Gordon Ng HW 2 CS 330

[5,7,2,8,10]

ex: n=5 [2,3,4,5,6]

1)
```
n = length(A);
swaps = 0;
for i=1 to n do
    for j=1 to n-1 do
        if A[j] > A[j+1] then
            A[j] = A[j] + A[j+1];
            A[j+1] = A[j] - A[j+1];
            A[j] = A[j] - A[j+1]
            swaps ++;
Output: swaps
```

1 2 3 4 5

1 2 3 4

A[1] > A[2] [5,3,4,5,6]

[5,2,4,5,6]

[3,2,4,5,6]

3 4 2 5 6

3 4 5 2 6

3 4 5 6 2

a) $O(n^2)$, the first loop runs n times, and the 2nd loop runs $(n-1)$ times. n^2-n; upper bound would be $O(n^2)$

b) This algorithm counts the amount of swaps needed to sort the array in increasing order. Since the array is already sorted it returns 0

2)
```
1  n = length(A)
2  dec = 0;
3  for i=2 to n+1 do
4      j = i-1;
5      while j > 1 and A[j-1] > A[j] do
6          temp = A[j-1]
7          A[j-1] = A[j];
8          A[j] = temp;
9          j--;
10         dec++;
   Output dec
```

a) Line 3 is $O(n)$, line 4 and 5 would be $(n-1)$ run time, so this gives a total of n^2-1, which has an upper bound of $\boxed{O(n^2)}$

b) swap is smaller than dec if it is sorted increasing

swap is equal than dec if half the numbers are both greater / smaller

swap is greater than dec if it is decreasingly ordered.

[1,2,3,4,5] would become [5,4,3,2,1] in decAlg and vice versa.

Swap and dec add to the same sum if elements of A are not changed.

ex: [1,2,3,4,5], [5,2,3,1,4] ... and so on.

2)

```
Function ShortAndAlternatingTest():
    Nodes for Altering = BFS_Alternating(G,V,s,c)
    Nodes = BFS(G,V,s,c)
    n = length(V)
    pathsalt = Empty_Array_length_n
    paths = Empty_Array_length_n
    for i=1 to n do
        pathalt = []
        path = []
        while parentAlt != s do:
            if parentAlt in Nodes for Altering then:
                child = parentAlt;
                parentAlt = NodesforAltering[child]'s parent;
                pathalt.append((parentAlt, child))
            else:
                pathsalt = Emptylist
        while parent != s do:
            if parent in Nodes then:
                child = parent;
                parent = Nodes[child]'s parent;
                path.append((parent, child))
            else:
                paths = EmptyList
        pathalt.reverse()
        path.reverse()
        pathsalt[i] = pathalt
        paths[i] = path
    for i=1 to n do
        if pathsalt[i] != paths[i] then
            Output "None"
    Output pathsalt
```

Computes two arrays with BFS algorithms, finding the alternating paths and the shortest paths, then it loops through the paths arrays to return an array that satisfies both alternating colors and shortest path.

The run time of this algorithm would be $O(m+n^3)$, since BFS runs on an output of n nodes and m edges in $O(m+n)$ time. The looping of both trees will run in $O(n)$ time, so we get $O(n^2)$ for both trees and another $O(n)$ for the last comparison of arrays. So it has a final run time of $O(m+n^3)$