```cpp
#include <iostream>
#include <QuickSort.h>
using namespace std;
void Max_e_Min(int arr[], int mySize){//Size means the number of elements
    QuickSorting sortThis;
    int finArray[mySize];
    int k=0;
    int i=0;
    int j=(mySize-1);
    int m=mySize/2;//m is the index number of the midpoint
    sortThis.actualSort(arr,i,j);
    while(i<=m){
        if(i!=j){
            finArray[k]=arr[i];
            k++;
            i++;
            finArray[k]=arr[j];
            j--;
            k++;
        }
        else if (i==j){
            finArray[k]=arr[i];
        }
    }
    for(i=0,k=0;i<mySize;i++,k++){
        arr[i]=finArray[k];
    }
    }

int main()
{
int X[]={4,7,5,2,8,4,0,9};
    Max_e_Min(X,8);

    for(int i:X){
        cout<<i<<" ";
    }

    return 0;
}
```

Problem 4 Explanation:

I believe that the code for the problem is not linear due to the sorting method that I used is a normal quicksort. Quick sort itself has a nLog(n) time. This alone disproves the idea that this code will run linearly. Not only that but also depending on the size of the array, being even or off, it may take and additional conditional statement into consideration. That again proves that this code will not run linearly. So I feel like depending on which type of sorting algorithm the user uses, it can change the time factor of this code. As for the complexity of the problem. It was not that hard. If you wrote out the output, the pattern stands out and is easy to see. All that needs to be done is sort the array and then have one point at the beginning and one at the end. Then create another array with the same size and then first put in the lowest value. Then highest. Then second lowest, and alternate until the two pointers cross.