**SOLUTION TO QUESTION - 6 OF ASSIGNMENT:**(10 points) Write a program that does the following.

* Create an instance of your BinaryHeap class with n entries with diﬀerent priorities. Make sure the entries are inserted in random order of priorities.
* Create an instance of your vEBtree class with n entries with diﬀerent priorities. Make sure the entries are inserted in random order of priorities.
* Perform the same sequence of m operations (chosen at random among ExtractMax()and IncreaseKey(value,priority)) in each of the data structures created.)
* For each of those four cases, measure the running times and ﬁll the table below (adjust the values of n as needed according to your platform to obtain at least 4 measurements).

FOR BINARY HEAP

//these are random generated build times and running times for BinaryHeap and they may vary in each case where m operations include Insert(value, priority).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Construction time | N = 100 | N = 1000 | N = 10000 | N = 100000 | N = 1000000 |
| Binary Heap | 1854955 ns | 5051320 ns | 6344334 ns | 27507654 ns | 98747357 ns |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| M operations | N = 100 | N = 1000 | N = 10000 | N = 100000 | N = 1000000 |
| Binary Heap | 13460518 ns | 49690702 ns | 122961743 ns | 686072093 ns | 5131356037 ns |

FOR Veb TREE

//these are random generated build times and running times for VeB Tree and they may vary in each case where m operations include IncreasKey(value, priority) and extractMax().

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Construction time | N = 100 | N = 1000 | N = 10000 | N = 100000 | N = 1000000 |
| vEB Tree | 1593384797 ns | 1522203887 ns | 1502978479 ns | 1554419437 ns | 1643439355 ns |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| M operations | N = 100 | N = 1000 | N = 10000 | N = 100000 | N = 1000000 |
| vEB Tree | 14338530 ns | 47894277 ns | 209544873 ns | 1003427611 ns | 10114858727 ns |

**SOLUTION TO QUESTION - 7 OF ASSIGNMENT**: (20 points) How does these measurements compare with your conjectures? You are NOT being asked which one is faster. If the results differ from your conjecture, investigate the reason by looking carefully at the code and explain what may have happened.

FOR BINARY HEAP

The given graph depicts the build and running case analysis of Binary Heap for n insertions and m operations. The times is calculated in nanoseconds and the instance has values of n=100 to n=1000000 where m operations are being performed and m >= n.

From the graph, the construction time(or Build Time) for n = 1000000 comes up to be a NON-LINEAR function of n in which the dedicated structure to Binary tree is build and the function is displaying construction of an array into the Binary tree structure, whereas the running time comes out to be a function of O(m log n)that involves all other functions that include Insert(value, priority), extractMax(), Increasekey(value, priority) at r1andom.

Hence, the conjectures do not match. As the construction time is not real, in a technical design is our own architectural flaw. Maybe three is not self-balancing. This gives us a brief idea about how the running times can be made linear.

FOR Veb TREE

The given graph depicts the build and running case analysis of vEB Tree for n insertions and m operations. The times is calculated in nanoseconds and the instance has values of n=100 to n=1000000 where m operations are being performed and m >= n.

From the graph, the construction time(or Build Time) for n = 1000000 comes up to U be a function of log and builds up space pretty much fast in construction of a vEB Tree structure. Whereas, the total running time comes out to be a function of O(m log u) that involves functions like extractMax() and IncreaseKey(value, priority).

Hence, the insert function takes O(m log u), extractmax() takes O(m log u) and increaseKey(value, priority)takes O(m log u) running times.

Therefore, the conjectures made were not achieved I the running cases as the big issue comes out to be memory issues. So a better implementation of search algorithm may be a suitable design case for the betterment of running times.