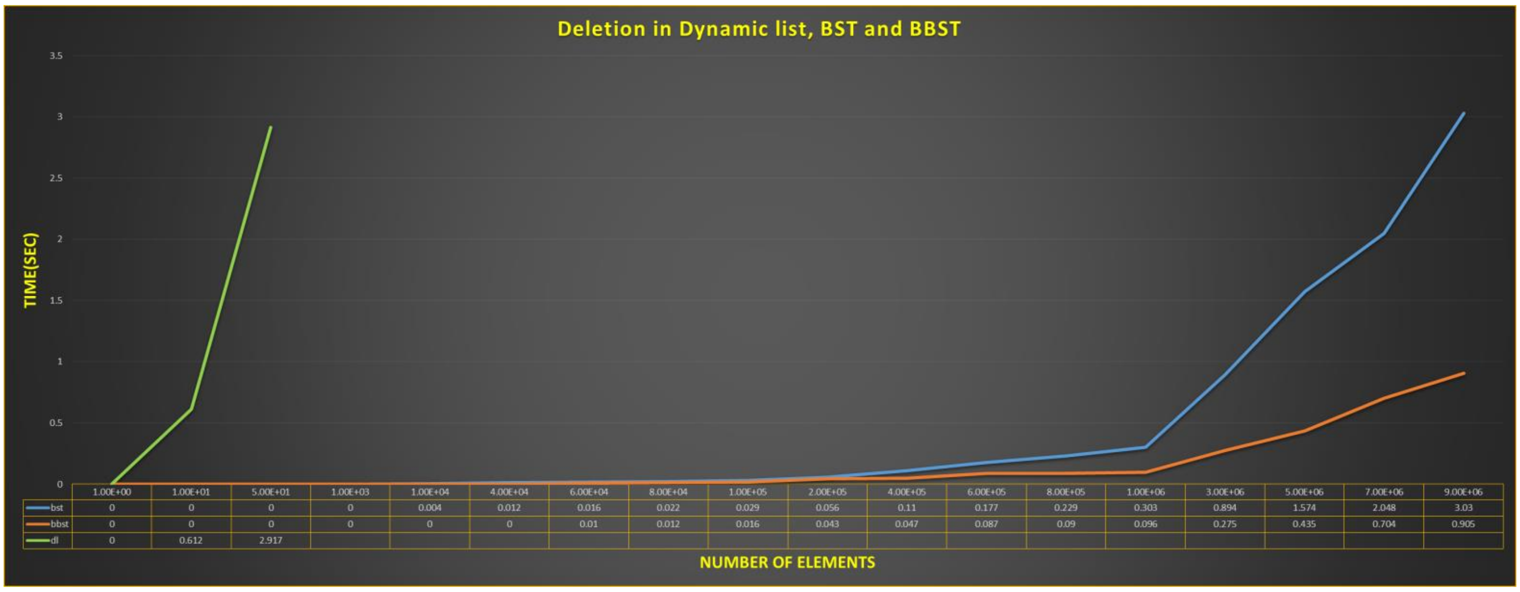
Conclusions:

According to the chart above, it can be concluded that Dynamic Ordered List shows the best results when it comes to inserting new elements. That can be explained by the complexity of O(1) for adding elements to the beginning of a list (in that case, it replaces head keeping following links as it was before). However, adding elements to the particular place, a program would need to activate “search” function as well which would lead to an increase of time complexity to O(n).

As for BST and BBST data structures, it can be noticed that they have almost the same results which is due to equal time complexity of O(log n) in average case. Nevertheless, BBST performs a bit better because of the its structure specificity (i.e., balanced type of levels depth).

Comparing experiment results of all the described structures together, the speed of Dynamic Ordered List is significantly higher than BST and BBST ones. That also corresponds to the theoretical background that states that Dynamic Ordered List with the complexity of O(1) takes less time for performing then O(log n) / O(n) of BST and BBST (i.e. O(1) < O(log n)).

II. Deletion of n elements from data structures.



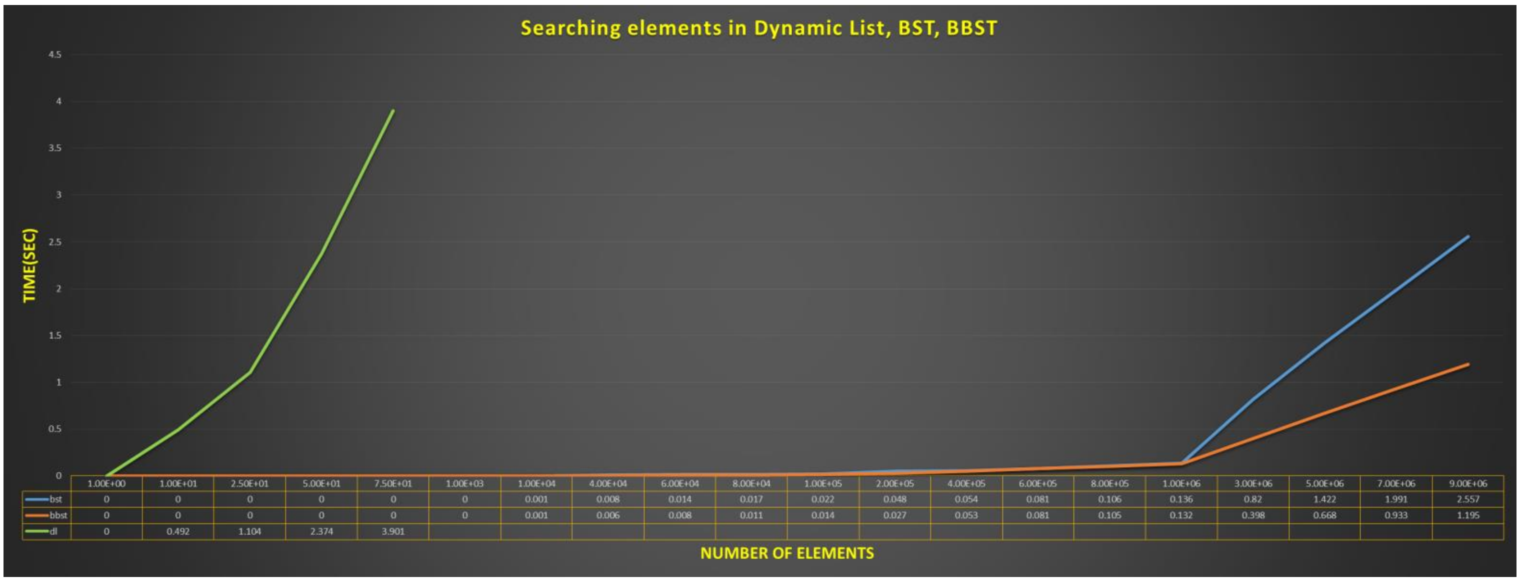
	1.00E+00	1.00E+01	5.00E+01	1.00E+03	1.00E+04	4.00E+04	6.00E+04	8.00E+04	1.00E+05	2.00E+05	4.00E+05	6.00E+05	8.00E+05	1.00E+06	3.00E+06	5.00E+06	7.00E+06	9.00E+06
bst	0	0	0	0	0.004	0.012	0.016	0.022	0.029	0.056	0.11	0.177	0.229	0.303	0.894	1.574	2.048	3.03
bbst	0	0	0	0	0	0	0.01	0.012	0.016	0.043	0.047	0.087	0.09	0.096	0.275	0.435	0.704	0.905
dl	0	0.612	2.917															

Conclusions:

As it could be noticed, Dynamic Ordered List deletes elements much slower than the other data structures (that is why only three measuring points are presented for this). That was expected because to delete an element it is firstly needed to find it, so that it takes O(n), whereas it is equal to O(log n) for BST and BBST.

Summing up, BBST works faster for data sets of all considered sizes which is the same for deletion of all elements (in our case, 9.00E+06 measuring point in the chart above). That happens due to the difference of BBST and BST worst cases that are O(log n) and O(n), respectively.

III. <u>Dependence of the searching time of three data structures: Dynamic List, BST, BBST on the number of evaluated elements.</u>



	1.00E+00	1.00E+01	2.50E+01	5.00E+01	7.50E+01	1.00E+03	1.00E+04	4.00E+04	6.00E+04	8.00E+04
bst	0	0	0	0	0	0	0.001	0.008	0.014	0.017
bbst	0	0	0	0	0	0	0.001	0.006	0.008	0.011
dl	0	0.492	1.104	2.374	3.901					

	1.00E+05	2.00E+05	4.00E+05	6.00E+05	8.00E+05	1.00E+06	3.00E+06	5.00E+06	7.00E+06	9.00E+06
bst	0.022	0.048	0.054	0.081	0.106	0.136	0.82	1.422	1.991	2.557
bbst	0.014	0.027	0.053	0.081	0.105	0.132	0.398	0.668	0.933	1.195
dl										

<u>Conclusions:</u>

When it comes to searching elements, Dynamic Ordered List has the slowest speed among supplied data structures which supports theoretical part of the experiment that is as follows: DOL O(n) > BST O(log n) = BBST O(log n). Nevertheless, BBST is still faster than BST because of the different worst cases (O(log n) < O(n)).

<u>Overall:</u>

Searching and deletion functions give us approximately the same trend of results: BBST is faster than BST that is faster than DOL. Such conclusions are supported by the same time complexities for deletion and searching among required data structures. As for adding elements, Dynamic Ordered List shows the best results because it simply inserts requested amount of elements at the beginning of the linked list that does not change its structure, in opposite to BST and BBST that need to parse, reform and rebuild all the tree in such a case. In other words, organization of each elements of a tree plays a significant role (each element has its own place), whereas there is no such need for linked list construction.

	add			
n elements	dl	bst	bbst	
1.00E+02	0	0.001	0	
1.00E+03	0	0.002	0.001	
1.00E+04	0.002	0.007	0.004	
4.00E+04	0.012	0.024	0.015	
6.00E+04	0.017	0.038	0.024	
8.00E+04	0.023	0.057	0.033	
1.00E+05	0.028	0.069	0.042	
2.00E+05	0.06	0.168	0.102	
4.00E+05	0.116	0.455	0.309	
6.00E+05	0.185	0.842	0.615	
8.00E+05	0.234	1.35	1.057	
1.00E+06	0.296	2.017	1.652	
1.50E+06	0.435	4.395	3.704	
2.00E+06	0.57	7.542	6.694	
2.50E+06	0.7	11.568	10.224	
3.00E+06	0.882	16.406	14.71	
5.00E+06	1.422	43.647	4.09E+01	
7.00E+06	2.015			
9.00E+06	2.6			

	search			
n elements	bst	bbst	dl	
1.00E+00	0	0	0	
1.00E+01	0	0	0.492	
2.50E+01	0	0	1.104	
5.00E+01	0	0	2.374	
7.50E+01	0	0	3.901	
1.00E+03	0	0		
1.00E+04	0.001	0.001		
4.00E+04	0.008	0.006		
6.00E+04	0.014	0.008		
8.00E+04	0.017	0.011		
1.00E+05	0.022	0.014		
2.00E+05	0.048	0.027		
4.00E+05	0.054	0.053		
6.00E+05	0.081	0.081		
8.00E+05	0.106	0.105		
1.00E+06	0.136	0.132		
3.00E+06	0.82	0.398		
5.00E+06	1.422	0.668		
7.00E+06	1.991	0.933		
9.00E+06	2.557	1.195		

	del		
n elements	bst	bbst	dl
1.00E+00	0	0	0
1.00E+01	0	0	0.612
5.00E+01	0	0	2.917
1.00E+03	0	0	
1.00E+04	0.004	0	
4.00E+04	0.012	0	
6.00E+04	0.016	0.01	
8.00E+04	0.022	0.012	
1.00E+05	0.029	0.016	
2.00E+05	0.056	0.043	
4.00E+05	0.11	0.047	
6.00E+05	0.177	0.087	
8.00E+05	0.229	0.09	
1.00E+06	0.303	0.096	
3.00E+06	0.894	0.275	
5.00E+06	1.574	0.435	
7.00E+06	2.048	0.704	
9.00E+06	3.03	0.905	