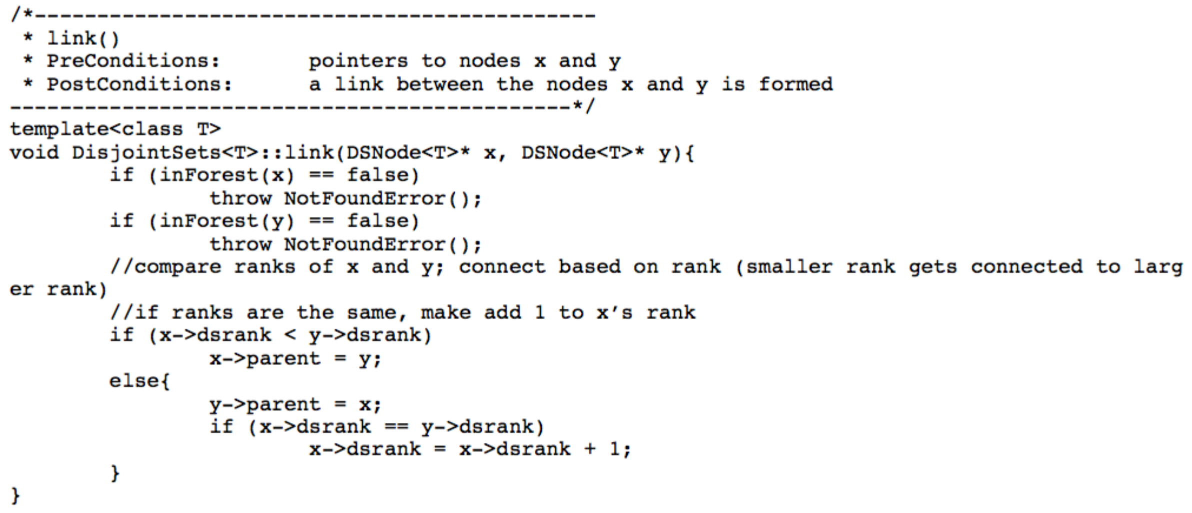
Analysis of Algorithms

Proficiency: Mastery

It is an important skill to be able to examine an algorithm and be able to tell the runtime of said algorithm. This is a skill that we've been working on throughout the year. To determine run time an algorithm you ignore any operation that takes constant time and only focus on the raw components of the algorithm. This way of analyzing run time focuses more on the algorithm and less on the speed of the computer it is being run on. To find the asymptotic runtime of an algorithm we examine the algorithm when the size of the problem goes to infinity. This is known as Big O notation. Once we find the asymptotic runtime of an algorithm we can prove its correctness using induction. Like any induction proof, we solve the base case of the algorithm, with its input n is minimum. Then we do the induction hypothesis where we assume that for arbitrary n, $T(n) \leq n$. Lastly we prove $T(n+1) \leq n+1$. Using this format we are able to prove the runtime of algorithms. We can prove loops in a similar fashion.

A loop invariant is a property of a loop that holds true before and after each iteration of the loop. To prove a loop invariant we first show that the invariant holds prior to the first iteration. Then we assume the invariant holds after some iteration k of the loop. If we show that the invariant holds after the k+1 iteration of the loop we have proved the loop invariant using induction. While examining and proving algorithms is easy to do in practice, messy, complicated code is a reality we all have dealt with. To help make code more understandable it is important to comment with proper pre and post conditions.

A precondition is a condition that must be true of the parameters of a function **before** running the code in the function. A post condition is a condition that is true **after** the code has been run.



```
/*--------------------------------------------
 * link()
 * PreConditions:       pointers to nodes x and y
 * PostConditions:      a link between the nodes x and y is formed
--------------------------------------------*/
template<class T>
void DisjointSets<T>::link(DSNode<T>* x, DSNode<T>* y){
        if (inForest(x) == false)
                throw NotFoundError();
        if (inForest(y) == false)
                throw NotFoundError();
        //compare ranks of x and y; connect based on rank (smaller rank gets connected to larg
er rank)
        //if ranks are the same, make add 1 to x's rank
        if (x->dsrank < y->dsrank)
                x->parent = y;
        else{
                y->parent = x;
                if (x->dsrank == y->dsrank)
                        x->dsrank = x->dsrank + 1;
        }
}
```

An example of proper pre and post conditions is above. As long as the link function gets pointers to nodes x and y (precondition) the post condition will always be true. In every project completed this year, every function was properly commented with pre and post conditions. This allows editors of the code to understand what we, the programmers, were thinking as we were coding. Attached are examples of proving a loop invariant and proving time complexity.

1-29-16 Proving loop invarient

Proof { Initialization (Base case)
Maintenance (Induction hypothesis & step) }

State termination condition

Insertion Sort:

While loop: Before each iteration of the while loop,
$A[i+1..j] \geq key$ & $A[i+2..j] \Leftarrow A[i+1..j-1]$

for loop: Before iteration j of the for loop, $A[0..j-1]$
contains sorted elements originally in $A[0..j-1]$

Initialization - Prove that before the first iteration of
the while loop, $A[j-1+1..j] \geq key$ and
$A[j-1+2..j] = A[j+1..j] \Leftarrow A[j-1+1..j-1] =$
$A[j..j-1]$

The first part is true because $A[j] = key$.
The second part is vacuosly true.

Maintenance - Assume the loop invarient is true
before some iteration of the while loop.
that is, that $A[i+1..j] \geq key$ & $A[i+2..j] \Leftarrow A[i+1..j-1]$

Since we are starting an iteration of this loop,
we know that $A[i] > key$ is true
Combing this w/ our assumption, we know
$A[i..j] \geq key$
Also, since $A[i+1] = A[i]$, by combining this with the
2nd part of our assumption, we know that $A[i+1..j] \leftarrow A[i..j-$

Before iteration j of the for loop, A[0..j-1] contains the elements originally in A[0..j-1] in sorted order.

Proof:

Initialization:

Before the first iteration A[0..1-1] contains the elements originally in A[0..1-1] in sorted order. True

Maintenance:

Assume loop invariant is true before some iteration j, that is A[0..j-1] contains the elements originally in A[0..j-1] in sorted order. In particular A[0...i] & A[i+1..j] is sorted. By the loop invariant of the while loop, $A[i+2...j] \leq A[i+1...j-1]$; Therefor A[i+2..j] is sorted. Also by the inner loop invariant, $key \leq A[i+2]$

After the while loop, either:

(a) $A[i] \leq key$, since $A[i+1] = key$ & $A[i] \leq key \leq A[i+2]$, A[0..j] is sorted

(b) $i = -1$, It must be that $A[0..j] > key$ Since $A[0] = key$, we know that A[0..j] is sorted

Before the next iteration, j is incremented. Therefore A[0..j-1] is sorted

Termination: A[0...n-1] contains . . . . . in sorted order

Prove that $T(n) = an + cn\log_2 n$

<u>Proof By Induction</u>

Base: let $n=1$

$T(1) = a = a(1) + c(1)\log_2(1) = a(1) + 0 = a$

Induc. Hyp: Assume that $T(k) = ak + ck\log_2 k$ for all $k = 1, 2 \ldots n-1$

I.S. $T(n) = 2T(\frac{n}{2}) + cn$

$= 2[a(\frac{n}{2}) + c(\frac{n}{2})\log_2(\frac{n}{2})] + cn$

$= an + cn\log_2(\frac{n}{2}) + cn$

$= an + cn(\log_2(n) - \log_2(2)) + cn$

$= an + cn\log_2(n) - cn + cn$

$\star = an + cn\log_2(n) \star$