Concurrent Programming

Final Project Report

Author: Sarayu Managoli

Program Overview:

The program is intended to implement parallelized binary search tree. The BST consists of applications that include insert(put), search(get), and deletion of nodes. These functions are parallelized across multiple threads. The program also includes a function which prints the values present in a range. These functions have been demonstrated as a part of the main function as well for better representation and understanding. The user is expected to provide the number of nodes and the number of threads along with the locking mechanism they would like to use, hand-over-hand or reader-writer locks. Memory leaks have been checked and rectified in the implementation.

The implementation also contains a unit test which exhibits the basic functionalities of all the functions on the trees.

Brief Introduction of Binary Search Tree

A binary search tree is an existent data structure that is used to sort a set of values. It helps in maintaining a sorted list of numbers. Each node in the tree has two children, left and right. It searches for the presence of a number in 0(log(n)) time. The fundamental property of the tree remains such that the nodes on the left of the root node are lesser in value than the root itself. All the nodes on the right are of a greater value. This minimizes the effort of sorting and searching elements of the tree.

For the search operation, if the value is below the root, we can say for sure that the value is not in the right subtree; we need to only search in the left subtree and if the value is above the root, we can say for sure that the value is not in the left subtree; we need to only search in the right subtree. Insertion and deletion also follow a similar suit.

The struct below defines a node:

```
typedef struct bstree{

int key;

int value;

struct bstree *left;

struct bstree *right;

pthread_mutex_t lock;

pthread_rwlock_t rwlock;

}bst_node;
```

Deliverables

As a part of the deliverables, the following artefacts are present in the ZIP folder:

1. Lab write-up
2. Source Files (in folder Code)
3. Makefile (in folder Code)

Source File - Code Organization

This file contains the source code for the implementation of Final project. The following functions are included as a part of the source file:

In main.cpp,

1. **void mutex_initialization(void):** The function is used to initialize all locks, barriers and mutexes needed for the multithreading operation.
2. **void assign_value(int* key, int* value):** The function is used to generate various random key and values required for put, get and delete operation.
3. **void *low_contention(void *arg):** The thread handler function is used to demonstrate low contention between the threads generating various key and values required for put, get and delete operation. Here, each thread randomly searches for values using the get function.
4. **void *high_contention(void *arg):** The thread handler function is used to demonstrate high contention between the threads generating various key and values required for put, get and delete operation. Here, one value is put onto the BST and that value is required to be searched by all the threads. This creates a high contention for that one particular value thus demonstrating the phenomenon.
5. **void *main_thread_handler(void* arg):** The function is used to demonstrate all the basic functionalities on the BST. Here, depending upon the thread ID, the function either gets, puts or deleted values from the BST.
6. **int main(int argc, char **argv):** Initializes the variables and threads and calls other functions.
7. **void free_tree(bstree *root):** Frees the entire tree that is malloc'd including the child nodes.
8. **void close(void):** This function frees the required memory and destroys locks and barriers.

In bst.cpp,

9. **void print_tree(bstree *root):** The function is used to print all the nodes with their respective keys. It is done inorder.
10. **bstree *new_node(int key, int value):** The function is used to create a new node if there is a requirement. It provides a null to the left and the right child nodes upon initialization.
11. **bstree* minValueNode(bstree* node):** The function is used to find the node with the minimal value that is that is useful while deleting the nodes recursively. It is called in the delete function.
12. **bstree* deleteNode(bstree* func_node, int key):** The function is used to delete the nodes recursively based on the input node and key. If the node is found, the function returns a NULL.
13. **bstree *put(bstree *func_node, int key, int value):** The function is used to add the requested node onto the BST. It too is done recursively based on the value of the other keys and the nodes.
14. **bstree *get(bstree *func_node, int key):** The function is used to search for the requested node onto the BST. It too is done recursively based on the value of the other keys and the nodes.
15. **void bst_range_query(bstree* node, int start_key, int end_key):** The function is used to print the values between the start and the end keys in the BST.

In unit_test.cpp,

16. **void test_range():** Tests the bst_range_query function.
17. **bool test_delete():** Tests the deleteNode function.
18. **void BST_Array(bstree *root, int* arr):** Converts the BST into a 1-D array, for better interpretation.
19. **bool test_put():** Tests the put function of the BST.
20. **bool test_get():** Tests the get function of the BST.
21. **int main():** Calls all the test functions.

Makefile

Makefile consists of two targets, all and clean. The steps to execute are also printed in the file.

Compilation steps:

Open command prompt in the folder SarayuManagoli_FinalProject/Code, type "**make**". This will create the executable "**binarysearchtree**". To delete the executable, type "**make clean**". To compile the unit tests, type "**make test**".

Execution steps:

To print the author's name, type "**./binarysearchtree --name**"

To run the program in normal mode, type **"./binarysearchtree [--name] [--node=NUM_NODES] [-t NUM_THREADS] [--lock=HOH,RW] [--test=low,high]**"

To run the program in test mode, type "**./unittest**"

Error handling and extant bug:

Error handling has been done if some of user inputs are missing. The initial number of threads and nodes are set to 10 each. If low and high contention is not selected, the normal mode of operation is selected. The program also exits if the number of threads is greater than the number of nodes.

The program also has been checked for memory leaks and there appear to be none. The way the tree was free'd recursively helped with the memory leak issues.

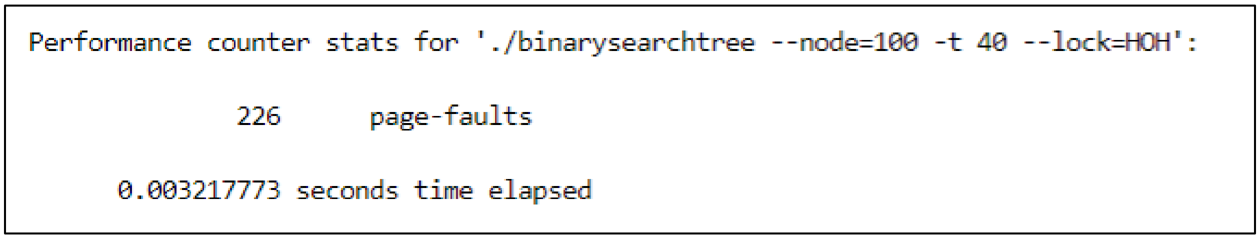
Extant bugs: The program does not handle the errors related to the inputs from the user being out of order. This can be considered as future scope of work.

Timing Analysis:

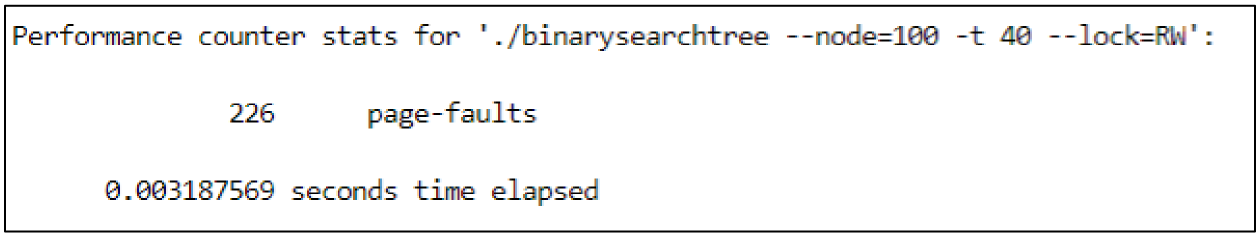
Number of nodes	**Number of threads**	**Lock**	**Contention**	**Time taken in s**
100	40	Hand-over-hand	None	0.002592
100	40	Hand-over-hand	Low	0.001291
100	40	Hand-over-hand	High	0.001417
10	**2**	**Hand-over-hand**	**None**	**0.000203**
10	**2**	**Hand-over-hand**	**Low**	**0.000207**
10	**2**	**Hand-over-hand**	**High**	**0.000214**
10	10	Hand-over-hand	None	0.001249
10	10	Hand-over-hand	Low	0.000615
10	10	Hand-over-hand	High	0.001142
100	**40**	**Reader-writer**	**None**	**0.002255**
100	**40**	**Reader-writer**	**Low**	**0.000771**
100	**40**	**Reader-writer**	**High**	**0.001278**
10	2	Reader-writer	None	0.000206
10	2	Reader-writer	Low	0.000155
10	2	Reader-writer	High	0.000225
10	**10**	**Reader-writer**	**None**	**0.001339**
10	**10**	**Reader-writer**	**Low**	**0.000567**
10	**10**	**Reader-writer**	**High**	**0.000903**

From the above analysis it can be found that hand-over-hand locking takes more time compared to the use of reader-writer locks. Also, it can be seen that, among these implementations, high contention takes up more CPU time. This is justified in the sense that more threads are contending for the same resource and there would be a lot of time wasted in waiting to acquire the locks. This wait is further reduced in terms of reader-writer locks since the reader lock can be held by multiple threads at once. This helps in reducing the overall time taken in executing the program. The behavior of various locks under various conditions can be said to be consistent. In most of the above cases, it can be seen that time taken under no contention > high contention > low contention. There can be some anomalies, but holistically looking, this seems to be the trend.

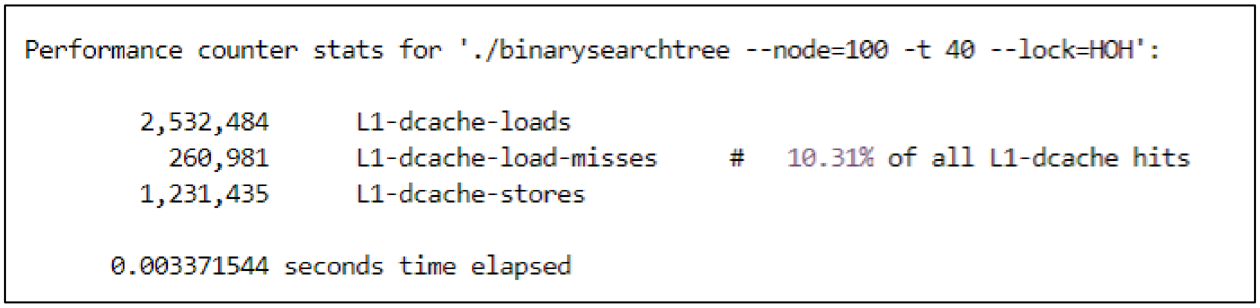
Perf analysis:



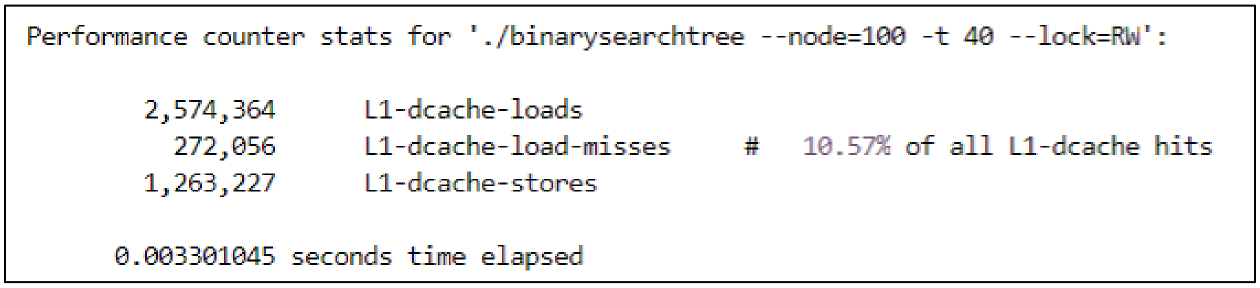
```
Performance counter stats for './binarysearchtree --node=100 -t 40 --lock=HOH':

             226      page-faults

     0.003217773 seconds time elapsed
```



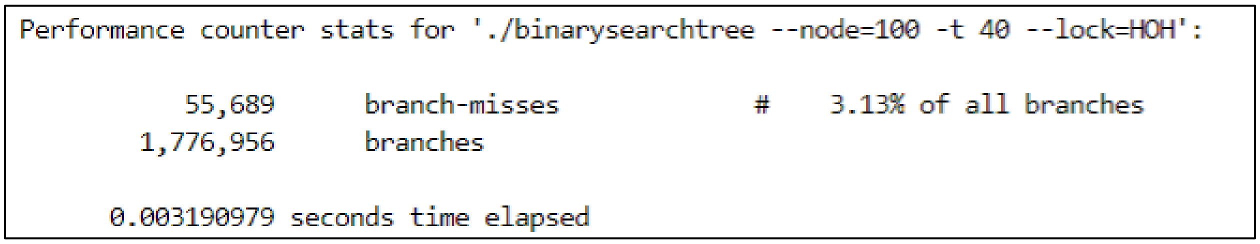
```
Performance counter stats for './binarysearchtree --node=100 -t 40 --lock=RW':

             226      page-faults

     0.003187569 seconds time elapsed
```



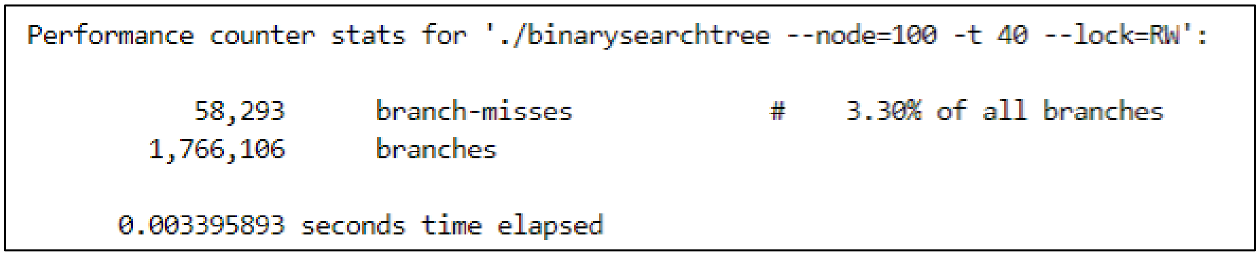
```
Performance counter stats for './binarysearchtree --node=100 -t 40 --lock=HOH':

       2,532,484      L1-dcache-loads
         260,981      L1-dcache-load-misses     #   10.31% of all L1-dcache hits
       1,231,435      L1-dcache-stores

     0.003371544 seconds time elapsed
```



```
Performance counter stats for './binarysearchtree --node=100 -t 40 --lock=RW':

       2,574,364      L1-dcache-loads
         272,056      L1-dcache-load-misses     #   10.57% of all L1-dcache hits
       1,263,227      L1-dcache-stores

     0.003301045 seconds time elapsed
```



```
Performance counter stats for './binarysearchtree --node=100 -t 40 --lock=HOH':

          55,689      branch-misses             #    3.13% of all branches
       1,776,956      branches

     0.003190979 seconds time elapsed
```

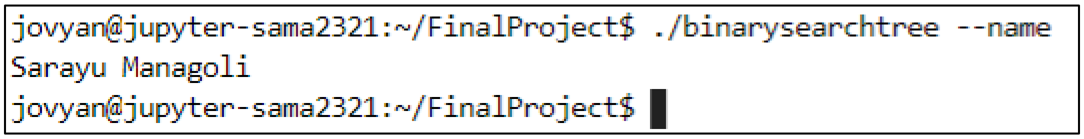


```
Performance counter stats for './binarysearchtree --node=100 -t 40 --lock=RW':

          58,293      branch-misses             #    3.30% of all branches
       1,766,106      branches

     0.003395893 seconds time elapsed
```

It can be seen that most of the performance stats are comparable under the normal mode of operation. This is because both the locks are equally CPU intensive. However, it could be seen that, for hand-over-hand locking under high contention, the cache misses were at 11.26% compared to that of low contention at 10.18%. For RW locks, 10.14% was the miss at low contention compared to 11.02% at high contention. It can still be seen that between the two locks, RW locks are less intensive by a small margin. Another reason why the locks would be similarly lossy in terms of the cache misses could be because of the fact that there are as many write operations as there are read. This could be the reason the RW lock might not be used till its fullest potential since only one thread can obtain the write lock at a time. It can be a reason to slow the execution of the program if not slower than hand-over-hand locking.

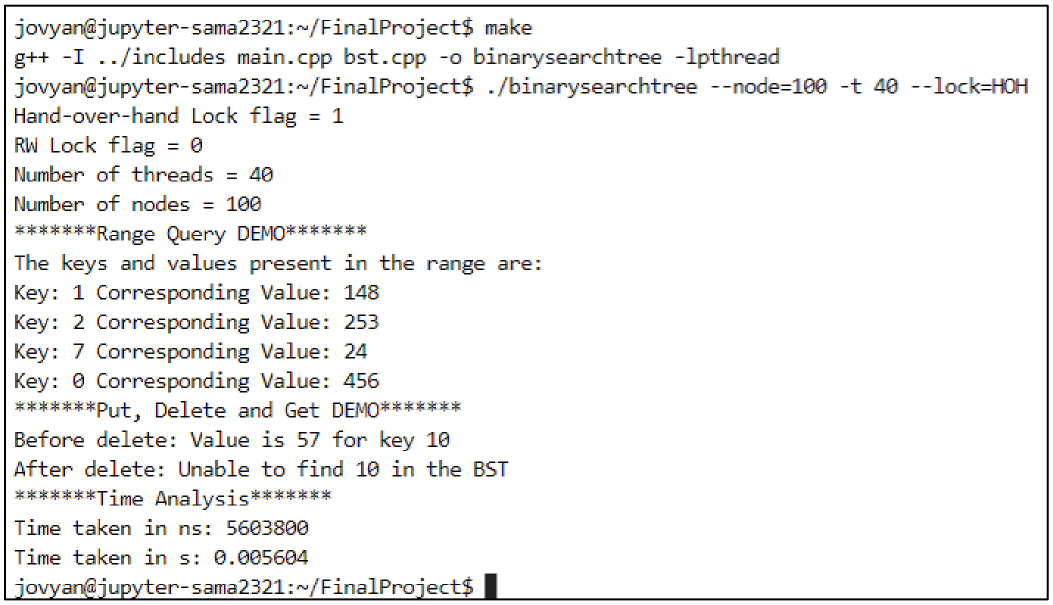
Output:

Name output:

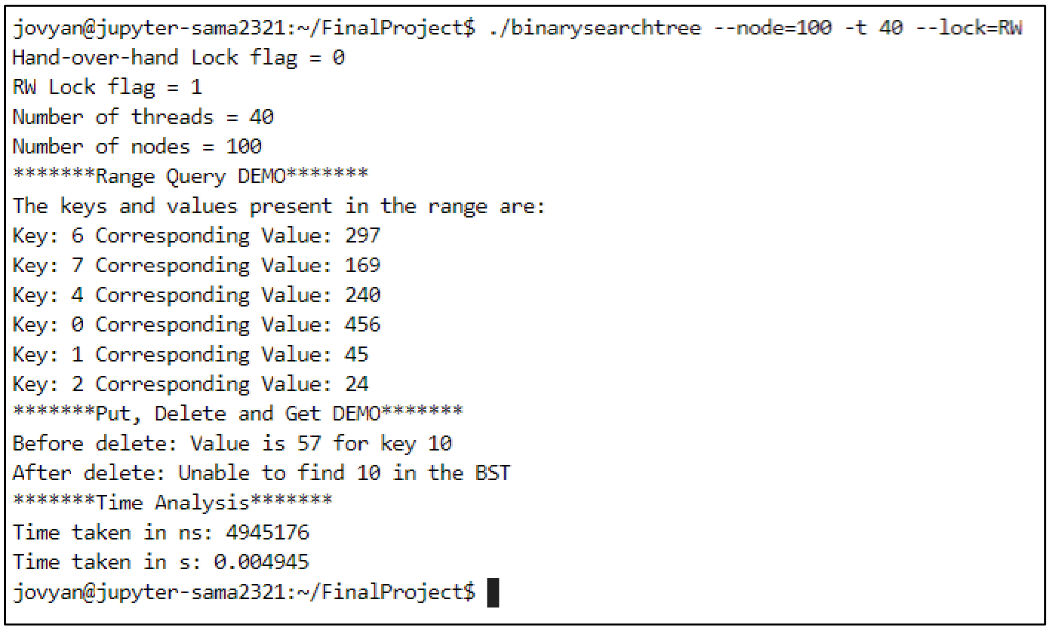


```
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --name
Sarayu Managoli
jovyan@jupyter-sama2321:~/FinalProject$
```

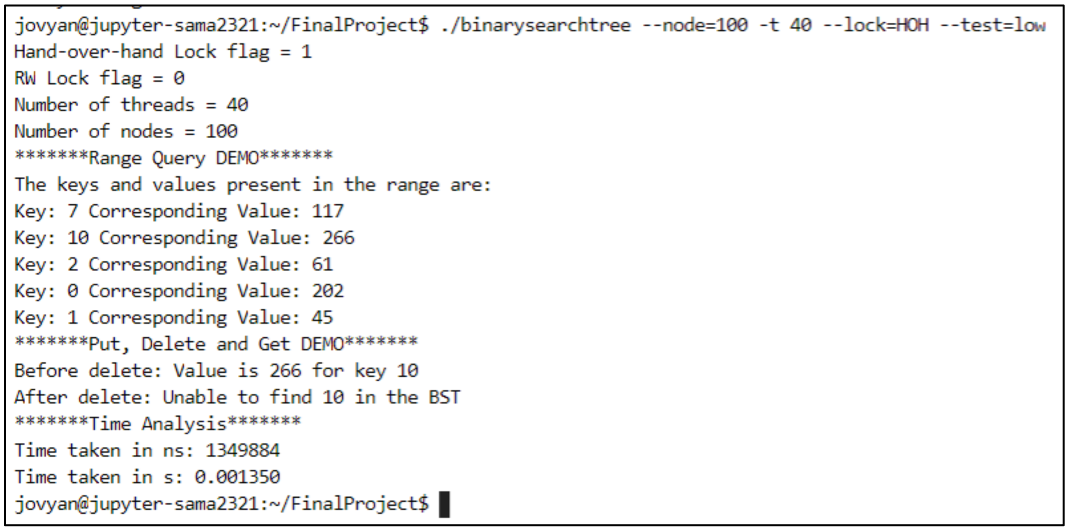
Various mode output:



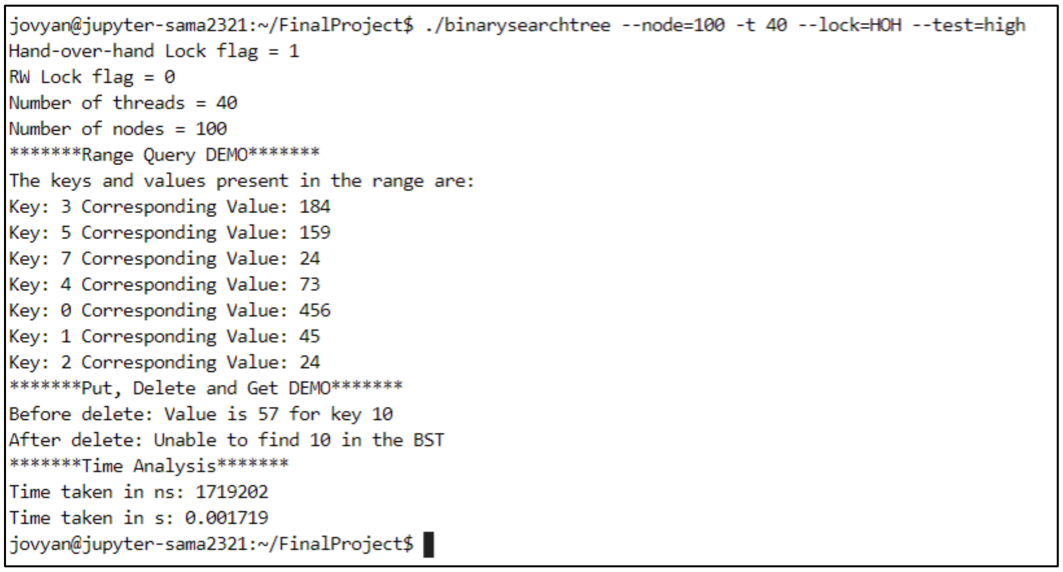
```
jovyan@jupyter-sama2321:~/FinalProject$ make
g++ -I ../includes main.cpp bst.cpp -o binarysearchtree -lpthread
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --node=100 -t 40 --lock=HOH
Hand-over-hand Lock flag = 1
RW Lock flag = 0
Number of threads = 40
Number of nodes = 100
*******Range Query DEMO*******
The keys and values present in the range are:
Key: 1 Corresponding Value: 148
Key: 2 Corresponding Value: 253
Key: 7 Corresponding Value: 24
Key: 0 Corresponding Value: 456
*******Put, Delete and Get DEMO*******
Before delete: Value is 57 for key 10
After delete: Unable to find 10 in the BST
*******Time Analysis*******
Time taken in ns: 5603800
Time taken in s: 0.005604
jovyan@jupyter-sama2321:~/FinalProject$
```



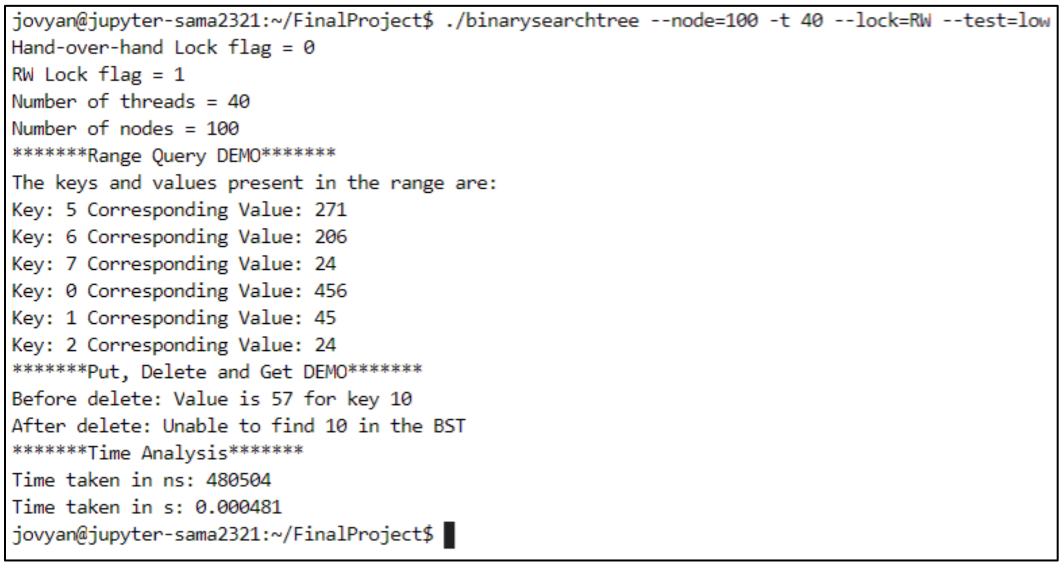
```
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --node=100 -t 40 --lock=RW
Hand-over-hand Lock flag = 0
RW Lock flag = 1
Number of threads = 40
Number of nodes = 100
*******Range Query DEMO*******
The keys and values present in the range are:
Key: 6 Corresponding Value: 297
Key: 7 Corresponding Value: 169
Key: 4 Corresponding Value: 240
Key: 0 Corresponding Value: 456
Key: 1 Corresponding Value: 45
Key: 2 Corresponding Value: 24
*******Put, Delete and Get DEMO*******
Before delete: Value is 57 for key 10
After delete: Unable to find 10 in the BST
*******Time Analysis*******
Time taken in ns: 4945176
Time taken in s: 0.004945
jovyan@jupyter-sama2321:~/FinalProject$
```



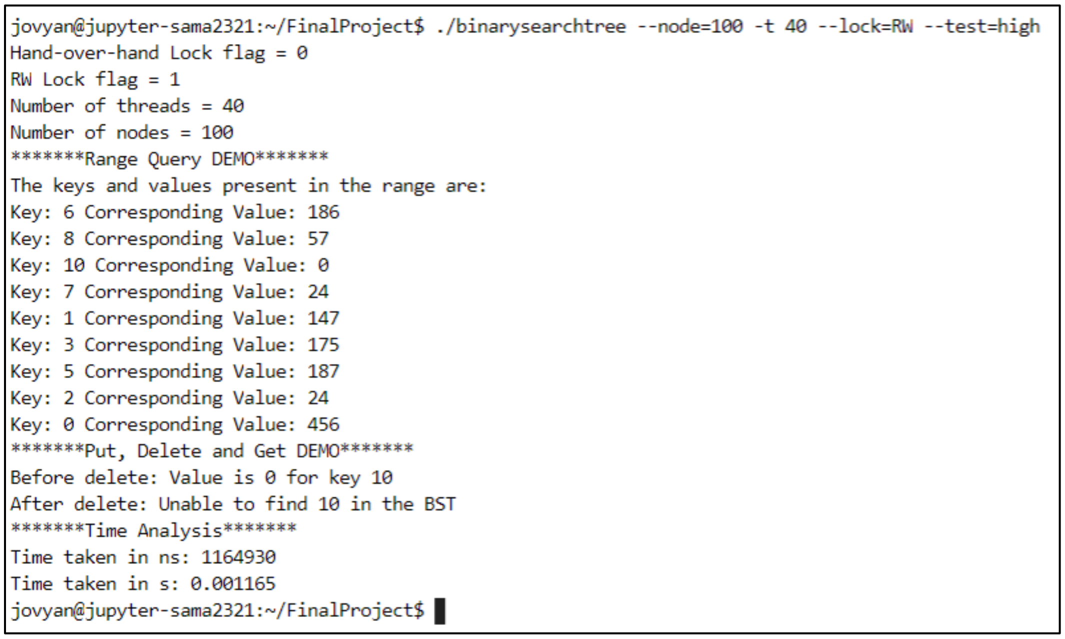
```
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --node=100 -t 40 --lock=HOH --test=low
Hand-over-hand Lock flag = 1
RW Lock flag = 0
Number of threads = 40
Number of nodes = 100
*******Range Query DEMO*******
The keys and values present in the range are:
Key: 7 Corresponding Value: 117
Key: 10 Corresponding Value: 266
Key: 2 Corresponding Value: 61
Key: 0 Corresponding Value: 202
Key: 1 Corresponding Value: 45
*******Put, Delete and Get DEMO*******
Before delete: Value is 266 for key 10
After delete: Unable to find 10 in the BST
*******Time Analysis*******
Time taken in ns: 1349884
Time taken in s: 0.001350
jovyan@jupyter-sama2321:~/FinalProject$ █
```



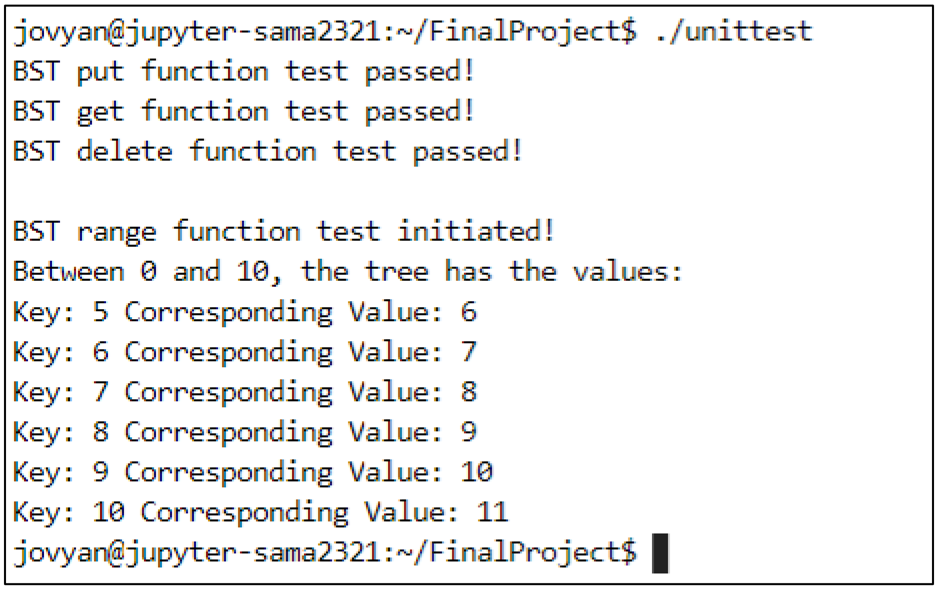
```
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --node=100 -t 40 --lock=HOH --test=high
Hand-over-hand Lock flag = 1
RW Lock flag = 0
Number of threads = 40
Number of nodes = 100
*******Range Query DEMO*******
The keys and values present in the range are:
Key: 3 Corresponding Value: 184
Key: 5 Corresponding Value: 159
Key: 7 Corresponding Value: 24
Key: 4 Corresponding Value: 73
Key: 0 Corresponding Value: 456
Key: 1 Corresponding Value: 45
Key: 2 Corresponding Value: 24
*******Put, Delete and Get DEMO*******
Before delete: Value is 57 for key 10
After delete: Unable to find 10 in the BST
*******Time Analysis*******
Time taken in ns: 1719202
Time taken in s: 0.001719
jovyan@jupyter-sama2321:~/FinalProject$ █
```



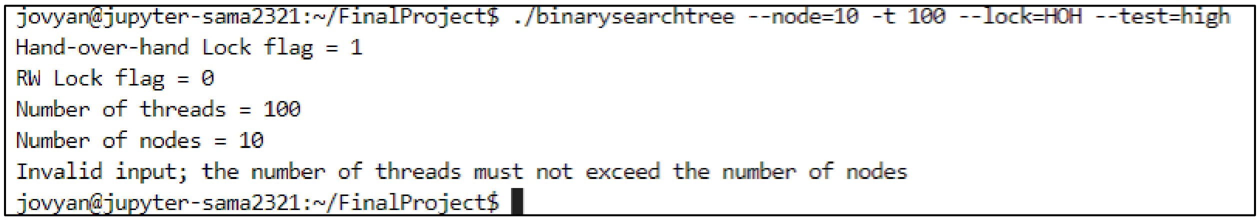
```
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --node=100 -t 40 --lock=RW --test=low
Hand-over-hand Lock flag = 0
RW Lock flag = 1
Number of threads = 40
Number of nodes = 100
*******Range Query DEMO*******
The keys and values present in the range are:
Key: 5 Corresponding Value: 271
Key: 6 Corresponding Value: 206
Key: 7 Corresponding Value: 24
Key: 0 Corresponding Value: 456
Key: 1 Corresponding Value: 45
Key: 2 Corresponding Value: 24
*******Put, Delete and Get DEMO*******
Before delete: Value is 57 for key 10
After delete: Unable to find 10 in the BST
*******Time Analysis*******
Time taken in ns: 480504
Time taken in s: 0.000481
jovyan@jupyter-sama2321:~/FinalProject$ █
```

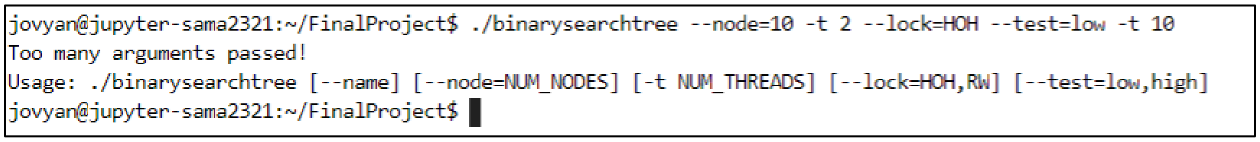

```
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --node=100 -t 40 --lock=RW --test=high
Hand-over-hand Lock flag = 0
RW Lock flag = 1
Number of threads = 40
Number of nodes = 100
*******Range Query DEMO*******
The keys and values present in the range are:
Key: 6 Corresponding Value: 186
Key: 8 Corresponding Value: 57
Key: 10 Corresponding Value: 0
Key: 7 Corresponding Value: 24
Key: 1 Corresponding Value: 147
Key: 3 Corresponding Value: 175
Key: 5 Corresponding Value: 187
Key: 2 Corresponding Value: 24
Key: 0 Corresponding Value: 456
*******Put, Delete and Get DEMO*******
Before delete: Value is 0 for key 10
After delete: Unable to find 10 in the BST
*******Time Analysis*******
Time taken in ns: 1164930
Time taken in s: 0.001165
jovyan@jupyter-sama2321:~/FinalProject$
```

Unit test:

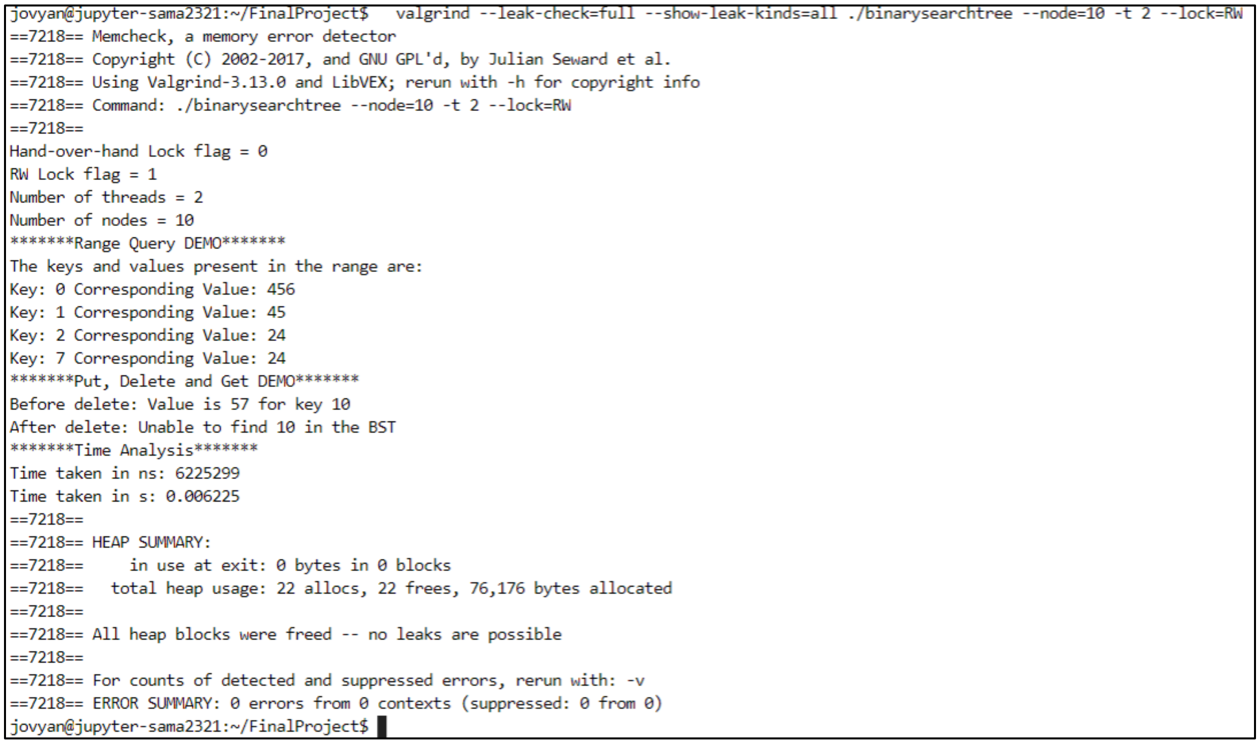

```
jovyan@jupyter-sama2321:~/FinalProject$ ./unittest
BST put function test passed!
BST get function test passed!
BST delete function test passed!

BST range function test initiated!
Between 0 and 10, the tree has the values:
Key: 5 Corresponding Value: 6
Key: 6 Corresponding Value: 7
Key: 7 Corresponding Value: 8
Key: 8 Corresponding Value: 9
Key: 9 Corresponding Value: 10
Key: 10 Corresponding Value: 11
jovyan@jupyter-sama2321:~/FinalProject$
```

Error handling:

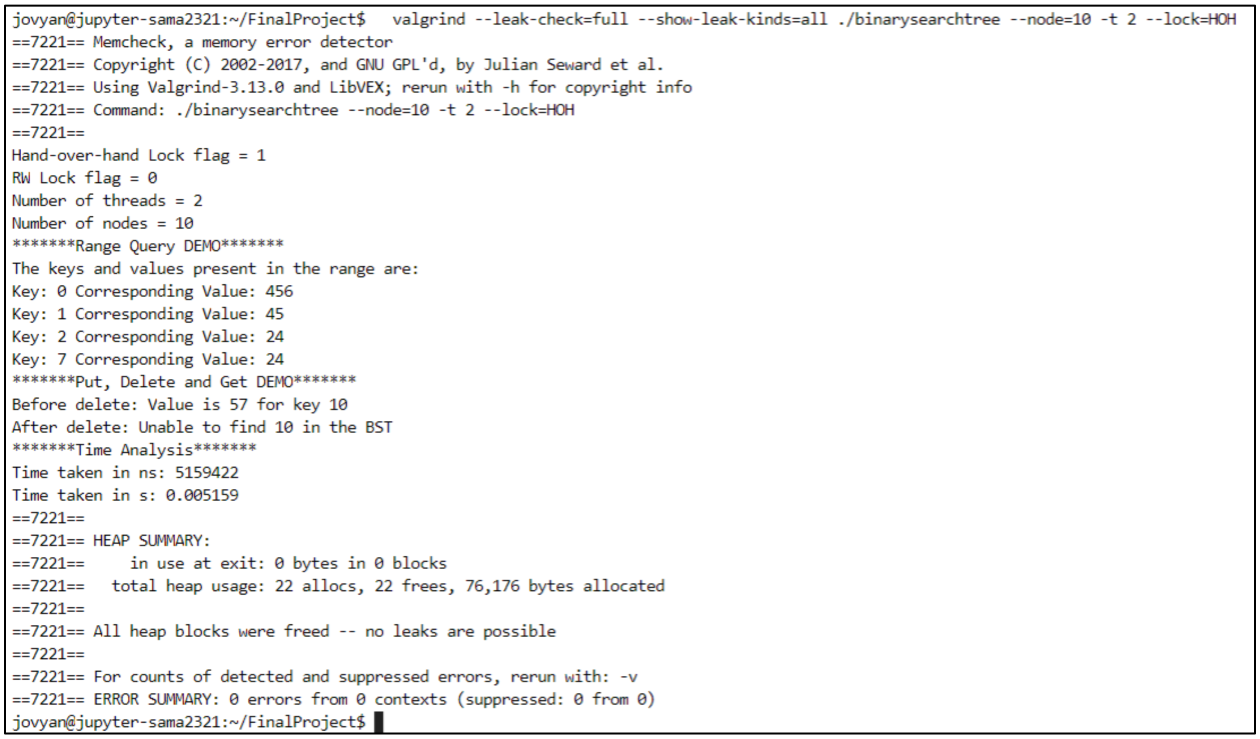

```
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --node=10 -t 100 --lock=HOH --test=high
Hand-over-hand Lock flag = 1
RW Lock flag = 0
Number of threads = 100
Number of nodes = 10
Invalid input; the number of threads must not exceed the number of nodes
jovyan@jupyter-sama2321:~/FinalProject$
```


```
jovyan@jupyter-sama2321:~/FinalProject$ ./binarysearchtree --node=10 -t 2 --lock=HOH --test=low -t 10
Too many arguments passed!
Usage: ./binarysearchtree [--name] [--node=NUM_NODES] [-t NUM_THREADS] [--lock=HOH,RW] [--test=low,high]
jovyan@jupyter-sama2321:~/FinalProject$
```

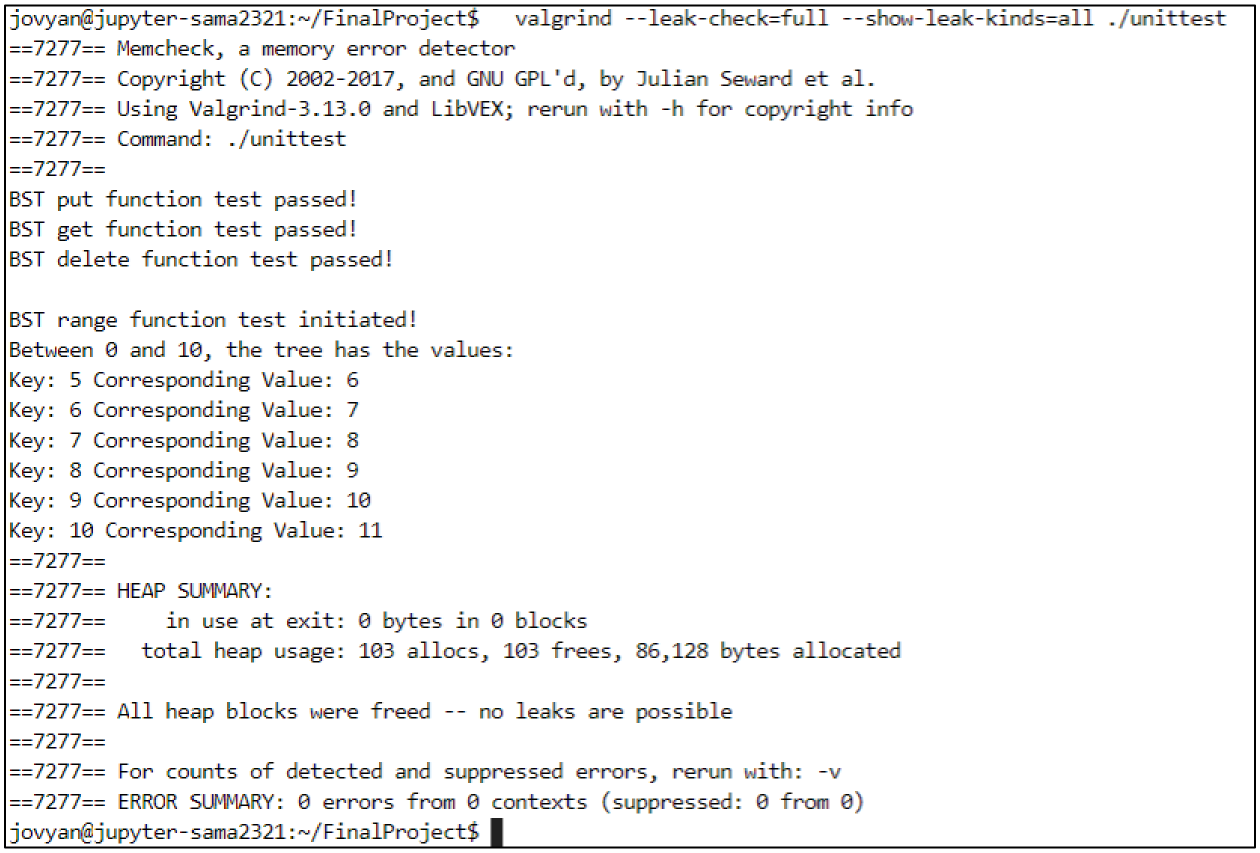
Memory leak checks:



```
jovyan@jupyter-sama2321:~/FinalProject$  valgrind --leak-check=full --show-leak-kinds=all ./binarysearchtree --node=10 -t 2 --lock=RW
==7218== Memcheck, a memory error detector
==7218== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7218== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7218== Command: ./binarysearchtree --node=10 -t 2 --lock=RW
==7218==
Hand-over-hand Lock flag = 0
RW Lock flag = 1
Number of threads = 2
Number of nodes = 10
*******Range Query DEMO*******
The keys and values present in the range are:
Key: 0 Corresponding Value: 456
Key: 1 Corresponding Value: 45
Key: 2 Corresponding Value: 24
Key: 7 Corresponding Value: 24
*******Put, Delete and Get DEMO*******
Before delete: Value is 57 for key 10
After delete: Unable to find 10 in the BST
*******Time Analysis*******
Time taken in ns: 6225299
Time taken in s: 0.006225
==7218==
==7218== HEAP SUMMARY:
==7218==     in use at exit: 0 bytes in 0 blocks
==7218==   total heap usage: 22 allocs, 22 frees, 76,176 bytes allocated
==7218==
==7218== All heap blocks were freed -- no leaks are possible
==7218==
==7218== For counts of detected and suppressed errors, rerun with: -v
==7218== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
jovyan@jupyter-sama2321:~/FinalProject$
```



```
jovyan@jupyter-sama2321:~/FinalProject$  valgrind --leak-check=full --show-leak-kinds=all ./binarysearchtree --node=10 -t 2 --lock=HOH
==7221== Memcheck, a memory error detector
==7221== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7221== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7221== Command: ./binarysearchtree --node=10 -t 2 --lock=HOH
==7221==
Hand-over-hand Lock flag = 1
RW Lock flag = 0
Number of threads = 2
Number of nodes = 10
*******Range Query DEMO*******
The keys and values present in the range are:
Key: 0 Corresponding Value: 456
Key: 1 Corresponding Value: 45
Key: 2 Corresponding Value: 24
Key: 7 Corresponding Value: 24
*******Put, Delete and Get DEMO*******
Before delete: Value is 57 for key 10
After delete: Unable to find 10 in the BST
*******Time Analysis*******
Time taken in ns: 5159422
Time taken in s: 0.005159
==7221==
==7221== HEAP SUMMARY:
==7221==     in use at exit: 0 bytes in 0 blocks
==7221==   total heap usage: 22 allocs, 22 frees, 76,176 bytes allocated
==7221==
==7221== All heap blocks were freed -- no leaks are possible
==7221==
==7221== For counts of detected and suppressed errors, rerun with: -v
==7221== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
jovyan@jupyter-sama2321:~/FinalProject$
```



```
jovyan@jupyter-sama2321:~/FinalProject$   valgrind --leak-check=full --show-leak-kinds=all ./unittest
==7277== Memcheck, a memory error detector
==7277== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7277== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7277== Command: ./unittest
==7277==
BST put function test passed!
BST get function test passed!
BST delete function test passed!

BST range function test initiated!
Between 0 and 10, the tree has the values:
Key: 5 Corresponding Value: 6
Key: 6 Corresponding Value: 7
Key: 7 Corresponding Value: 8
Key: 8 Corresponding Value: 9
Key: 9 Corresponding Value: 10
Key: 10 Corresponding Value: 11
==7277==
==7277== HEAP SUMMARY:
==7277==     in use at exit: 0 bytes in 0 blocks
==7277==   total heap usage: 103 allocs, 103 frees, 86,128 bytes allocated
==7277==
==7277== All heap blocks were freed -- no leaks are possible
==7277==
==7277== For counts of detected and suppressed errors, rerun with: -v
==7277== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
jovyan@jupyter-sama2321:~/FinalProject$
```

References

https://stackoverflow.com/questions/25833541/reading-numbers-from-file-c

https://codeyarns.com/2015/01/30/how-to-parse-program-options-in-c-using-getopt_long/

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/

https://www.tutorialspoint.com/binary-search-tree-delete-operation-in-cplusplus

http://www.brendangregg.com/perf.html