Patrick Austin
CS 477
Homework # 3
9/26/2016

1. a. Here is the implemented algorithm:

```
template <typename ItemType>
int findMaxIndex( ItemType* data, int search_start, int search_end )
{

//declare variables

int leftResult;
int rightResult;
int median;

//base case: if considering one element, just return the current index

if ( search_start == search_end )

   return search_start;

//recursive case

else
   {

   //calculate the median

   median = ( search_start + search_end ) / 2;

   //recursive call: get max index from the start to the median

   leftResult = findMaxIndex ( data, search_start, median);

   //recursive call: get max index from median + 1 to the end

   rightResult = findMaxIndex ( data, median + 1, search_end );

   //return the larger of the values

   if ( data[leftResult] >= data[rightResult] )
         return leftResult;

   else
         return rightResult;
   }

}
```

Consult the submitted program to see detailed output of this algorithm on the specified array.

b. My implementation of the algorithm will select the leftmost element in the array if it contains several elements of equal value. This can be seen on the specified array in the submitted program. The program could be trivially modified to select the rightmost element by replacing the line:

if (data[leftResult] >= data[rightResult])

With:

if (data[leftResult] > data[rightResult])

If that change is made, the algorithm will always return the right index instead of the left one in cases of equal values at the indices being compared.

c. Attached.

2. a. Mergesort can be implemented as a stable sorting algorithm, depending on the algorithm used for merge. If values from the left subarray are copied back to the main array if they are <= to the right subarray values then stability will be maintained. If < is used at this point in merge it's possible for it to be unstable.

Assuming <= is used: If two elements are in the same subarray then it's not possible for a later incidence of the same value to "jump" an earlier one, since the elements will go back into the main array in left to right order (possibly with entries from the other subarray spliced in between).

If the two equal elements are in different subarrays, then the <= comparison will make sure that the left one will go in first in case they are compared for entry, maintaining stability.

So in any case, the stability is maintained so long as <= is used in the merge.

b. Quicksort is not a stable sorting algorithm. Consider the attached counterexample.

3. Here is the implemented algorithm:

```
template <typename ItemType>
void negBeforePos ( ItemType* data, int data_start, int data_end )
{

//declare variables

int startIndex = data_start;
int endIndex = data_end;
int swapTemp;

//stop when the two indices cross

while ( startIndex <= endIndex )
   {

   //if the element at the left index is a negative number, its location is fine. Increment left index and move on

   if ( data[startIndex] < 0 )
         startIndex++;

   //otherwise, the value at left index needs to be swapped with the value at right index.
   //Now the value at right index is fine, so decrement right index and move on

   else
         {

         swapTemp = data[startIndex];
         data[startIndex] = data[endIndex];
         data[endIndex] = swapTemp;

         endIndex--;

         }

   }

}
```

Consult the submitted program to see output of this algorithm on the specified array.

4. In the average case for an array of size n, a sequential search will find the element halfway through the array, ie in time proportional to n/2 . A binary search will find the element in time proportional to $\log_2 n$.

Therefore the ratio of n/2 over $\log_2 n$ for n=100,000 will be how many times faster binary search is than sequential search for an input size of 100,000.

100,000 / 2 = 50,000. $\log_2(100{,}000) \approx 16.61$.

50,000 / 16.61 ≈ 3010. Therefore the binary search would be around 3000 times faster for an input size of 100,000.

1. c. $C(n) = C(\frac{n}{2}) + C(\frac{n}{2}) + 1, \quad C(1) = 0$

let $n = 2^k$ for some k.

$$C(n) = C(2^k) = C(2^{k-1}) + C(2^{k-1}) + 1 = 2C(2^{k-1}) + 1$$

$$= 2(2C(2^{k-2}) + 1) + 1 = 2^2 C(2^{k-2}) + 2 + 1$$

$$= 2(2(2C(2^{k-3}) + 1) + 2) + 1 = 2^3 C(2^{k-3}) + 4 + 2 + 1$$

$$= \ldots$$

$$= 2^k C(2^{k-k}) + 2^{k-1} + 2^{k-2} + \ldots + 1$$

$$= 2^k C(2^0) + 2^{k-1} + 2^{k-2} + \ldots + 1$$

$$= 2^k + 2^{k-1} + 2^{k-2} + \ldots + 1 \quad \text{(induction)}$$

$$= 2^k - 1 = n - 1.$$

$n-1$ key comparisons are made for input size n.

2. b. Consider a run of quicksort on this array.

2 5₁ 5₂ 5₃ 1 [4] ← pivot

P₁: [2] 5₁ 5₂ 5₃ 1 [4]

P₁ P₂: [2] [5₁] 5₂ 5₃ 1 [4]

P₁ P₂: [2] [5₁ 5₂] 5₃ 1 [4]

P₁ P₂: [2] [5₁ 5₂ 5₃] 1 [4]

P₁ P₂: [2 1] [5₁ 5₂ 5₃ 4]

P₁ P₂: [2 1] [4] [5₁ 5₂ 5₃]

The relative order of equal elements was not maintained, so quicksort is not stable.