Homework 2

Sunday, February 10, 2019 12:18 PM

1.

A)
The problem is solvable in polynomial time.

Array A = [$coin_1$, $coin_2$, …. , $coin_n$]
Array P1 = []
Array P2 = []
X = value of the smaller coin
Y = value of the larger coin
Val1 = 0
Val2 = 0
it1 = 0
it2 = 0

```
For i to n:
    If Val1 > Val2:
        P2[it2] = A[i]
        it2++
        Val2 += A[i]
    If Val2 > Val1:
        P1[it1] =A[i]
        it1++
        Val1 += A[i]
    If Val1 == Val1:
        If it1 > it2:
            P2[it2] += A[i]
            it2++
            Val2 += A[i]
        If it2 > it2:
            P1[it1] += A[i]
            it1++
            Val1 += A[i]
If Val1 == Val2:
    Return true
If Val1 != Val2:
    Return false
```

The above loop runs in O(n) time and is therefore polynomial solvable.

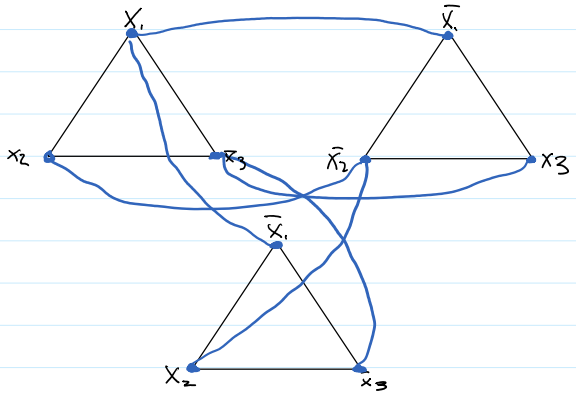
B) This problem can be solved the same way as above and therefore always runs in polynomial time.

C) The problem is NP complete because Independent Set is a known NP-Complete Problem. We can prove this by mapping 3SAT->Independent Set.

Reduction of an independent set with k = 3

3 SAT

$F = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3})$



IS of k = 3 : $\{x_1, \bar{x_2}, \bar{x_3}\}$

Mapping 3-SAT → IS can be done in Polynomial Time

If 3-SAT is yes IS is yes

$x_1 = 1 \quad x_2 = 0 \quad x_3 = 0$

$F = (1 \vee 0 \vee 1) \wedge (0 \vee 1 \vee 0) \wedge (0 \vee 0 \vee 1) = 1 \wedge 1 \wedge 1 = 1$

IS: $\{x_1, \bar{x_2}, \bar{x_3}\}$

C.

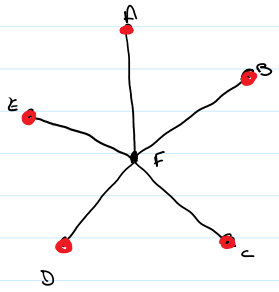
The problem is solvable in polynomial time.

C.

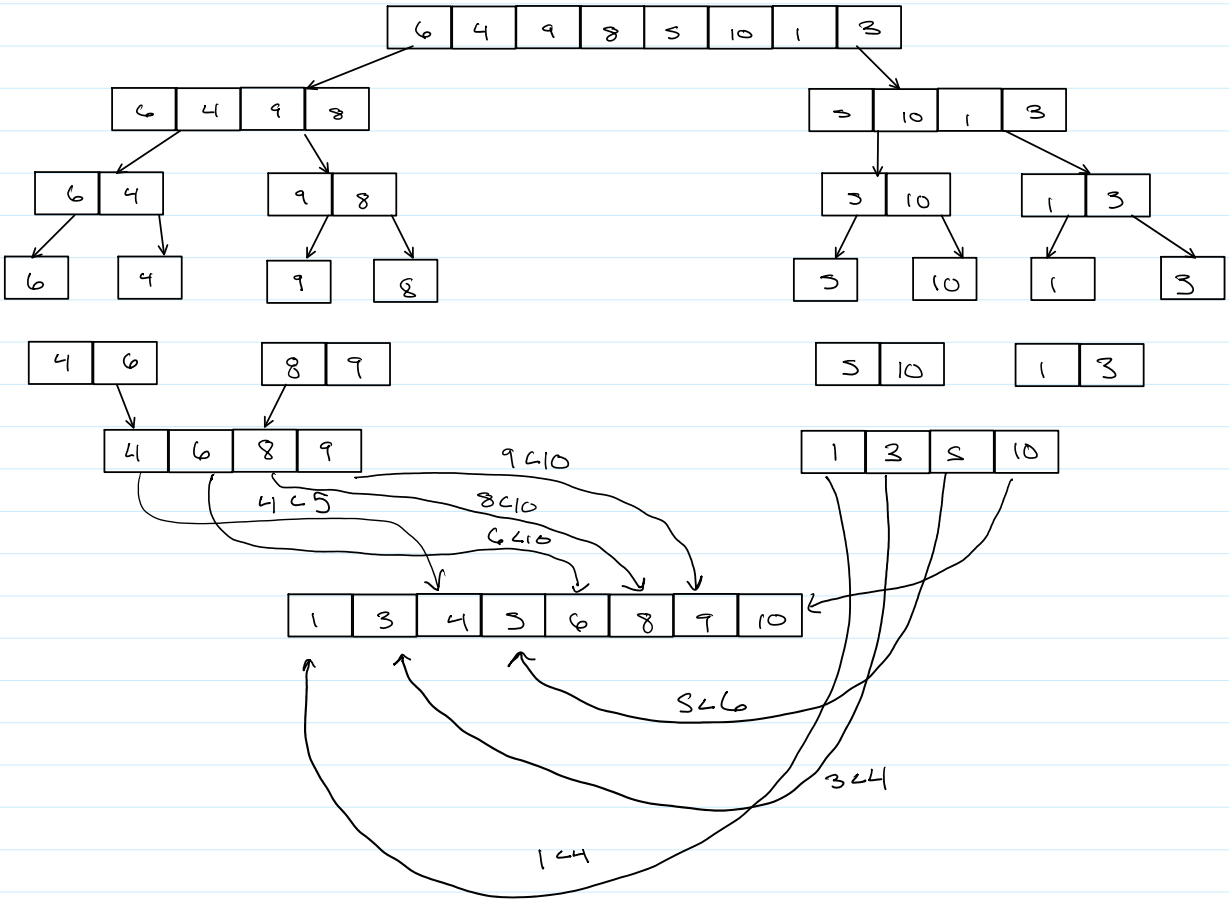
The problem is solvable in polynomial time.

{A, B, C, D, E} is an independent set

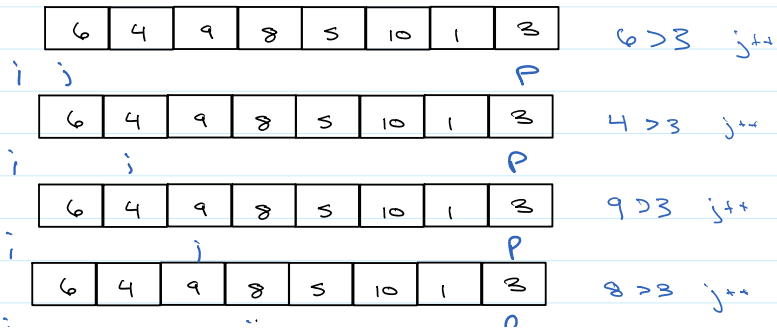
Independent Set is in NP-Complete

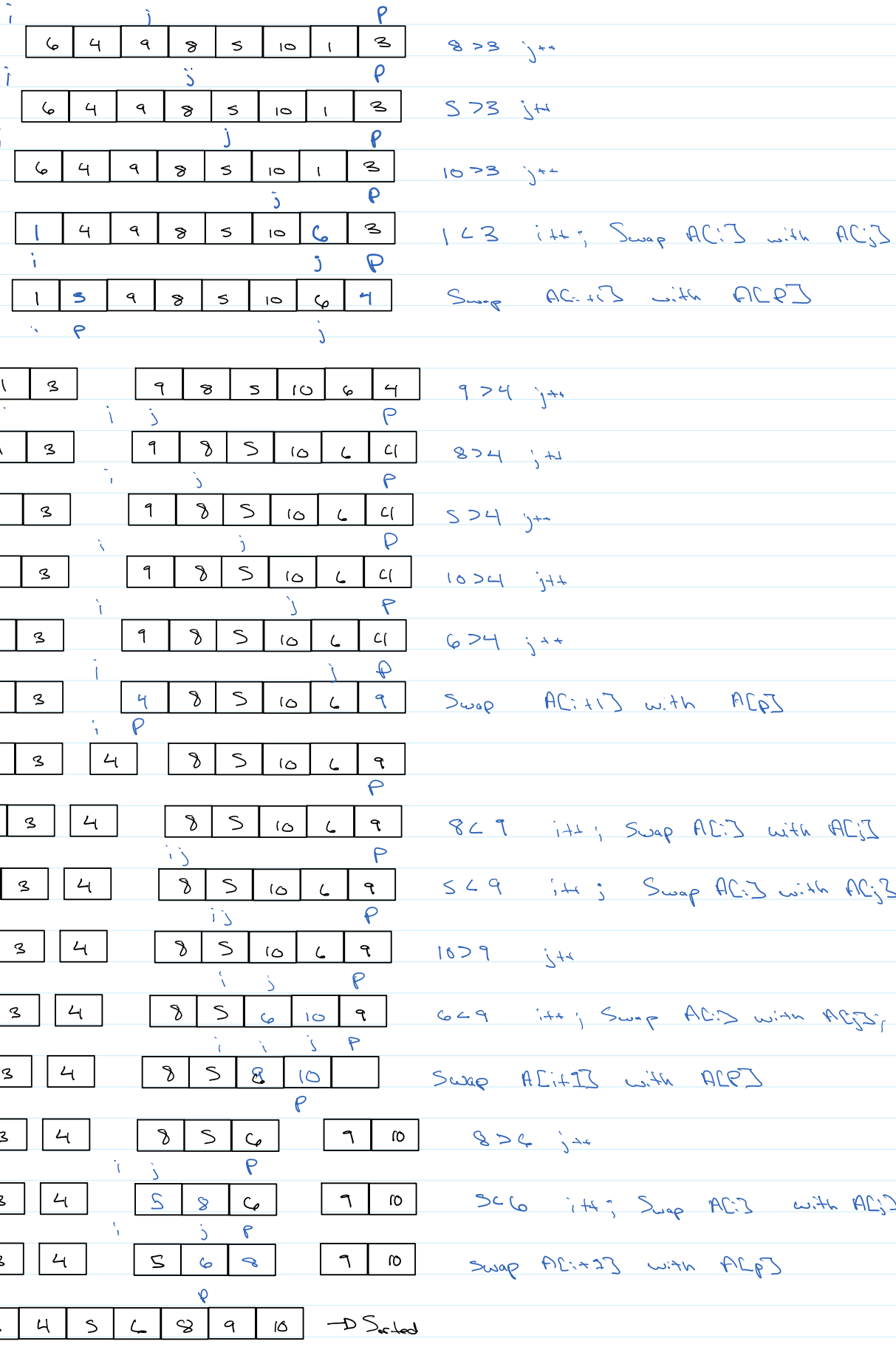


2.)



3.)




6 4 9 8 5 10 1 3
8 > 3 j++
6 4 9 8 5 10 1 3
5 > 3 j++
6 4 9 8 5 10 1 3
10 > 3 j++
1 4 9 8 5 10 6 3
1 < 3 i++; Swap A[i] with A[j]
1 3 9 8 5 10 6 4
Swap A[i+1] with A[p]
1 3 9 8 5 10 6 4
9 > 4 j++
1 3 9 8 5 10 6 9
8 > 4 j++
1 3 9 8 5 10 6 9
5 > 4 j++
1 3 9 8 5 10 6 9
10 > 4 j++
1 3 9 8 5 10 6 9
6 > 4 j++
1 3 4 8 5 10 6 9
Swap A[i+1] with A[p]
1 3 4 8 5 10 6 9
8 < 9 i++; Swap A[i] with A[j]; j++
1 3 4 8 5 10 6 9
5 < 9 i++; Swap A[i] with A[j]; j++
1 3 4 8 5 10 6 9
10 > 9 j++
1 3 4 8 5 6 10 9
6 < 9 i++; Swap A[i] with A[j]; j++
1 3 4 8 5 8 10
Swap A[i+1] with A[p]
1 3 4 8 5 6 9 10
8 > 6 j++
1 3 4 5 8 6 9 10
5 < 6 i++; Swap A[i] with A[j]; j++
1 3 4 5 6 8 9 10
Swap A[i+1] with A[p]
1 3 4 5 6 8 9 10
→ Sorted

4.)

Using the Binary Search Algorithm on the sorted array A[1...n] you can determine if there is an index such that A[i] = i in O(lgn) time

```
BinarySearch ( A, Key, f, l)
    if A.size = 1:
        if A[1] = 1:
            return true
        else
            return false
    if first ≤ last:
        middle = (f + l) ÷ 2
        if A[middle] == key:
            return true
        if key ≤ A[middle]
            return BinarySearch(A, key, f, middle-1)
        if key ≥ A[middle]
            return BinarySearch(A, key, middle+1, l)
```

5.)

A.)

Insertion sort will run in $O(k^2)$ time for items of length k. This is equivalent to ak^2+bk+c so $f(\frac{n}{k}) = \frac{n}{k}(ak^2+bk+c) = nak + nb + \frac{n}{k}c$ so for large values of n it will run in $O(nk)$ time

B.)

with n/k sublists of length k while merging it will take $\lg(\frac{n}{k})$ steps to merge each row row (both leaves of the recursion tree). Also at every step you must compare n elements n elements so the worst case run time is $O(n \lg \frac{n}{k})$

C.) $O(nk + n\lg\frac{n}{k}) = O(nk + n\lg n - n\lg k)$

For this to be true k cannot grow faster than $O(\lg n)$

if $k = O(\lg n)$ then

$O(nk + n\lg n - n\lg k) = O(n\lg n + n\lg n - n\lg(\lg n))$

$= O(2n\lg n - n\lg(\lg n))$

$n\lg(\lg n) < 2n\lg n$ so $O(2n\lg n)$ or $O(n\lg n)$

So when $k = O(\lg n)$ you have the same runtime as merge sort