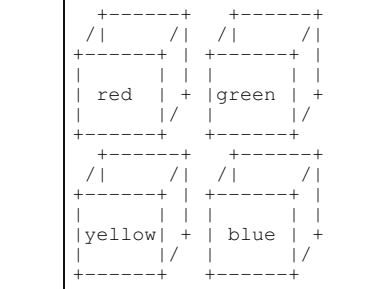
```
**************************************************************************
***                      COMP 15  Homework 5                           ***
**************************************************************************
```




```
        Hmm, difficult. VERY difficult.  Plenty
        of courage, I see.  Not a bad mind, either.
        There's talent, oh yes.  And a thirst to
        prove yourself.  But where to put you?
                                  - Sorting Hat
                                    (J. K. Rowling)
```

```
//////////////////////////////////////////////////////////////////////////
//                                                                      //
//                All Sorts of Sorta Nice Sortin' Algorithms            //
//                                                                      //
//////////////////////////////////////////////////////////////////////////


In this assignment you will implement a sorting program that can use
three different sorting algorithms to sort a list of an integers (but
theoretically it should be able to sort a list of any type).

We'll first describe the program, and then the implementation details.
Please read through the entire assignment before you start.


---------------------
Program specification
---------------------

    Your sorter will order integers based on their values (using simple
    boolean comparisons like ">", and "<").  Your alphabetizer should be
    able to store a list of integers (read from standard input or a file),
    sort them in ascending order, and print them to standard output (cout)
    or save them to a file.

    Your program will run in 2 different modes, the "interactive" mode, in
    which you can manually type in the integers, or in "automated" mode, in
    which it will read a list of integers from a file and print or save
    those integers to file in ascending order.

    Your program will be started from the command line like this:

            ./sorter sortAlg outputMode [fileName]

    "sortAlg" determines which sorting algorithm is used: It can be "-s1"
    to use the first sort, "-s2" to use the second sort, or "-s3" to use
    the third sort.  The "outputMode" determines whether the program will
    print the results to standard output or save the results to a file: It
    can be "--print" to print the results standard output (cout), "--save"
    to the program saves the results to file.  The optional "fileName"
    specifies the file containing the integers to sort (i. e., the input
    file).  The integers in the file are separated by whitespace
    characters. "fileName" is listed in square brackets because it is an
    optional command line parameter: you do not type the square brackets.
    If no input fileName is specified then it will start your program in
    "interactive" mode.
```



```
    If the program is started in "interactive" mode you will read in the
    list of integers from cin.  The program should read in numbers until
    it reaches end of file.  Once it reaches end of file, the program will
    sort the integers using the specified algorithm, and then output the
    sorted list depending on the output mode specified.  For example
    "interactive" mode could take the following input:

            23
            34 2 7
            23
            900 1

    and then sort and print the following to standard output (more on
    output style and formatting in "Output Formatting" under
    "Implementation Details")

            1
            2
            7
            23
            23
            34
            900

    If the user did not specify the correct number of command line
    parameters (2 or 3) or specified invalid command line parameters
    (e.g. -s is not 1-3), print this message on cerr:

            Usage:  sorter sortAlg outputMode [fileName]
                        where:  sortAlg is -s1, -s2, or -s3
                        and     outputMode is --print or --save

    and terminate the program (by returning 1 from main() or by calling
    exit(1)).

    You are not responsible for checking/validating the input.  You can
    assume you will get a sequence of white-space separated integers and
    that the file ends in a new line.  Furthermore, you may assume the
    input consists of non-negative integers of 12 digits or less.

    For this assignment you will use the Standard Template Library (STL)
    vector implementation.  A description of the essential parts of the
    vector interface is included later in its own section "STL Vector
    Interface Essentials" under "Implementation Details" below.
```

```
----------------------
The Files to Implement
----------------------

    For this assignment, you will not be writing any classes.  You will
    write the following three files:

        - sorter.cpp:   This file contains the end user (people) interface
                        implementation with the functionality described in
                        the program specifications above.  It contains your
                        "main", which runs your entire program.

        - sortAlgs.h:   This contains the function declarations (interface)
                        for the sort functions you will have to implement,
                        and that "sorter.cpp" can use to sort the lists of
                        numbers.

        - sortAlgs.cpp: This contains the function definitions
                        (implementation) for the sorts you declared in
                        "sortAlgs.h".

    You will want to write test programs to test your sorting algorithms.

    We'll describe the interfaces of the different functions you will
    declare and define in sortAlgs.h and .cpp respectively first, then give
    some implementation specifics, and finally submission instructions.

    Below are the descriptions for the interfaces of the sort functions
    you have to implement.

    Your sortAlgs.h file must contain the declarations for all three of
    your initial sorting algorithm function calls. The name of the function
    is up to you, however you should give them meaningful names (e.g. the
    initial function call for insertion sort (sort 1) could be
    insertionSort()). Each function takes a reference to a STL vector of
    ints, and returns nothing. You will need to implement one sort from
    each group of sorting algorithms described in the "Sorting Algorithms"
    section under "Implementation Details".

    You may add helper function in addition to the sort function specified
    above, but be sure to declare them in your sortAlgs.cpp and NOT in the
    .h file.  We encourage the use of helper functions that help you
    produce a more modular solution that you and we can better understand.

    In addition to this sorting module you will also write/implement a
    main() function for your final submission (in addition to any testing
    mains you may write).  This main() will be in sorter.cpp.  It should
    handle the checks of the command line arguments and if all the checks
    pass it should read in and store the data from the specified point of
    origin (file or cin) and then call the specified sort function from
    "sortAlgs.h".
```



```
----------------------
Implementation Details
----------------------

    # STL Vector Interface Essentials #

        For this assignment you will have to use the STL vector template
        (which you used in Lab 8).  No other use of the STL is allowed.
        Make sure that you compile your program with the flag
        "-std=c++11".  You will use an STL vector to store the data
        internally as it will help with the random access speed, which is
        important when sorting data.

        The STL vector can be used very similarly to the sequences that you
        have implemented and used.  The functions that we believe are
        essential to complete this assignment are described below.

            - constructor
                o Initializes an empty vector of a specified type.

            - size_t size()
                o Returns the number of elements that the vector contains.
                  size_t is equivalent to the type "unsigned long" on our
                  system.

            - bool empty()
                o Returns "true" if the vector is empty (contains no
                  elements).

            - operator[int i]
                o Allows access to the data at index i. For example we
                  could access the data at index 5  of vector v using v[5].
                  This is the same way you would access data in the static
                  and dynamically allocated arrays of data that you have
                  used before.

            - void push_back(int i)
                o Adds i to the end of the vector.

        For more on using STL templates you can refer to Prelabs 6 & 8 and
        Labs 6 & 8.

        If you want to learn more about the STL vector interface you can
        find more information at:

            http://www.cplusplus.com/reference/vector/vector/
```

```
# Output Formatting #

    For this assignment there will be two specified output formats
    depending on the mode that the Sorter runs in.

    If you are printing the results to standard output (cout) then you
    will print the sorted list in ascending order where each number is
    separated followed by a new line (\n or endl) as shown below:

        1
        2
        3
        4
        5

        NOTE: THERE IS A NEW LINE AFTER THE LAST NUMBER.

    If you are saving the results to a file then you will save them in
    the file the same way as indicated above.  However you must name the
    file using the following style.

        ListOrigin_SortAlg_NumElements_sorted.txt

    Where "ListOrigin" is either the name of the file you read the
    list from, or "cin" if you read it from standard input.  "SortAlg"
    is the sorting algorithm number that you used to sort the list of
    numbers (just the number, not "-s1").  "NumElements" is the number
    of elements you read in and then subsequently sorted.  You'll find
    this naming convention convenient for running tests and making
    sure they all get the same answer!


# Sorting Algorithms #

    You will implement 3 sorting algorithms.  You have some choices of
    sorting algorithms.  Your sort number 1 must come from group 1,
    your sort number 2 must come from group 2, and your sort number 3
    will come from group 3:

        - Group 1:
            o Insertion Sort  --- you must implement this one.

        - Group 2 --- Pick one of:
            o Merge  Sort
            o Quick  Sort
            o Quick3 Sort
            o Shell  Sort
            o Radix  Sort

        - Group 3 --- Pick:
            o Any Group 2 sorting algorithm
            o Any sorting algorithm that
                (1) you find interesting,
                (2) will complete sorting a large list in a reasonable
                    amount of time (say, a 100,000-element list in
                    under 3 minutes).
                (3) not Bogo Sort or Bubble Sort (due to point 2 above)
                (4) not Heap Sort or Selection Sort because we did
                    those for you in lab and class.
                (5) not a modified HW4 Alphabetizer (due to point 2
                    above, and because you already did it).

              We suggest that you do some research on various sorting
              algorithms and find one that you are genuinely interested
              in.  The following link is a great place to start your
              research:

                  http://panthema.net/2013/sound-of-sorting/
```



```
    # Other Implementation Details #

                    *DO NOT IMPLEMENT EVERTHING AT ONCE!*

        The programming for this assignment may not be as much as previous
        assignments, but don't take it for granted.  Some of the
        algorithms can be very tricky to get right --- managing indices
        can be subtle.  Nonetheless, if you break it into pieces, and do
        it over several sittings, it's perfectly manageable.  Just do it
        one bit at a time.  You can probably do the argument processing,
        for example, in one sitting.

        Write a function, then write code in your test file that
        performs one or more tests for that function.  Write a function,
        test a function, write a function, test function, ...

        Follow the same testing approach for every module you write!

        ************************************************************************
        * NOTE:  YOU WILL NOT BE SUBMITTING A TESTING MAIN ALONG WITH      *
        *        YOUR SORTER IMPLEMENTATION.  HOWEVER YOU ARE REQUIRED     *
        *        TO DETAIL YOUR TESTING METHODS IN YOUR README             *
        ************************************************************************

        You may refer to the introduction to file input and output and the
        example program that reads strings from a file provided with HW 3.
        Study the example, but don't use that code exactly: use the
        information and patterns it shows you to write your own code.

        If you need help, TAs will ask about your testing plan and ask to
        see what tests you have written.  They will likely ask you to
        comment out the most recent (failing) tests and ask you to
        demonstrate your previous tests.


------
README
------

    In addition to your code files you will also submit a README file.
    You can format your README however you like.  However it should have
    the following sections:

        A. The title of the homework and the author's name (you)
        B. The purpose of the program
        C. Acknowledgements for any help you received
        D. The files that you provided and a short description of what each
           file is and its purpose
        E. How to compile and run your program
        F. An outline of the data structures and algorithms that you used.
           Given that this is a data structures class, you need to always
           discuss the the data structure that you used and justify why you
           used it. The algorithm overview may not be relevant depending on
           the assignment. For this assignment in addition to describing
           any algorithms that you think are interesting or complicated you
           are required to explain the algorithm of each sorting algorithm
           that you implemented.  Make sure to  discuss its asymptotic
           run time behaviour (Big O).
        G. Details and an explanation of how you tested the various parts
           of your classes and the program as a whole.  You may reference
           the testing files that you submitted to aid in your explanation.

    Each of the sections should be clearly delineated and begin with a
    section heading which describes the content of the section.
```

```
--------------------
Submitting Your Work
--------------------

    Be sure your files have header comments, and that those header comments
    include your name, the assignment, the date, the purpose of the
    particular file, and acknowledgements for any help you received.

    The command is:

         provide comp15 hw5 sorter.cpp sortAlgs.h sortAlgs.cpp README ...

    The ... can include any test files you want to turn in.

-------
Grading
-------

    For this assignment grading will be done differently. Your submissions
    will be graded in-person at a grading appointment that you sign up for
    (after your final submission) with the TAs from your lab section.  If
    you do not sign up for a grading slot over the week following your
    submission, your assignment will not be graded.

    For your in-person grading appointment you will need to be able to
    discuss:
        - The basic algorithm of each sort you implemented (without looking
          at your code).
        - The big-O run time for best, average, and worse case of each
          sort and also what kind of data will result in the best, average
          and worst cases (as applicable).
        - The details of your implementation of each sort with regards to
          its basic algorithm.
        - Whether, using some empirical data (which we provide), whether your
          implementation is behaving as it should
        - Do some basic arithmetic.
```