|  | | Input Datafile | | | |
| --- | --- | --- | --- | --- | --- |
| in.50.50 | in.50.1000 | in.1000.1000 | in.1000.10000 |
| **Executable** | TestappQF | 0.001031s | 0.004571s | 0.013017s | 0.115125s |
| TestappQU | 0.00074s | 0.00492s | 0.01898s | 0.083871s |
| TestappQUPCR | 0.001029 | 0.004376s | 0.015348s | 0.043186s |

Time includes reading the file, performing the operations, and dislaying the data

Restrictions:

1. Of the operations in the data file, approximately 2/3 are find operations.

Observations:

TestappQF

* between 50 and 1000 operations, an increase of 20 times the operations, there is an increase in time by a multiple of 4.4.
* between 1000 and 10000 operations, an increase of 10 times the operations, there is an increase in time by a multiple of 8.8
* between 50 and 1000 operands completing 1000 operations, an increase of 20 times the operands, there is an increase in time by a multiple of 2.8

TestappQU

* between 50 and 1000 operations, an increase of 20 times the operations, there is an increase in time by a multiple of 6.6.
* between 1000 and 10000 operations, an increase of 10 times the operations, there is an increase in time by a multiple of 4.4
* between 50 and 1000 operands completing 1000 operations, an increase of 20 times the operands, there is an increase in time by a multiple of 3.8

TestappQUPCR

* between 50 and 1000 operations, an increase of 20 times the operations, there is an increase in time by a multiple of 4.2.
* between 1000 and 10000 operations, an increase of 10 times the operations, there is an increase in time by a multiple of 2.8
* between 50 and 1000 operands completing 1000 operations, an increase of 20 times the operands, there is an increase in time by a multiple of 3.5

|  |  |  |  |
| --- | --- | --- | --- |
|  | 50 – 1000 operations | 1000 – 10000 operations | 50 – 1000 operands |
| TestappQF | 4.4 | 6.6 | 4.2 |
| TestappQU | 8.8 | 4.4 | 2.8 |
| TestappQUPCR | 2.8 | 3.8 | 3.5 |

Analysis of Algorithms

The initialization of the arrays containing the data has to cycle through each element and set the value of the element. If we have n elements, then the initialization time is O(n). Since the initialization in all three programs is the same, the time should be the same, with the exception of TestappQUPCR which has to initialize the Rank array as well. Since the element and rank are initialized through the single sweep of n elements, this time differenct will be a constant and the time will be C \* n; therefore, it is still O(n).

For all three programs, the printing of the array has to cycle through each element in the array and print the element to the screen; therefore, the print function is O(n).

TestappQF

1. The Find algorithm compares the root of one number to the root of another, and therefore is dependant on the Root algorithm; therefore it would take O(h1) + O(h2).
2. The Root algorithm checks if the data in the element is equal to the id of the element. If not, it recursively calls the Root of the data, until the data matches the id. This algorithm would have to cycle through each node until it hit the root. Letting h be the height of the tree, the algorithm's time would be O(h).
3. The Union algorithm cycles through all the elements in the array; hence O(n), and compares each elements parent against the parent of the first arguement which would take constant time. If the comparison is true then the parent of the element is assigned the parent of the second arguement, which is constant time. Therefore the Union algorithm worst case is O(n).

TestappQU

1. The Find algorithm compares the root of one number to the root of another, and therefore is dependant on the Root algorithm; therefore it would take O(1) + O(1), or 2\*O(1).
2. The Root algorithm checks if the parent in the element is equal to the item of the element. If not, it assigns the parent's item to the item. This algorithm, at worst case, does one comparison and one assignment, which would take a constant time; therefore O(1).
3. The Union algorithm finds the root of each arguement or 2\*O(1) and makes the second arguement's root the parent of the first arguement's root, which takes constant time or O(1). Therefore, 2\*O(1) + O(1)=3\*O(1).

TestappQUPCR

1. The Find algorithm compares the root of one number to the root of another, and therefore is dependant on the Root algorithm; therefore it would take O(h1) + O(h2).
2. The Root algorithm checks if the parent in the element is equal to the item of the element. If not, it recursively assigns the Root of the data, until the data matches the id. Letting h be the height of the tree, the algorithm's time would be O(h). Then, the algorithm checks if the item is equal to its' parent. If it does, then a rank of 0 is set; if not, a rank of 1 is set. This would be done in constant time or O(1), therefore O(h)+C\*h.
3. The Union algorithm checks the Find of the two elements [(O(h1)+(C\*(h1)) + O(h2)+(C\*(h1))] and if they are not equal, compares the rank of each element. If the rank of element 1 is less than element 2, the parent of element 1 is assigned the root of element 2, which is O(h)+C\*h. If the rank of element 1 is equal to or greater than the rank of element 2, the parent of element 2 is assigned the root of element 1, again O(h)+C\*h. Therefore, the time for the union is [(O(h1)+(C\*(h1)) + O(h2)+(C\*(h1))]+ O(h)+C\*h which is O(h); but the rank (height) of any element is never more than one; therefore the time is O(1).