```
void Problem4A(int a[], int arraySize) {

	int temp;
	int n = 6;//figure out why n wont equal properly
	//cout << n << ", ";
	quickSort(a, 0, arraySize -1);


	for (int l = 1; l < arraySize; l = l + 2) {
		if (l + 1 != 6) {
			temp = a[l];
			a[l] = a[l + 1];
			a[l + 1] = temp;
		}
	}

}
```

This code uses quick sort to sort the element in ascending order. After the code is sorted, the array elements are swapped in increments of two starting at the second element. If the array contains an even number of elements, the final value will not be swapped. The code for the swap is shown in the for loop above. The time complexity of the swap is the integer value of n/2. This means that the total time complexity is the sum of the quicksort time complexity + n/2. Since the best-case time complexity of quick sort is O(n*(log(n)). The total time complexity of the code is O(quicksort). The space complexity is also equal to O(quicksort) because the space requirement of the swapping portion of the code is constant.

There is no guaranteed linear time solution to this problem because the array always needs to be sorted to be done in an efficient time. Since the time complexity of the sorting algorithm with the smallest worst-case time complexity is O(n*(log(n)), it is not possible that the solution will always be linear. If the problem is attempted to be solved without sorting the array, you would need to have a for loop that passes through the array from the beginning to the end and increments by 1. If the current value of the array (a[i]) does not satisfy the conditions for a local minima or local maxima a nested loop will need to run to find a value that will satisfy the necessary conditions. This means that the worst time complexity will be $O(n^2)$ if the problem is solved without sorting the array.