Patrick Austin
CS 477
Homework # 4
9/30/2016

1. Attached.

2. Here is the implemented algorithm. To see it tested, consult the submitted program:

```
int numberInRange ( int* data, int arrayMin, int arrayMax, int rangeMin, int rangeMax )
   {

   //if array or range min > max, no processing needed. Return an error value

   if ( rangeMin > rangeMax || arrayMin > arrayMax )
         return -1;

   //create an array of size 0-max value in data array. Uses std::max_element method

   int maxValue = *max_element ( data, data + arrayMax ) ;

   int counts[ maxValue ];

   //zero out the values in the new array

   for ( int i = 0; i <= maxValue; i++ )
         counts[i] = 0;

   //c[i] will contain the number of elements equal to i in data

   for ( int j = 0; j <= arrayMax; j++ )
         counts[data[j]]++;

   //c[i] will contain the number of elements <= to i in data

   for ( int k = 1; k <= maxValue; k++ )
         counts[k] = counts[k] + counts[k-1];

   //the final range will be the number of values <= to rangeMax minus the number of values
   //<= rangeMin - 1

   return counts[rangeMax] - counts[rangeMin - 1];

   }
```

3. Attached.

5. Attached.

II. [a] list = { b, c, d, c, b, a, a, b }

create frequency and distribution lists:

freq = { 2, 3, 2, 1 } dist = { 2, 5, 7, 8 }

list [7] = b, dist = {2,5,7,8}, sorted list = { _ _ _ _ b _ _ _ }

list [6] = a, dist = {2,4,7,8}, sorted list = { _ a _ _ b _ _ _ }

list [5] = a, dist = {1,4,7,8}, sorted list = { a a _ _ b _ _ _ }

list [4] = b, dist = {0,4,7,8}, sorted list = { a a _ b b _ _ _ }

list [3] = c, dist = {0,3,7,8}, sorted list = { a a _ b b _ c _ }

list [2] = d, dist = {0,3,6,8}, sorted list = { a a _ b b _ c d }

list [1] = c, dist = {0,3,6,7}, sorted list = { a a _ b b c c d }

list [0] = b, dist = {0,3,5,7}, sorted list = { a a b b b c c d }

Sort complete. sorted list = { a a b b b c c d }

[b.]

COW	SEA	TAB	BAR
DOG	TEA	BAR	BIG
SEA	MOB	EAR	BOX
RUG	TAB	TAR	COW
ROW	DOG	SEA	DIG
MOB	RUG	TEA	DOG
BOX	DIG	DIG	EAR
TAB →	BIG →	BIG →	FOX
BAR	BAR	MOB	MOB
EAR	EAR	DOG	NOW
TAR	TAR	COW	ROW
DIG	COW	ROW	RUG
BIG	ROW	NOW	SEA
TEA	NOW	BOX	TAB
NOW	BOX	FOX	TAR
FOX	FOX	RUG	TEA
↑	↑	↑	final list.

2. Attached & submitted online.

3. Will prove by induction that a RB tree with n nodes formed by RB-insert with $n > 1$ will have at least 1 red node.

base case: for $n = 2$, the root must be black (property 2) and the child will be red (since if black property 5 would be violated for the root).

So base case is satisfied.

inductive step: I assume a tree of n: $n > 1$ nodes has at least one red node.

I will show a tree of $n+1$: $n > 1$ nodes has at least one red node.

So a tree of n satisfies, and I will show it still satisfies after node $n+1$ is added.
2 cases: node $n+1$ inserted as child of black or child of red.

if child of black, node $n+1$ will be colored red, so at least 1 node will be red.

if child of red, 3 possible cases for RB-insert-fixup:

case 1: node $n+1$ will be red after insert-fixup, so at least 1 red.

case 2: parent of $n+1$ will be red after insert-fixup rotation, so at least 1 red.

case 3: node $n+1$ will be red after insert-fixup rotation, so at least 1 red.

Since this covers all cases of inserting, I have shown a tree of $n+1$: $n > 1$ nodes has at least 1 red node. Inductive step complete.

With base case and inductive step shown, proof is complete.

5. Comparison sorts are $O(n \log n)$ at best - they're out.
Counting sort is $O(n + (n^3 - 1)) = O(n^3)$ in this case, will not work.

Consider radix sort, $O(d(n+k))$ where d = number of digits and k = number of possible values of each digit, ie the number base.

A number n in base 10 has $d = \lceil \log_{10}(n) \rceil$ digits, so number $n^3 - 1$ has $d = \lceil \log_{10}(n^3) \rceil$ digits and radix sort has $O((\lceil \log_n(n^3) \rceil)(n+10))$, so radix sort will not work in linear time.

But: if we convert the numbers to base n, then $d = \log_n(n^3) = 3$.
This conversion happens in $O(n)$.

Then radix sort on the converted numbers would run in $O(3(n+n)) = O(n)$.
$O(n)$ to convert base and $O(n)$ to sort means $O(n)$ overall, so linear time sorting was achieved with radix sort.