**Devansh Gupta**

**01609278**

**CIS 455 – Homework #3**

**Problem 3-1.** (10 points) Jones & Pevzner, (*modified version* of Problem 5.1, page 143). Suppose you have a maximization algorithm, *A*, that has an approximation ratio of 1/4 When run on some input π, *A*(π) = 12.

1. What can you say about the true (correct) answer OPT = OPT(π)?

a. OPT(π) >= 3;

b. OPT(π) <= 3;

c. OPT(π) >= 12;

d. OPT(π) <= 12;

e. OPT(π) >= 48;

f. OPT(π) <= 48;

Solution:

Maximization algorithms, the approximation ratio is

max A(π)/ OPT(π)

| π|=n

Here A(π)=12 and approximation ratio is 1/4 When run on input π

1/4= max 12/ OPT(π)

| π|=n

OPT(π)=max 12\*4

| π|=n

=max 48

So we can say that OPT(π) <= 48.

2. What if A is a minimization algorithm? (*Hint:* This is a trick question)

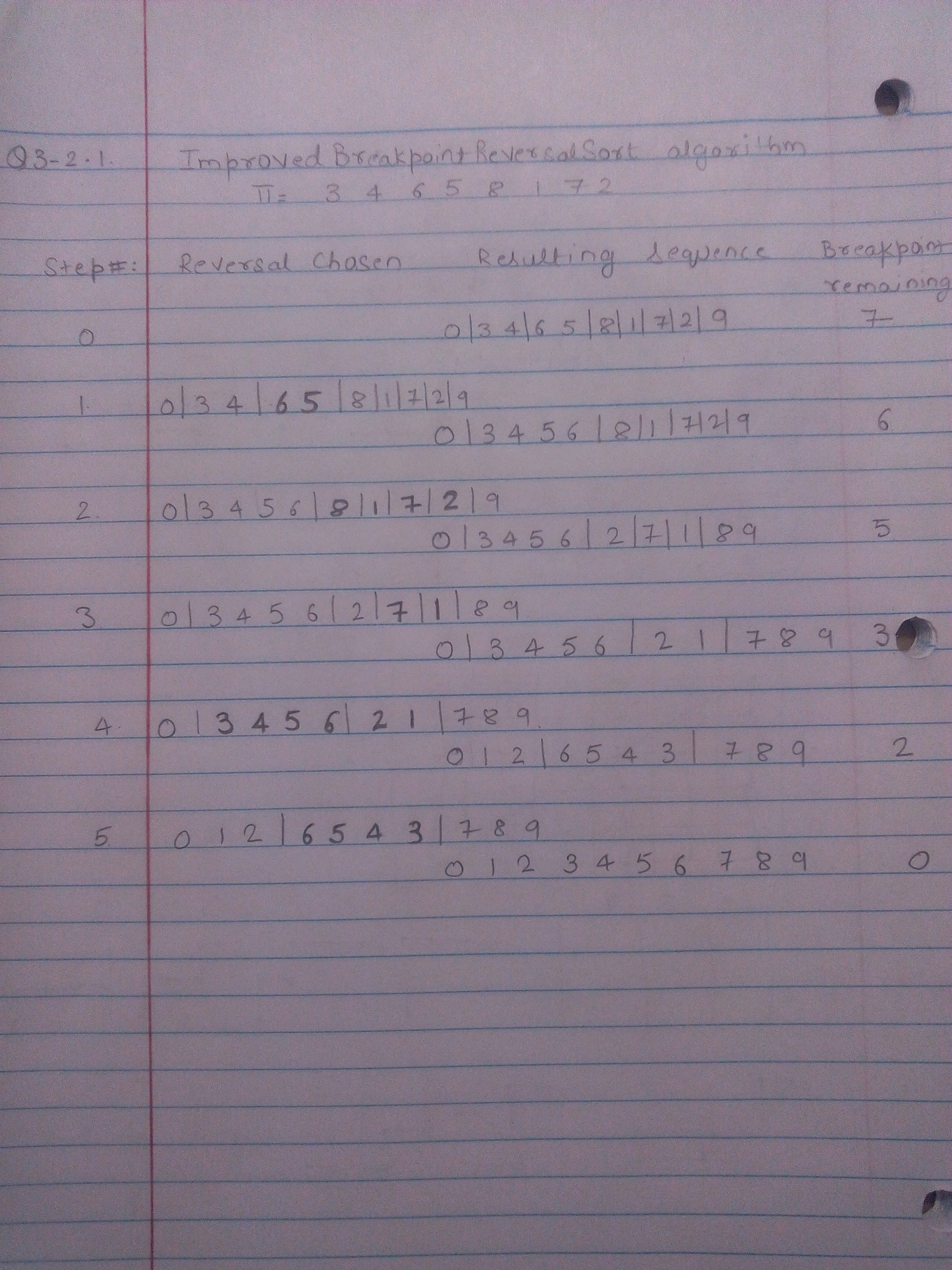
Solution: For a minimization algorithm, the approximation should at least be 1 and for maximization algorithm the ratio should be maximum 1. If ratio for minimization algorithm is less than 1 i.e. 1/4 then it can return a optimal solution.

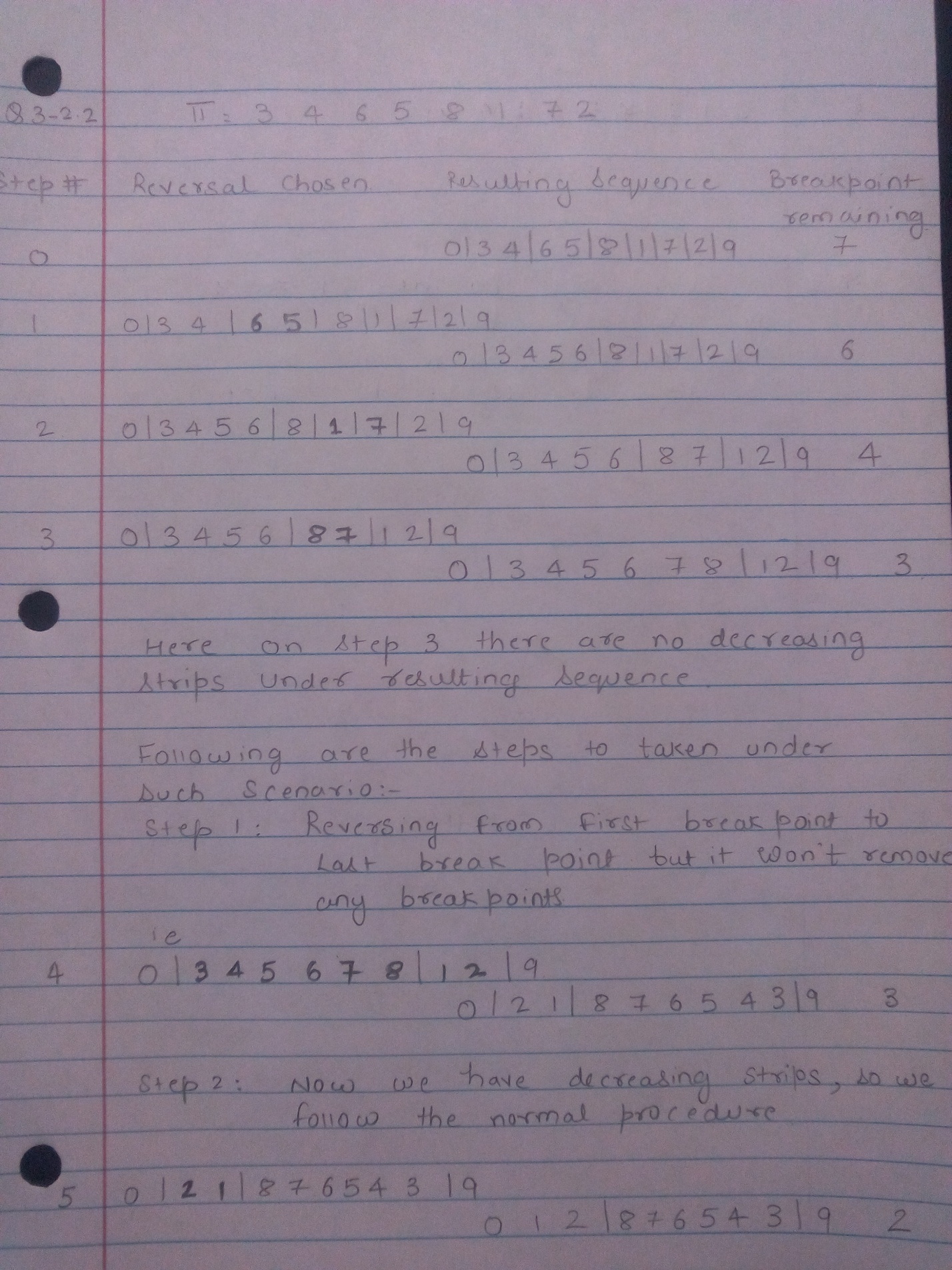
**Problem 3-2.** (28 points) Jones & Pevzner, (*modified version* of Problem 5.4, page 143).

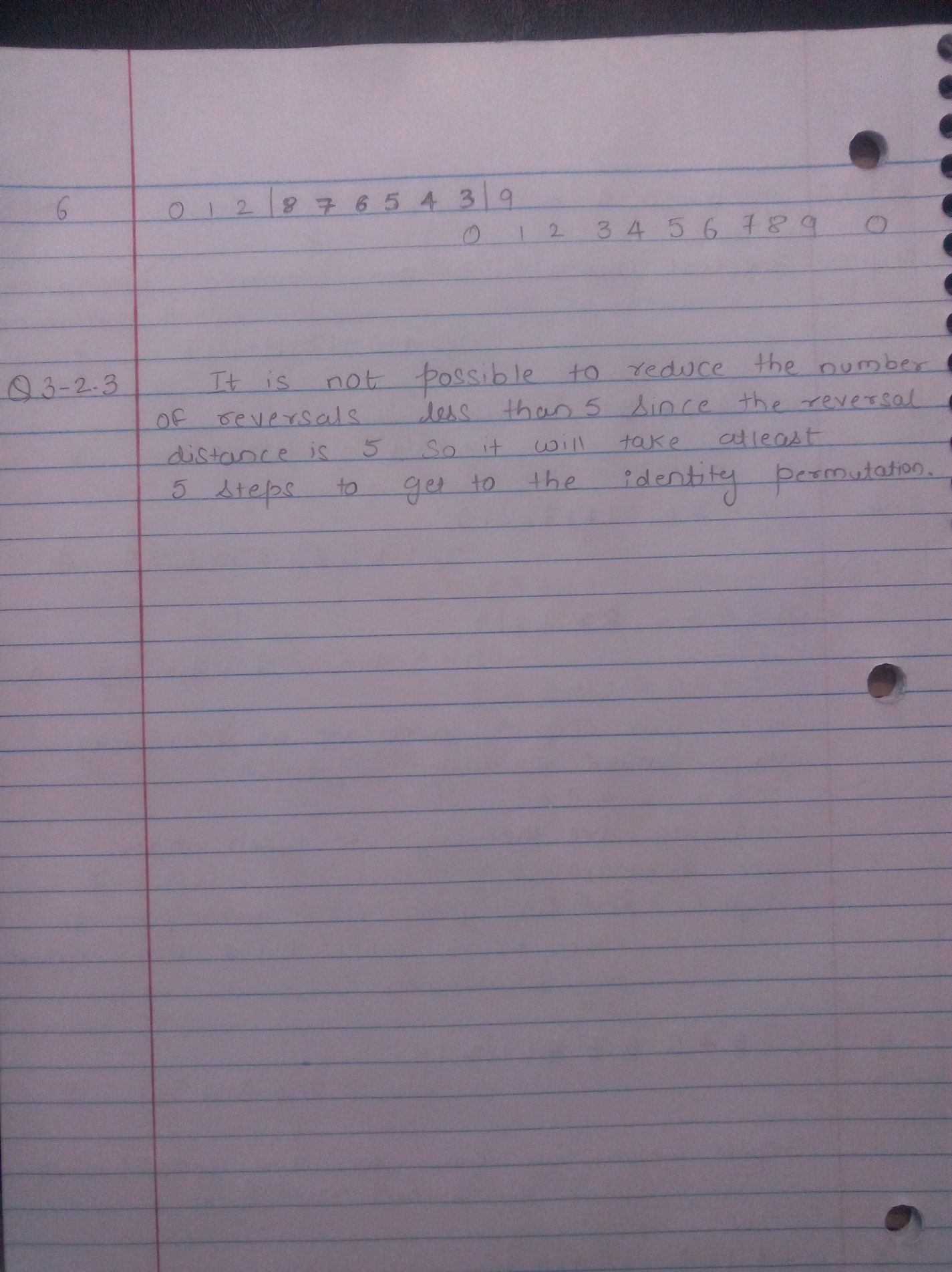
1. Perform the *ImprovedBreakpointReversalSort* algorithm with π = 3 4 6 5 8 1 7 2

(Remember to start with **0** 3 4 6 5 8 1 7 2 **9** and to follow the algorithm below, **particularly line 3**).

Solution:







**Problem 3-3.** (10 points) Jones & Pevzner, Problem 5.5, page 143

Find a permutation with no decreasing strips for which there exists a reversal that reduces the number of breakpoints.

Solution:The example of permutation with no decreasing steps is as follows:

0 1 | 5 6 | 3 7 | 2 | 4

The number of breakpoint in this increasing strip is 4.

While reversing one of the strips (3, 7) we get,

0 1 | 5 6 7 | 3 2 | 4

Here, the permutation consists of a decreasing strip (3, 2). Also the number of breakpoints reduced to 3.

Therefore, there is a permutation with no decreasing strips which there exists a reversal that reduces the number of breakpoints.

**Problem 3-4.** (12 points) Jones & Pevzner, Problem 5.13, page 144 (*read the description between 5.11 and 5.12*).



Given permutations π1 = 124356, π2 = 143256, and π3 = 123465, compute the number of breakpoints between:

(1) π1 and π2

(2) π1 and π3

(3) π2 and π3

(Use the first permutation as the given sorting order. For example, given π3 = 123465, 4 and 6 are adjacent and 4

and 5 are not.)

Solution:

(1) π1 and π2

Considering π1 and π2 will take pair from π1 and compare with π2. In π1, 1 and 2 are adjacent but in π2, it is not adjacent. So there exists a breakpoint. Next in π1, 2 and 4 are not adjacent and neither in π2. So there exists one more breakpoint. Next comparing 4 and 3,it is adjacent in both π1 and π2. So there is no breakpoint. 5 and 6 are adjacent in both π1 and π2,so no breakpoint. Hence there are three breakpoint 12, 23 and 35.

(2) π1 and π3

Consider π1 and π3 will take pair from π1 and compare with π3. In π1 and π2, 1 and 2 are adjacent, so there exists no breakpoint. Next in π1, 2 and 4 are not adjacent and neither in π3. So there exists one more breakpoint. Next comparing 4 and 3,it is adjacent in both π1 and π3. So there is no breakpoint. 3 and 5 are not adjacent in π3so there is one more breakpoint. Hence there are two breakpoint 24 and 25.

(3) π2 and π3

Consider π2 and π3 will take pair from π2 and compare with π3. In π2 1 and 4 are adjacent but not in π3, 1 and 2 are adjacent, so there exists a breakpoint. Next 4 and 3 are adjacent in both. So there exists no breakpoint. Next is 3 and 2,they are adjacent in both π2 and π3. So there is no breakpoint. 2 and 5 are not adjacent in π3 so there is one more breakpoint. Hence there are two breakpoint 14 and 25.

**Problem 3-5.** (10 points) Jones & Pevzner, Problem 6.4, page 212.

Modify DPChange (on page 151) to return not only the smallest number of coins but also the correct combination of coins.

Solution:

DPCHANGE(M, c, d)

1. bestNumCoins0<--0
2. for m<--1 toM
3. bestNumCoinsm<--∞
4. for i<--1 to d
5. if m >=ci
6. if bestNumCoinsm−ci + 1 < bestNumCoinsm
7. bestNumCoinsm<--bestNumCoinsm−ci + 1
8. return bestNumCoinsM

DPChange algorithm can be modified by using one or more variable to store denomination against the pre computed value of each of M. The variable will return the correct combination of coins and also in smallest numbers.

Modified: DPCHANGE(M, c, d)

1. bestNumCoins\_0←0
2. for m ←1 to M
3. bestNumCoins\_m←∞
4. for i←1 to d
5. if m≥ci
6. if bestNumCoinsm−ci+ 1< bestNumCoinsm
7. bestNumCoinsm←bestNumCoins(m−ci)+ 1
8. bestDenomination [A] 🡨 Amount M
9. bestDenomnation [A][B] 🡨 Number of coins of M
10. return best NumCoinsM

This modified algorithm will return correctcombinations of coins and also has the smallest number of coins combination.

**Problem 3-6.** (10 points) Jones & Pevzner, Problem 6.6, page 212.

Find the number of different paths from source (0,0) to sink (n,m) in an n by m rectangular grid.

Solution:

#include<stdio.h>

#include<stdlib.h>

int main()

{

int m,n;

int nos=0;

printf("\nPlease enter the size of matrix:");

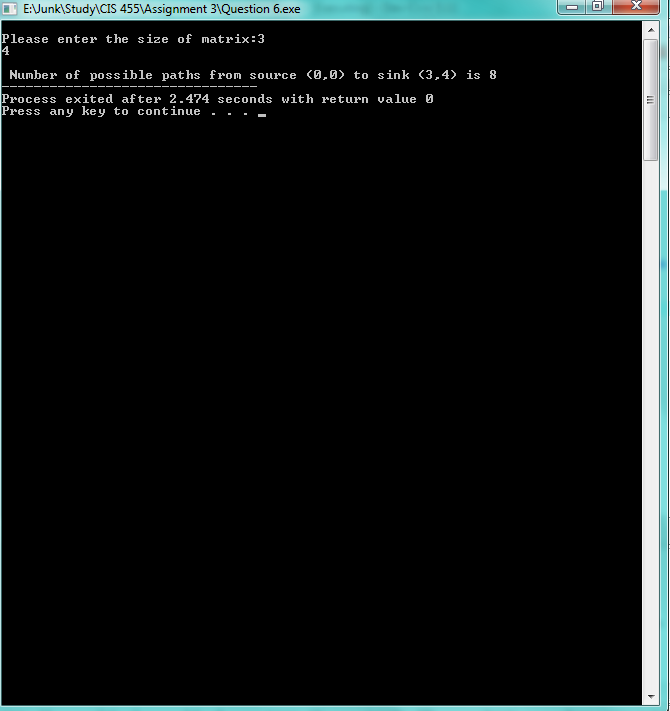
scanf("%d %d",&m,&n);

nos=m\*n;

printf("\n Number of possible paths from source (0,0) to sink (%d,%d) is %d",m,n,nos-4);

return 0;

}



**Problem 3-7**. (20 points) Jones & Pevzner, (modified version of Problem 6.14, page 213).

Two players play the following rock game with two piles of rocks of heights n and m.

At every turn a player must take two rocks from one pile (either the first pile or the second pile) and one rock from the other. The player who cannot complete their turn loses.

1. Who will win? Using dynamic programming, describe the winning strategy for each n and m.

2. Show the details of the winning strategy for n=m=6.

Solution:

Some algorithms break a problem into smaller sub problems and use the solutions of the sub problems to construct the solution of the larger one. During this process, the number of sub problems may become very large, and some algorithms solve the same sub problem repeatedly, needlessly increasing the running time. Dynamic programming organizes computations to avoid re computing values that you already know, which can often save a great deal of time.

In rock problem a player can take one rock from ONE pile or TWO rocks from piles. Once the rocks are taken, they are removed from play. The player that takes the last 2 rocks or 1 rock wins the game. You make the first move. To find the winning strategy for the 6+6 game, we can construct a table, which we can call R, shown below. Instead of solving a problem with 6 rocks in each pile, we will solve a more general problem with n rocks in one pile and m rocks in another (the n + m game) where n and m are arbitrary. If Player 1 can always win the game of 5 + 6, then we would say R5,6 = W, but if Player 1 has no winning strategy against a player that always makes the right moves, we would write R5,6 = L. Computing Rn,m for an arbitrary n and m seems difficult, but we can build on smaller values. Some games, notably R0,1 and R1,0 are clearly winning propositions for Player 1 but R1,1 is not a wining proposition. Thus, we fill in entries (1, 1) as L and (0, 1) and (1, 0) as W.

0 1 2 3 4 5 6

0 W

1 W W

2

3

4

5

6

After the entries (0, 1), (1, 0), and (1, 1) are filled, one can try to fill other entries. For example, in the (2, 0) case, the only move that Player 1 can make leads to the (2, 0) case that, as we already know, is a winning position for him. A similar analysis applies to the (0, 2) case, leading to the following result:

0 1 2 3 4 5 6

0 W W

1 W W

2 W

3

4

5

6

In the (2, 1) case, Player 1 can make three different moves that lead respectively to the games of (1, 1), (2, 0), or (1, 0). One of these cases, (2, 0), leads to a losing position for his opponent and therefore (2, 1) is a winning position. The case (1, 2) is symmetric to (2, 1), so we have the following table:

0 1 2 3 4 5 6

0 W W

1 W L W

2 W W

3

4

5

6

We can proceed filling in R in this way by noticing that for the entry (i, j) to be L, the entries above, diagonally to the left and directly to the left, must be W. These entries ((i − 1, j-1),

(i − 2, j-1),(i-1, j − 1) and (i-1, j - 2) correspond to the four possible moves that player 1 can make.

0 1 2 3 4 5 6

0 W W L W W W

1W W W L W W W

2W W L L W W W

3L L L W W W W

4W W W W L W W

5W W W W W W L

6W W W W W L L