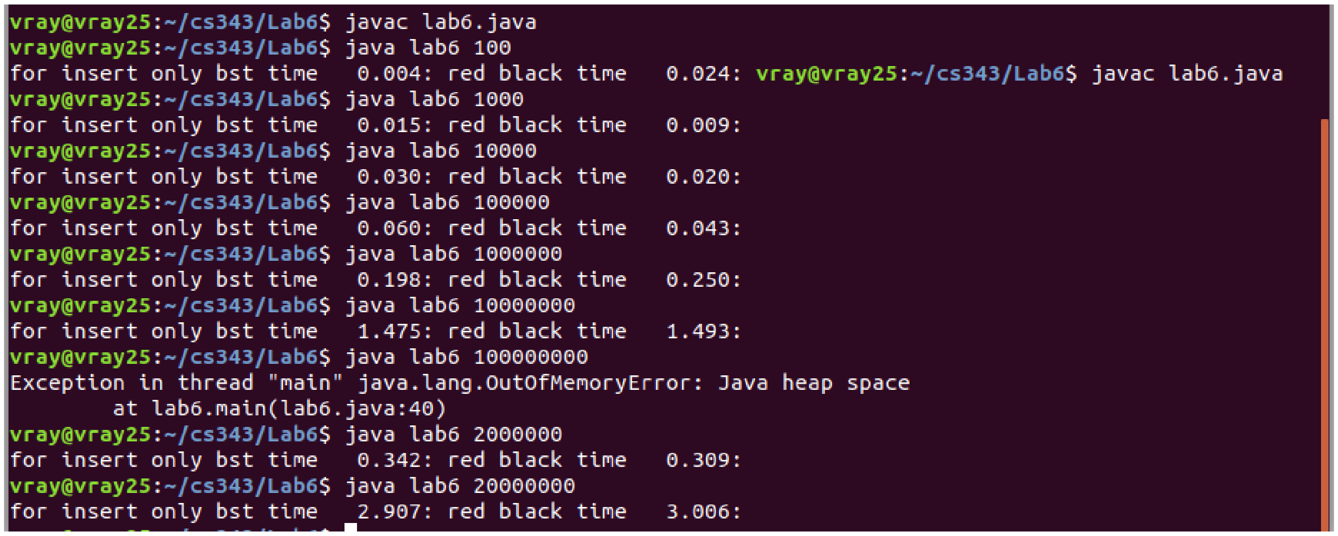
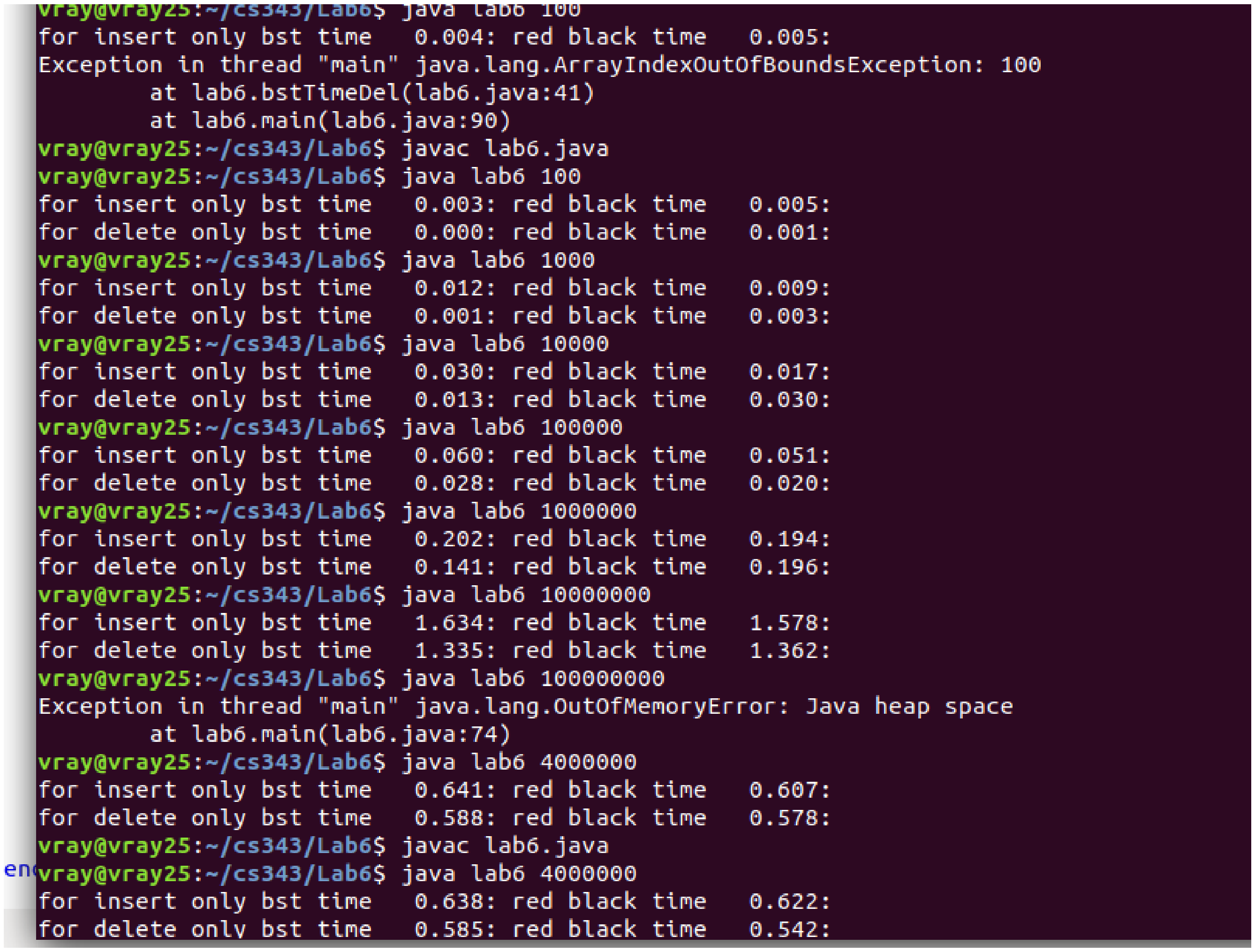


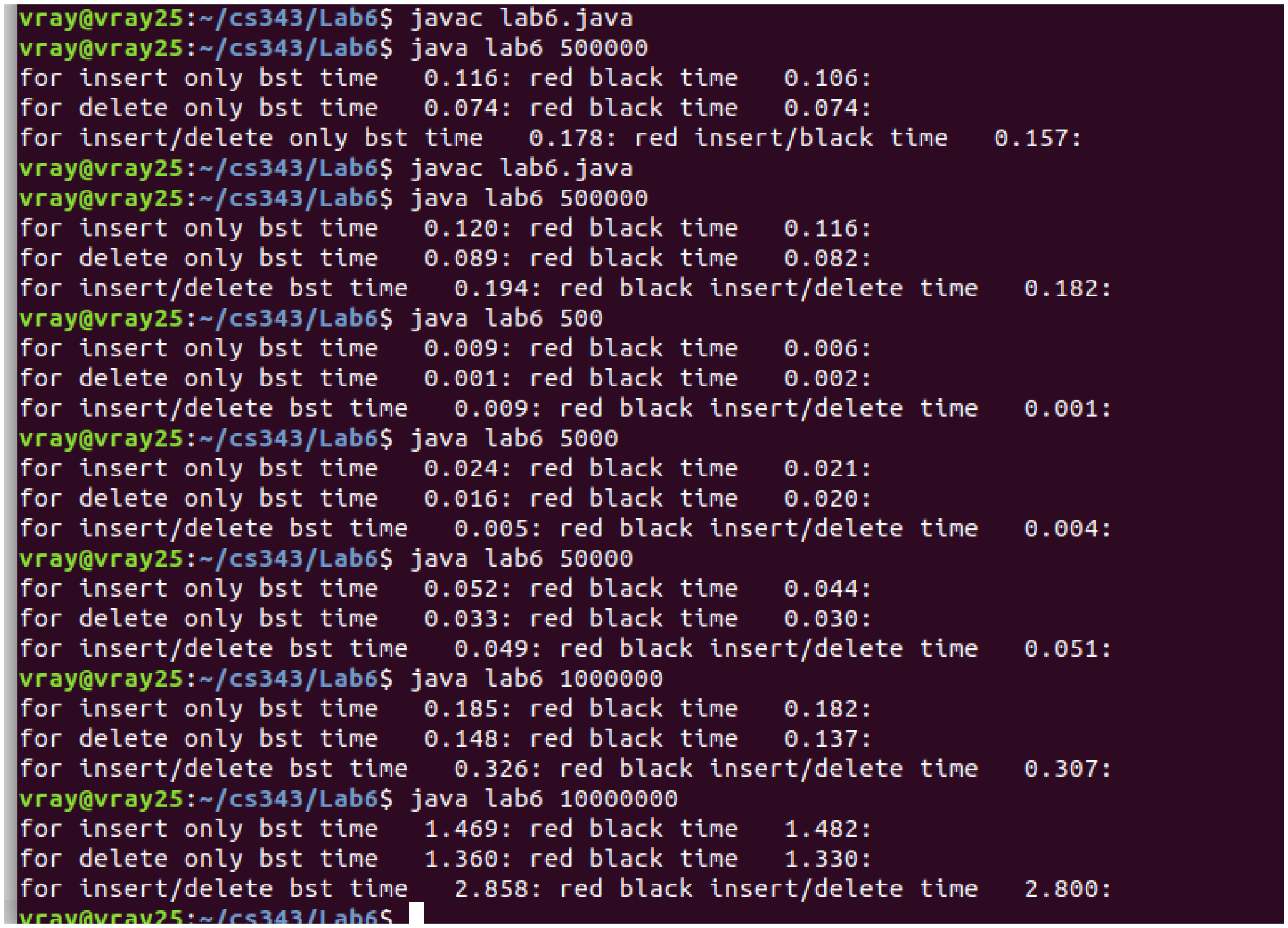
```
vray@vray25:~/cs343/Lab6$ javac lab6.java
vray@vray25:~/cs343/Lab6$ java lab6 100
for insert only bst time   0.004: red black time   0.024: vray@vray25:~/cs343/Lab6$ javac lab6.java
vray@vray25:~/cs343/Lab6$ java lab6 1000
for insert only bst time   0.015: red black time   0.009:
vray@vray25:~/cs343/Lab6$ java lab6 10000
for insert only bst time   0.030: red black time   0.020:
vray@vray25:~/cs343/Lab6$ java lab6 100000
for insert only bst time   0.060: red black time   0.043:
vray@vray25:~/cs343/Lab6$ java lab6 1000000
for insert only bst time   0.198: red black time   0.250:
vray@vray25:~/cs343/Lab6$ java lab6 10000000
for insert only bst time   1.475: red black time   1.493:
vray@vray25:~/cs343/Lab6$ java lab6 100000000
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at lab6.main(lab6.java:40)
vray@vray25:~/cs343/Lab6$ java lab6 2000000
for insert only bst time   0.342: red black time   0.309:
vray@vray25:~/cs343/Lab6$ java lab6 20000000
for insert only bst time   2.907: red black time   3.006:
```

Based on my studies above I have learned that for insert only red black tree is more efficient than bst but once the number of insertions is over 1000000 than bst becomes more efficient.



```
vray@vray25:~/cs343/Lab6$ java lab6 100
for insert only bst time   0.004: red black time   0.005:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 100
        at lab6.bstTimeDel(lab6.java:41)
        at lab6.main(lab6.java:90)
vray@vray25:~/cs343/Lab6$ javac lab6.java
vray@vray25:~/cs343/Lab6$ java lab6 100
for insert only bst time   0.003: red black time   0.005:
for delete only bst time   0.000: red black time   0.001:
vray@vray25:~/cs343/Lab6$ java lab6 1000
for insert only bst time   0.012: red black time   0.009:
for delete only bst time   0.001: red black time   0.003:
vray@vray25:~/cs343/Lab6$ java lab6 10000
for insert only bst time   0.030: red black time   0.017:
for delete only bst time   0.013: red black time   0.030:
vray@vray25:~/cs343/Lab6$ java lab6 100000
for insert only bst time   0.060: red black time   0.051:
for delete only bst time   0.028: red black time   0.020:
vray@vray25:~/cs343/Lab6$ java lab6 1000000
for insert only bst time   0.202: red black time   0.194:
for delete only bst time   0.141: red black time   0.196:
vray@vray25:~/cs343/Lab6$ java lab6 10000000
for insert only bst time   1.634: red black time   1.578:
for delete only bst time   1.335: red black time   1.362:
vray@vray25:~/cs343/Lab6$ java lab6 100000000
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at lab6.main(lab6.java:74)
vray@vray25:~/cs343/Lab6$ java lab6 4000000
for insert only bst time   0.641: red black time   0.607:
for delete only bst time   0.588: red black time   0.578:
vray@vray25:~/cs343/Lab6$ javac lab6.java
vray@vray25:~/cs343/Lab6$ java lab6 4000000
for insert only bst time   0.638: red black time   0.622:
for delete only bst time   0.585: red black time   0.542:
```

After studying delete only I seem to find that both bst and red black tree are pretty even. In some cases BST wins and in other cases red black tree wins. With small number of deletes I seem to find that the difference is quite big compared to the difference when the deletion operations are greater than 1 million.



```
vray@vray25:~/cs343/Lab6$ javac lab6.java
vray@vray25:~/cs343/Lab6$ java lab6 500000
for insert only bst time   0.116: red black time   0.106:
for delete only bst time   0.074: red black time   0.074:
for insert/delete only bst time   0.178: red insert/black time   0.157:
vray@vray25:~/cs343/Lab6$ javac lab6.java
vray@vray25:~/cs343/Lab6$ java lab6 500000
for insert only bst time   0.120: red black time   0.116:
for delete only bst time   0.089: red black time   0.082:
for insert/delete bst time   0.194: red black insert/delete time   0.182:
vray@vray25:~/cs343/Lab6$ java lab6 500
for insert only bst time   0.009: red black time   0.006:
for delete only bst time   0.001: red black time   0.002:
for insert/delete bst time   0.009: red black insert/delete time   0.001:
vray@vray25:~/cs343/Lab6$ java lab6 5000
for insert only bst time   0.024: red black time   0.021:
for delete only bst time   0.016: red black time   0.020:
for insert/delete bst time   0.005: red black insert/delete time   0.004:
vray@vray25:~/cs343/Lab6$ java lab6 50000
for insert only bst time   0.052: red black time   0.044:
for delete only bst time   0.033: red black time   0.030:
for insert/delete bst time   0.049: red black insert/delete time   0.051:
vray@vray25:~/cs343/Lab6$ java lab6 1000000
for insert only bst time   0.185: red black time   0.182:
for delete only bst time   0.148: red black time   0.137:
for insert/delete bst time   0.326: red black insert/delete time   0.307:
vray@vray25:~/cs343/Lab6$ java lab6 10000000
for insert only bst time   1.469: red black time   1.482:
for delete only bst time   1.360: red black time   1.330:
for insert/delete bst time   2.858: red black insert/delete time   2.800:
vray@vray25:~/cs343/Lab6$
```

I assumed bst would be faster at insert and deleting but based on my studies which can be wrong, I found that red black tree is faster than bst. What I did was create random arrays and used the timer tool in java to get the execution time. I inserted and deleted millions and hundreds of times and I made sure I deleted in a different order from insertion. I am surprised to see that red black is faster and insert and delete which makes me worry because I thought bst was faster. There was only one case where bst was faster and that’s when I inserted and deleted 50000 times.