4iii.

Code Snippet:

```
void Problem4a(int a[], int arraySize){
   int c = 0;
   int p = 1;


         for (int i = 0; i < arraySize/2; i++)
         {
            swap (a[c],a[p]);
            p = p+2;
         }
}
```

This piece of code is the code who actually takes a sorted array (already sorted from least to greatest due to sorting earlier on in the program), and goes ahead and sorts a array of any length in terms of local maxima and local minima. What I realized, before building my algorithm, that in order to sort a sorted array in terms of local minima and maxima in terms of: a>b<c>d<e>f<g>h is that the following steps need to be executed:

0th elements swapped with the 1st element

0th element swapped with the 3rd element

0th element swapped with the 5th element

Etc….

So the pattern that stood out to me is that the 0th element is ALWAYS swapped with odd numbered elements, in ascending odd values order (what I mean by odd valued order: 1 →3→5→7 etc.).S So being that the element that was being swapped was always the odd numbered element, in ascending order, it allowed me to build my for loop body and initialize the first odd element, then iterate it for every loop by two in order for it to be continuously odd. The only thing that was left for me to figure out was how long the for loop iterated for and I noticed that the iteration amount was always half the value of the passed array length.

4iv.

Being that my algorithm is working with indexed array elements, depending on a inputted array size, there actually does not exist a linear time solution since my algorithm operates under O(1) . this is so because no matter how big the array is, the program will always run the same exact way