Q1. Common elements problems

Q1.1 Find common elements in two arrays

Assumptions:

- sorted?
- data type
- length of array 1 vs array 2
- duplicate
- **Optimize space or time?**

What if sorted?

Solution 1: 谁小移谁

```
        xxxxxxxxxxxx              1 1 1 2 2 3          length = m
        i →
        yyyyyyyyyyyyyy            1 1 2 2 2 3 3        length = n
        j →
```

Case 1: If A1[i] < A2[j]: i++

Case 2: If A1[i] > A2[j]: j++

Case 3: If A1[i] == A2[j]: print first, then i++ j++ 1 1 2 2 3

What if we only print one copy of duplicate elements?

print first, then while() i++ j++ until the values are different 1 2 3

Time = O(m+n)

Space = O(1)

What if sorted, but the size of two arrays are very different (m << n)

Solution 2:

For each element form the array with the smaller size, run binary search against the array of larger size.

Time = O(m log n)

What if unsorted?

Solution 3: hash set

think about the length of two arrays. hash the array with the smaller size.

Time = O(m+n)

Space = O(min(m,n))

What if unsorted but we want to optimize space?

- if limited data range?
- otherwise, sort first. (selection sort / heap sort)

Q1.2 Find common elements in 3 sorted arrays **preferred O(1) space**

Solution 1:

Step1: find common elements between A1 and A2 → result1

Step2: find common elements between result1 and A3 → final result

Time = O(n)

Solution 2:

谁小移谁

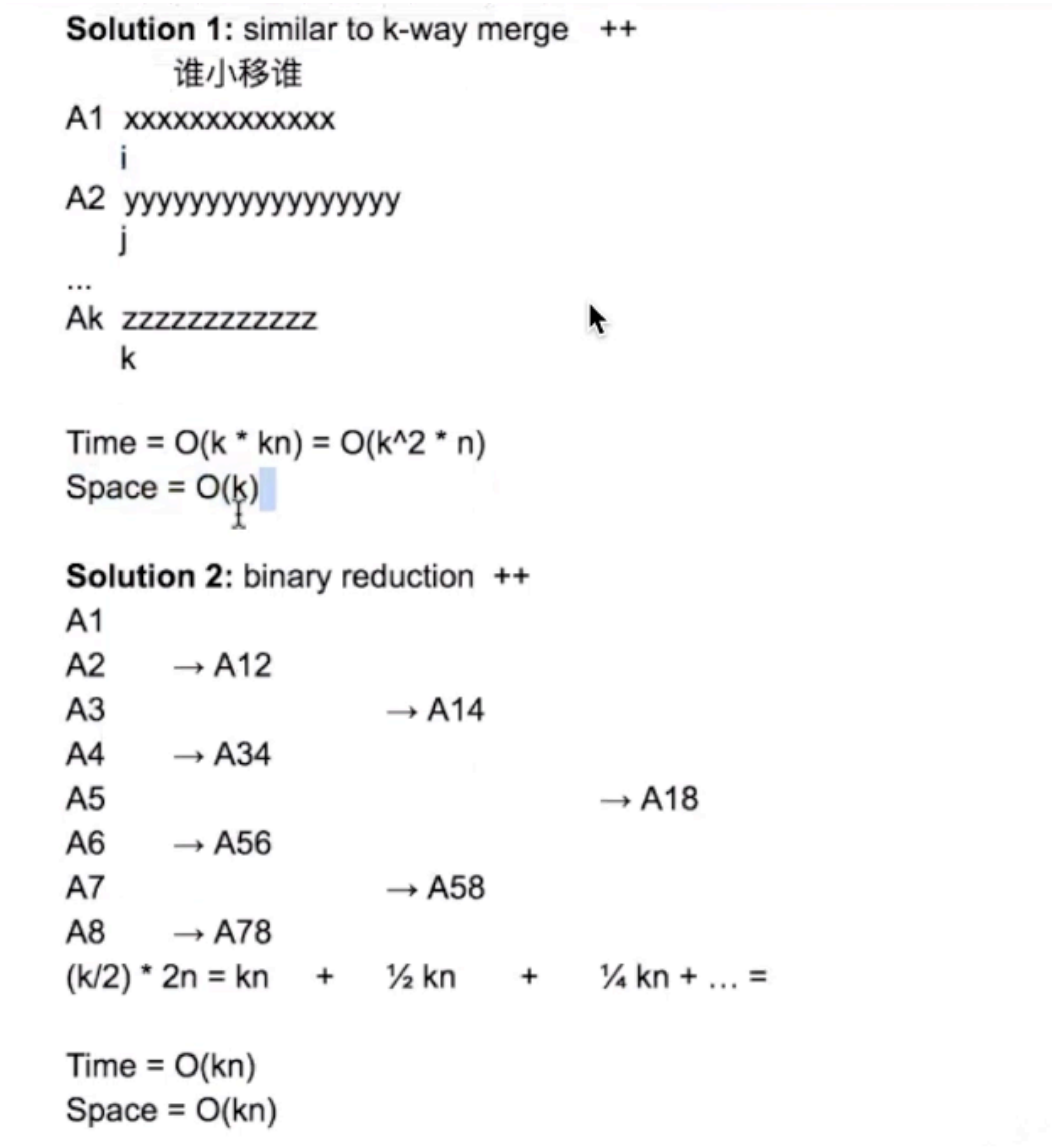
```
A  xxxxxxxxxxxxx          1111
    i                      i
B  yyyyyyyyyyyyyyyy       1111
    j                      j
C  zzzzzzzzzzzz           1111
    k                      k
```

How to move each pointer?

- move the smallest pointer
- move the non-largest two pointers

Time = O(n)

Q1.3 Find common elements in **k** **sorted** arrays



Solution 1: similar to k-way merge ++

谁小移谁

A1 xxxxxxxxxxxxx

i

A2 yyyyyyyyyyyyyyyyy

j

...

Ak zzzzzzzzzzzz

k

Time = O(k * kn) = O(k^2 * n)

Space = O(k)

Solution 2: binary reduction ++

A1			
A2	→ A12		
A3		→ A14	
A4	→ A34		
A5			→ A18
A6	→ A56		
A7		→ A58	
A8	→ A78		

(k/2) * 2n = kn + ½ kn + ¼ kn + … =

Time = O(kn)

Space = O(kn)

Solution 3: iterative ++++

A1 (n)

A2 (n) → A12 (n) 2n

A3 (n) → A13 (n) 2n

A4 → A14 2n

Time = O(kn)

Space = O(n)

Q2.1 一个字典有给一系列strings，要求找两个string，使得它们没有共同字符，并且长度乘积最大. (Assumption: **all letters in the word is from 'a-z' in ASCII,** and sorted in the length)

Example:

s1 abcde size = 5
s2 adzz size = 4
s3 abd size = 3
s4 fgz size = 3;

on average the word.length = m
there are n words in the dictionary.

Solution: abcde x fgz = 5 x 3 == 15

Potential ways to solve min/max problems:

1. DP
2. BFS2
   a. initial state <s1 x s2>
   b. expansion/generation rule
      expand a state <si x sj>
      1. generate <si+1 x sj>
      2. generate <si x sj+1>
   c. termination condition
      i. when we pop a state <si x sj> satisfying that si and sj do not share any common letter ⇒ get answer
      ii. p-queue is empty ⇒ no answer
3. DFS

```
There are n^2 pairs of state {
        Step 1: Use the p-queue to pop. Worst case, the size of the p-queue = n^2.  O(log(n^2)).
        Step 2: Compare two strings to determine if they share common letters.      O(m)
}
```

Total time = O((2log n + m) * n^2)

Solution 2: brute force

```
for i = 0...n-1 {
        for j = i+1 … n {
                check                    O(m)
        }
}
Time = O(m * n^2)
```

Example:

s1 abcde	size = 5
s2 adzz	size = 4
s3 efgh	size = 4
s4 fgz	size = 3
s5 ab	
s6 cd	
s7 x	
s8 y	

on average the word.length = k
there are n words in the dictionary.

Solution: adzz x efgh = 4 x 4 = 16

BFS2:

expand <s1, s2>	(5*4=20)	generate <s1, s3>
expand <s1, s3>	(5*4=20)	generate <s1, s4> (5*3=15) and <s2, s3> (4*4=16)
expand <s2, s3>	(4*4=16)	and we know for sure that this is the largest product.

Brute force:

check <s1, s2>	5*4=20	
check <s1, s3>	5*4=20	
check <s1, s4>	5*3=15	← is this our solution?
check <s1, s5>		
....		
check <s1, s8>		
check <s2, s3>	5	

Q2.2 How to find the k-th smallest number in the **f(x,y,z)** = 3^x * 5^y * 7^z (int x>0, y>0, z>0).

1. initial state: <x=1, y=1, z=1>
2. exp/gen rule: <x, y, z> → <x+1, y, z>, <x, y+1, z>, <x, y, z+1>
3. termination condition: when the k-th element is popped out of the heap.
4. **Deduplication when generating a new state:**
    a. example: <5, 5, 5> can be generated from <4, 5, 5> <5, 4, 5> <5, 5, 4>

Q2.3 Given three arrays with numbers in ascending order. Pull one number from each array to form a coordinate <x,y,z> in a 3D space. (1) How to find the coordinates of the points that is k-th closest to <0,0,0>?

```
A1[m] = { 1, 3, 5, 7, 9, ,... }   -> X
          i
A2[n] = {  2, 3, 4, 6, 8, ….}  -> Y
          j
A3[L] = { 1, 3, 4, 5, 6, ...}   -> Z
          k
```

Solution: f = sqrt(x^2 + y^2 + z^2)

Q2.4 Given a gym with k equipments, and some obstacles. Let's say we bought a chair and wanted to put this chair into the gym such that the sum of the shortest path cost from the chair to the k equipments is minimal.

```
xxOxxxxxxxxxxx
xxxxCxxxxxxOxxx
xxxxOE1xxxxxxx
xxxxxxE2xxxxxxx
xxxxxxxE3xxxxx
```

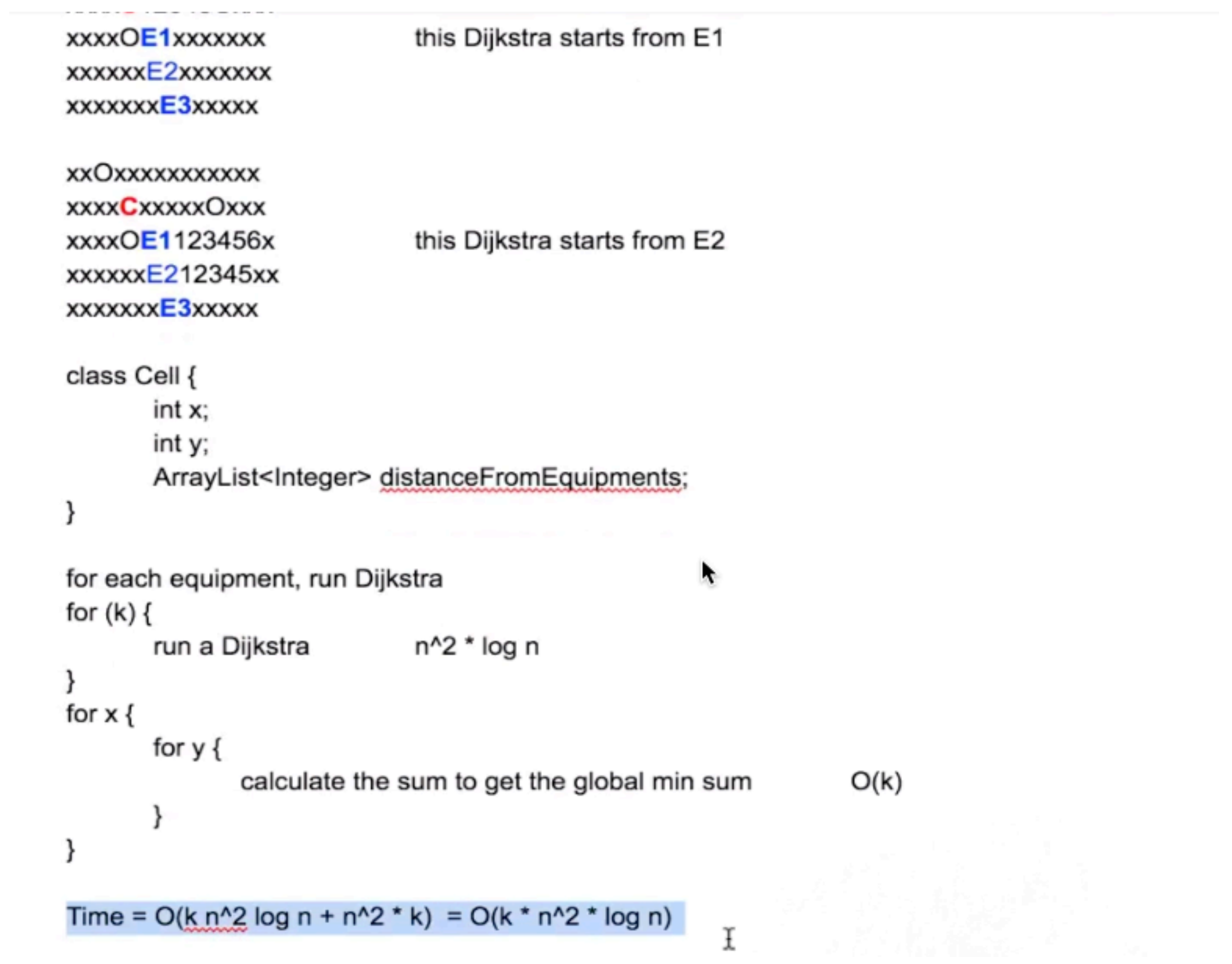
First of all, how to model the gym?
Use a 2D matrix

- 4-connected grid: cost = 1 for horizontal/vertical moves
- 8-connected grid: cost = 1 for horizontal/vertical moves, cost = sqrt(2) for diagonal moves

Solution:
for all equipment, run a Dijkstra.

Solution 1:

```
for x {
    for y {
        Try to put a chair at <x, y>.
        Run Dijkstra from <x, y> to all other locations.        O(n^2 * log n)

        Calculate the sum of the k shortest paths
        Update the global min sum
    }
}
Time = O(n^4 * log n)
```

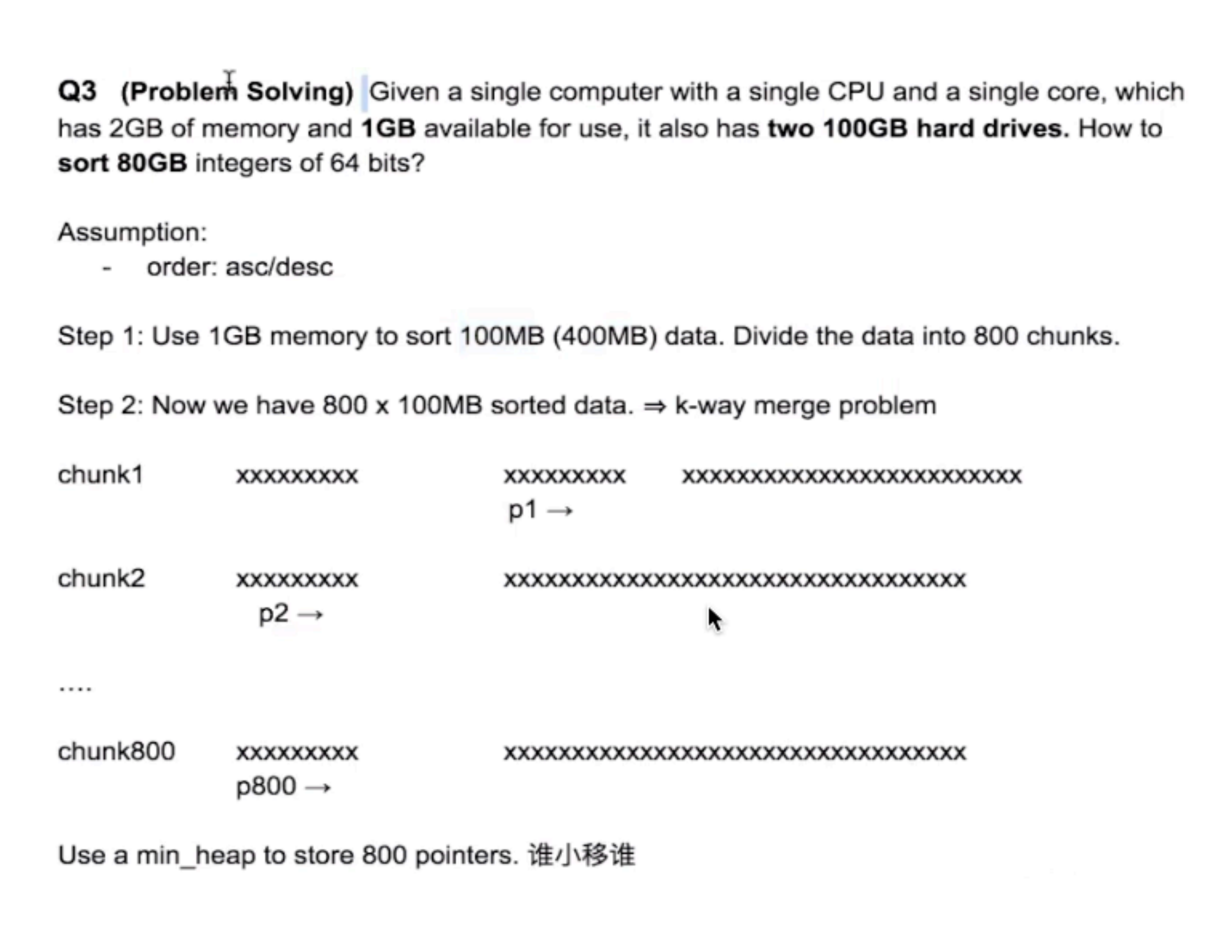


```
xxxxOE1xxxxxxx              this Dijkstra starts from E1
xxxxxxE2xxxxxxx
xxxxxxxE3xxxxx

xxOxxxxxxxxxxx
xxxxCxxxxxOxxx
xxxxOE1123456x              this Dijkstra starts from E2
xxxxxxE212345xx
xxxxxxxE3xxxxx
```

```
class Cell {
        int x;
        int y;
        ArrayList<Integer> distanceFromEquipments;
}
```

```
for each equipment, run Dijkstra
for (k) {
        run a Dijkstra          n^2 * log n
}
for x {
        for y {
                calculate the sum to get the global min sum          O(k)
        }
}
```

Time = O(k n^2 log n + n^2 * k) = O(k * n^2 * log n)



Q3 (Problem Solving) Given a single computer with a single CPU and a single core, which has 2GB of memory and **1GB** available for use, it also has **two 100GB hard drives.** How to **sort 80GB** integers of 64 bits?

Assumption:

- order: asc/desc

Step 1: Use 1GB memory to sort 100MB (400MB) data. Divide the data into 800 chunks.

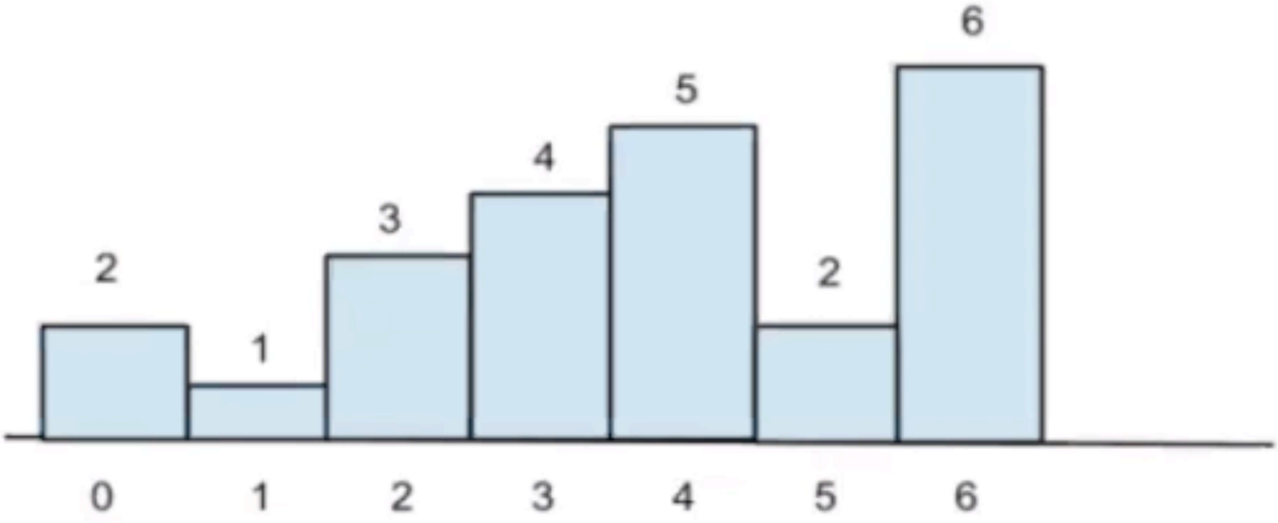
Step 2: Now we have 800 x 100MB sorted data. ⇒ k-way merge problem

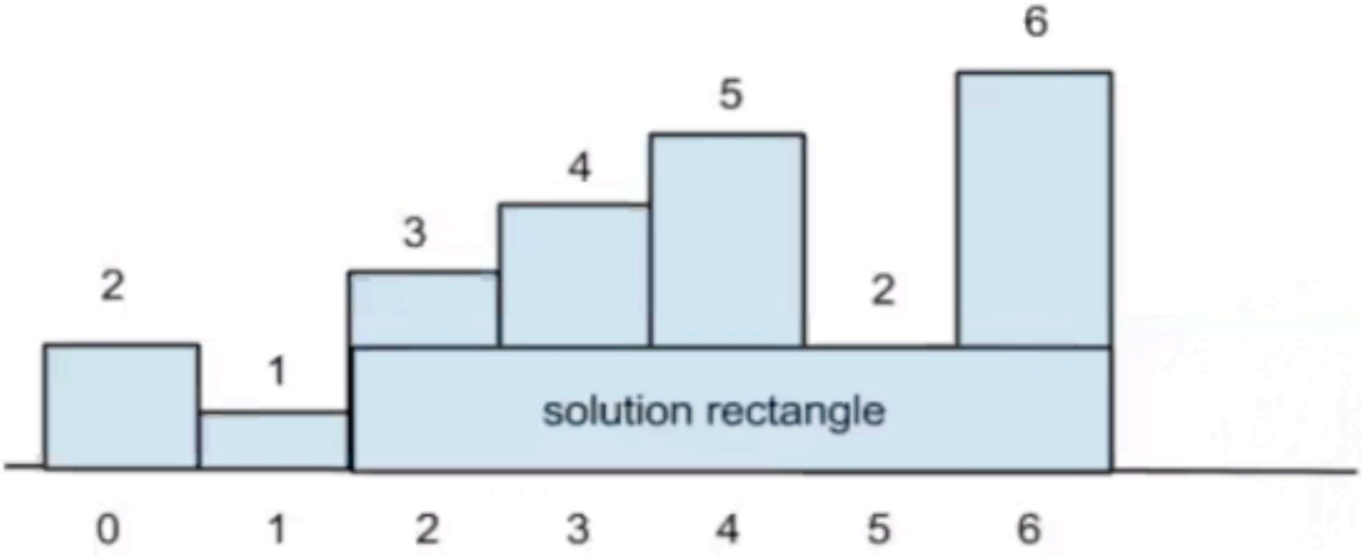
```
chunk1      xxxxxxxxx           xxxxxxxxx    xxxxxxxxxxxxxxxxxxxxxxxxx
                                 p1 →

chunk2      xxxxxxxxx           xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              p2 →

....

chunk800    xxxxxxxxx           xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            p800 →
```

Use a min_heap to store 800 pointers. 谁小移谁

Q4 Histogram questions (直方图问题)

Q4 .1直方图中找最大矩形

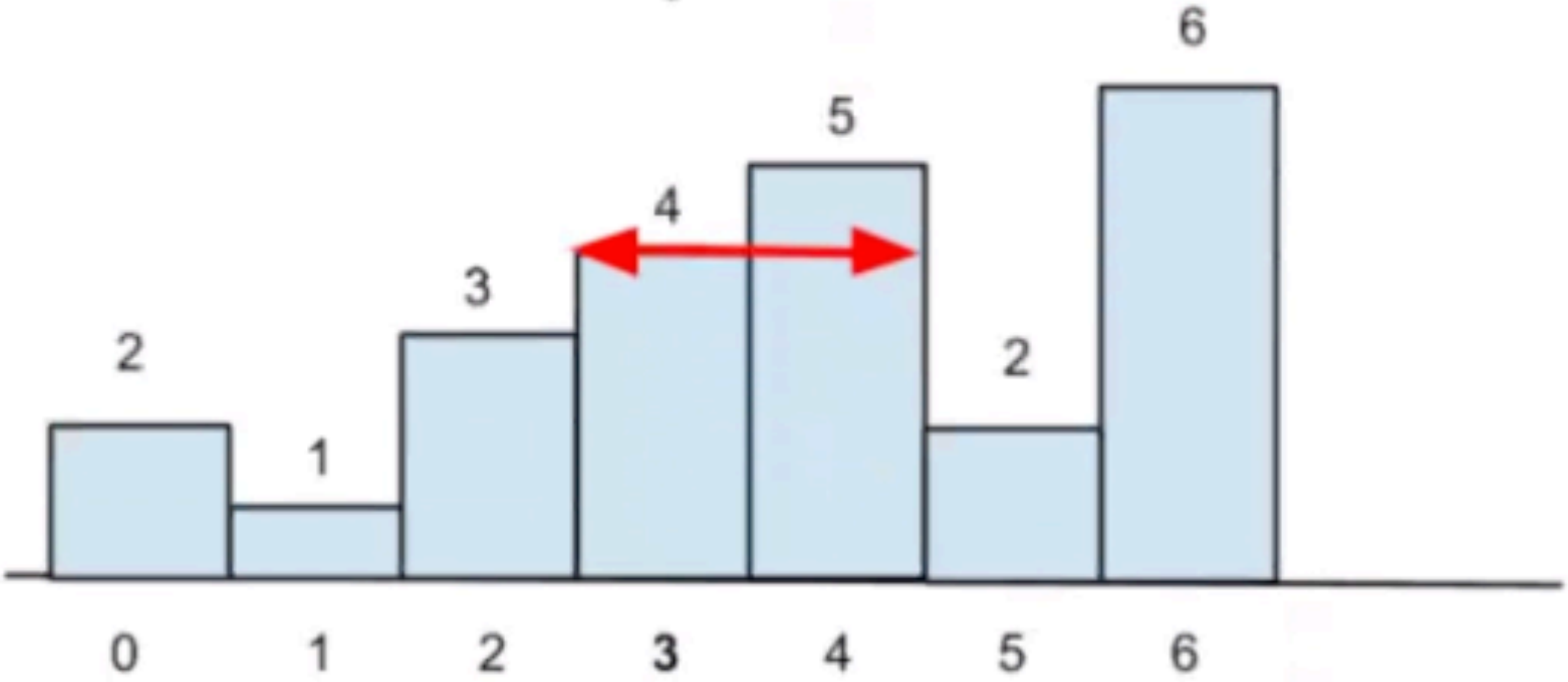




Index 3

[Left border: 3, Right border: 4]

height: 4



Solution 1: 中心开花

for each index i, 做中心开花，左走，右走

Time = O(n * (n+n)) = O(n^2)

Solution 1: 中心开花

for each index i, 做中心开花，左走，右走
Time = O(n * (n+n)) = O(n^2)

Solution 2:
Better idea:
Use a stack to store all the indices of the columns that form an ascending order
stack that stores the indices in ascending order Bottom|| [1, 2, 3, 4,

When scanning the element with index = 5, M[5] == 2 < M[4] == 5, so we keep checking left column of index 5, and calculate the area of index 4, 3, 2, and pop them out of the stack, after this step , the stack is Bottom||[1, 5

Principle, to maintain the stack to make sure the columns whose indices are stored in the stack form an ascending order.
细节：When popped an element out of the stack, the element's right border == the current index - 1, the left border of the element = the index of the element on top of the statck + 1;

Time = O(n) because every single element can only be inserted and popped out of the stack once and only once.