INFO 6205 PSA SUMMER FINAL PROJECT

PRASAD PATHAK (002925486)
ANVITHA LAKSHMISHA (002965377)

TOPICS

1. Hibbard deletion of BST
   According to the course lecture notes, after several (Hibbard) deletions have been made, the average height of the tree is $\sqrt{n}$. Do you agree with this? How does it look after modifying the deletion process to either (a) randomly choose which direction to look for the node to be deleted or (b) choose the direction according to the size of the candidate nodes.

2. Quicksort
   Do you agree that the number of swaps in "standard" quicksort is 1/6 times the number of comparisons? Is this figure correct? If not, why not. How do you explain it.

3. 2-3 tree
   Implementation of 2-3 Tree. Benchmark various operations of 2-3 Tree like Insert, Search, and depth of the tree, draw comparisons against BST

BST DELETION

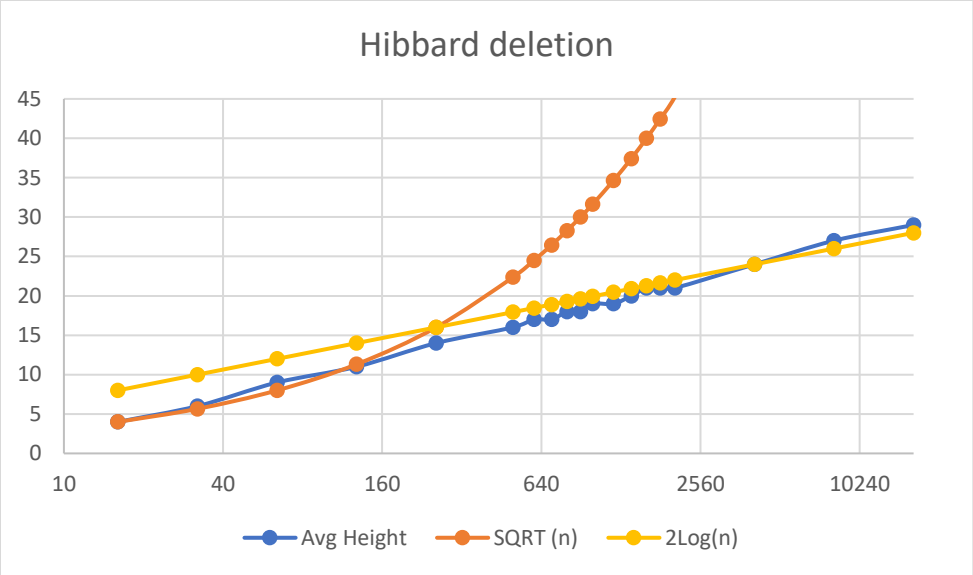
Code Links

BSTOptimizeDelete.java
BSTOptimizeDeleteTest.java

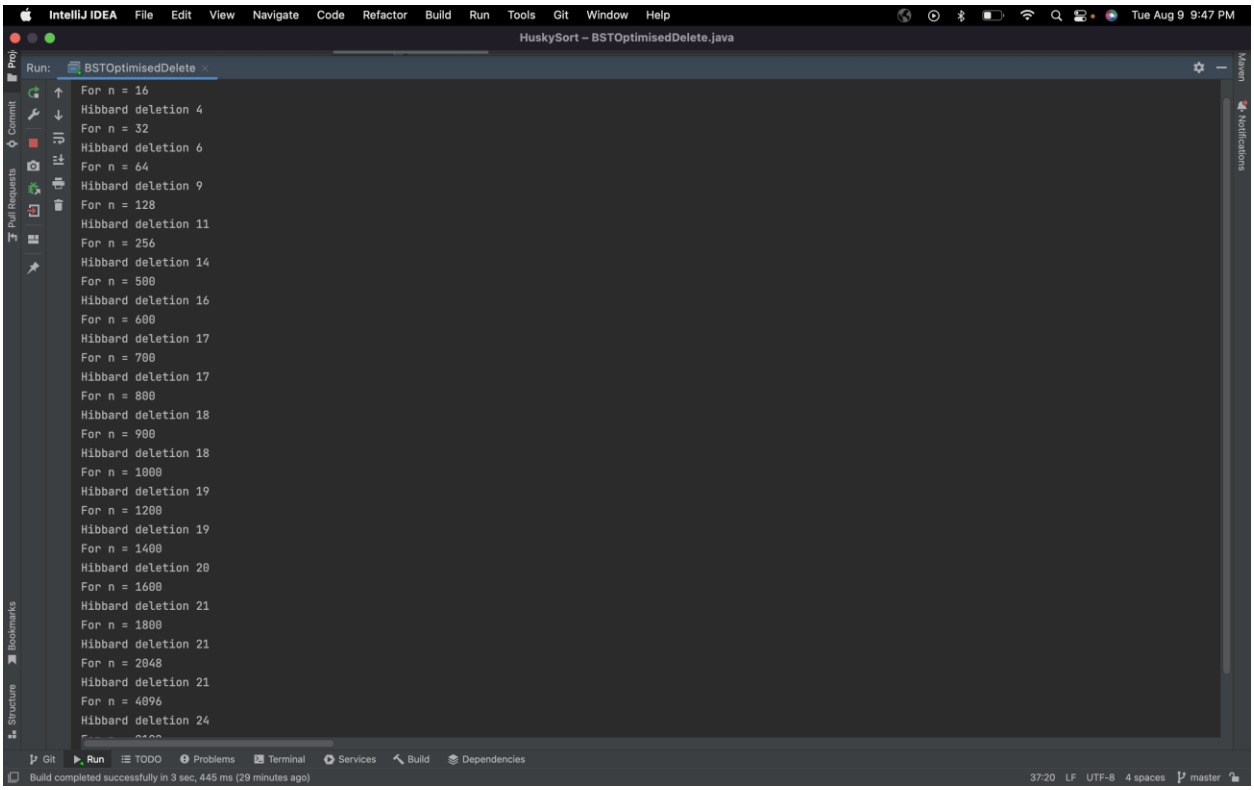
Analysis

Hibbard deletion values for different n

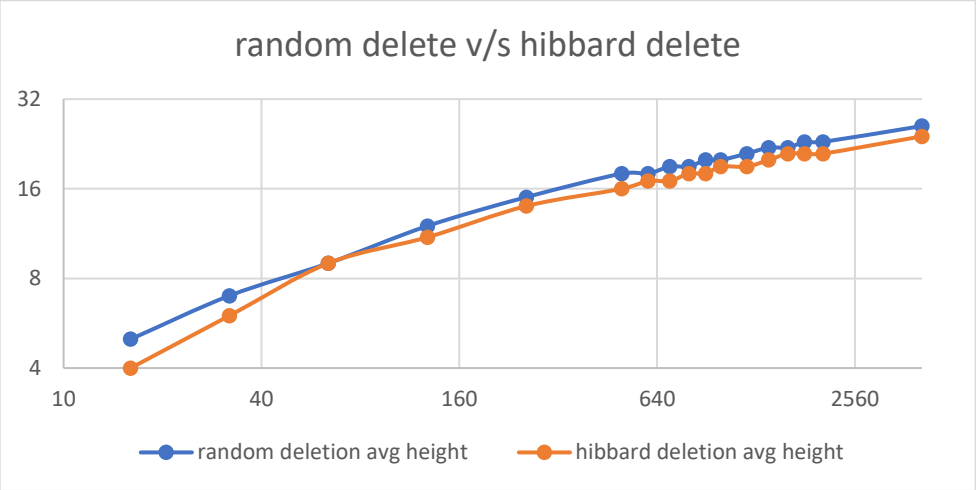
n	Avg Height	SQRT (n)	2Log(n)
16	4	4.0	8.0
32	6	5.7	10.0
64	9	8.0	12.0
128	11	11.3	14.0
256	14	16.0	16.0
500	16	22.4	17.9
600	17	24.5	18.5
700	17	26.5	18.9
800	18	28.3	19.3
900	18	30.0	19.6
1000	19	31.6	19.9
1200	19	34.6	20.5
1400	20	37.4	20.9
1600	21	40.0	21.3
1800	21	42.4	21.6
2048	21	45.3	22.0
4096	24	64.0	24.0
8192	27	90.5	26.0
16384	29	128.0	28.0


Hibbard deletion
45
40
35
30
25
20
15
10
5
0
10
40
160
640
2560
10240
Avg Height
SQRT (n)
2Log(n)

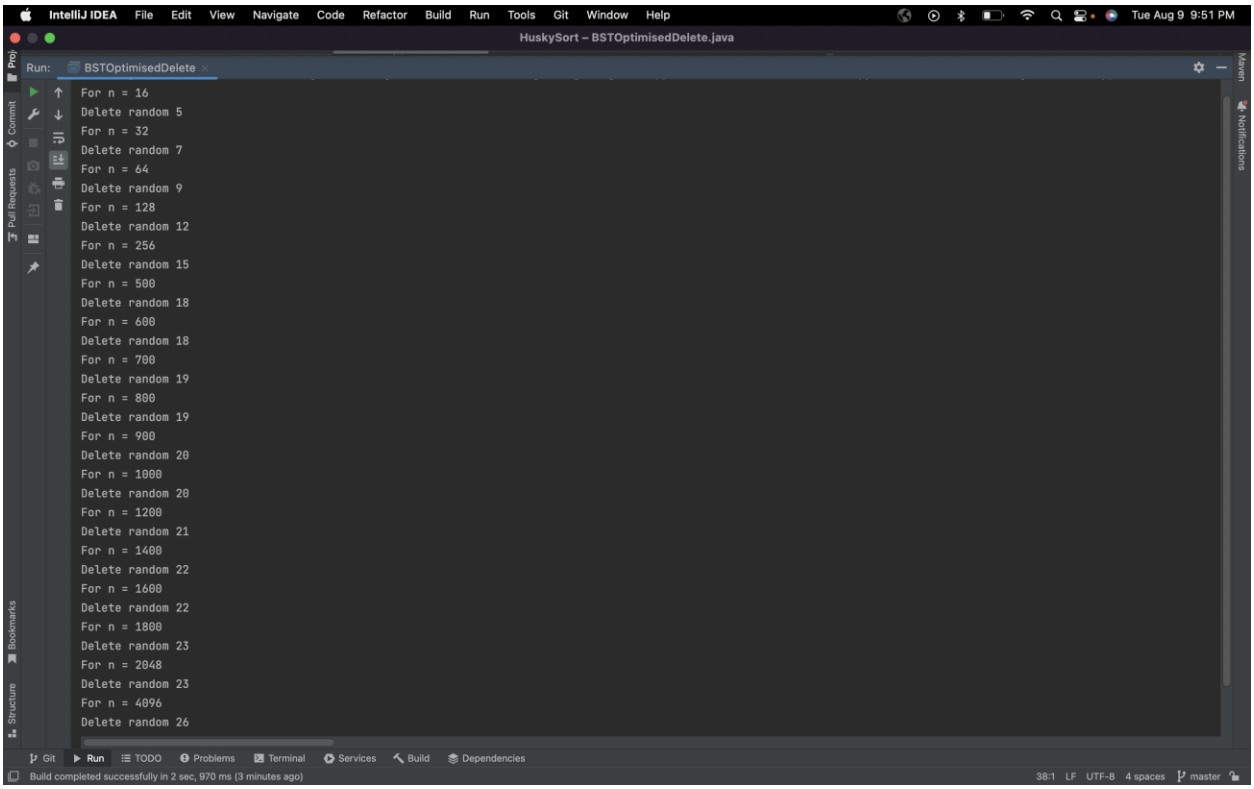
Code Output


IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Tue Aug 9 9:47 PM
HuskySort – BSTOptimisedDelete.java
Run: BSTOptimisedDelete
For n = 16
Hibbard deletion 4
For n = 32
Hibbard deletion 6
For n = 64
Hibbard deletion 9
For n = 128
Hibbard deletion 11
For n = 256
Hibbard deletion 14
For n = 500
Hibbard deletion 16
For n = 600
Hibbard deletion 17
For n = 700
Hibbard deletion 17
For n = 800
Hibbard deletion 18
For n = 900
Hibbard deletion 18
For n = 1000
Hibbard deletion 19
For n = 1200
Hibbard deletion 19
For n = 1400
Hibbard deletion 20
For n = 1600
Hibbard deletion 21
For n = 1800
Hibbard deletion 21
For n = 2048
Hibbard deletion 21
For n = 4096
Hibbard deletion 24
Git Run TODO Problems Terminal Services Build Dependencies
Build completed successfully in 3 sec, 445 ms (29 minutes ago)
37:20 LF UTF-8 4 spaces master

a) Randomly choose which direction to look for the node to be deleted

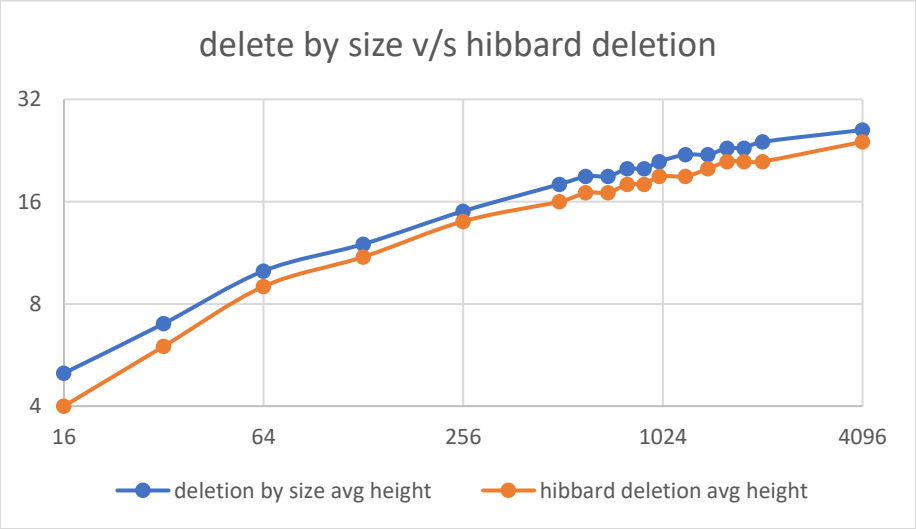


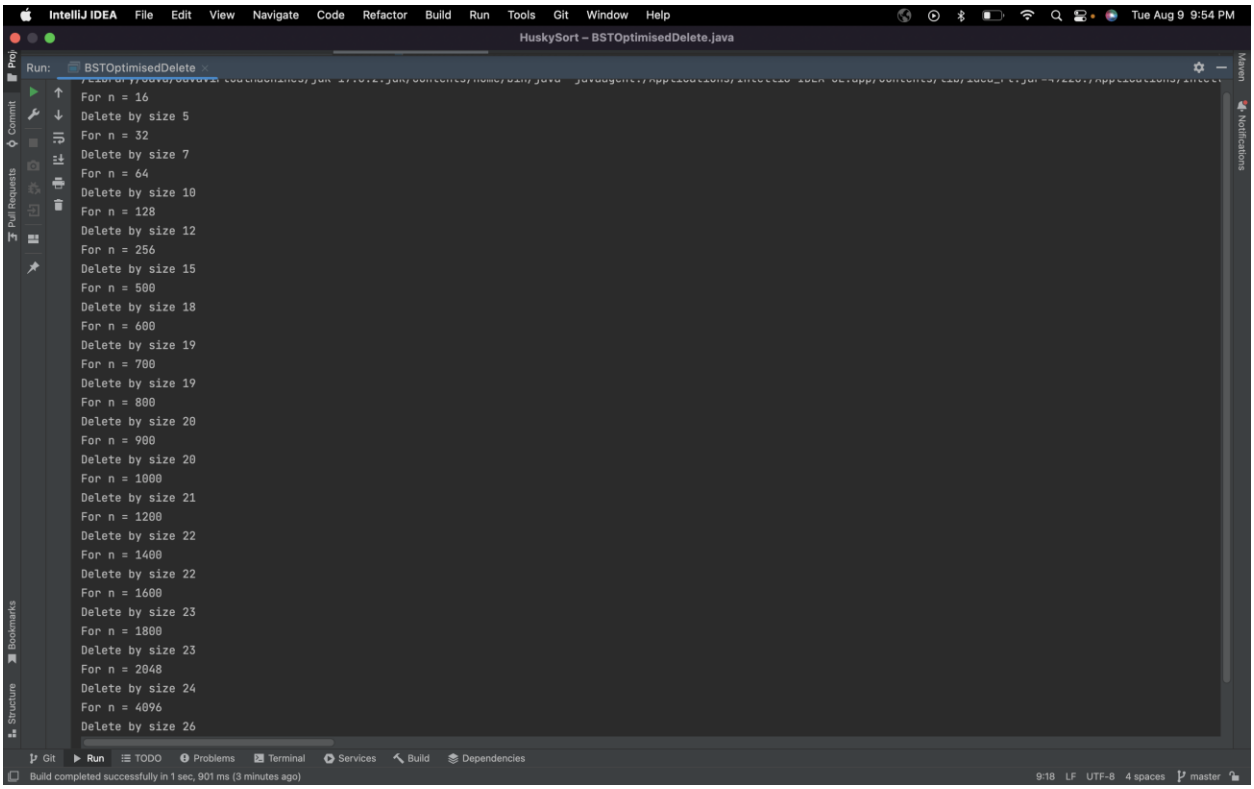
n	random deletion avg height	hibbard deletion avg height
16	5	4
32	7	6
64	9	9
128	12	11
256	15	14
500	18	16
600	18	17
700	19	17
800	19	18
900	20	18
1000	20	19
1200	21	19
1400	22	20
1600	22	21
1800	23	21
2048	23	21
4096	26	24


IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Tue Aug 9 9:51 PM
HuskySort – BSTOptimisedDelete.java
Run: BSTOptimisedDelete
For n = 16
Delete random 5
For n = 32
Delete random 7
For n = 64
Delete random 9
For n = 128
Delete random 12
For n = 256
Delete random 15
For n = 500
Delete random 18
For n = 600
Delete random 18
For n = 700
Delete random 19
For n = 800
Delete random 19
For n = 900
Delete random 20
For n = 1000
Delete random 20
For n = 1200
Delete random 21
For n = 1400
Delete random 22
For n = 1600
Delete random 22
For n = 1800
Delete random 23
For n = 2048
Delete random 23
For n = 4096
Delete random 26
Git Run TODO Problems Terminal Services Build Dependencies
Build completed successfully in 2 sec, 970 ms (3 minutes ago)
38:1 LF UTF-8 4 spaces master

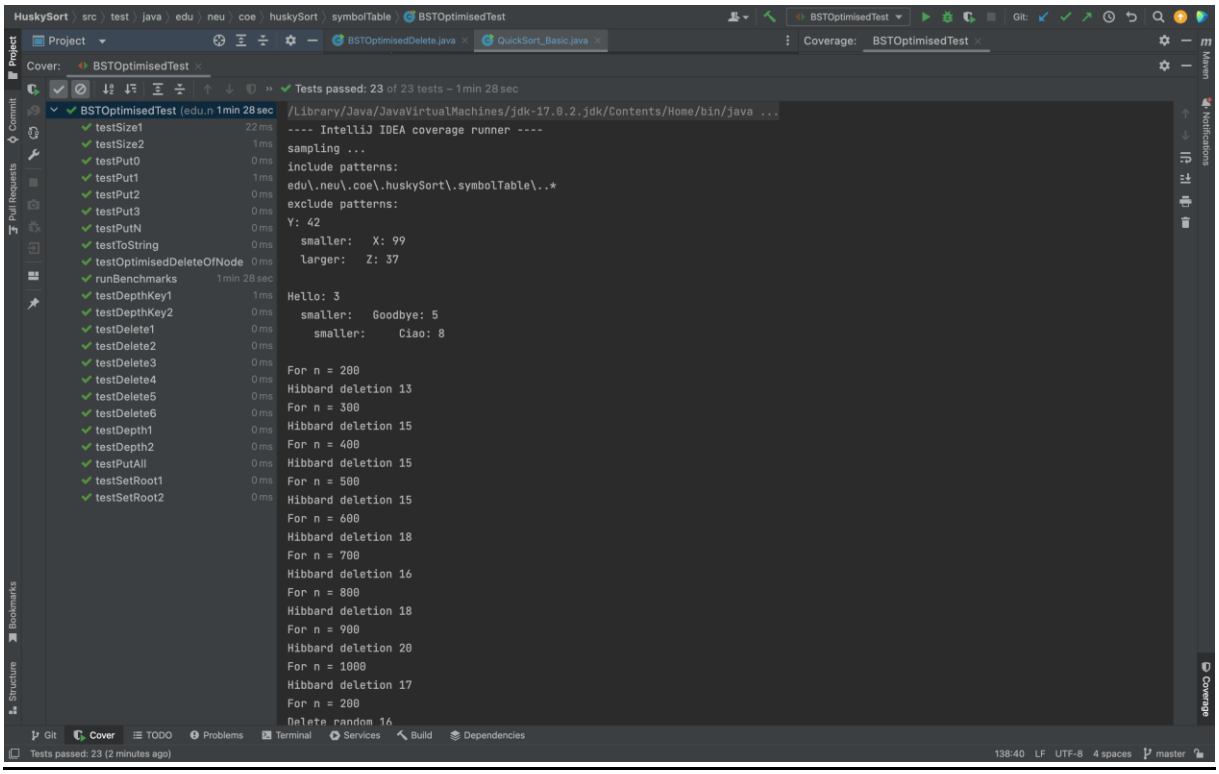
b) choose the direction according to the size of candidate nodes

n	deletion by size avg height	hibbard deletion avg height
16	5	4
32	7	6
64	10	9
128	12	11
256	15	14
500	18	16
600	19	17
700	19	17
800	20	18
900	20	18
1000	21	19
1200	22	19
1400	22	20
1600	23	21
1800	23	21
2048	24	21
4096	26	24

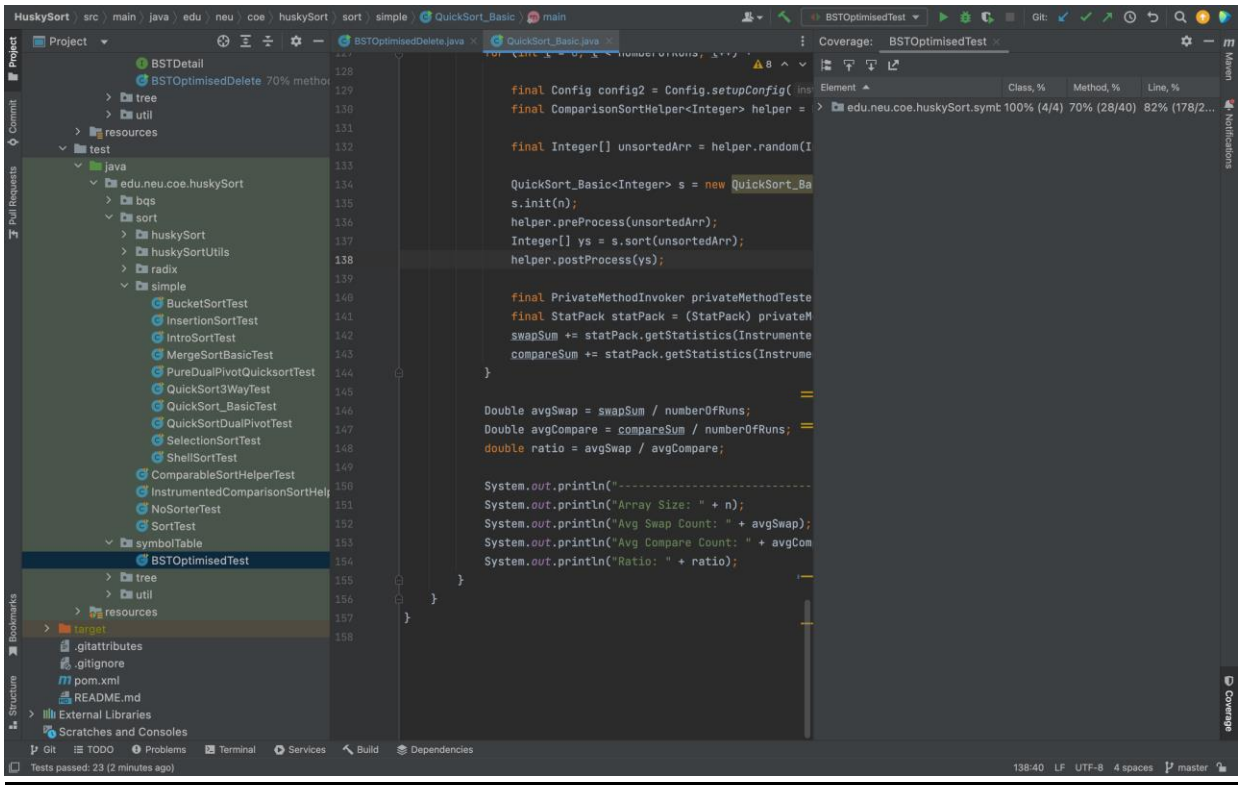



IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Tue Aug 9 9:54 PM
HuskySort – BSTOptimisedDelete.java
Run: BSTOptimisedDelete
For n = 16
Delete by size 5
For n = 32
Delete by size 7
For n = 64
Delete by size 10
For n = 128
Delete by size 12
For n = 256
Delete by size 15
For n = 500
Delete by size 18
For n = 600
Delete by size 19
For n = 700
Delete by size 19
For n = 800
Delete by size 20
For n = 900
Delete by size 20
For n = 1000
Delete by size 21
For n = 1200
Delete by size 22
For n = 1400
Delete by size 22
For n = 1600
Delete by size 23
For n = 1800
Delete by size 23
For n = 2048
Delete by size 24
For n = 4096
Delete by size 26
Git Run TODO Problems Terminal Services Build Dependencies
Build completed successfully in 1 sec, 901 ms (3 minutes ago)
9:18 LF UTF-8 4 spaces master

Unit Tests


HuskySort src test java edu neu coe huskySort symbolTable BSTOptimisedTest
Project
BSTOptimisedDelete.java
QuickSort_Basic.java
Coverage: BSTOptimisedTest
Cover: BSTOptimisedTest
Tests passed: 23 of 23 tests – 1 min 28 sec
BSTOptimisedTest (edu.n 1 min 28 sec
testSize1 22 ms
testSize2 1 ms
testPut0 0 ms
testPut1 1 ms
testPut2 0 ms
testPut3 0 ms
testPutN 0 ms
testToString 0 ms
testOptimisedDeleteOfNode 0 ms
runBenchmarks 1 min 28 sec
testDepthKey1 1 ms
testDepthKey2 0 ms
testDelete1 0 ms
testDelete2 0 ms
testDelete3 0 ms
testDelete4 0 ms
testDelete5 0 ms
testDelete6 0 ms
testDepth1 0 ms
testDepth2 0 ms
testPutAll 0 ms
testSetRoot1 0 ms
testSetRoot2 0 ms
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
edu\.neu\.coe\.huskySort\.symbolTable\..*
exclude patterns:
Y: 42
smaller: X: 99
larger: Z: 37
Hello: 3
smaller: Goodbye: 5
smaller: Ciao: 8
For n = 200
Hibbard deletion 13
For n = 300
Hibbard deletion 15
For n = 400
Hibbard deletion 15
For n = 500
Hibbard deletion 15
For n = 600
Hibbard deletion 18
For n = 700
Hibbard deletion 16
For n = 800
Hibbard deletion 18
For n = 900
Hibbard deletion 20
For n = 1000
Hibbard deletion 17
For n = 200
Delete random 16
Git Cover TODO Problems Terminal Services Build Dependencies
Tests passed: 23 (2 minutes ago)
138:40 LF UTF-8 4 spaces master

Code Coverage



Conclusion

1. The average height of BST after several (almost half of array size) Hibbard deletion is near to $\sqrt{n}$ when the value of the array size is less than 256. As the value get larger the average tends to get closer to 2log(n). We conclude the cutoff for the relation to be 256.
2. When compared to selecting a node to be eliminated at random and Hibbard deletion, Hibbard deletion performs marginally better.
3. Similarly, Hibbard deletion performs better than choosing the node to be deleted based on the size.
4. It can be concluded that the Hibbard deletion is the most efficient deletion method for BST from the benchmarks above.

QUICKSORT

Code Links

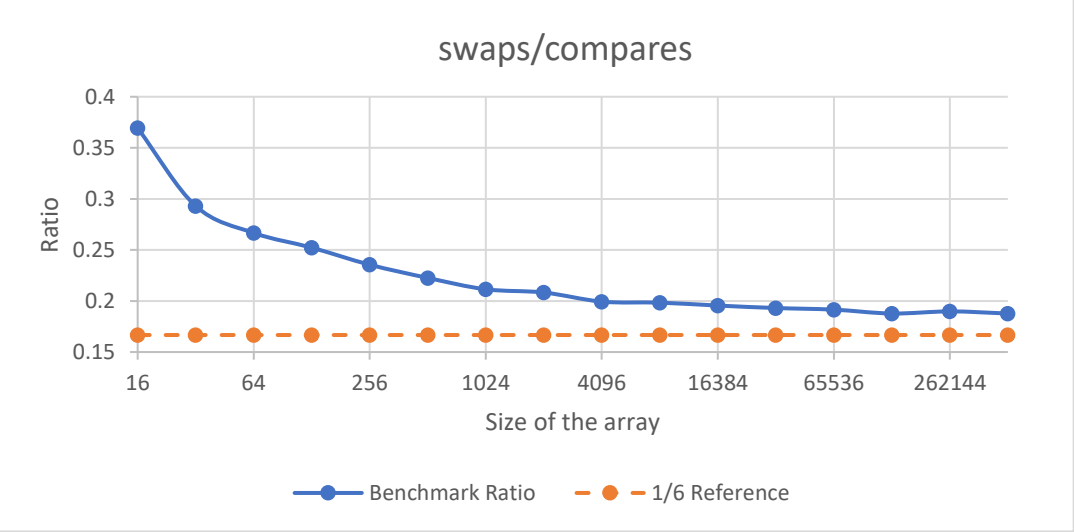
QuickSort_Basic.java
QuickSort_BasicTest.java

Analysis

** Note: Due to hardware limitations as the array size increases the avg runs need to be lower*

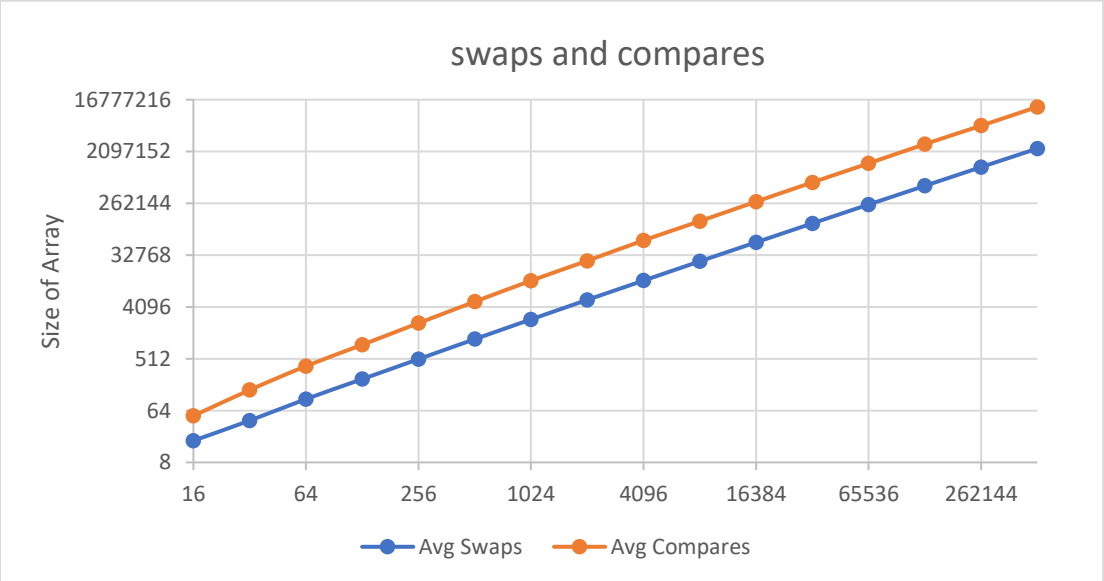
Swaps/Compare Ratio

No of Runs	n	Benchmark Ratio	1/6 Reference
1000	16	0.36929108	0.166666667
1000	32	0.293010991	0.166666667
1000	64	0.266663868	0.166666667
1000	128	0.252224209	0.166666667
1000	256	0.235538471	0.166666667
1000	512	0.222596969	0.166666667
1000	1024	0.211444878	0.166666667
1000	2048	0.208324657	0.166666667
1000	4096	0.199367588	0.166666667
100	8192	0.198375098	0.166666667
100	16384	0.195519294	0.166666667
100	32768	0.193108923	0.166666667
3	65536	0.191516039	0.166666667
3	131072	0.187745357	0.166666667
3	262144	0.189798017	0.166666667
3	524288	0.187732235	0.166666667

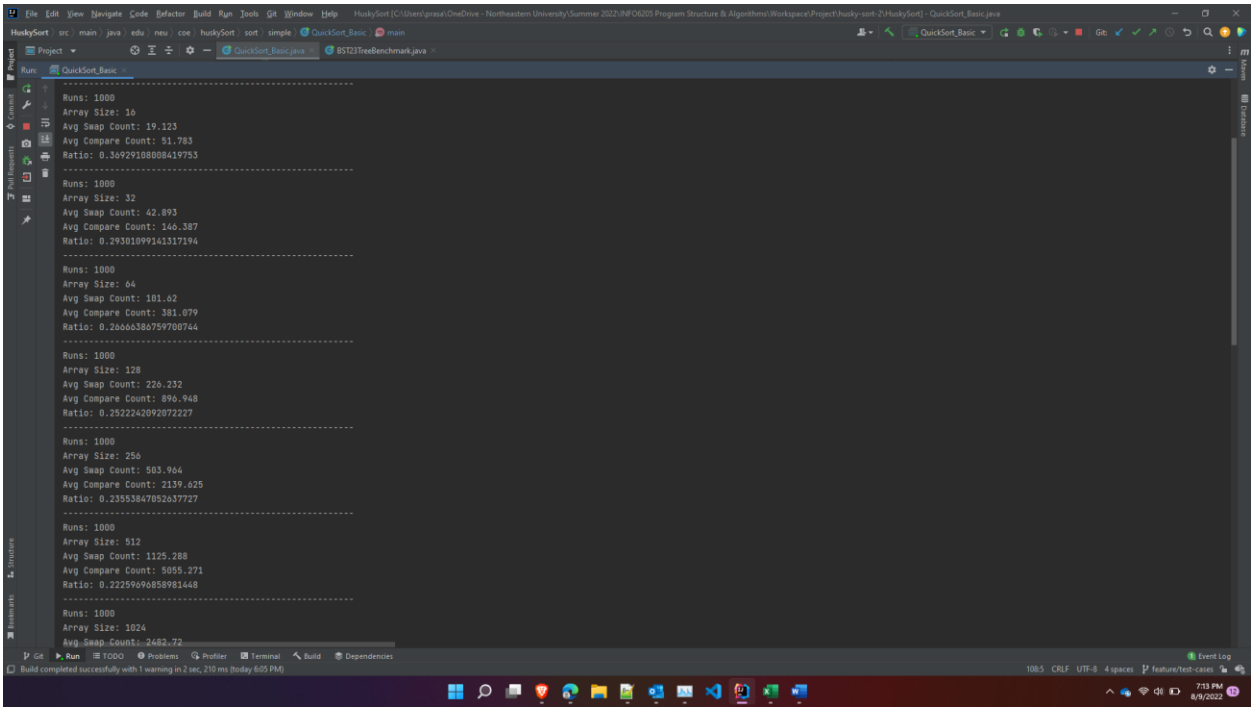


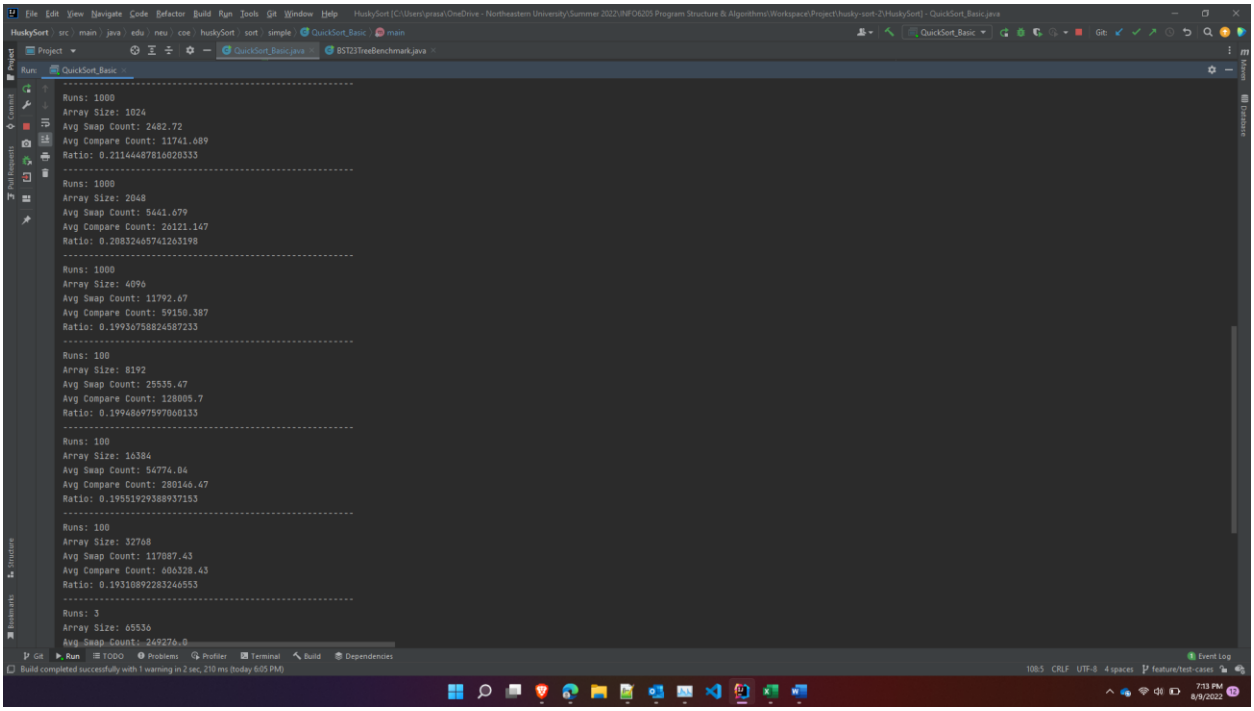
Swaps and Compares

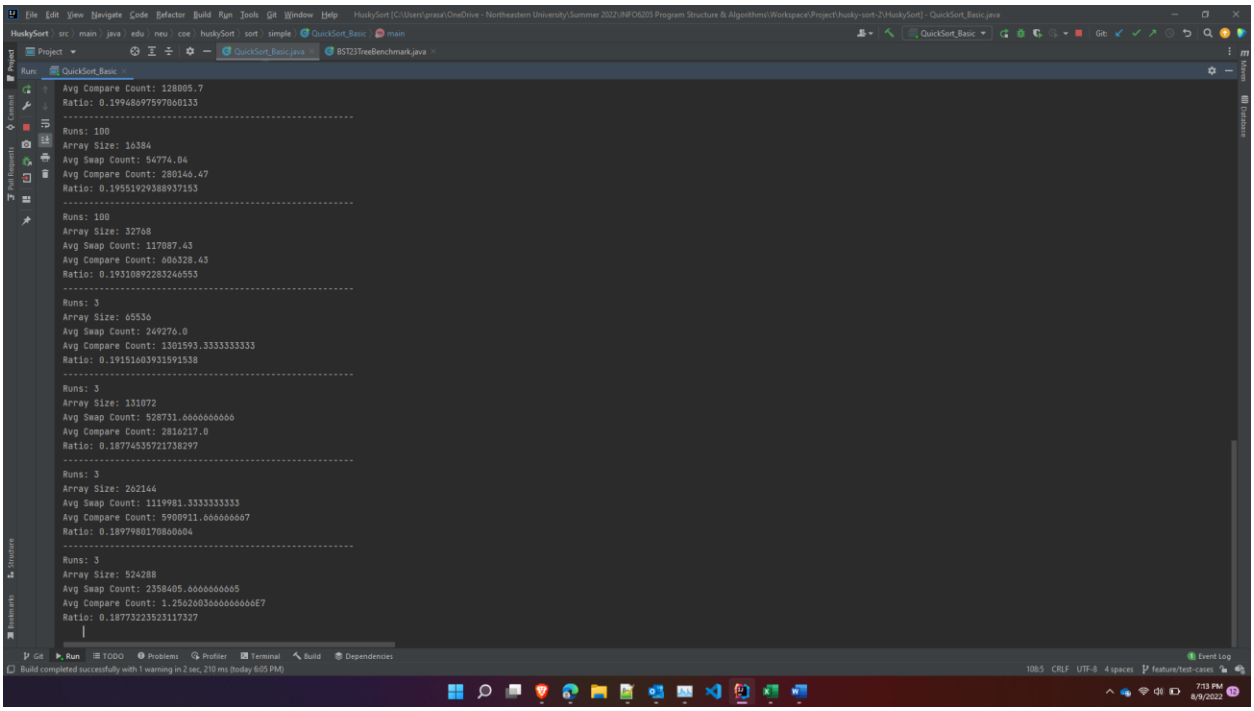
No of Runs	n	Avg Swaps	Avg Compares
1000	16	19.123	51.783
1000	32	42.893	146.387
1000	64	101.62	381.079
1000	128	226.232	896.948
1000	256	503.964	2139.625
1000	512	1125.288	5055.271
1000	1024	2482.72	11741.689
1000	2048	5441.679	26121.147
1000	4096	11792.67	59150.387
100	8192	25535.47	128005.7
100	16384	54774.04	280146.47
100	32768	117087.43	606328.43
3	65536	249276	1301593.333
3	131072	528731.667	2816217
3	262144	1119981.33	5900911.667
3	524288	2358405.67	12562603.67



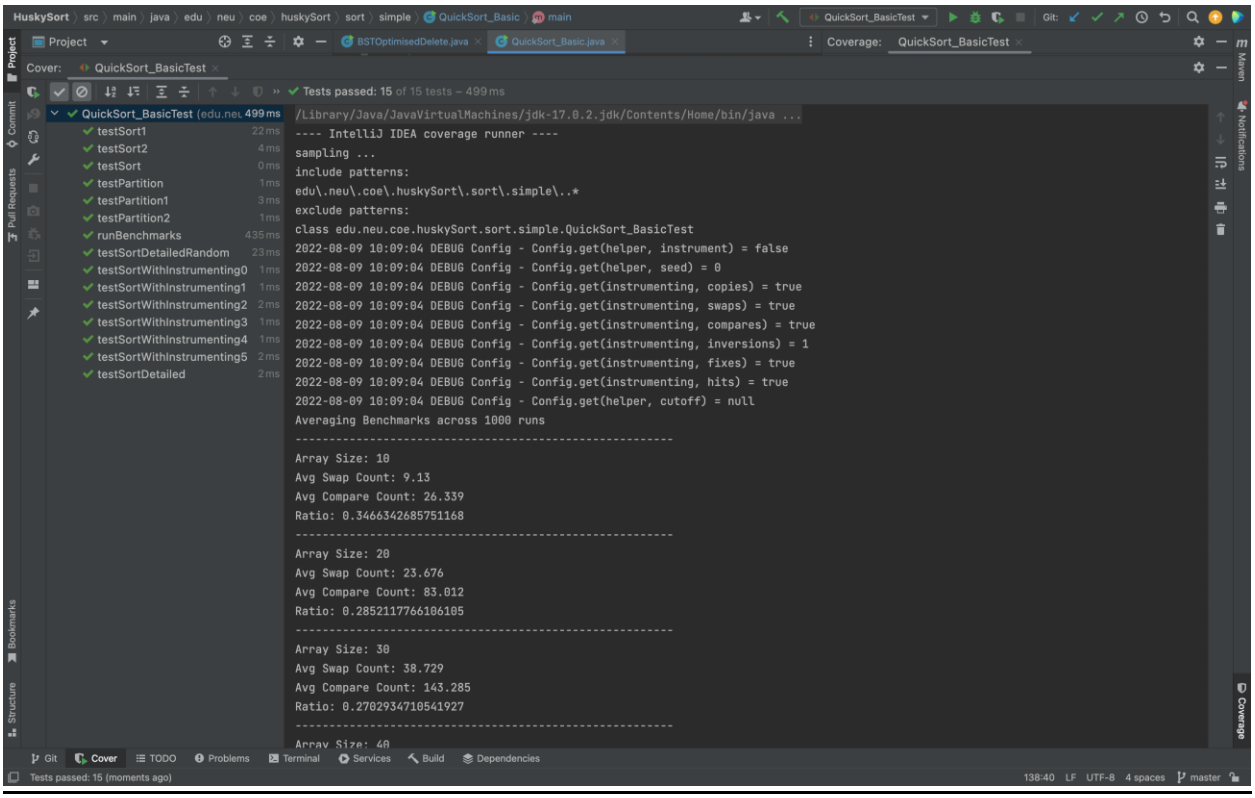
Output


Runs: 1000
Array Size: 16
Avg Swap Count: 19.123
Avg Compare Count: 51.783
Ratio: 0.3692918808419753
Runs: 1000
Array Size: 32
Avg Swap Count: 42.893
Avg Compare Count: 146.387
Ratio: 0.2930109914317194
Runs: 1000
Array Size: 64
Avg Swap Count: 101.62
Avg Compare Count: 381.079
Ratio: 0.26666386759700744
Runs: 1000
Array Size: 128
Avg Swap Count: 226.232
Avg Compare Count: 896.948
Ratio: 0.2522242092072227
Runs: 1000
Array Size: 256
Avg Swap Count: 503.964
Avg Compare Count: 2139.625
Ratio: 0.2355384705263727
Runs: 1000
Array Size: 512
Avg Swap Count: 1125.288
Avg Compare Count: 5055.271
Ratio: 0.22259696858981448
Runs: 1000
Array Size: 1024
Avg Swap Count: 2482.72

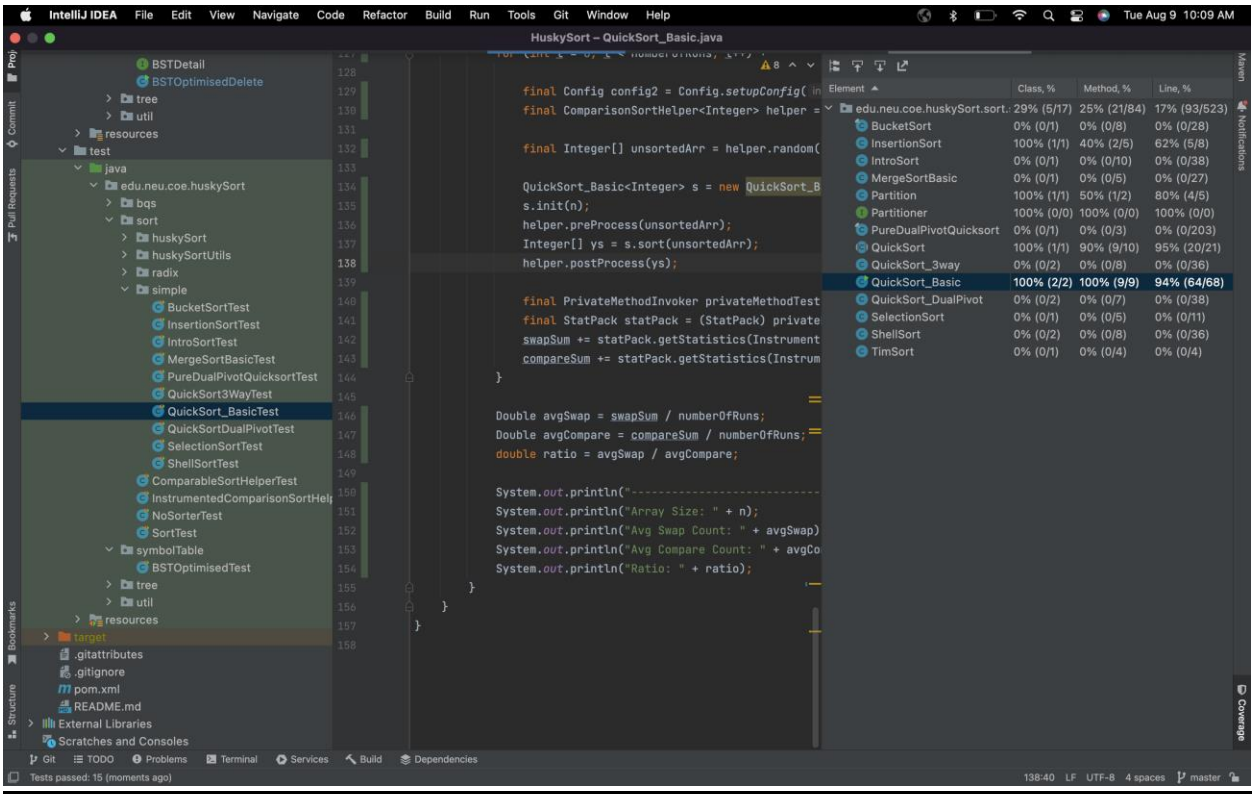

Runs: 1000
Array Size: 1024
Avg Swap Count: 2482.72
Avg Compare Count: 11741.689
Ratio: 0.2114487816020333
Runs: 1000
Array Size: 2048
Avg Swap Count: 5441.679
Avg Compare Count: 26121.147
Ratio: 0.20832465741263198
Runs: 1000
Array Size: 4096
Avg Swap Count: 11792.67
Avg Compare Count: 59150.387
Ratio: 0.19936758824587233
Runs: 100
Array Size: 8192
Avg Swap Count: 25535.47
Avg Compare Count: 128005.7
Ratio: 0.19948697597060133
Runs: 100
Array Size: 16384
Avg Swap Count: 54774.04
Avg Compare Count: 280146.47
Ratio: 0.19551929388937153
Runs: 100
Array Size: 32768
Avg Swap Count: 117087.43
Avg Compare Count: 606328.43
Ratio: 0.19310892283246553
Runs: 3
Array Size: 65536
Avg Swap Count: 249276.0


Avg Compare Count: 128005.7
Ratio: 0.19948697597060133
Runs: 100
Array Size: 16384
Avg Swap Count: 54774.04
Avg Compare Count: 280146.47
Ratio: 0.19551929388937153
Runs: 100
Array Size: 32768
Avg Swap Count: 117087.43
Avg Compare Count: 606328.43
Ratio: 0.19310892283246553
Runs: 3
Array Size: 65536
Avg Swap Count: 249276.0
Avg Compare Count: 1301593.3333333333
Ratio: 0.19151603931591538
Runs: 3
Array Size: 131072
Avg Swap Count: 528731.6666666666
Avg Compare Count: 2816217.0
Ratio: 0.18774535721738297
Runs: 3
Array Size: 262144
Avg Swap Count: 1119981.3333333333
Avg Compare Count: 5900911.666666667
Ratio: 0.1897980170860604
Runs: 3
Array Size: 524288
Avg Swap Count: 2358405.6666666665
Avg Compare Count: 1.2562603666666666E7
Ratio: 0.18773223523117327

Unit Test


Tests passed: 15 of 15 tests – 499 ms
QuickSort_BasicTest (edu.neu 499 ms
testSort1 22 ms
testSort2 4 ms
testSort 0 ms
testPartition 1 ms
testPartition1 3 ms
testPartition2 1 ms
runBenchmarks 435 ms
testSortDetailedRandom 23 ms
testSortWithInstrumenting0 1 ms
testSortWithInstrumenting1 1 ms
testSortWithInstrumenting2 2 ms
testSortWithInstrumenting3 1 ms
testSortWithInstrumenting4 1 ms
testSortWithInstrumenting5 2 ms
testSortDetailed 2 ms
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
edu\.neu\.coe\.huskySort\.sort\.simple\..*
exclude patterns:
class edu.neu.coe.huskySort.sort.simple.QuickSort_BasicTest
2022-08-09 10:09:04 DEBUG Config - Config.get(helper, instrument) = false
2022-08-09 10:09:04 DEBUG Config - Config.get(helper, seed) = 0
2022-08-09 10:09:04 DEBUG Config - Config.get(instrumenting, copies) = true
2022-08-09 10:09:04 DEBUG Config - Config.get(instrumenting, swaps) = true
2022-08-09 10:09:04 DEBUG Config - Config.get(instrumenting, compares) = true
2022-08-09 10:09:04 DEBUG Config - Config.get(instrumenting, inversions) = 1
2022-08-09 10:09:04 DEBUG Config - Config.get(instrumenting, fixes) = true
2022-08-09 10:09:04 DEBUG Config - Config.get(instrumenting, hits) = true
2022-08-09 10:09:04 DEBUG Config - Config.get(helper, cutoff) = null
Averaging Benchmarks across 1000 runs
Array Size: 10
Avg Swap Count: 9.13
Avg Compare Count: 26.339
Ratio: 0.3466342685751168
Array Size: 20
Avg Swap Count: 23.676
Avg Compare Count: 83.012
Ratio: 0.2852117766106185
Array Size: 30
Avg Swap Count: 38.729
Avg Compare Count: 143.285
Ratio: 0.2702934710541927
Array Size: 40

Coverage



Conclusion

1. The ratio of swaps to compare is near to 1/6 (0.1666) but not 1/6
2. As the size of the array increases the ratio tends towards 1/6
3. One reasonable explanation to this could be that as the size of array increases, there are more compares relative to swaps.

2-3 TREE

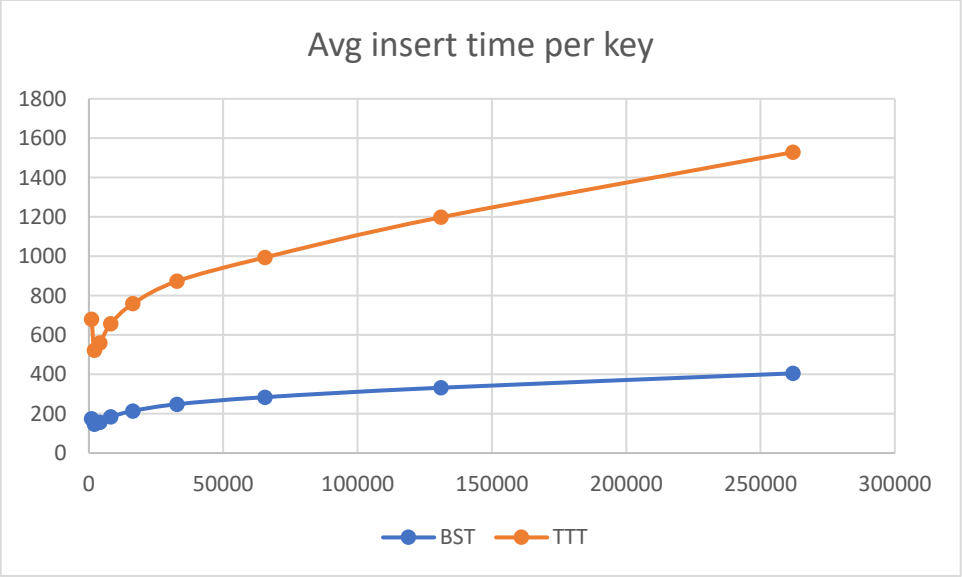
Code Links

[TwoThreeTree.java]
[BST23TreeBenchmark.java]
[TwoThreeTreeTest.java]

Analysis

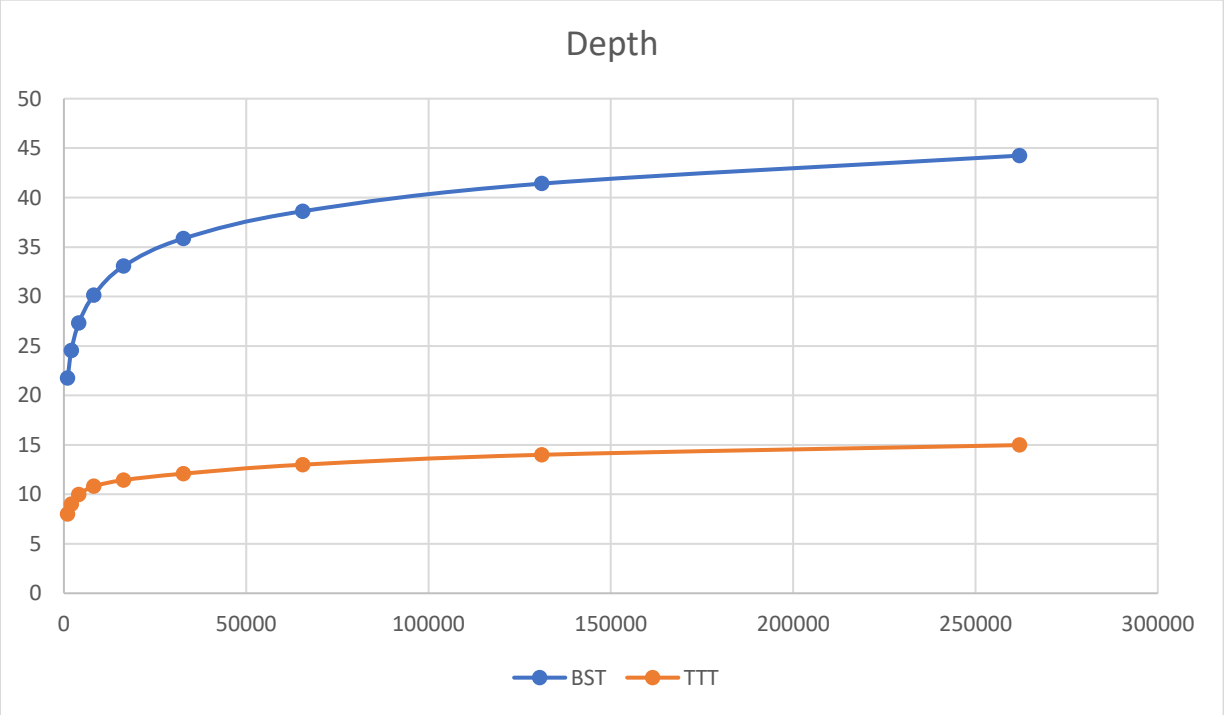
Avg Time To Insert Per Key (Ns)

n	BST	TTT
1024	174	680
2048	146	522
4096	156	560
8192	184	658
16384	214	759
32768	248	873
65536	284	994
131072	332	1198
262144	405	1529



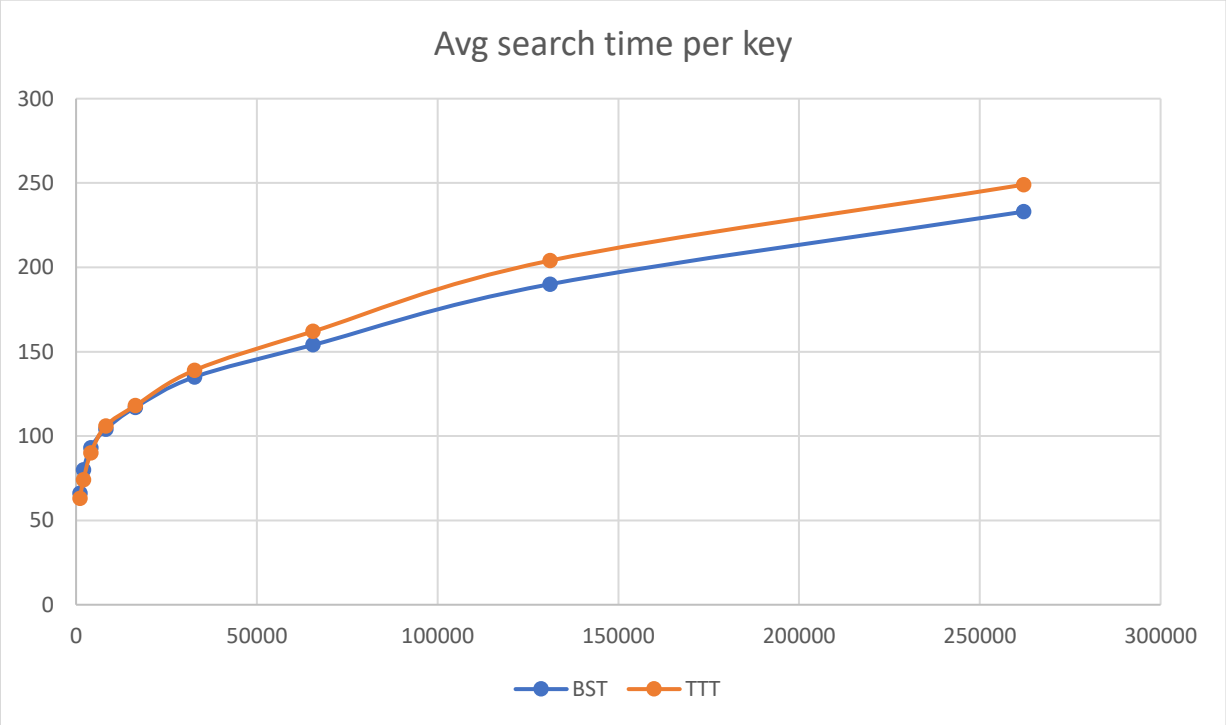
Avg Depth of the Tree

n	BST	TTT
1024	21.75	8.01
2048	24.55	9
4096	27.32	9.97
8192	30.13	10.81
16384	33.07	11.44
32768	35.86	12.09
65536	38.62	13
131072	41.41	14
262144	44.24	14.98

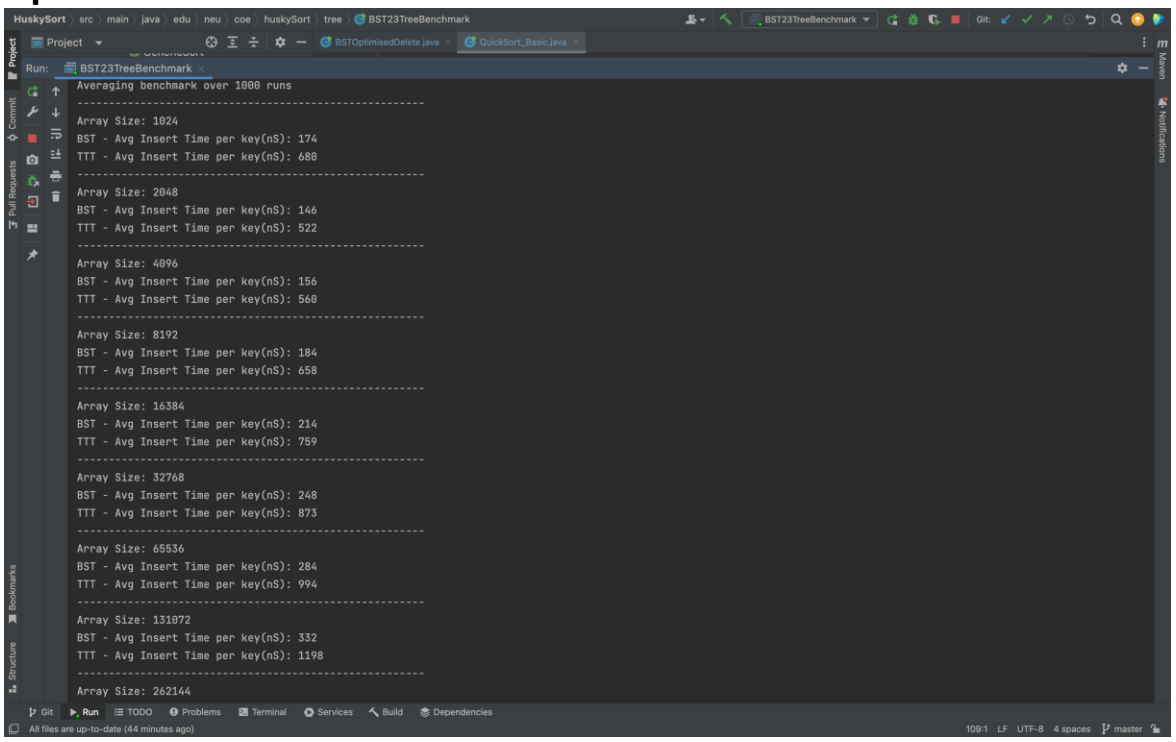


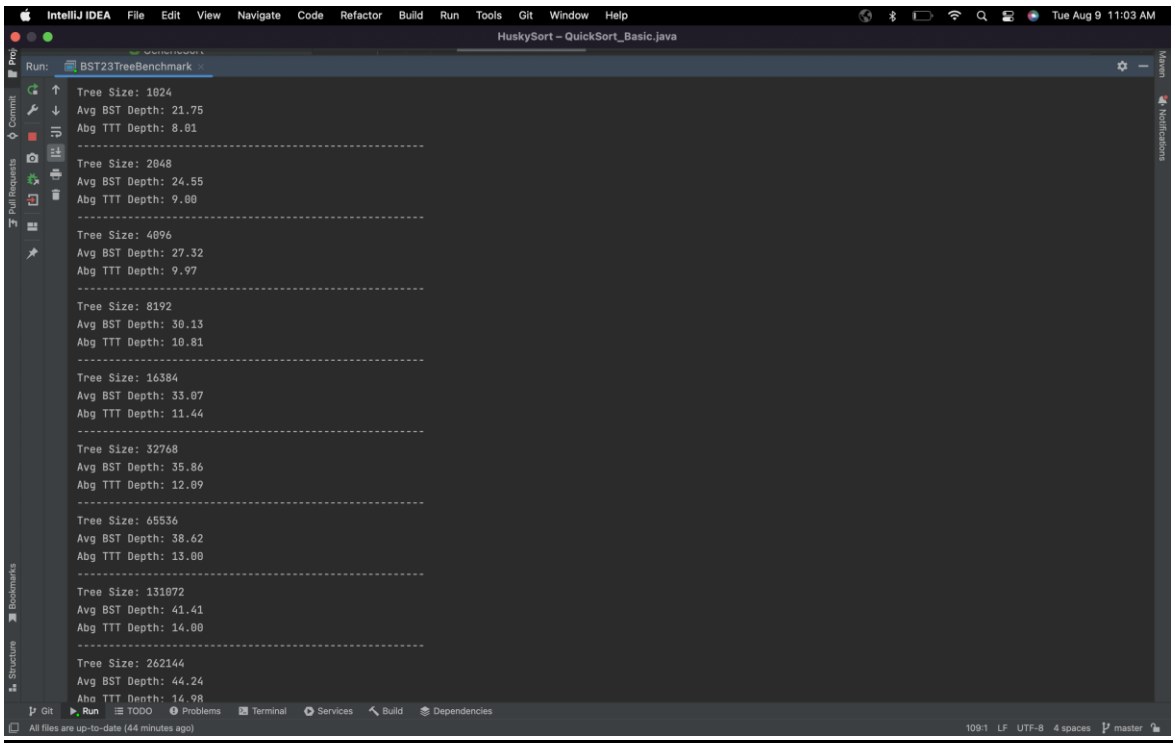
Avg Time To Search Per Key (nS)

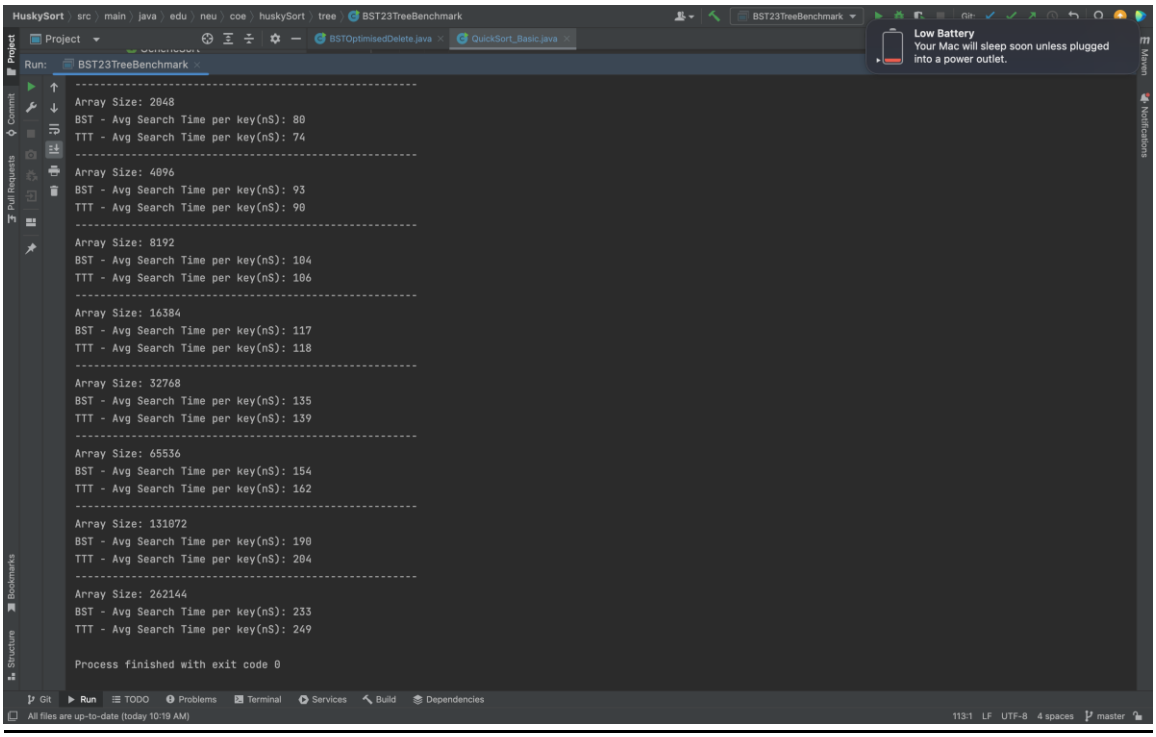
n	BST	TTT
1024	66	63
2048	80	74
4096	93	90
8192	104	106
16384	117	118
32768	135	139
65536	154	162
131072	190	204
262144	233	249



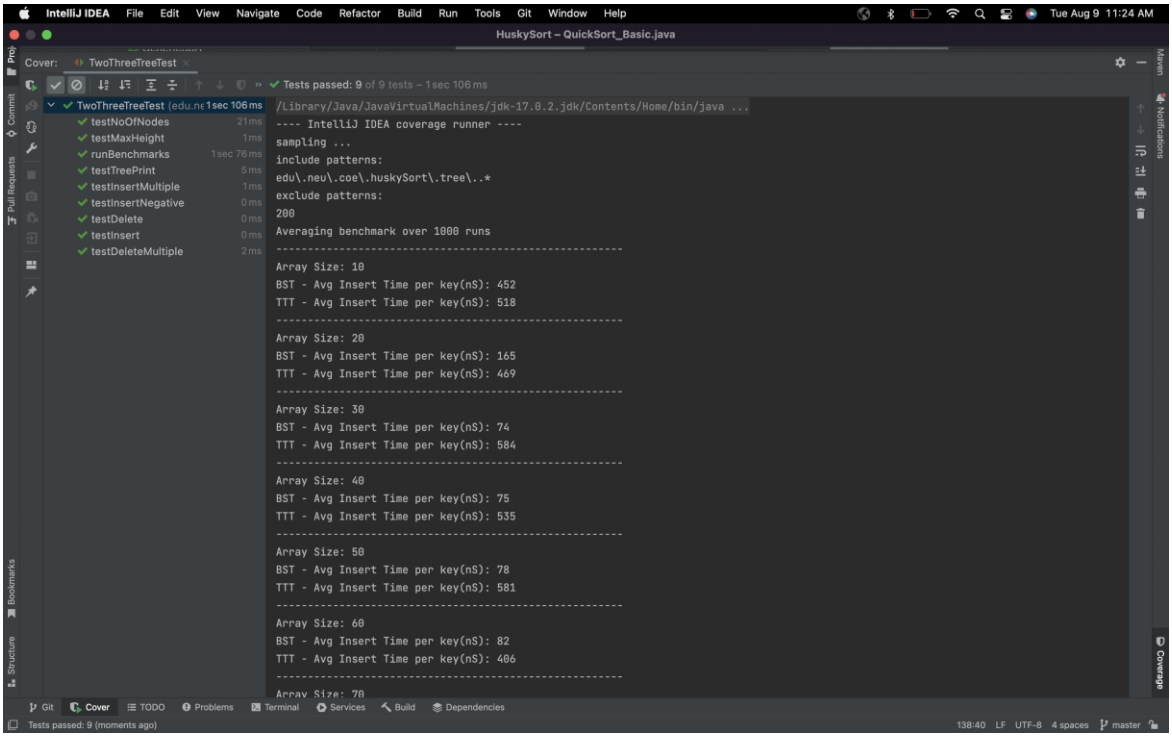
Output


Run: BST23TreeBenchmark
Averaging benchmark over 1000 runs
Array Size: 1024
BST - Avg Insert Time per key(nS): 174
TTT - Avg Insert Time per key(nS): 680
Array Size: 2048
BST - Avg Insert Time per key(nS): 146
TTT - Avg Insert Time per key(nS): 522
Array Size: 4096
BST - Avg Insert Time per key(nS): 156
TTT - Avg Insert Time per key(nS): 560
Array Size: 8192
BST - Avg Insert Time per key(nS): 184
TTT - Avg Insert Time per key(nS): 658
Array Size: 16384
BST - Avg Insert Time per key(nS): 214
TTT - Avg Insert Time per key(nS): 759
Array Size: 32768
BST - Avg Insert Time per key(nS): 248
TTT - Avg Insert Time per key(nS): 873
Array Size: 65536
BST - Avg Insert Time per key(nS): 284
TTT - Avg Insert Time per key(nS): 994
Array Size: 131072
BST - Avg Insert Time per key(nS): 332
TTT - Avg Insert Time per key(nS): 1198
Array Size: 262144

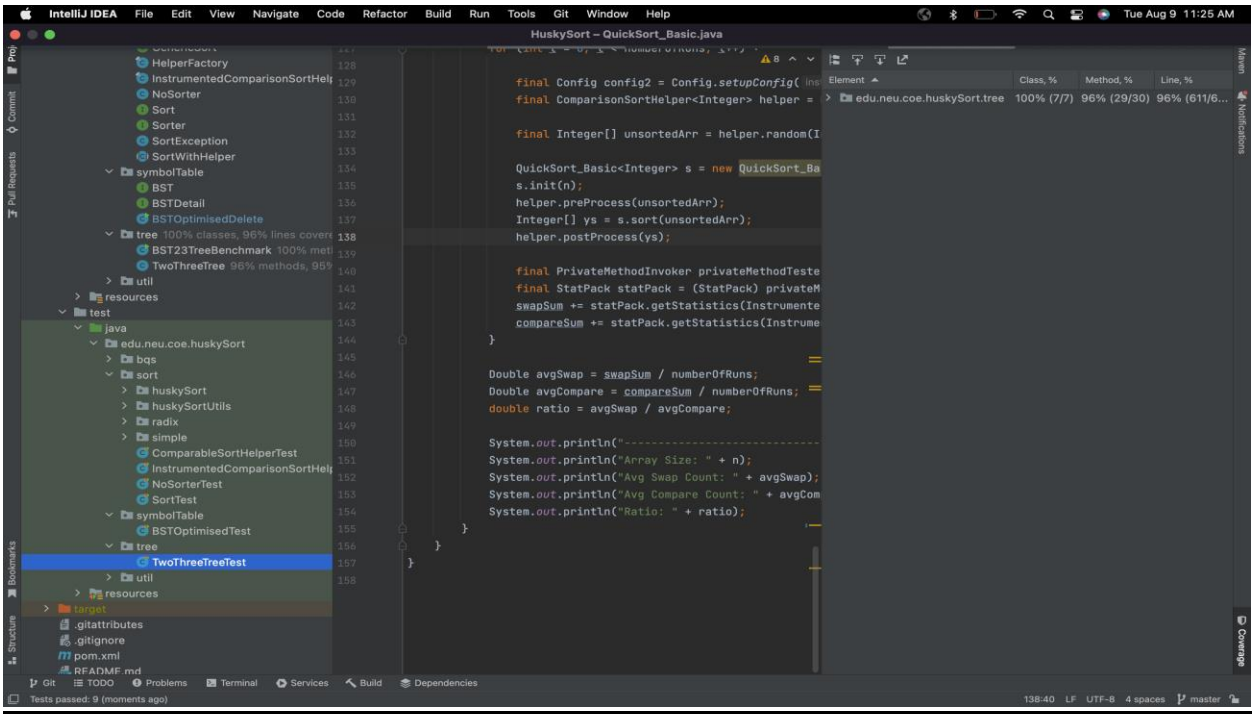

HuskySort – QuickSort_Basic.java
Run: BST23TreeBenchmark
Tree Size: 1024
Avg BST Depth: 21.75
Abg TTT Depth: 8.01
Tree Size: 2048
Avg BST Depth: 24.55
Abg TTT Depth: 9.00
Tree Size: 4096
Avg BST Depth: 27.32
Abg TTT Depth: 9.97
Tree Size: 8192
Avg BST Depth: 30.13
Abg TTT Depth: 10.81
Tree Size: 16384
Avg BST Depth: 33.07
Abg TTT Depth: 11.44
Tree Size: 32768
Avg BST Depth: 35.86
Abg TTT Depth: 12.09
Tree Size: 65536
Avg BST Depth: 38.62
Abg TTT Depth: 13.00
Tree Size: 131072
Avg BST Depth: 41.41
Abg TTT Depth: 14.00
Tree Size: 262144
Avg BST Depth: 44.24


Low Battery
Your Mac will sleep soon unless plugged into a power outlet.
Run: BST23TreeBenchmark
Array Size: 2048
BST - Avg Search Time per key(nS): 80
TTT - Avg Search Time per key(nS): 74
Array Size: 4096
BST - Avg Search Time per key(nS): 93
TTT - Avg Search Time per key(nS): 90
Array Size: 8192
BST - Avg Search Time per key(nS): 104
TTT - Avg Search Time per key(nS): 106
Array Size: 16384
BST - Avg Search Time per key(nS): 117
TTT - Avg Search Time per key(nS): 118
Array Size: 32768
BST - Avg Search Time per key(nS): 135
TTT - Avg Search Time per key(nS): 139
Array Size: 65536
BST - Avg Search Time per key(nS): 154
TTT - Avg Search Time per key(nS): 162
Array Size: 131072
BST - Avg Search Time per key(nS): 190
TTT - Avg Search Time per key(nS): 204
Array Size: 262144
BST - Avg Search Time per key(nS): 233
TTT - Avg Search Time per key(nS): 249
Process finished with exit code 0

Unit Test


HuskySort – QuickSort_Basic.java
Tests passed: 9 of 9 tests – 1 sec 106 ms
TwoThreeTreeTest
testNoOfNodes
testMaxHeight
runBenchmarks
testTreePrint
testInsertMultiple
testInsertNegative
testDelete
testInsert
testDeleteMultiple
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
edu\.neu\.coe\.huskySort\.tree\..*
exclude patterns:
200
Averaging benchmark over 1000 runs
Array Size: 10
BST - Avg Insert Time per key(nS): 452
TTT - Avg Insert Time per key(nS): 518
Array Size: 20
BST - Avg Insert Time per key(nS): 165
TTT - Avg Insert Time per key(nS): 469
Array Size: 30
BST - Avg Insert Time per key(nS): 74
TTT - Avg Insert Time per key(nS): 584
Array Size: 40
BST - Avg Insert Time per key(nS): 75
TTT - Avg Insert Time per key(nS): 535
Array Size: 50
BST - Avg Insert Time per key(nS): 78
TTT - Avg Insert Time per key(nS): 581
Array Size: 60
BST - Avg Insert Time per key(nS): 82
TTT - Avg Insert Time per key(nS): 486

Coverage



Conclusion

1. The insert operation for 23Tree takes longer due to additional merge operations
2. The 23Tree is much shorter when compared to BST
3. Even though the tree is shorter, 23Tree and BST have similar metrics for search. This could be due to additional compares required in the 23Tree