Morgan Baccus
CptS 350
Homework 2'

Problem 1

The babies either have brown or purple hair. We will use 0 to denote a baby having brown hair and 1 to denote a baby having purple hair. Now, we have an array filled only with 0s and 1s.

a) The three constraints on the pointers are: one-pass, in-place, and linear-time

One-pass: This means that we should create an algorithm that will process each elements at most one time. Once an element has been visitied, it should not be again.

In-place: This means that all operations should be done within the array versus using temporary space.

Linear-time: This means that the time complexity of the algorithm should be linear and thus O(n). Big-O notation tells us that for an array with *n* elements, the algorithm should take O(n) time to execute.

b) It is best to first consider the linear-time and then the one-pass constraints. Our last consideration will be one-pass as it is fairly simple to modify code to fit this constraint.

We will be using only two pointers in this algorith. We will call the pointers 'left' and 'right'. We will initialize left with the first position in the array, 0, and initialize right with the last index in the array, n-1 for n number of babies. We will then compare the elements held at each index of the array which can have two possible cases…

Case 1: The element held in *left* is 1, which needs to go after all the 0 elements. Here, we will swap the elements held within *left* and *right*. This means that the value in *right* is a 1 and we can now move *right* one index to the left by decreasing the position by one. Since we still don't know if the value in *left* is a 1 or 0, that pointer will stay at the first element.

Case 2: The element held in *left* is 0 and is sorted how we want. This means that that element can stay where it is and we will simply index *left* one to the right by increasing the position by one.

We will repeat the above steps until *left* becomes bigger than *right* as that will indicate the entire array has been processed. At most, one pointer could traverse the entire array by *left* being indexed all the way from 0 to n-1, or *right* being indexed from n-1 to 0. In either case, one element will be processed each pass through the loop which makes it linear-time and one-pass. Additionally, since we are only using the variables *left* and *right*, the algorithm is in-place.

c) Psuedo-code

```
int left = 0, right = n-1; //set left to the first element and right to the last
while (left < right) //iterate through the array until left and right meet
 {
 if (array[left] == 1) //check if the left array is equal to 1
  {
  swap (array[left], array[right]) // if it is, swap with the element at right
```

```
    --right; // move right to the left by one since we know the last element is a 0 now
    }
    else
    ++left; //if left is a 0, then it is where it should be and we can move left to the right by one
  }
```

Problem 2

We are able to use a similar approach here as we did in problem 1, but now we will use a pointer for each hair color rather than a left and right. Since we want the elements to be sorted by brown, purple, and black, we will begin by moving all the brown elements to the beginning of the array.

```
int brown = 0, purple = 0, black = 0; //set all pointers to 0 so they start at the first element in the array
for (int i = array.size(); brown < array.size() )
//set i equal to the last element in the array and iterate through until brown reaches the end of the
array
  {
  if (array[brown] == 0)
    {
    swap(array[brown], array[i]);
    ++i;
    }
  else
    ++brown;
  }

//repeat with purple and black where array[purple] = 1 and array[black] = 2
```