CS2134 HOMEWORK 5*
Due Thursday March 10, 2016 at 11:00 p.m.

Be sure to include your name at the beginning of each file! Assignment 5 include a programming portion and a written part. The programming portion must compile and consist of a single file (hw05.cpp). The written portion should consist of a single file (hw05written) in a .pdf format. Be sure to include your name at the beginning of each file! You must hand in the file via NYU Classes.

Programming Part:

1. Rewrite the recursive part of the merge sort algorithm presented in class to work with any container which has random access iterators, and has an overloaded `operator<` for comparison of items in the container. You will *not* write your own `merge` method. Instead you will use the STL `merge` algorithm.

   Here is the driver for the mergesort algorithm you will write.

```
template <class RandItr>
void mergeSort( RandItr start, RandItr end )
{
     int  sz = end - start;  // or use auto sz = end-start;
     typedef typename iterator_traits< RandItr >::value_type Object; //Xcode
     // typedef  iterator_traits< RandItr >::value_type Object; //Other compilers
     // Don't worry about this line of code

      vector<Object> tmp( sz );

      mergeSort( tmp.begin(), start, end );
}
```

   The STL algorithm `merge` takes five arguments, `first1, last1, first2, last2, result`. The merge algorithm combines "the elements in the sorted ranges [first1,last1) and [first2,last2), into a new range beginning at result with all its elements sorted."[1]

2. Create a functor called `meFirst` which has a private member variable, `me`, of type string. Its constructor will take one argument of type string that it uses to initialize its private private member variable, `me`. The functor's overloaded `operator()` takes two arguments of type `student` and returns a boolean value. It returns `true` if the first students' name is less than the second students' name unless either of the students' name equals the variable `me` then that name is always less than the other name.

   Test your functor using the STL 3 parameter algorithm `sort`, the `student` class we discussed in class and some data you create yourself.[2]

[*] 10% extra credit will be given if you turn this assignment in on Wednesday March 9

[1] The quote is from http://www.cplusplus.com/reference/algorithm/merge/. **Don't forget to copy** the elements back into the original container after calling the merge function. The STL merge function does **NOT** have the same signature as the merge function we discussed in class. You might need to include `#include< algorithm >`

[2] This is a biased sort. One item is always first regardless of the other items.

3. [3] Write an algorithm to do the following: given a `vector` of *boolean* values `(true/false)`, order the container such that the `false` values come before the `true` values. Your algorithm must run in $O(n)$ and use $O(1)$ space.

   You algorithm may not simply count the number of `true` or `false` values and then assign the correct number of `false` and `true` values in the `vector`. You should think of your algorithm as the first step in creating an algorithm that sorts based on `true/false` values, where the items are not simply `true/false` values, but large objects that evaluate to `true/false` by using a functor.

   Hint: Be inspired by one of the sorting algorithms we discussed in class.

4. (Extra Credit)[4] Rather than sorting the numbers in either ascending or descending order, suppose we wanted to sort the an array in a new way. The criteria for this sort is given as: if the index of the i'th element is odd, then that element should be less than it's neighboring elements. If the index of the i'th element is even, it should be greater than it's neighboring elements. Your algorithm must run in $O(n \log n)$ time and can use at most $O(n)$ extra space

   Ex: Consider the array $[5, 9, 8, 2, 3, 4]$. After the sort the array should have: $[5, 2, 4, 3, 9, 8]$

   Explanation: 5 is at index 0, so it must be greater than it's neighbors (2)
   2 is at index 1, so it must be less than it's neighbors (5 & 4)
   4 is at index 2, so it must be greater than it's neighbors (2 & 3).... and so on

   Note: The solution may not be unique.

[3] This problem is from Shahzaib, our TA!

[4] This problem is from Shahzaib, our TA!

Written Part:

1. The C++ STL has many functions and functors. Here is your chance to try some of them. In a program when you use an STL algorithm add #include<algorithm>, and when you use an STL functor add #include<functional>. Fill in the correct code where you see a ****

   ```
   vector<int> A = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   vector<int> B = {1, 2, 1, 2, 1, 2};
   vector<int> C(16);
   ```

   (a) Sort the first 4 items in vector B
   `sort(B.begin( ), ****);` // B now contains 1, 1, 2, 2, 1, 2

   (b) Sort the the vector A in reverse order using the functor greater
   `sort(A.begin( ), ****, ****);` //A now contains 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

   (c) Sort the items in vector B in reverse order
   `sort(B.begin( ), ****, ****);`// B now contains 2, 2, 2, 1, 1, 1

   (d) Merge A and B into C. Note that C must have enough space to store both A and B. Since both A and B are sorted in reverse order, you need to pass in a functor to compare the items in the containers
   `merge(A.begin( ), ****, B.begin( ), ****, C.begin( ), greater<int>());`
   // C now contains 10, 9, 8, 7, 6, 5, 4, 3, 2, 2, 2, 2, 1, 1, 1, 1

2. Is our mergeSort algorithm *stable*[5]?

3. What would be printed out by the following code, if the vector a contained $\{11, 10, 1, 3, 2, 5\}$, and the function printVec printed out contents of a? (Don't worry about the exact format of the output, the point is to show how the contents of the vector is or is not changing)

   ```
   template<class Comparable>
   void insertionSort( vector<Comparable> & a )
   {
        int j;
        for( int p = 1; p < a.size( ); p++ )
        {
             Comparable tmp = a[ p ];
             for( j = p; j > 0 && tmp < a[ j - 1 ]; j-- )
                       a[ j ] = a[ j - 1 ];
             a[ j ] = tmp;
             printVec(a);  // prints the contents of the vector in order
        }
    }
   ```

[5] "Stable sorting algorithms maintain the relative order of records with equal keys (i.e. values). That is, a sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list." from https://en.wikipedia.org/wiki/Category:Stable_sorts

4. What would be printed out by the following code, if the vector `a` contained $\{11, 10, 1, 3, 2, 5\}$, and the function `printVec` printed out contents of `a`? (Don't worry about the exact format of the output, the point is to show how the contents of the vector is or is not changing)

```
template <class Comparable>
void mergeSort( vector<Comparable> & a, vector<Comparable> & tmpArray, int left, int right )
{
    if( left < right )
    {
        int center = ( left + right ) / 2;
        mergeSort( a, tmpArray, left, center );
        mergeSort( a, tmpArray, center + 1, right );
        mymerge( a, tmpArray, left, center + 1, right );
        printVec(a); // prints the contents of the vector in order

    }
}
```

5. What would be printed out by the following code, if the vector `a` contained $\{11, 10, 1, 3, 2, 5\}$, and the function `printVec` printed out contents of `a`? (Don't worry about the exact format of the output, the point is to show how the contents of the vector is or is not changing)

```
void quickSort( vector<int> & a, int low, int high )
{
     if (low < high)
     {
         int mid = ( low + high )/2; // select pivot to be element in middle position
         int pivot = a[ mid ];
         swap( a[high], a[mid] ); // put pivot in a[high]

         // Begin partitioning
         int i, j;
         for( i = low, j = high - 1; ; )
         {
             while (  a[i ] < pivot  )  ++i;
             while(  j > i && pivot < a[j ] ) --j;
             if( i < j )
                   swap( a[ i++ ], a[ j-- ]  );
             else
                   break;
         }
         swap( a[ i ], a[ high  ] );  // Restore pivot
         printVec(a); // prints the contents of the vector in order
         quickSort( a, low, i - 1 );     // Sort small elements
         quickSort( a, i + 1, high );    // Sort large elements
     }
}
```

6. Show all the function calls organized as a recursion tree where you include the contents of the container `a` for:

   (a) mergeSort on input `a = {28, 10, 2, 27, 5, 1}`

   (b) quickSort on input `a =  {28, 10, 2, 27, 5, 1}`

7. When all the items in the vector are in ”almost” sorted order (e.g. `a` contains $\{2, 1, 4, 3, 6, 5, \ldots, n/2, n/2-1, \ldots, n, n-1\}$, what is the *average* running time in Big-Oh notation for:

   (a) `insertionSort`

   (b) `quickSort`

8. For the quickSelect algorithm we discussed in class, if after the partition it turns out that $i + 1 = k$ why does the function not recursively call itself?

9. For the following function:

   (a) what is printed by the following function call: `myRecFunc1(4)`

   (b) what is the running time of `myRecFunc1(n)`

```
void myRecFunc1(int n)
{
    if (n < 1) return;

    cout << n << ", ";
    myRecFunc1(n/2);
    cout << n << ", ";
}
```

10. For the following function:

    (a) what is printed by the following function call: `myRecFunc2(4)`

    (b) what is the running time of `myRecFunc2(n)`

```
void myRecFunc2(int n)
{
    if (n < 1) return;

    cout << n << ", ";
    myRecFunc2(n/2);
    cout << n << ", ";
    myRecFunc2(n/2);
}
```