January 11th

* To log in from home we need
  + Secure Shell – sftp (secure file transfer protocol)
  + Putty
  + Login into host: jbh3-1.cse.csusb.edu
  + Using user id and password
  + Login on jbh358-0 or jb358-1 …. all to jb358-30 any number works. Allows us to use the compiler g++.
  + Wants us to log in to those machines.
  + $ssh jb358-0 is the command line to log into the machines
  + To transfer the program back to our computer. Use winscp to transfer copy from computer to computer
* Linux commands
  + Cd (change directory)
  + Mkdir (make directory)
  + Cp (copy) old file to newfile name
  + Ls (listing)
  + Cat (file)
  + Mv (copies old file to new file)
  + . (current directory)
  + .. (parent directory)
  + Gedit is only for lab machines
  + VI is a visual editor that will allow us to use in machine and at home
  + To compile and run a program
    - IE file name is expr.cpp
    - To compile ($ ls) to see the file directory current
      * $ g++ expr.cpp
      * If there is no errors you get a.out
      * If you type $ ls again then you see the cpp and the a.out
    - To run the file you type ./a.out
    - After typing the expression you have to press return then control-D to create EoF
    - Then the result should print out of the command prompt
  + I/O Redirections
    - If you create a file for input instead of the keyboard being used
    - $ vi expr.in then you type in your expression (a + b \* c)
    - $ls
      * Expr.cpp, a.out, expr.in
    - $ a.out < expr.in
    - To create the output into a file
      * a.out < expr.in > expr.out
* Big Oh Notation
  + Time Complexity
    - We are interested in the maximum number of times an algorithm "loops" not how much time it takes to run.
      * While loop
      * For loop
      * Go to (don’t use this lol)
      * Calls itself (recursive)
      * Calls another algorithm which calls the original algorithm.
    - Example
      * Alg1:
        + Input n
        + Sum = 0
        + For I = 1 to n

Sum +=I < - C

* + - * + Output sum
        + We looped a N amount of times
        + Time complexity is O(n) ----- O is order of.
        + 2c +cn + c = cn + 3c
      * Alg2:
        + Input n
        + Sum = 0
        + For I = 1 to n

Sum += I

Output sum

* + - * + Loops n times = O(n)
        + The difference between 1 and 2 is that 1 the sum of the algorithm is a constant amount of time.
        + 2c + n(2c)
        + T(n) = 2cn + 2c
      * Alg 3
        + Input n (c)
        + Sum = 0 (c)
        + For I = 1 to n (cn)

Sum += I

* + - * + Output sum (c)
        + Sum = 0 (c)
        + For I = 1 to n (cn)

For j = 1 to n (cn) Total is (cn^2)

Sum+=j

* + - * + Output sum (c)
        + 5c + cn + cn^2
        + You drop 5c because of the constant. Drop the cn because the cn^2 is bigger and dominates.
        + The big Oh is O(n^2)
  + We define O notation
    - T(n) = O(g(n) iff T(n) <= c \* g(n)
      * C >0 && n >= n\_o
      * Log(n) is the fastest possible algorithm
      * The simpler of the algorithm the better the program. Write the programs correct and the most effiecient.
  + Lab 1 Time complexity
    - The number of the characters in the expression is the time for the program to run.
    - N = is the number of operators and operands.
    - We loop n times
    - O(n) is the time complexity of lab 1.
  + Example
    - Input n
    - Sum = 0
    - I =1
    - While I < n
      * Sum += I
      * I = 2 \*I
    - Output sum
    - O(Log (base 2) n)
  + Example
    - Input n
    - Sum = 0
    - I = n
    - While I >= 1
      * ------------
      * I = i/2
    - O(log(n))
  + + -> - are inverses of each other
  + \* -> + are repeated additions
  + / -> \* are inverses of each of other
  + / -> - are repeated subtractions
  + Repeated \* is exponentiation
  + Inverse of exponentiation is log
  + Repeated / is log
* Selection Sort
  + You have a arrary or vector of numbers arr[6] for example. If you have to move numbers so they go in increasing order
  + Simplest algorithm to remember
  + I j -------->
  + - I j --------> they move together thru the arrary
  + For (I= 0; I < n-1; I++)
    - For (j = I+1; j < n; j++)
      * If (A[I] > A[j])
        + Swap(A[I], A[j]);
  + The amount of times the if statement executes is the amount of loops
    - N^2/2 - n/2 = O(n^2)