Time Complexity of Data Structures

A data structure is a systematic way of organizing and accessing data.

We tend to look at the worst case run time, instead of the average or best, as that will give a better idea of the O(n).

**Primitive Operations Counting example**  
2: Assignment and array access = 2

3:Loop for n times, start at 1 = 3(n-1)… the 3 is (1) comparing i<n; (2) incrementing I value … = I +1 counts as 1; (3) assignming I value the value i=… [i++ is 2 vaues]. In addition int I = 1, counts as a constant 2 as creating of I, and assigning I a value. So 2+3(n-1)

4: Access array + comparing values = 2 … \*(n-1) of the loop as it runs that many times

5: accessing array and assigning value = 2 \* (n-1) of the loop as it runs that many times

7: return value = 1

End Result: 7n-2

**Primitive Operations II example.**

2: 1 assign  
3: 1 assign  
4: 1logn assign

{(Between the while loop, we are doing a c\*logn calc)  
5: 3 (2 calculations and 1 assigning) (logn as in while)  
6: 2 getArrValue and compare (logn as in while)  
7: 2 assignAndCalc (logn as in while)  
8: 2 assign and getArrValue (logn as in while)  
9: 1 return (no logn as is return, only runs once)  
11: 2 calcAndAssign (logn as in while)  
}(Between the while loop, we are doing a c\*logn calc)  
13 1 return

12logn + 4

**Big O Notation**

For function f(n) and g(n), we say f(n) is O(g(n)), you need a constant c>0; n >=n0 [n0 is a determined value]; such that   
eg 8n-2 <= cn [f(n) <= c g(n)]  
f(n) is **big Oh** of g(n) or f(n) is **order** of g(n)

You can see here, if c=8, then n>=1 works.

Eg  
1) 2n+10 is O(n)  
2n+10 <= cn  
(c-2)n >= 10  
n >= 10/(c-2)  
Therefore let c=3 and n0=10

Eg  
2) n^2 is not O(n)  
n^2 <= cn  
n <= c  
There is no c or n= which will satisfy this.

Eg  
4) 3n^3 + 20n^2 + 5 is O(n^3)  
3n^3 + 20n^2 + 5 <= cn^3  
20n^2+5 >= cn^3-3n^3  
5(4n^2+1) >= n^3(c-3)  
5(4n^2+1) /n^3 >= (c-3)  
------  
Instead of doing the above calc: let 3n^3+ 20n^3+5n^3 <cn^3… such that add to c=28, then if n0>=1 this holds.

Let C=28 and n0=1

Eg  
5) 3logn + 5 is O(logn)  
  
3logn + 5 <= clogn  
let 3logn+5logn <= clogn  
such that c = 8, n0=16… note that in this case n0=1 would result in log(1)=0 which would not work

**Use smallest possible upper bound O(\_\_\_)…???**

Good Runtime Calculation do in slide… Quadratic

**Big-Omega**if c>0, n0>=1 such that f(n) >= c\*g(n)  
This finds out the lower bound of f(n)  
Previously we were finding the upper bound.

**Big-Thata**Upper and lower bound  
such that c1\*g(n) <= f(n) <= c2\*g(n)