Como vimos na primeira aula de ED2, “quick find” é lento para problemas enormes e devemos utilizar uma outra alternativa de algoritmo , como por exemplo a “quick union” que seria uma forma de algoritmo preguiçoso, onde se utiliza da mesma estrutura de dados/id de array com tamanho N, mas com uma diferente interpretação, como se fosse um conjunto de árvores em que cada entrada do array iria conter uma referência ao seu pai na árvores, sendo assim, cada entrada de um array iria está associada a uma raiz e essa será a raiz de sua árvore e os elementos que estão sozinhos conectados nos próprios componentes apenas irão apontar para si mesmo, assim poderíamos calcular as raízes e implementar as operações de busca, apenas verificando se eles estão conectados,onde tem as mesmas raízes.

Com a operação de union, iria ser fácil mesclar componentes contendo dois itens diferentes apenas definindo o id da raiz “p” para o id da raiz “q”, fazendo a árvore p apontar para q, isso torna o algoritmo de quick union mais rápido que o “quick find”, entretanto esse método também pode ser lento, pois quando a árvore fica muito alta, a operação para busca se torna muito cara, por exemplo, se for necessário buscar dois ou mais objetos e estes objetos se encontram nas extremidades da árvore, seria necessário percorrer uma extremidade e em seguida, percorrer todo o caminho até encontrar o outro objeto, envolvendo N acessos no arrays.

Tomando como base o “quick find” e o “quick union” podemos melhorar fazendo o chamado “weighting”, a ideia de implementação deste algoritmo se dá ao modo de evitar as árvores altas, por exemplo, se temos uma uma árvore pequena para combinar com uma árvore grande, teremos que tentar evitar colocar a árvore grande na parte mais baixa da associação, controlando o número de objetos contidas em cada árvore, sempre mantendo o equilíbrio e ligando a raiz da árvore menor à raiz da árvore maior.

No weighting, podemos fazer uma compressão de caminho e deixar as árvores quase completamente planas, usando uma variante simples, fazendo todos os nós visitados apontar para o seu avô, onde o tempo no pior caso seria de N + M lg∗ N, muito menor que no caso anterior de o quick union, que seria no pior caso, de N acessos/tempo.

Uma representação do algoritmo e tempo/acessos associados a ele:

Algoritmo	Tempo de pior caso
quick-find	MN
quick-union (QU)	MN
weighted QU	$N + M \lg N$
QU + *path compression*	$N + M \lg N$
weighted QU + *path compression*	$N + M \lg^* N$