**Assignment 1**

**Q1**.   We discussed two versions of the 3-sum problem: A "naive" implementation O(n3) and a "sophisticated" implementation O(n2log n). Implement these algorithms.  **Your implementation should be able to read data in from regular data/text file with each entry on a separate line.**Using Data provided in a .txt file, determine the run time cost of your implementations as function of input data size.  Plot and analyze (discuss) your data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File Name**  **(.txt)** | **Brute Force Algorithm** | | **Sophisticated Algorithm** | |
| **Triplets** | **Time(s)** | **Triplets** | **Time(s)** |
| **8int** | 0 | 0.006 | 0 | 0.002 |
| **32int** | 9 | 0.015 | 9 | 0.005 |
| **128int** | 509 | 0.906 | 509 | 0.046 |
| **512int** | 33151 | 44.015 | 33151 | 0.855 |
| **1024int** | 265426 | 348.304 | 265426 | 3.13 |
| **4096int** | Unobservable | Unobservable | 17135744 | 61.049 |
| **4192int** | Unobservable | Unobservable | 18693542 | 70.715 |
| **8192int** | Unobservable | Unobservable | 137073412 | 295.38 |

Figure 1: Plot of Execution times of the 3-sum algorithms

(X-axis => Input file size, Y-axis => Time in seconds)

As we can see from the plot, the time taken by the ‘Brute Force Algorithm’ to execute increases exponentially. It tends to infinity in a sense that it becomes impossible to observe the results for input sizes above 1024 in a reasonable amount of time. As it uses three nested for loops, one going up to (n-1), the other up to (n-2) and the third one up to (n-3), the complexity becomes O(n3).

On the other hand, the sophisticated algorithm gives much faster results. In the sophisticated algorithm, the sorting takes up O(log n) and the searching takes up O(n2log n), the total complexity becomes O(n2log n).

Q2. We discussed the Union-Find algorithm in class. Implement the three versions: (i) Quick Find, (ii) Quick Union, and (iii) Quick Union with Weight Balancing. Using Data provided in a .txt file determine the run time cost of your implementation (as a function of input data size). Plot and analyze your data. Note:  The maximum value of a point label is 8192 for all the different input data set. This implies there could in principle be approximately 8192 x 8192 connections.  Each line of the input data set contains an integer pair (p, q) which implies that p is connected to q.  Recall: UF algorithm should

// read in a sequence of pairs of integers (each in the range 1 to N) where N=8192

// calling find() for each pair: If the members of the pair are not already connected

// call union() and print the pair.

The data and plot suggest that there is a bit improvement in time as the data size increases when we compare the three algorithms against each other. We also have to observe here that for small data sizes, the Quick Find algorithm performs better than Quick Union. But as the data size increases further, this trend reverses. This is because of the constant factor, or ‘pre-factor’ associated with the complexity term. The Complexity of Quick Find algorithm is O(n), that of Quick Union is O(n) and that of weighted is O(log2n)

A.C. = Number of Already Connected pairs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name**  **(.txt)** | **Quick Find** | | **Quick Union** | | **Weighted Quick Union** | |
| **A.C.** | **Time(s)** | **A.C.** | **Time(s)** | **A.C.** | **Time(s)** |
| **8pair** | 0 | 0.099 | 0 | 0.05 | 0 | 0.032 |
| **32pair** | 0 | 0.19 | 0 | 0.189 | 0 | 0.078 |
| **128pair** | 0 | 0.567 | 0 | 0.66 | 0 | 0.548 |
| **512pair** | 0 | 2.477 | 0 | 2.509 | 0 | 2.54 |
| **1024pair** | 0 | 5.567 | 0 | 4.925 | 0 | 4.607 |
| **4096pair** | 4 | 21.763 | 4 | 20.503 | 4 | 17.852 |
| **8192pair** | 1314 | 35.933 | 1314 | 34.831 | 1314 | 31.077 |

Figure 1: Plot of Execution times of the Union-Find algorithms

(X-axis => Input file size, Y-axis => Time in seconds)

Q3. Recall the definition of "Big Oh" (where F(N) is said to be in O(g(N), when F(N) < c (g(N)), for N > Nc). Estimate the value of Nc for both Q1 and Q2. More important than the specific value, is the process and reasoning your employ.

Three-Sum Algorithm: -

1. Brute Force Algorithm has three for loops, one running for (n), other for (n-1) and the last one for (n-2). So, the time taken is like (n)(n-1)(n-2) = n3 - 3n2 + 2n. But, number of combinations of selecting 3 numbers out of n is nC3 = (n3 - 3n2 + 2n)/ 6. Thus, the complexity is O(n3).

At n=Nc, the graphs of these two equations will intersect.

Equating the two equations, (f(n) = c\*g(n))

(n3 - 3n2 + 2n)/6= n3

Solving the equation, we get the value of Nc = 2/5 (Assuming C=1)

Since the data size is integral, we consider the nearest integer. So, Nc = 1.

1. Sophisticated Algorithm has two for loops, one running for (n) and the other for (n-1). We select two values from n using these loops. So it is nC2 = n(n-1)/2. We deploy the binary search (having worst case complexity as log2n). There is a quicksort algorithm that runs before the for loops and it has the worst case complexity as n2.

So the total becomes = n2 + (n(n-1)/2)log2n. And the total complexity is n2logen

Equating the two equations, (f(n) = c\*g(n))

n2 + (n(n-1)/2)log2n = n2logen

Solving the equation, we get the value of Nc = 0.398. (Assuming C=1)

Since the data size is integral, we consider the nearest integer. So, Nc = 1.

Union-Find Algorithm: -

1. Quick find algorithm requires n array accesses to initialize, 2 accesses for checking if there is already a connection and (2n+1) accesses to make a union. So, the total accesses in worst case will be n+2+(2n+1) and the complexity is O(n).

Equating the two equations, (f(n) = c\*g(n))

n+2+2n+1 = n

1. Quick Union algorithm takes n array access to get initialized, 2n-2 to see if there is a connection, n-1 accesses to find the root in worst case, and 2n-1 accesses to make a connection. So, the total complexity is O(n).

Equating the two equations, (f(n) = c\*g(n))

n+(2n-2)+(2n-1) = n

i.e. 5n-3 = n

i.e. n = 3/4

Since the data size is integral, we consider the nearest integer. So, Nc = 1.

1. Weighted Quick Union takes 2n array accesses to get initialized, 2log2n to check a connection and 2+2log2n+1 for making a connection. So, total will be 2n+(2log2n)+(3+2log2n).