```
if(first<last){
  int mid = first + (last-first)/2;

  merger(a,first,mid, arraySize);

  merger(a,mid+1,last, arraySize);


  merge(a,first,mid,last,arraySize);
```

My sort for Problem Four A is of the same complexity as the mergesort, as it uses the same code to accomplish the initial sort. However, it will take longer, as there are

```
for(int i = 0; i<arraySize; i++){

  reversecopy[i] = a[i];

}



int i = 0;
int j = arraySize-1;
while(i < j){

  int temp = reversecopy[i];
  reversecopy[i] = reversecopy[j];
  reversecopy[j] = temp;
  i++;
  j--;

}

cout<<endl;



int temp2;

for(int i = 0; i<arraySize-1; i++){



  temp2 = a[i+1];
  a[i+1] = reversecopy[i];
  reversecopy[i+1] = temp2;

}
```

several more linear run throughs of the array. So although the highest order of complexity is O(Nlog$_2$N), it would actually by closer to O(Nlog$_2$N +3N).

C. There is no true linear solution to this problem, as this kind of sort will always require you to sort the array, and then do some other step to place the mins and maxes next to each other. However, the best case for this would be linear, as if the array were already sorted in this manner, than only one loop through the array would be required to check this and return it.