CS 201 Data Structures Library Phase 1 Due 9/20

For phase 1 of the CS201 programming project, we will start with a dynamic array class and extend it to implement some of the algorithms discussed in class.

Your dynamic array class should be called CircularDynamicArray. The CircularDynamicArray class should manage the storage of an array that can grow and shrink. The class should be implemented using templates. As items are added and removed from both the front and the end of the array, the items will always be referenced using indices 0…size-1.

The public methods of your class should include the following (elmtype indicates the type from the template):

Function	Description	Runtime
CircularDynamicArray();	Default Constructor. The array should be of capacity 2.	O(1)
CircularDynamicArray(int s);	For this constructor the array should be of capacity and size s.	O(1)
~CircularDynamicArray();	Destructor for the class.	O(1)
elmtype& operator[](int i);	Traditional [] operator. Should print a message if i is out of bounds and return a reference to value of type elmtype stored in the class for this purpose..	O(1)
void addEnd(elmtype v);	increases the size of the array by 1 and stores v at the end of the array. Should double the capacity when the new element doesn't fit.	O(1) amortized
void addFront(elmtype v);	increases the size of the array by 1 and stores v at the beginning of the array. Should double the capacity when the new element doesn't fit. The new element should be the item returned at index 0.	O(1) amortized
void delEnd();	reduces the size of the array by 1 at the end. Should shrink the capacity when only 25% of the array is in use after the delete.	O(1) amortized
void delFront();	reduces the size of the array by 1 at the beginning of the array. Should shrink the capacity when only 25% of the array is in use after the delete.	O(1) amortized
int length();	returns the size of the array.	O(1)
int capacity();	returns the capacity of the array.	O(1)
int clear();	Frees any space currently used and starts over with an array of size 2.	O(1)
Elmtype QuickSelect(int k);	returns the k^{th} smallest element in the array using the quickselect algorithm.	O(size) expected
Elmtype WCSelect(int k);	returns the k^{th} smallest element in the array using the worst case O(N) time algorithm.	O(size)
void stableSort();	Sorts the values in the array using a comparison based O(N lg N) algorithm. The sort must be stable.	O(size lg size)
void radixSort(int i);	Sorts the values in the array using a radix based sort on the low order i bits of elmtype.	O(i size)

int linearSearch(elmtype e)	Performs a linear search of the array looking for the item e. Returns the index of the item if found or -1 otherwise.	O(size)
int binSearch(elmtype e)	Performs a binary search of the array looking for the item e. Returns the index of the item if found or -1 otherwise. Assumes that the array is in sorted order.	O(lg size)

Your class should include proper memory management, including a destructor, a copy constructor, and a copy assignment operator.