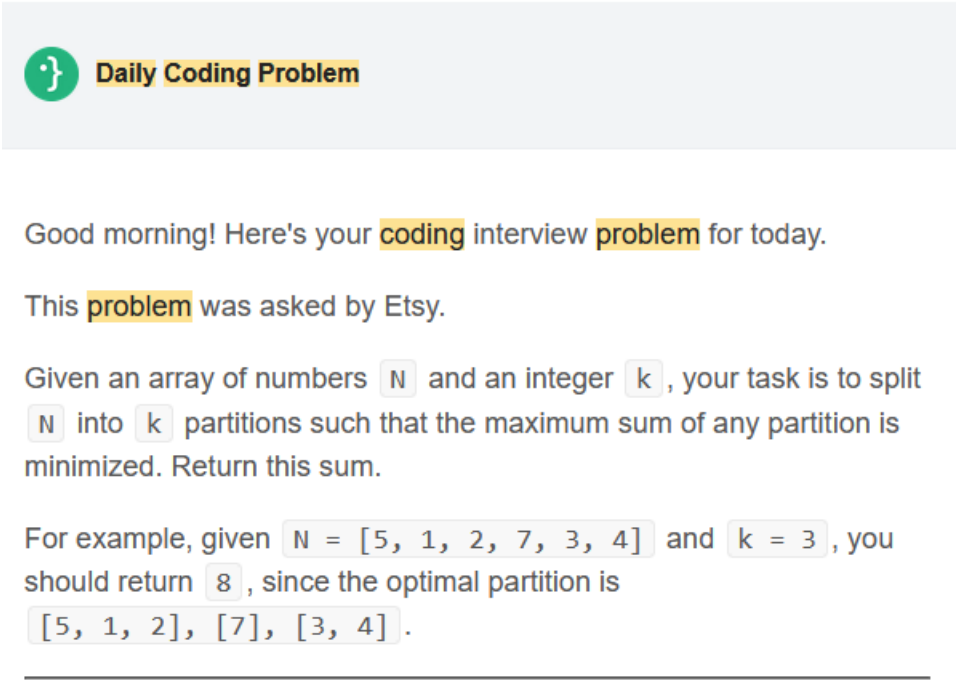
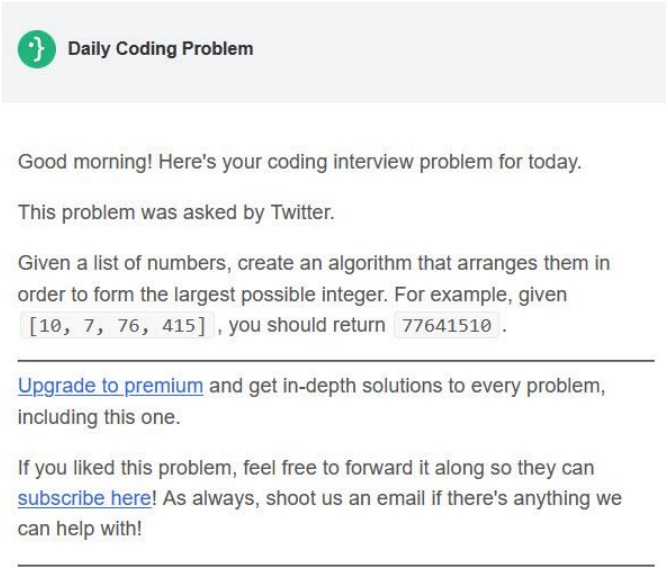


For this example (Etsy), can use the example on combination (without replacement) for generating the highest number (selecting numbers from an array and placing adjacently), completed on 18 October 2024 (Twitter example as below).



That example (Twitter) already has all combination arranging the numbers.
Note limit: C(n,r) where r can not be 64 or greater (this is the limit for arithmetic operations).
There is also if combinations exceeds limit of highest long value in Java.
Let this be stored in Set <String > arrangingN
This can be incorporated in the calculator

In Etsy example, we also have k partitions. Not used this logic before.
So size of partitions need to equal N.length.
Partition size is j=1 up to N.length (1,2,3,4,5,6)
Need to calculate all the combinations for reaching N.length
(this is similar to Amazon steps example). Example would be [1,2,3] stored in String in the Set totalsToLengthN
Once all combinations have been figured out, it would need to execute:
totalsToLengthN.toArray() and place all the values in
String [] totalsToLengthNconvertedString

Also, every time it satisfies the condition where total = 6 (once processed r items), it would populate an integer array as follows:

int [ ] partitionSizes = new int [totalsToLengthN.size()];

For above example [1,2,3], it would have performed following:
partitionSizes[0] = 3

Now need to use the partition {1,2,3} total is equal to 6 and size is **partitionSizes[0]**
Let's assume the first combination of arranging the numbers is
[5,7,1,2,3,4]

Need to fill this array (which will be stored in the Set as a String) into all arrangements of k partitions.

int [] [] [] partitionsFilled = new int [0 up to all combinations for arranging numbers in N]

[0 up to totalsToLengthN.size()]
[0 up to maximum value in partitionSizes....] The value is stored exactly here...

The array can initialised as follows for instance (given as example).

int [][][] partitionsFilled = new int [5][3][];

******* LIVE EXAMPLE COMBINING ALL VARIABLES *********

If ALL combination partitions were {1,2,3} and {5.1} for reaching N.length = 6
There are the current arrangements

If first combination (**arrangingN[0]** for arranging N [5,7,1,2,3,4] .
NOTE: It would need to execute a StringTokenizer or arranging[0].split(",").
It would fill the values in int [ ] currentCombinationN = new int [N.length]

The array can be populated as follows:

```
do
{

for (int i=0; i<partitionSizes.length;  i++)
{

for (j=0; j< partitionSizes[i].length; j++)

{
   partitionsFilled [combinationFromTotalstoLengthN ]  [i ]  [j ]  =  currentCombinationN[j];

}


counter++;

} while  (counter<currentCombinationN);
```

****

At this point, all arrangements of N have been stored in all possible partition arrangements and sizes!!!

The question states maximum sum of any partition is minimised.
In layman terms, it is suggesting the total values in each partition should be as close as possible to each other.

This in itself is another tricky concept. If the following was presented (
({3} {6} {8} **OR** {3} {4} {8})
I am totally unsure which satisfies conditions of the question better.
My initial instinct leads me toward difference (maximumPartitionTotal - minimumPartitionTotal), since it will create smallest difference in intermediate partitions...

For now, I would process each filled partition as below (see red):

int [] [] [] partitionsFilled = new int

[0 up to all combinations for arranging numbers in N]
[0 up to totalsToLengthN.size()]
[0 up to maximum value in partitionSizes....] The value is stored exactly here...

```
for (int [] []  m:   partitionsFilled)    //examining each row
  for (int [ ]  n:   m)                          //this would be examining at partition level
    total[combinationN][count]=runningTotal;
    runningTotal = 0;       //this would erase running total at partition once stored above.
     for (int p:  n)
           runningTotal = runningTotal + p;
       end for
      count++;
     end for
         combinationN++;
          count=0;   //it would be ready to start next row, hence it would start first partition
                          //again
end for
```

AND FINALLY, I WOULD HAVE THE TOTAL OF ALL SIZED PARTITIONS FOR ARRANGEMENT OF VALUES IN N.

THIS MUST BE FAIRLY CLOSED TO EXPECTED OUTCOME...