**ECS 60 TIMETEST3.CPP PROGRAM WRITEUP**

**Daivik Dinesh**

**(SID- 913296561)**

**Time Values for each run**

|  |  |  |
| --- | --- | --- |
| **FILE** | **ADT #** | **TIME** |
| **File 1** | **6** | 0.171992 |
| **File 2** | **6** | 0.170305 |
| **File 3** | **6** | 0.279047 |
| **File 4** | **6** | 0.477169 |
| **File 1** | **7** | (More than 5 min) |
| **File 2** | **7** |  |
| **File 3** | **7** | 0.068817 |
| **File 4** | **7** | 50.9677 |
| **File 1** | **8** | 0.254635 |
| **File 2** | **8** | 0.196205 |
| **File 3** | **8** | 0.256295 |
| **File 4** | **8** | 0.386966 |
| **File 1** | **9** | 0.111965 |
| **File 2** | **9** | 0.101951 |
| **File 3** | **9** | 0.109017 |
| **File 4** | **9** | 0.331784 |
| **File 1** | **10** |  |
| **File 2** | **10** | 0.039404 |
| **File 3** | **10** | 0.040295 |
| **File 4** | **10** | 0.022987 |
| **File 1** | **11** |  |
| **File 2** | **11** | 0.130281 |
| **File 3** | **11** | 0.146732 |
| **File 4** | **11** | 0.147275 |

|  |  |  |
| --- | --- | --- |
| **FILE** | **ADT #** | **TIME** |
| **File 1** | **12** |  |
| **File 2** | **12** | **Word Settings (15)** |
| **File 3** | **12** | **Word Settings (15)** |
| **File 4** | **12** | **Word Settings (15)** |
| **File 1** | **13** | 0.117129 |
| **File 2** | **13** | 0.122231 |
| **File 3** | **13** | 0.231396 |
| **File 4** | **13** | 0.252465 |
| **File 1** | **14** |  |
| **File 2** | **14** | **Word Settings (15)** |
| **File 3** | **14** | **Word Settings (15)** |
| **File 4** | **14** | **Word Settings (15)** |

**Time Complexities**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **FILE** | **ADT #** | **Insertion (Individual)** | **Deletion (Individual)** | **Insertion (N-times)** | **Deletion (N-times)** | **Entire File** |
| **File 1** | **6** | O(1) | N/A | O(N) | N/A | O(N) |
| **File 2** | **6** | O(1) | O(N) | O(N) | O(N^2) | O(N^2) |
| **File 3** | **6** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 4** | **6** | O(1) | O(N) | O(N) | O(N^2) | O(N^2) |
| **File 1** | **7** | O(1) | N/A | O(N) | N/A | O(N) |
| **File 2** | **7** | O(1) | O(N) | O(N) | O(N^2) | O(N^2) |
| **File 3** | **7** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 4** | **7** | O(1) | O(N) | O(N) | O(N^2) | O(N^2) |
| **File 1** | **8** | O(1) | N/A | O(N) | N/A | O(N) |
| **File 2** | **8** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 3** | **8** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 4** | **8** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 1** | **9** | O(1) | N/A | O(N) | N/A | O(N) |
| **File 2** | **9** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 3** | **9** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 4** | **9** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 1** | **10** | O(1) | N/A | O(N) | N/A | O(N) |
| **File 2** | **10** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 3** | **10** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 4** | **10** | O(1) | O(1) | O(N) | O(N) | O(N) |
| **File 1** | **11** | O(logN) | N/A | O(NlogN) | N/A | O(NlogN) |
| **File 2** | **11** | O(logN) | O(logN) | O(NlogN) | O(NlogN) | O(NlogN) |
| **File 3** | **11** | O(logN) | O(logN) | O(NlogN) | O(NlogN) | O(NlogN) |
| **File 4** | **11** | O(logN) | O(logN) | O(NlogN) | O(NlogN) | O(NlogN) |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

For File1.dat, since there only consists an insertion sequence from 1-500000, and for File2.dat there’s an insertion sequence of 1-250000 and a deletion sequence of 1-250000. For file3.dat the insertion sequence is the same as file2.dat but the deletion sequence is 250000-1 (reverse of file2.dat). For file4.dat, the insertion and deletion sequences are random and unique from 1-124999 (125000 sequences).

For both File1.dat and File2.dat, in ADT 1 and ADT 2 (Linked List and Cursor List), we notice that the big-O values are similar. Since file1 doesn’t have deletions, deletion-N and individual deletion aren’t applicable there for any of the ADTs. But for ADT 1 and ADT 2, for files 2 and 4, the deletion routines for individual elements is O(N), since the ADT is traversed completely, to find the element, taking a time of O(N). Since the individual deletion takes O(N), naturally, the deletion- N times routine would take O(N^2) time complexity to run. For file3, individual deletion for ADTS 1 and 2 will take O(1) time because since the list is reversed, the elements 250000, 249999, etc. are now at the very beginning of the list, so deletion takes only O(1) time, in this case. So, because of this, deletion-N times will take O(N) time complexity. For individual insertion into ADTs 1 and 2 for ALL files, it is O(1) because whether it is a linked list or a cursor list, element insertion is always done at the beginning of the list or in front of the header. For insertion-N times, since individual insertion is O(1), the time complexity will then turn out to be, O(N) for all the files here, as well.

For ADTs 3, 4, and 5, Stack Array, Stack List and Queue Array, it is observed that these data structures have similar characteristics. Since insertion and deletion is always done at one end, regardless of the number of the element, i.e. the element is always either inserted at the top of the stack or the ‘front’ of the queue, and is ‘popped’ from the top of the stack or deleted from the ‘back’ of the queue, these ADTs have very similar time complexities. For individual insertion into these 3 ADTs for file1, file2, file3 or file4, since the element is always pushed to one ‘end’ of the data structure it will always have a complexity of O(1). Hence, the time complexity for insertion-N times into ADTs 3, 4 and 5 for files 1, 2, 3 or 4 will have a complexity of O(N), as N elements are now being pushed into the structures. Again, as File1.dat has no deletion sequences whatsoever, there will be no big-O values for individual deletion or deletion-N times for ANY of these ADTs. Whereas for individual deletion into these ADTs for files 2, 3 or 4, the big-O value would definitely be O(1) since the data structure would have an element to be deleted by simply ‘popping’ it from the end of the ADT, easily. Since the time complexity for individual deletion of these ADTs is O(1), the deletion-N times routine for ADTs 3, 4 and 5 would have to be O(N), since N elements are now being ‘popped’ or ‘removed’ from the back of these data structures.

For ADT 6, which is the Skip List, the algorithm used is very similar to that of the binary search algorithm. So, when an element is being searched for, it compares the element with the highest node, if it doesn’t match, it continues by searching from the middle of the list nodes, and keeps comparing whether the element is greater than or less than elements in the ‘middle’ which is why it makes the algorithm very similar to the binary search algorithm. So for individual insertion into the skiplist, for all files 1, 2, 3 and 4, the time complexity is O(logN) because the list is traversed completely using a Binary Search-like algorithm that takes logN complexity to find the element, in this case. Similarly, for individual deletion, the big-O value remains O(logN) for files 2,3 and 4. Thus, as it is pretty clear to understand, it would take O(NlogN) time complexity to compute the N-times insertion OR deletion for all files 1, 2, 3 and 4. But, in the above mentioned cases, it is important to note that since deletion sequences don’t exist for file1.dat, the individual deletion or deletion-N times big-O values aren’t applicable to file 1.

Now for Stack List, Stack Array and Queue Array, it is easy to understand that List implementation takes more time to create nodes and delete or insert new nodes than to merely increment or decrement for space or elements in an array. This is why as we observe in the average time values, Stack Arrays and Queue arrays are faster than Stack Lists.

Weiss’ implementation of Cursor list involves unnecessary creation of cursor list objects and the cursor space vector which makes the program slower, so this is why Cursor list is generally slower than Linked list, as we’ve observed in the average time column of the table, although it’s supposed to be more efficient.

Also, as explained above in the essay using time complexities, it is pretty evident that O(N^2) takes more time than O(logN), which takes more time than O(N), which takes more time than O(1). The precedence goes as follows:

**O(1) > O(logN) > O(N) > O(NlogN) > O(N^2)**