1. The code of Main:

```
package edu.neu.coe.info6205.symbolTable;

import java.util.Random;

public class test {
   public static void main(String[] args) {
      Random random = new Random(); int value = 1;
      BSTSimple<Integer, Integer> bst;
      //i is the initial number of seeds
      //if operation = 0, choose "put";
      //if operation = 1,choose "delete";
      //if operation = 2,choose "get"
      //the number of operation times are 1000
      for(int j = 100; j <= 2000; j+=20) {
         bst = new BSTSimple<>();
      long averageput = 0;
      long averagedelete = 0;
      long averageget = 0;
      long count0 = 0;
      long count1 = 0;
      long count2 = 0;
      long totaltime0 = 0;
      long totaltime1 = 0;
      long totaltime2 = 0;
      for(int i = 0; i < j; i++) {
             bst.put(i, value);
         }
      for(int ot = 0; ot < j*1000; ot++) {
         int operation = random.nextInt(3);
         int key = random.nextInt(j *2);
      if (operation == 0) {
      long stime0 = System.nanoTime();
      bst.put(key, value);
      long etime0 = System.nanoTime();
      totaltime0 += (etime0 - stime0);
      count0++;
      }
      else if(operation == 1) {
      long stime1 = System.nanoTime();
      bst.delete(key);
      long etime1 = System.nanoTime();
          totaltime1 += (etime1 - stime1);
      count1++;
      }
      else {
      long stime2 = System.nanoTime();
      bst.get(key);
      long etime2 = System.nanoTime();
      totaltime2 += (etime2 - stime2);
      count2++;
      }
      }
      averageput = totaltime0/count0;
      averagedelete = totaltime1/count1;
      averageget = totaltime2/count2;
      System.out.println(j + " "+ averageput + " " +
averagedelete + " " + averageget);
      System.out.println(Math.log(j)+" "+Math.sqrt(j));
      System.out.println(bst.getmaxDepth(bst.getRoot()));
      }
   }
}
```

The part of code of Queue_Elements:

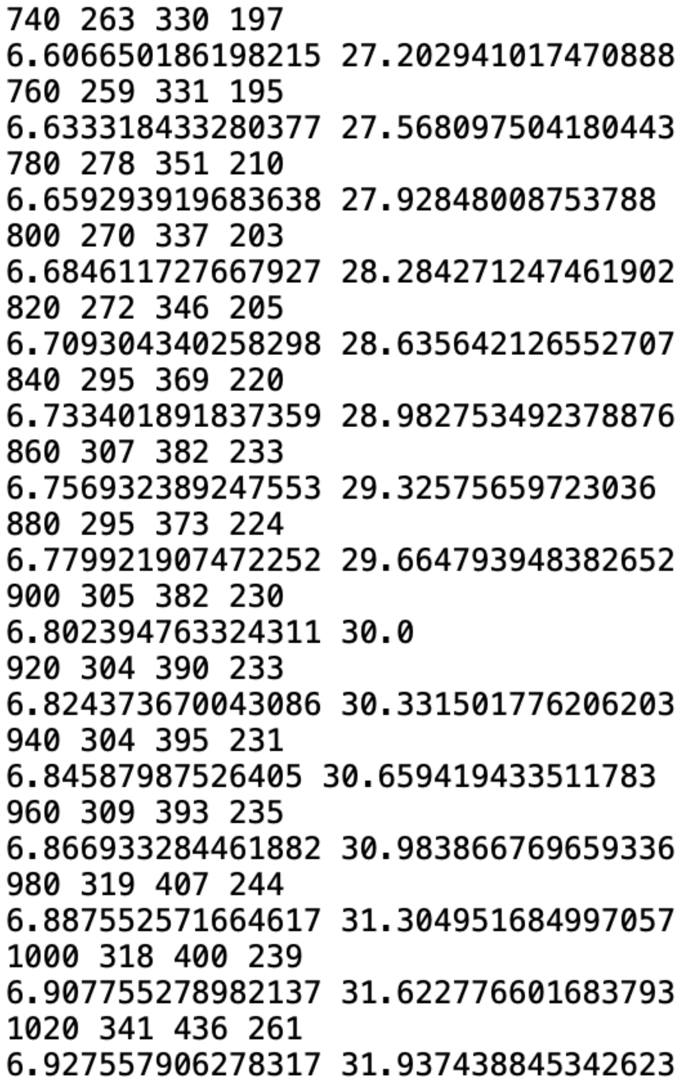
```
public void enqueue( Item item) {

      Element element = new Element<>(item);
      Element secondNewest = newest;
      if(isEmpty()) oldest = element;
      else {
         assert secondNewest != null;
         secondNewest.next =element;

      }
      this.newest =element;
      n++;
   }
   /**
    * Dequeue an element from the oldest list and return
the item.
    *
    * @return the value of the oldest element.
    */
   public Item dequeue() {
      n--;
```

```
        if (isEmpty()) return null;
        else {
          assert oldest != null;
          Item result =oldest.item;
          oldest =oldest.next;
          if(isEmpty()) newest =null;
           return result;
        }
    }
```

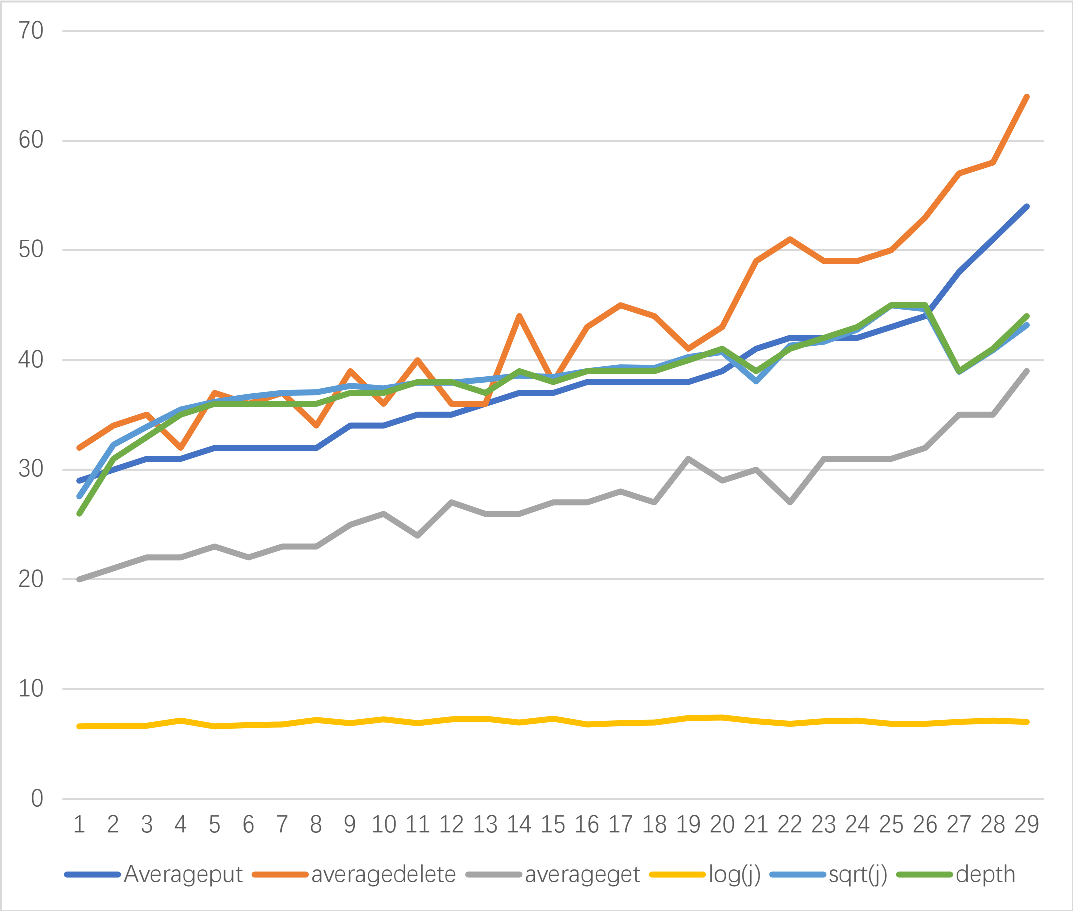
2. The results of code running



```
740 263 330 197
6.606650186198215 27.202941017470888
760 259 331 195
6.633318433280377 27.568097504180443
780 278 351 210
6.659293919683638 27.92848008753788
800 270 337 203
6.684611727667927 28.284271247461902
820 272 346 205
6.709304340258298 28.635642126552707
840 295 369 220
6.733401891837359 28.982753492378876
860 307 382 233
6.756932389247553 29.32575659723036
880 295 373 224
6.779921907472252 29.664793948382652
900 305 382 230
6.802394763324311 30.0
920 304 390 233
6.824373670043086 30.331501776206203
940 304 395 231
6.84587987526405 30.659419433511783
960 309 393 235
6.866933284461882 30.983866769659336
980 319 407 244
6.887552571664617 31.304951684997057
1000 318 400 239
6.907755278982137 31.622776601683793
1020 341 436 261
6.927557906278317 31.937438845342623
```

This test implements insertion function, delete function, and search function. The number of nodes I randomly generated was 100 and the number of operations was 1000, and then the number of randomly generated nodes was gradually increased. I made 96 tests in all. The number of nodes ranged from

1000 to 2000. The value is between log(n) and n. What’s more, the depth of the tree tends to sqrt(n).



Averageput	averagedelete	averageget	log(j)	sqrt(j)	depth
29	32	20	6.63332	27.56811	26
30	34	21	6.68461	32.28427	31
31	35	22	6.65929	33.92848	33
31	32	22	7.138867	35.496478	35
32	37	23	6.60665	36.20294	36
32	36	22	6.70931	36.63564	36
32	37	23	6.75693	36.98275	36

32	34	23	7.17012	37.05551	36
34	39	25	6.90775	37.62278	37
34	36	26	7.24423	37.41657	37
35	40	24	6.92756	37.93743	38
35	36	27	7.27239	37.94733	38
36	36	26	7.28619	38.20994	37
37	44	26	6.96602	38.55764	39
37	38	27	7.29981	38.47077	38
38	43	27	6.80239	39	39
38	45	28	6.88755	39.30495	39
38	44	27	6.94698	39.24903	39
38	41	31	7.39018	40.24922	40
39	43	29	7.41457	40.7431	41
41	49	30	7.05618	38.05877	39
42	51	27	6.82437	41.33152	41
42	49	31	7.09008	41.64102	42
42	49	31	7.15462	42.77708	43
43	50	31	6.86693	44.98387	45
44	53	32	6.84588	44.65942	45
48	57	35	7.02108	38.92831	39
51	58	35	7.10661	40.92851	41
54	64	39	7.00306	43.16625	44

The search operation is the least complex and the insertion operation is not as complex as the deletion operation. Their complexity are all between O(N^1/2) and 0(lg N), the put and delete operation are tend to O(N^1/2) and the search operation tends to O(lg N).