Implement (3) sorting algorithms - quicksort, bubblesort and "roll-your-own".  You will generate 50 random numbers between 1 and 1000, place them in a data structure of your own choosing, implement all 3 sorting algorithms, and then you MUST time/compare them.  Bonus points if your "own" algorithm does better than the other two.

For you to get FULL points on this assignment, you will need to hand in ALL as follows:

* Cover page displaying your answers, the reason for these results written up in a paragraph.
* Code
* Screen CAP of your output.

Remember, NO ZIP files!!

**If you are missing any of the above, you will receive a CHECK MINUS on this assignment.**

Quicksort Algorithm

function quicksort('array')

if length('array') ≤ 1

return 'array' // an array of zero or one elements is already sorted

select and remove a pivot element 'pivot' from 'array' // see 'Choice of pivot' below

create empty lists 'less' and 'greater'

for each 'x' in 'array'

if 'x' ≤ 'pivot' then append 'x' to 'less'

else append 'x' to 'greater'

return concatenate(quicksort('less'), list('pivot'), quicksort('greater')) // two recursive calls

Quicksort is a [divide and conquer algorithm](http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm). Quicksort first divides a large [list](http://en.wikipedia.org/wiki/List_(computing)) into two smaller sub-lists: the low elements and the high elements. Quicksort can then recursively sort the sub-lists.

The steps are:

1. Pick an element, called a **pivot**, from the list.
2. Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the **partition** operation.
3. [Recursively](http://en.wikipedia.org/wiki/Recursion_(computer_science)) apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values.

The [base case](http://en.wikipedia.org/wiki/Recursion_(computer_science)) of the recursion are lists of size zero or one, which never need to be sorted.

int partition( void \*a, int low, int high )

{

int left, right;

void \*pivot\_item;

pivot\_item = a[low];

pivot = left = low;

right = high;

while ( left < right ) {

/\* Move left while item < pivot \*/

while( a[left] <= pivot\_item ) left++;

/\* Move right while item > pivot \*/

while( a[right] > pivot\_item ) right--;

if ( left < right ) SWAP(a,left,right);

}

/\* right is final position for the pivot \*/

a[low] = a[right];

a[right] = pivot\_item;

return right;

def partition(list, start, end):

pivot = list[end]

bottom = start-1

top = end

done = 0

while not done:

while not done: place element...

bottom = bottom+1 # ... move the bottom up.

if bottom == top: # If we hit the top...

done = 1 # ... we are done.

break

if list[bottom] > pivot: # Is the bottom out of place?

list[top] = list[bottom] # Then put it at the top...

break # ... and start searching from the top.

while not done: # Until we find an out of place element...

top = top-1 # ... move the top down.

if top == bottom: # If we hit the bottom...

done = 1 # ... we are done.

break

if list[top] < pivot: # Is the top out of place?

list[bottom] = list[top] # Then put it at the bottom...

break # ...and start searching from the bottom.

list[top] = pivot # Put the pivot in its place.

return top # Return the split point

def quicksort(list, start, end):

if start < end: # If there are two or more elements...

split = partition(list, start, end) # ... partition the sublist...

quicksort(list, start, split-1) # ... and sort both halves.

quicksort(list, split+1, end)

else:

return