"Simplest lab yet" - Z

* + Constructors for stack will auto call the constructor and deconstructor of the container class that the stack uses.
  + Bool empty() const { return container.empty();}
  + Unsigned int size() const {return container.size()}
  + Push = return container.push\_back(x);
  + Pop = return container.pop\_back();
  + Top = return container.back();
* End of stacks
* Queue
  + Uses a list as the container. Because list has a push front and a pop back.
  + Front in a queue becomes back in the list.
  + Make our own Queue class and out own Queue\_test.cpp
* seBinary search analysis
  + T(n) = time taken by binSearch() for n elements.
    - T(n) = T(n/2) + c -> Recurrence
    - Then T(n) = T(n/4) + c + c
    - Continues till the end of the function running.
    - Then T(n) = T(n/8) + 3c
    - Then T(n) = T(n/16) + 4c
    - What if n = 16
      * T(1) + 4c = meaning that T(1) is some constant time
      * C + 4c.
      * Log (base 2) n = 4 for n = 16 = 2^4
      * Making the recursion of T(n) = O(log n)
* At some point depending on number of searchs the N^2 + logn for the sort and search of a array would be better then the linear search of O(n)
* However you can get better then n^2 with a merge sort
* Merge sort:
  + Takes a array to sort
  + Splits the array into half, each half is sorted individually
  + Then merged together after the sorting
  + Uses two pointers during the split arrays and you compare each number in those locations and the lower numbers gets merged into the new array and you increment that pointer.
  + During the split you have to sort those halfs before merging back. You will then split the halfs again and run the same merge on the quarters. RECURSION
  + Merge\_sort(vector)
    - If #elemet <= 1 then return
    - Ms(1st half of v)
    - Ms(2nd half of v)
    - Merge (1st and 2nd half.
    - T(n) = 2T(n/2) + Cn