Högskolan i Gävle

Laboration 3

William Tiderman
williamtiderman@live.se
19wit01

John Engblom Sandin
engblomsandinjohn@gmail.com
19joen03

2020-12-13

Kurs: Algoritmer och datastrukturer 7.5hp

Lärare: Anders Jackson
Lärare/handledare: Hanna Holmgren
Lärare/handledare: Atique Ullah

Uppgifter

1

Uppgift 1 var att undersöka hur en prioritetskö utför insättning och borttagning. Prioritetskön gjordes med två olika metoder, en heapkö och BST-kö, BST var baserat på ett binärt sökträd. Sedan skulle de båda typer av en prioritetskö undersökas för osorterat, sorterat och sorterat baklänges, för att kolla var som var snabbast av insättning och borttagning av element.

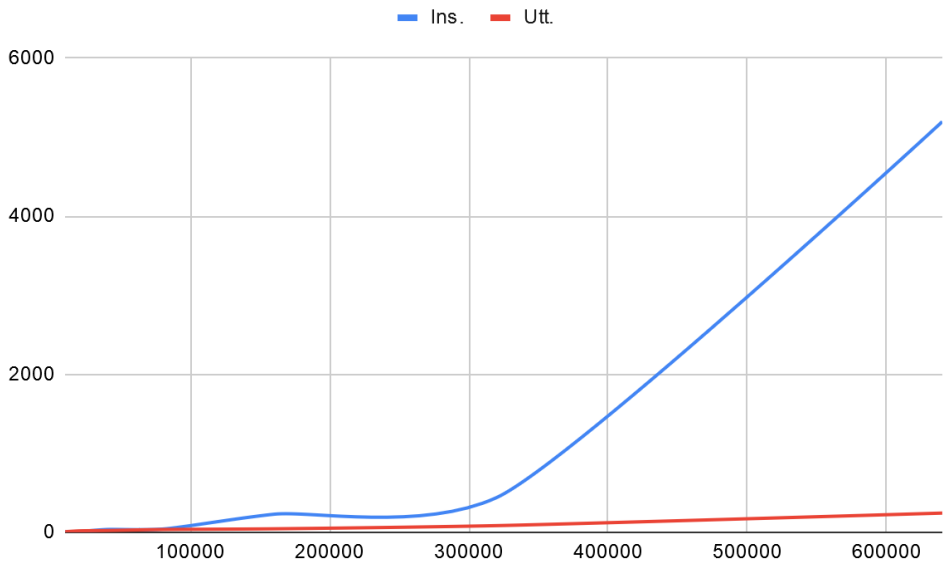
```java
@Test
void testSortedHeapEnqueue() throws FileNotFoundException, IOException, DuplicateItemException, EmptyQueueException {
    reader = new fileRead(filePath, sizeTest);
    list = reader.loadListFromFile();
    Collections.sort(list, Collections.reverseOrder());
    for(int i = 0; i<sizeTest; i++) {
        Heap.enqueue(list.get(i));
    }
    long t1 = System.currentTimeMillis();
    for(int i = 0; i<sizeTest; i++) {
        Heap.dequeue();
    }

    long exeTime = System.currentTimeMillis() - t1;
    System.out.println(exeTime);
}
```

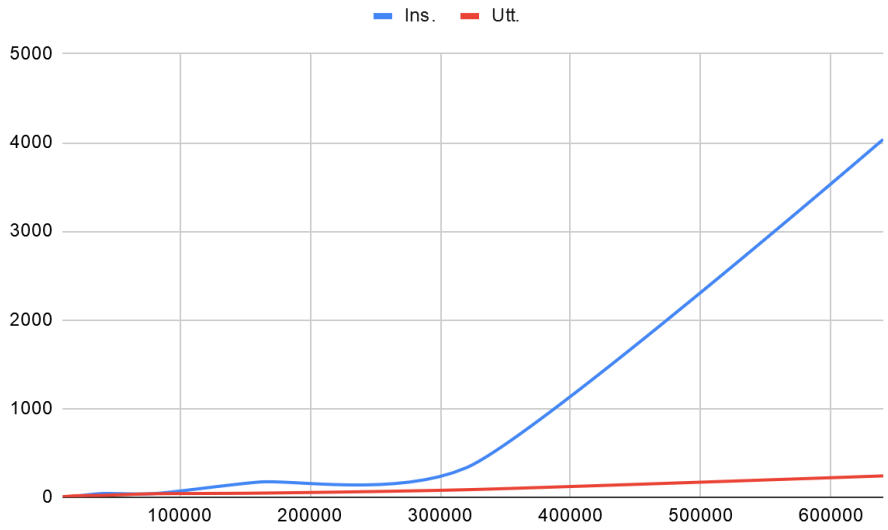
I figuren ovan så visas våran borttagning av element med hjälp av Heap queuen. Det är en av de 12 olika tester vi gjorde med våra prioritetsköar.

Sedan skulle förhållandet mellan insättning och uttagning av element representeras i diagram:

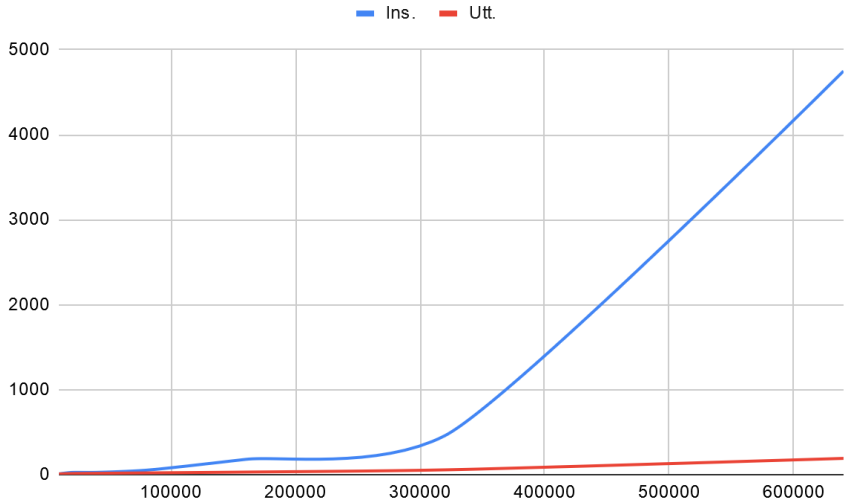
Osorterad Heap:



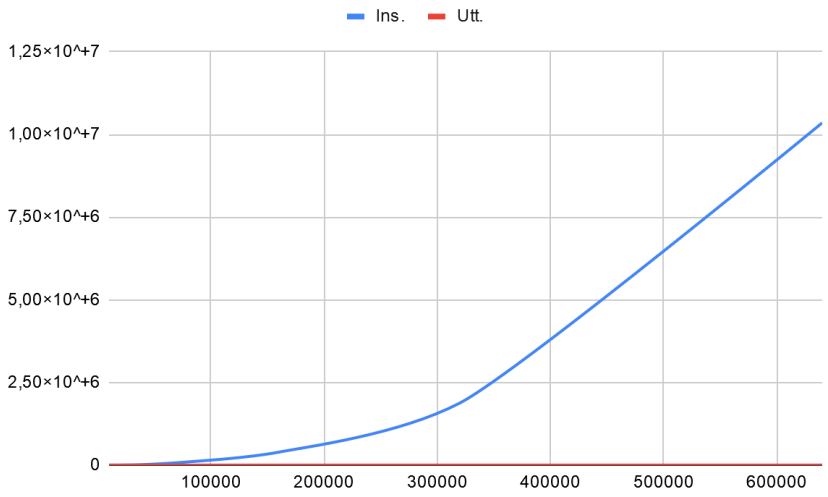
Sorterad Heap:



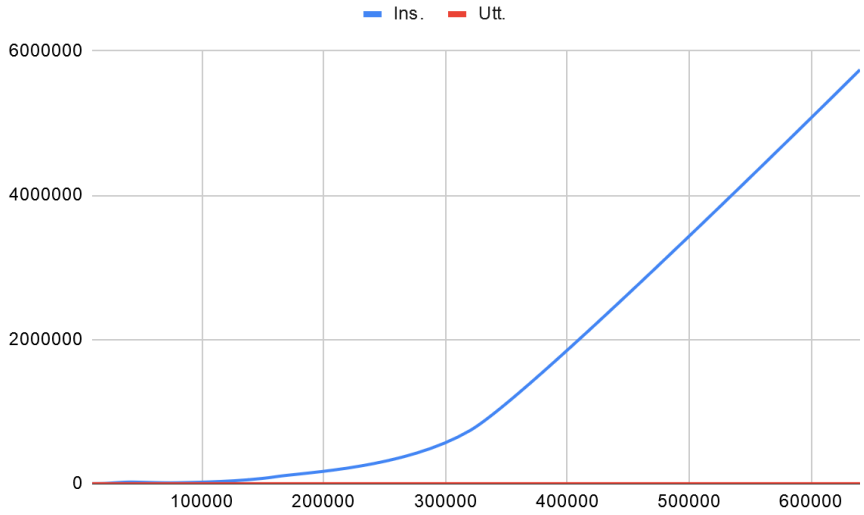
Sorterad bakvänt Heap:



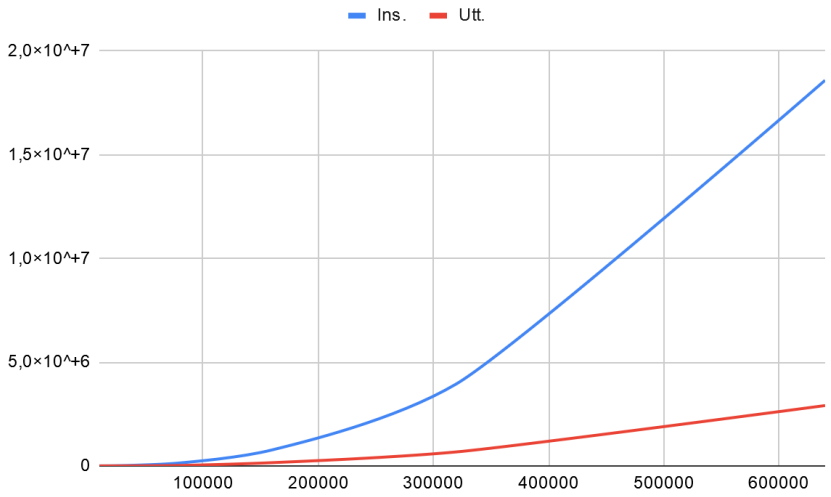
Osorterad BST:



Sorterad BST:



Sorterad bakvänt BST:



2

Uppgift 2 så skulle ett interface skapas för prioritetskön.

3

Uppgift 3 så skulle ett testprogram för prioriteskön skapas, men detta testprogram kan inte implementeras för det inte finns en implmentation.

Diskussionsfrågor

1

Det finns inget som stoppar dubletter i heap-prioritetskön eftersom den bryr sig inte om dubletter. Om det ska vara ej-tillåtna så måste det implementeras själv. I BST kön så är det implementerat av det binära sökträdet att dubletter inte ska finnas.

2

(A):
Det värsta och genomsnittliga fallet så är det O(n). För att den måste gå igenom varje nod i kön.
(B):
Det värsta fall är O(n), och det genomsnittliga om den är (O(Log(n))
(C):
Värsta fallet är (OLog(n)), genomsnittliga är (OLog(n))

3

(A):
Värsta fallet O(1), genomsnittliga O(1)
(B):
Värsta fallet O(n), genomsnittliga O(Log(n))
(C):
Värsta fallet O(Log(n)), genomsnittliga O(Log(n))

Appendix

PriorityQueue

```
1. package lab3;
2.
3. /**
```

```
 * @author John Engblom Sandin
 * @author William Tiderman
 * @version 2020-12-13
 */

public interface PriorityQueue <T extends Comparable<? super T>> {

    public void clear();

    public boolean isEmpty();

    public boolean isFull();

    public int size();

    public void enqueue(T t);

    public T dequeue();

    public T getFront();
}
```

PriorityQueueEmptyException

```
package lab3;
/**
 * Exception thrown when no data in Queue<V>
 *
 * @author Anders Jackson
 * @author John Engblom Sandin
 * @author William Tiderman
 * @version 2019-11-15
 */
public class PriorityQueueEmptyException extends RuntimeException {
    static final long serialVersionUID = 1;

    public PriorityQueueEmptyException() {
        super();
    }

    public PriorityQueueEmptyException(String message) {
        super(message);
    }

    public PriorityQueueEmptyException(String message, Throwable cause) {
        super(message, cause);
    }

    public PriorityQueueEmptyException(Throwable cause) {
```

```
26.         super(cause);
27.     }
28.
29.     protected PriorityQueueEmptyException(String message, Throwable cause,
30.             boolean enableSuppression,
31.             boolean writableStackTrace) {
32.         super(message, cause, enableSuppression, writableStackTrace);
33.     }
34. }
```

PriorityQueueFullException

```
1.  package lab3;
2.  /**
3.   * Exception thrown when no data in Queue<V>
4.   *
5.   * @author Anders Jackson
6.   * @author John Engblom Sandin
7.   * @author William Tiderman
8.   * @version 2020-12-13
9.   */
10. public class PriorityQueueFullException extends RuntimeException {
11.     static final long serialVersionUID = 1;
12.
13.     public PriorityQueueFullException() {
14.         super();
15.     }
16.
17.     public PriorityQueueFullException(String message) {
18.         super(message);
19.     }
20.
21.     public PriorityQueueFullException(String message, Throwable cause) {
22.         super(message, cause);
23.     }
24.
25.     public PriorityQueueFullException(Throwable cause) {
26.         super(cause);
27.     }
28.
29.     protected PriorityQueueFullException(String message, Throwable cause,
30.             boolean enableSuppression,
31.             boolean writableStackTrace) {
32.         super(message, cause, enableSuppression, writableStackTrace);
33.     }
34. }
```

PriorityQueueStub

```
1. package lab3;
2. 
3. 
4. /**
5.  * @author John Engblom Sandin
6.  * @author William Tiderman
7.  * @version 2020-12-13
8.  */
9. 
10. public class PriorityQueueStub<T extends Comparable<? super T>> implements Prior
    ityQueue<T> {
11. 
12.     @Override
13.     public void clear() {
14.         // TODO Auto-generated method stub
15. 
16.     }
17. 
18.     @Override
19.     public boolean isEmpty() {
20.         // TODO Auto-generated method stub
21.         return false;
22.     }
23. 
24.     @Override
25.     public boolean isFull() {
26.         // TODO Auto-generated method stub
27.         return false;
28.     }
29. 
30.     @Override
31.     public int size() {
32.         // TODO Auto-generated method stub
33.         return 0;
34.     }
35. 
36.     @Override
37.     public void enqueue(T t) {
38.         // TODO Auto-generated method stub
39. 
40.     }
41. 
42.     @Override
43.     public T dequeue() {
44.         // TODO Auto-generated method stub
45.         return null;
46.     }
47. 
```

```
48.     @Override
49.     public T getFront() {
50.         // TODO Auto-generated method stub
51.         return null;
52.     }
53.
54. }
```

PriorityQueueTest

```
1.  package lab3;
2.
3.  import static org.junit.jupiter.api.Assertions.*;
4.
5.
6.
7.  import org.junit.jupiter.api.AfterEach;
8.  import org.junit.jupiter.api.BeforeEach;
9.  import org.junit.jupiter.api.Test;
10.
11. /**
12.  * @author John Engblom Sandin
13.  * @author William Tiderman
14.  * @version 2020-12-13
15.  */
16.
17.
18. class PriorityQueueTest {
19.
20.     PriorityQueue<Integer> testQueue;
21.     static final int[] INT_FIXTURE = { 1, 2, 3, 4, 5 };
22.
23.     @BeforeEach
24.     void setUp() throws Exception {
25.         testQueue = new PriorityQueueStub<Integer>();
26.         for (int data : INT_FIXTURE) {
27.             testQueue.enqueue(data);
28.         }
29.     }
30.
31.     @AfterEach
32.     void tearDown() throws Exception {
33.         testQueue = null;
34.     }
35.
36.     @Test
37.     void testClear() {
38.
```

```
39.         assertFalse(testQueue.isEmpty(),"Metod inte implementerad");
40.
41.         testQueue.clear();
42.
43.         assertTrue(testQueue.isEmpty(),"Metod inte implementerad");
44.
45.     }
46.
47.     @Test
48.     void testIsFull() {
49.
50.         assertFalse(testQueue.isFull(),"Metod inte implementerad");
51.
52.         testQueue.clear();
53.
54.         assertFalse(testQueue.isFull(),"Metod inte implementerad");
55.
56.     }
57.
58.     @Test
59.     void testEnqueue() {
60.         testQueue.clear();
61.
62.         assertTrue(testQueue.isEmpty(),"Metod inte implementerad");
63.
64.         testQueue.enqueue(6);
65.
66.         assertFalse(testQueue.isEmpty(),"Metod inte implementerad");
67.
68.     }
69.
70.     @Test
71.     void testDequeue() {
72.
73.         assertFalse(testQueue.isEmpty(),"Metod inte implementerad");
74.         assertEquals( (double) testQueue.dequeue(), (double) 5,"Metod inte implementer
    ad");
75.         assertEquals( (double) testQueue.dequeue(), (double) 4,"Metod inte implementer
    ad");
76.
77.
78.     }
79.
80.     @Test
81.     void testGetFront() {
82.
83.         assertEquals( (double) testQueue.getFront(), (double) 5,"Metod inte implementer
    ad");
84.         assertEquals( (double) testQueue.getFront(), (double) 5,"Metod inte implementer
    ad");
```

```
85.         assertEquals( (double) testQueue.dequeue(), (double) 5,"Metod inte implementer
    ad");
86.         assertEquals( (double) testQueue.getFront(), (double) 4,"Metod inte implementer
    ad");
87.
88.
89.     }
90.
91.     @Test
92.     void testDequeueOnEmptyQueue(){
93.
94.         testQueue.clear();
95.         assertThrows(PriorityQueueEmptyException.class,
96.                 () -> testQueue.dequeue(),
97.                 "Expected: PriorityQueueEmptyException");
98.
99.     }
100.
101.            @Test
102.            void testGetFrontOnEmptyQueue(){
103.                testQueue.clear();
104.                assertThrows(PriorityQueueEmptyException.class,
105.                        () -> testQueue.getFront(),
106.                        "Expected: PriorityQueueEmptyException");
107.            }
108.
109.            @Test
110.            void testEnqueueOnFullQueue(){
111.
112.                assertFalse(testQueue.isFull(),"Metod inte implementerad");
113.                assertThrows(PriorityQueueFullException.class,
114.                        () -> testQueue.enqueue(6),
115.                        "Expected: PriorityQueueFullException");
116.            }
117.
118.        }
```