Project (2): -

Marmara University - Faulty of Engineering

Department of Computer Engineering

Data Structures (Autumn 2022)

Submit Date: 30/12/2022

Name: Abdelrahman Zahran

Student ID: 150120998

AVL-Tree Vs Splay Tree

Sections Of the Report: -

* Section (1): Problem Definition.
* Section (2): Implementation Details.
* Section (3): Test Cases.

To compare the performance of AVL and Splay trees, you can follow these steps:

1. Read the input text file and store the characters in a list or array.
2. Create two empty AVL and Splay trees.
3. Iterate over the list of characters and perform the following operations for each character:
   * Insert the character in both the AVL and Splay trees if it does not exist. If it already exists, update its frequency in the text.
   * For the AVL tree, if there is an AVL condition violation after inserting the new node, perform the necessary rotations to restore the balance of the tree. Keep track of the total number of rotations performed.
   * For the Splay tree, perform the necessary splay(s) after reading each character. Keep track of the total number of splays performed.
4. At the end of the iteration, calculate the total cost for both the AVL and Splay trees as follows:
   * For the AVL tree, the total cost is the sum of the number of comparisons (successful and unsuccessful) and the number of rotations performed.
   * For the Splay tree, the total cost is the sum of the number of comparisons (successful and unsuccessful) and the number of splays performed.
5. Compare the total costs of the AVL and Splay trees to determine which tree performed better.

It is important to note that the performance of AVL and Splay trees may vary depending on the input data. You may want to test your program with different input datasets to see how the trees perform under different conditions.

The recurrence formulas for AVL and Splay trees can be used to calculate the asymptotic complexity (i.e., the time and space complexity) of these data structures.

AVL Tree:

The time complexity of an AVL tree is determined by the height of the tree, which is O(log n) in the average case and O(n) in the worst case.

Insertion:

* Time complexity: O(log n)
* Space complexity: O(1)

Deletion:

* Time complexity: O(log n)
* Space complexity: O(1)

Search:

* Time complexity: O(log n)
* Space complexity: O(1)

Splay Tree:

The time complexity of a Splay tree is also determined by the height of the tree, which is O(log n) in the average case and O(n) in the worst case.

Insertion:

* Time complexity: O(log n)
* Space complexity: O(1)

Deletion:

* Time complexity: O(log n)
* Space complexity: O(1)

Search:

* Time complexity: O(log n)
* Space complexity: O(1)

It is important to note that the above time and space complexities are for the average and worst cases, and the actual complexity may vary depending on the specific implementation and the input data.

AVL Tree:

An AVL tree is a self-balancing binary search tree, which means that the height of the tree is kept balanced to ensure that the time complexity of the basic operations (insertion, deletion, and search) is O(log n). An AVL tree is named after its inventors, G.M. Adelson-Velsky and E.M. Landis, who published it in their 1962 paper "An algorithm for the organization of information".

The balance factor of an AVL tree is defined as the difference in the heights of the left and right subtrees of a node. An AVL tree is considered balanced if the balance factor of every node is in the range [-1, 1]. If the balance factor of a node is outside this range, then the tree is considered unbalanced and rotations are performed to restore the balance.

Insertion:

To insert a new node in an AVL tree, follow the same steps as in a binary search tree, but also check the balance factor of the nodes after each insertion. If the balance factor of a node is outside the range [-1, 1], perform the necessary rotations to restore the balance of the tree.

The time complexity of insertion in an AVL tree is O(log n) in the average case and O(n) in the worst case.

Deletion:

To delete a node from an AVL tree, follow the same steps as in a binary search tree, but also check the balance factor of the nodes after each deletion. If the balance factor of a node is outside the range [-1, 1], perform the necessary rotations to restore the balance of the tree.

The time complexity of deletion in an AVL tree is O(log n) in the average case and O(n) in the worst case.

Search:

To search for a node in an AVL tree, follow the same steps as in a binary search tree. The time complexity of search in an AVL tree is O(log n) in the average case and O(n) in the worst case.

Splay Tree:

A Splay tree is a self-balancing binary search tree, similar to an AVL tree. The main difference between the two is that in a Splay tree, the recently accessed elements are moved to the top of the tree, while in an AVL tree, the balance factor of the nodes is maintained to ensure a balanced tree.

Insertion:

To insert a new node in a Splay tree, follow the same steps as in a binary search tree, but also perform splay operations to bring the recently inserted node to the top of the tree.

The time complexity of insertion in a Splay tree is O(log n) in the average case and O(n) in the worst case.

Deletion:

To delete a node from a Splay tree, follow the same steps as in a binary search tree, but also perform splay operations to bring the parent of the deleted node to the top of the tree.

The time complexity of deletion in a Splay tree is O(log n) in the average case and O(n) in the worst case.

Search:

To search for a node in a Splay tree, follow the same steps as in a binary search tree, but also perform splay operations to bring the searched node to the top of the tree. The time complexity of search in a Splay tree is O(log n) in the average case and O(n) in the worst case.

Input File (1): - Calculations:-

AVL Tree (1): -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num. of Nodes** | **Num. of Comparisons** | **Num. of Rotations** | **AVL Total Cost** | **AVL Performance** |
| 1 | 0 | 0 | 0 |  |
| 2 | 0 | 0 | 0 |  |
| 3 | 1 | 0 | 1 | 1 |
| 4 | 3 | 2 | 5 | 0.2 |
| 5 | 5 | 2 | 7 | 0.142857 |
| 6 | 8 | 4 | 12 | 0.083333 |
| 7 | 11 | 4 | 15 | 0.066667 |
| 8 | 13 | 4 | 17 | 0.058824 |
| 9 | 16 | 6 | 22 | 0.045455 |
| 10 | 19 | 6 | 25 | 0.04 |
| 11 | 22 | 6 | 28 | 0.035714 |
| 12 | 25 | 6 | 31 | 0.032258 |
| 13 | 29 | 7 | 36 | 0.027778 |
| 14 | 33 | 7 | 40 | 0.025 |
| 15 | 38 | 9 | 47 | 0.021277 |
| 16 | 41 | 9 | 50 | 0.02 |
| 17 | 44 | 9 | 53 | 0.018868 |
| 18 | 49 | 10 | 59 | 0.016949 |
| 19 | 54 | 11 | 65 | 0.015385 |
| 20 | 58 | 11 | 69 | 0.014493 |
| 21 | 62 | 11 | 73 | 0.013699 |
| 22 | 66 | 11 | 77 | 0.012987 |
| 23 | 70 | 11 | 81 | 0.012346 |
| 24 | 73 | 11 | 84 | 0.011905 |
| 25 | 76 | 11 | 87 | 0.011494 |
| 26 | 81 | 13 | 94 | 0.010638 |
| 27 | 85 | 13 | 98 | 0.010204 |
| 28 | 89 | 13 | 102 | 0.009804 |
| 29 | 94 | 13 | 107 | 0.009346 |
| 30 | 99 | 13 | 112 | 0.008929 |
| 31 | 102 | 13 | 115 | 0.008696 |
| 32 | 106 | 13 | 119 | 0.008403 |
| 33 | 111 | 15 | 126 | 0.007937 |
| 34 | 116 | 15 | 131 | 0.007634 |
| 35 | 120 | 15 | 135 | 0.007407 |
| 36 | 125 | 15 | 140 | 0.007143 |
| 37 | 129 | 15 | 144 | 0.006944 |
| 38 | 134 | 15 | 149 | 0.006711 |
| 39 | 138 | 15 | 153 | 0.006536 |
| 40 | 143 | 15 | 158 | 0.006329 |
| 41 | 147 | 15 | 162 | 0.006173 |
| 42 | 153 | 15 | 168 | 0.005952 |
| 43 | 158 | 15 | 173 | 0.00578 |
| 44 | 163 | 15 | 178 | 0.005618 |
| 45 | 167 | 15 | 182 | 0.005495 |
| 46 | 171 | 15 | 186 | 0.005376 |
| 47 | 177 | 17 | 194 | 0.005155 |
| 48 | 181 | 17 | 198 | 0.005051 |
| 49 | 186 | 17 | 203 | 0.004926 |
| 50 | 191 | 17 | 208 | 0.004808 |
| 51 | 195 | 17 | 212 | 0.004717 |
| 52 | 201 | 17 | 218 | 0.004587 |
| 53 | 207 | 17 | 224 | 0.004464 |
| 54 | 212 | 17 | 229 | 0.004367 |
| 55 | 217 | 19 | 236 | 0.004237 |
| 56 | 221 | 19 | 240 | 0.004167 |
| 57 | 226 | 19 | 245 | 0.004082 |
| 58 | 230 | 19 | 249 | 0.004016 |
| 59 | 236 | 20 | 256 | 0.003906 |
| 60 | 240 | 20 | 260 | 0.003846 |

Splay Tree (1): -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num. of Nodes** | **Num. of Comparisons** | **Num. of Rotations** | **Splay Total Cost** | **Splay Performance** |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 2 | 0.5 |
| 4 | 3 | 3 | 6 | 0.166667 |
| 5 | 5 | 5 | 10 | 0.1 |
| 6 | 7 | 7 | 14 | 0.071429 |
| 7 | 9 | 8 | 17 | 0.058824 |
| 8 | 14 | 13 | 27 | 0.037037 |
| 9 | 16 | 15 | 31 | 0.032258 |
| 10 | 18 | 17 | 35 | 0.028571 |
| 11 | 23 | 22 | 45 | 0.022222 |
| 12 | 29 | 28 | 57 | 0.017544 |
| 13 | 33 | 32 | 65 | 0.015385 |
| 14 | 37 | 36 | 73 | 0.013699 |
| 15 | 39 | 38 | 77 | 0.012987 |
| 16 | 47 | 46 | 93 | 0.010753 |
| 17 | 52 | 50 | 102 | 0.009804 |
| 18 | 55 | 53 | 108 | 0.009259 |
| 19 | 59 | 57 | 116 | 0.008621 |
| 20 | 68 | 66 | 134 | 0.007463 |
| 21 | 73 | 70 | 143 | 0.006993 |
| 22 | 74 | 70 | 144 | 0.006944 |
| 23 | 77 | 73 | 150 | 0.006667 |
| 24 | 83 | 79 | 162 | 0.006173 |
| 25 | 89 | 85 | 174 | 0.005747 |
| 26 | 93 | 89 | 182 | 0.005495 |
| 27 | 101 | 97 | 198 | 0.005051 |
| 28 | 107 | 103 | 210 | 0.004762 |
| 29 | 114 | 109 | 223 | 0.004484 |
| 30 | 116 | 110 | 226 | 0.004425 |
| 31 | 119 | 112 | 231 | 0.004329 |
| 32 | 126 | 119 | 245 | 0.004082 |
| 33 | 130 | 123 | 253 | 0.003953 |
| 34 | 135 | 128 | 263 | 0.003802 |
| 35 | 142 | 134 | 276 | 0.003623 |
| 36 | 147 | 138 | 285 | 0.003509 |
| 37 | 151 | 142 | 293 | 0.003413 |
| 38 | 157 | 147 | 304 | 0.003289 |
| 39 | 166 | 155 | 321 | 0.003115 |
| 40 | 173 | 162 | 335 | 0.002985 |
| 41 | 175 | 163 | 338 | 0.002959 |
| 42 | 180 | 167 | 347 | 0.002882 |
| 43 | 183 | 170 | 353 | 0.002833 |
| 44 | 187 | 174 | 361 | 0.00277 |
| 45 | 193 | 179 | 372 | 0.002688 |
| 46 | 200 | 185 | 385 | 0.002597 |
| 47 | 205 | 190 | 395 | 0.002532 |
| 48 | 212 | 196 | 408 | 0.002451 |
| 49 | 218 | 201 | 419 | 0.002387 |
| 50 | 225 | 207 | 432 | 0.002315 |
| 51 | 235 | 216 | 451 | 0.002217 |
| 52 | 239 | 219 | 458 | 0.002183 |
| 53 | 244 | 223 | 467 | 0.002141 |
| 54 | 250 | 228 | 478 | 0.002092 |
| 55 | 258 | 236 | 494 | 0.002024 |
| 56 | 260 | 237 | 497 | 0.002012 |
| 57 | 264 | 241 | 505 | 0.00198 |
| 58 | 269 | 246 | 515 | 0.001942 |
| 59 | 279 | 256 | 535 | 0.001869 |
| 60 |  |  | 549 | 0.001821 |

Input File (2): - Calculations:-

AVL-Tree (2): -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num. of Nodes** | **Num. of Comparisons** | **Num. of Rotations** | **AVL Total Cost** | **AVL Performance** |
| 1 | 0 | 0 | 0 |  |
| 2 | 0 | 0 | 0 |  |
| 3 | 1 | 0 | 1 | 1 |
| 4 | 3 | 2 | 5 | 0.2 |
| 5 | 5 | 2 | 7 | 0.142857 |
| 6 | 8 | 4 | 12 | 0.083333 |
| 7 | 11 | 6 | 17 | 0.058824 |
| 8 | 14 | 8 | 22 | 0.045455 |
| 9 | 17 | 8 | 25 | 0.04 |
| 10 | 20 | 8 | 28 | 0.035714 |
| 11 | 23 | 8 | 31 | 0.032258 |
| 12 | 26 | 8 | 34 | 0.029412 |
| 13 | 30 | 10 | 40 | 0.025 |
| 14 | 34 | 10 | 44 | 0.022727 |
| 15 | 37 | 10 | 47 | 0.021277 |
| 16 | 41 | 10 | 51 | 0.019608 |
| 17 | 45 | 11 | 56 | 0.017857 |
| 18 | 49 | 12 | 61 | 0.016393 |
| 19 | 53 | 12 | 65 | 0.015385 |
| 20 | 57 | 12 | 69 | 0.014493 |
| 21 | 62 | 13 | 75 | 0.013333 |
| 22 | 66 | 13 | 79 | 0.012658 |
| 23 | 70 | 13 | 83 | 0.012048 |
| 24 | 75 | 14 | 89 | 0.011236 |
| 25 | 79 | 14 | 93 | 0.010753 |
| 26 | 83 | 14 | 97 | 0.010309 |
| 27 | 88 | 15 | 103 | 0.009709 |
| 28 | 93 | 17 | 110 | 0.009091 |
| 29 | 97 | 17 | 114 | 0.008772 |
| 30 | 102 | 17 | 119 | 0.008403 |
| 31 | 106 | 17 | 123 | 0.00813 |
| 32 | 110 | 17 | 127 | 0.007874 |
| 33 | 115 | 19 | 134 | 0.007463 |
| 34 | 119 | 19 | 138 | 0.007246 |
| 35 | 124 | 20 | 144 | 0.006944 |
| 36 | 128 | 20 | 148 | 0.006757 |
| 37 | 132 | 20 | 152 | 0.006579 |
| 38 | 136 | 20 | 156 | 0.00641 |
| 39 | 141 | 20 | 161 | 0.006211 |
| 40 | 146 | 20 | 166 | 0.006024 |
| 41 | 151 | 22 | 173 | 0.00578 |
| 42 | 156 | 23 | 179 | 0.005587 |
| 43 | 162 | 24 | 186 | 0.005376 |
| 44 | 168 | 25 | 193 | 0.005181 |
| 45 | 173 | 25 | 198 | 0.005051 |
| 46 | 178 | 25 | 203 | 0.004926 |
| 47 | 181 | 25 | 206 | 0.004854 |
| 48 | 186 | 25 | 211 | 0.004739 |
| 49 | 191 | 25 | 216 | 0.00463 |
| 50 | 197 | 27 | 224 | 0.004464 |
| 51 | 202 | 27 | 229 | 0.004367 |
| 52 | 208 | 28 | 236 | 0.004237 |
| 53 | 213 | 28 | 241 | 0.004149 |
| 54 | 218 | 28 | 246 | 0.004065 |
| 55 | 223 | 28 | 251 | 0.003984 |
| 56 | 229 | 28 | 257 | 0.003891 |
| 57 | 235 | 30 | 265 | 0.003774 |
| 58 | 241 | 30 | 271 | 0.00369 |
| 59 | 246 | 30 | 276 | 0.003623 |
| 60 | 252 | 31 | 283 | 0.003534 |
| 61 | 258 | 31 | 289 | 0.00346 |
| 62 | 263 | 31 | 294 | 0.003401 |
| 63 | 269 | 31 | 300 | 0.003333 |
| 64 | 276 | 33 | 309 | 0.003236 |
| 65 | 281 | 33 | 314 | 0.003185 |
| 66 | 285 | 33 | 318 | 0.003145 |
| 67 | 290 | 33 | 323 | 0.003096 |
| 68 | 297 | 33 | 330 | 0.00303 |
| 69 | 303 | 33 | 336 | 0.002976 |
| 70 | 309 | 33 | 342 | 0.002924 |
| 71 | 316 | 35 | 351 | 0.002849 |
| 72 | 321 | 35 | 356 | 0.002809 |
| 73 | 325 | 35 | 360 | 0.002778 |
| 74 | 330 | 35 | 365 | 0.00274 |
| 75 | 331 | 35 | 366 | 0.002732 |
| 76 | 335 | 35 | 370 | 0.002703 |
| 77 | 341 | 35 | 376 | 0.00266 |
| 78 | 346 | 35 | 381 | 0.002625 |
| 79 | 352 | 35 | 387 | 0.002584 |
| 80 | 357 | 35 | 392 | 0.002551 |
| 81 | 364 | 37 | 401 | 0.002494 |
| 82 | 369 | 37 | 406 | 0.002463 |
| 83 | 376 | 39 | 415 | 0.00241 |
| 84 | 383 | 39 | 422 | 0.00237 |
|  |  |  | 427 | 0.002342 |

Splay Tree (2): -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num. of Nodes** | **Num. of Comparisons** | **Num. of Rotations** | **Splay Total Cost** | **Splay Performance** |
| 1 | 0 | 0 | 0 |  |
| 2 | 0 | 0 | 0 |  |
| 3 | 1 | 1 | 2 | 0.5 |
| 4 | 3 | 3 | 6 | 0.166667 |
| 5 | 5 | 5 | 10 | 0.1 |
| 6 | 7 | 7 | 14 | 0.071429 |
| 7 | 10 | 10 | 20 | 0.05 |
| 8 | 13 | 13 | 26 | 0.038462 |
| 9 | 17 | 17 | 34 | 0.029412 |
| 10 | 21 | 21 | 42 | 0.02381 |
| 11 | 26 | 26 | 52 | 0.019231 |
| 12 | 29 | 29 | 58 | 0.017241 |
| 13 | 35 | 35 | 70 | 0.014286 |
| 14 | 39 | 39 | 78 | 0.012821 |
| 15 | 46 | 46 | 92 | 0.01087 |
| 16 | 50 | 50 | 100 | 0.01 |
| 17 | 56 | 56 | 112 | 0.008929 |
| 18 | 61 | 61 | 122 | 0.008197 |
| 19 | 67 | 67 | 134 | 0.007463 |
| 20 | 74 | 74 | 148 | 0.006757 |
| 21 | 80 | 80 | 160 | 0.00625 |
| 22 | 84 | 83 | 167 | 0.005988 |
| 23 | 85 | 83 | 168 | 0.005952 |
| 24 | 87 | 85 | 172 | 0.005814 |
| 25 | 93 | 91 | 184 | 0.005435 |
| 26 | 98 | 96 | 194 | 0.005155 |
| 27 | 102 | 100 | 202 | 0.00495 |
| 28 | 106 | 104 | 210 | 0.004762 |
| 29 | 110 | 107 | 217 | 0.004608 |
| 30 | 113 | 110 | 223 | 0.004484 |
| 31 | 118 | 114 | 232 | 0.00431 |
| 32 | 126 | 122 | 248 | 0.004032 |
| 33 | 130 | 126 | 256 | 0.003906 |
| 34 | 136 | 132 | 268 | 0.003731 |
| 35 | 149 | 145 | 294 | 0.003401 |
| 36 | 154 | 149 | 303 | 0.0033 |
| 37 | 159 | 153 | 312 | 0.003205 |
| 38 | 168 | 161 | 329 | 0.00304 |
| 39 | 173 | 165 | 338 | 0.002959 |
| 40 | 178 | 170 | 348 | 0.002874 |
| 41 | 184 | 176 | 360 | 0.002778 |
| 42 | 195 | 187 | 382 | 0.002618 |
| 43 | 198 | 190 | 388 | 0.002577 |
| 44 | 201 | 193 | 394 | 0.002538 |
| 45 | 210 | 201 | 411 | 0.002433 |
| 46 | 221 | 212 | 433 | 0.002309 |
| 47 | 225 | 215 | 440 | 0.002273 |
| 48 | 232 | 221 | 453 | 0.002208 |
| 49 | 239 | 228 | 467 | 0.002141 |
| 50 | 247 | 236 | 483 | 0.00207 |
| 51 | 253 | 242 | 495 | 0.00202 |
| 52 | 259 | 248 | 507 | 0.001972 |
| 53 | 266 | 255 | 521 | 0.001919 |
| 54 | 276 | 264 | 540 | 0.001852 |
| 55 | 280 | 268 | 548 | 0.001825 |
| 56 | 288 | 276 | 564 | 0.001773 |
| 57 | 294 | 282 | 576 | 0.001736 |
| 58 | 297 | 284 | 581 | 0.001721 |
| 59 | 301 | 288 | 589 | 0.001698 |
| 60 | 306 | 293 | 599 | 0.001669 |
| 61 | 313 | 300 | 613 | 0.001631 |
| 62 | 324 | 310 | 634 | 0.001577 |
| 63 | 329 | 315 | 644 | 0.001553 |
| 64 | 333 | 319 | 652 | 0.001534 |
| 65 | 341 | 327 | 668 | 0.001497 |
| 66 | 348 | 333 | 681 | 0.001468 |
| 67 | 354 | 338 | 692 | 0.001445 |
| 68 | 358 | 341 | 699 | 0.001431 |
| 69 | 366 | 349 | 715 | 0.001399 |
| 70 | 381 | 364 | 745 | 0.001342 |
| 71 | 386 | 369 | 755 | 0.001325 |
| 72 | 389 | 371 | 760 | 0.001316 |
| 73 | 394 | 375 | 769 | 0.0013 |
| 74 | 399 | 379 | 778 | 0.001285 |
| 75 | 403 | 382 | 785 | 0.001274 |
| 76 | 412 | 390 | 802 | 0.001247 |
| 77 | 416 | 393 | 809 | 0.001236 |
| 78 | 424 | 401 | 825 | 0.001212 |
| 79 | 430 | 406 | 836 | 0.001196 |
| 80 | 436 | 411 | 847 | 0.001181 |
| 81 | 444 | 419 | 863 | 0.001159 |
| 82 | 454 | 429 | 883 | 0.001133 |
| 83 | 460 | 435 | 895 | 0.001117 |
| 84 | 466 | 440 | 906 | 0.001104 |
|  |  |  | 926 | 0.00108 |

From Previous Calculations: We conclude some points: -

# Difference in terms of Cost of operations and Performance:-

AVL and Splay trees are both self-balancing binary search trees, which means that they both have a time complexity of O(log n) for the basic operations (insertion, deletion, and search). However, there are some differences between the two in terms of the cost of operations and performance:

1. Balance maintenance: In an AVL tree, the balance of the tree is maintained by checking the balance factor of the nodes after each insertion or deletion and performing rotations if necessary. In a Splay tree, the balance of the tree is maintained by moving the recently accessed elements to the top of the tree.
2. Rotation cost: In an AVL tree, rotations are performed to maintain the balance of the tree. A single rotation in an AVL tree costs one time unit (tu), while a double rotation costs two tus. In a Splay tree, splay operations are performed to move the recently accessed elements to the top of the tree. The cost of a splay operation is equal to the number of depth levels that the node has moved through.
3. Search performance: In an AVL tree, the search performance is not affected by the recent access patterns, as the tree is balanced based on the balance factor of the nodes. In a Splay tree, the search performance may be affected by the recent access patterns, as the tree is balanced based on the access history of the elements.

Overall, AVL and Splay trees are both efficient data structures for storing and accessing data, but the specific performance and cost of operations may vary depending on the implementation and the input data.

# For Small Data Input: -

For small input sizes, the performance of AVL and Splay trees may not be significantly different, as the time complexity of both data structures is O(log n) for the basic operations (insertion, deletion, and search). However, there are some differences between the two data structures that may affect their performance for small input sizes:

1. Rotation cost: In an AVL tree, rotations are performed to maintain the balance of the tree. A single rotation in an AVL tree costs one time unit (tu), while a double rotation cost two tus. In a Splay tree, splay operations are performed to move the recently accessed elements to the top of the tree. The cost of a splay operation is equal to the number of depth levels that the node has moved through. For small input sizes, the cost of rotations or splays may not be significant compared to the overall cost of the operation.
2. Search performance: In an AVL tree, the search performance is not affected by the recent access patterns, as the tree is balanced based on the balance factor of the nodes. In a Splay tree, the search performance may be affected by the recent access patterns, as the tree is balanced based on the access history of the elements. For small input sizes, the difference in search performance between AVL and Splay trees may not be significant.

Overall, for small input sizes, the performance and cost of operations for AVL and Splay trees may not be significantly different. It is important to test the performance of both data structures with different input sizes and access patterns to determine which one performs better in a particular scenario.

# For Large Data Input: -

For large input sizes, the performance of AVL and Splay trees may differ, as the time complexity of both data structures is O(log n) for the basic operations (insertion, deletion, and search). However, there are some differences between the two data structures that may affect their performance for large input sizes:

1. Rotation cost: In an AVL tree, rotations are performed to maintain the balance of the tree. A single rotation in an AVL tree costs one time unit (tu), while a double rotation costs two tus. In a Splay tree, splay operations are performed to move the recently accessed elements to the top of the tree. The cost of a splay operation is equal to the number of depth levels that the node has moved through. For large input sizes, the cost of rotations or splays may be significant compared to the overall cost of the operation.
2. Search performance: In an AVL tree, the search performance is not affected by the recent access patterns, as the tree is balanced based on the balance factor of the nodes. In a Splay tree, the search performance may be affected by the recent access patterns, as the tree is balanced based on the access history of the elements. For large input sizes, the difference in search performance between AVL and Splay trees may be significant, depending on the access patterns.

Overall, for large input sizes, the performance and cost of operations for AVL and Splay trees may differ. It is important to test the performance of both data structures with different input sizes and access patterns to determine which one performs better in a particular scenario.

# For Inputs with Duplicates: -

For input with duplicates, both AVL and Splay trees will perform similarly, as the time complexity of both data structures is O(log n) for the basic operations (insertion, deletion, and search). However, there are some differences between the two data structures that may affect their performance for input with duplicates:

1. Rotation cost: In an AVL tree, rotations are performed to maintain the balance of the tree. A single rotation in an AVL tree costs one time unit (tu), while a double rotation costs two tus. In a Splay tree, splay operations are performed to move the recently accessed elements to the top of the tree. The cost of a splay operation is equal to the number of depth levels that the node has moved through. The cost of rotations or splays may be significant for input with many duplicates, as the number of operations may increase.
2. Search performance: In an AVL tree, the search performance is not affected by the recent access patterns, as the tree is balanced based on the balance factor of the nodes. In a Splay tree, the search performance may be affected by the recent access patterns, as the tree is balanced based on the access history of the elements. The search performance of both data structures may be affected by the presence of many duplicates, as the number of comparisons may increase.

Overall, for input with duplicates, the performance and cost of operations for AVL and Splay trees may not be significantly different. It is important to test the performance of both data structures with different input sizes and access patterns to determine which one performs better in a particular scenario.

The cost of operations in AVL and Splay trees is measured in time units (tus), which represent the number of basic operations (comparisons or rotations/splays) required to perform a particular task. The specific cost of operations in AVL and Splay trees may vary depending on the implementation and the input data.

Here is a comparison of the cost of some basic operations in AVL and Splay trees in terms of tus:

Insertion:

* In an AVL tree, the cost of insertion is determined by the number of comparisons required to find the appropriate position for the new node and the number of rotations required to maintain the balance of the tree. The total cost of insertion in an AVL tree is O(log n) in the average case and O(n) in the worst case.
* In a Splay tree, the cost of insertion is determined by the number of comparisons required to find the appropriate position for the new node and the number of splays required to move the recently inserted node to the top of the tree. The total cost of insertion in a Splay tree is O(log n) in the average case and O(n) in the worst case.

Deletion:

* In an AVL tree, the cost of deletion is determined by the number of comparisons required to find the node to be deleted and the number of rotations required to maintain the balance of the tree after the deletion. The total cost of deletion in an AVL tree is O(log n) in the average case and O(n) in the worst case.
* In a Splay tree, the cost of deletion is determined by the number of comparisons required to find the node to be deleted and the number of splays required to move the parent of the deleted node to the top of the tree. The total cost of deletion in a Splay tree is O(log n) in the average case and O(n) in the worst case.

Search:

* In an AVL tree, the cost of search is determined by the number of comparisons required to find the node. The time complexity of search in an AVL tree is O(log n) in the average case and O(n) in the worst case.
* In a Splay tree, the cost of search is determined by the number of comparisons required to find the node and the number of splays required to move the searched node to the top of the tree. The time complexity of search in a Splay tree is O(log n) in the average case and O(n) in the worst case.

It is important to note that the above cost estimates are for the average and worst cases, and the actual cost may vary depending on the specific implementation and the input data.