**Programming Assignment: Template Sorting/Searching Algorithms**

This assignment uses a previous assignment. You do NOT need to start from scratch. In the recursion inClass exercise, you built 2 sorts and one search (you have a previous sort under your belt, too. You did the Bubble Sort last semester). I’ve attached a copy of the exercises to remind you.

You are to take one of the sorting algorithms you built and make it a template algorithm. The sort will be able to sort integers, doubles, strings, even the Card objects you built previously. Be sure to test it with these. To test, you can simply use the array supplied in the inClass program. To build an array with random cards, you can use your deck classes to pick a set of cards. (Note, you can limit the array to few elements).

You are then to make a searching algorithm a template algorithm. I suggest the binary search done in the inClass exercise. However, you can build a new linear search algorithm if you want.

Submit through the group two things:

1. The source code.
2. A screen shot of the output of your test. Remember to take a screen shot of JUST the active window use ALT->PrintScreen.

Good luck!!
**Selection Sort**

There are many kinds of sorting routines. Some are easy to understand and to program, others are very complex. A couple of simple and well known sorts are **bubble sort** and **selection sort**. This section introduces selection sort.

Assume we have an array of n elements and we want to sort in **descending** order. The selection sort algorithm would go like this:

```
Set Current_Element to the start of the array
Repeat until Current_Element at the end of the array
  Linear search from Current_Element to the end of the array for the Largest_Element
  If the Largest_Element is not the Current_Element swap them
  Move Current_Element to next element
```

**Write the selection sort algorithm as a recursive function. A skeleton program is given to build the array. You can find it at the link [here](#).**

---

**Quicksort**

Quicksort is a recursive sort invented by C.A. R. Hoare. Quicksort works by deviding the array into partitions around a pivot element. Then sorting the elements around that pivot:

```
Select a pivot element from the array
Partition or Rearrange the array into sublist 1, the pivot element and sublist 2.
  Partitioning involves rearranging the array, by linearly looking through the array. If an element is less than the pivot it is moved to the left of the pivot, if it is greater than the pivot it is moved to the right of the pivot element.
  Sort the partition of the array before the pivot (sublist 1).
  Sort the partition of the array after the pivot element (sublist 2).
```

**Write the quicksort algorithm as a recursive function. A skeleton program is given to build the array. You can find it at the link [here](#).**
Binary Search

Binary Search is a recursive search on sorted information. The idea of binary search is to cut an array (sorted) into two parts and discard the part which could not contain our searching target.

```c
// Code for BinarySearch
int BinarySearch(IntArrayType IntArray, int Low, int High, int Target)
{
    int Mid, Difference;

    while (Low <= High)
    {
        Mid = (Low + High) / 2;
        Difference = IntArray[Mid] - Target;

        if (Difference == 0)    // IntArray[Mid] == Target
            return Mid;
        else if (Difference < 0)   // IntArray[Mid] < Target
            Low = Mid + 1;
        else
            High = Mid - 1;
    }

    return -1;   // If reach here, Target was not found.
}
```

Write the binary search algorithm as a recursive function. A skeleton program is given to build the array. It also contains the iterative version of the binary search to get you started. You can find it at the link [here](#).