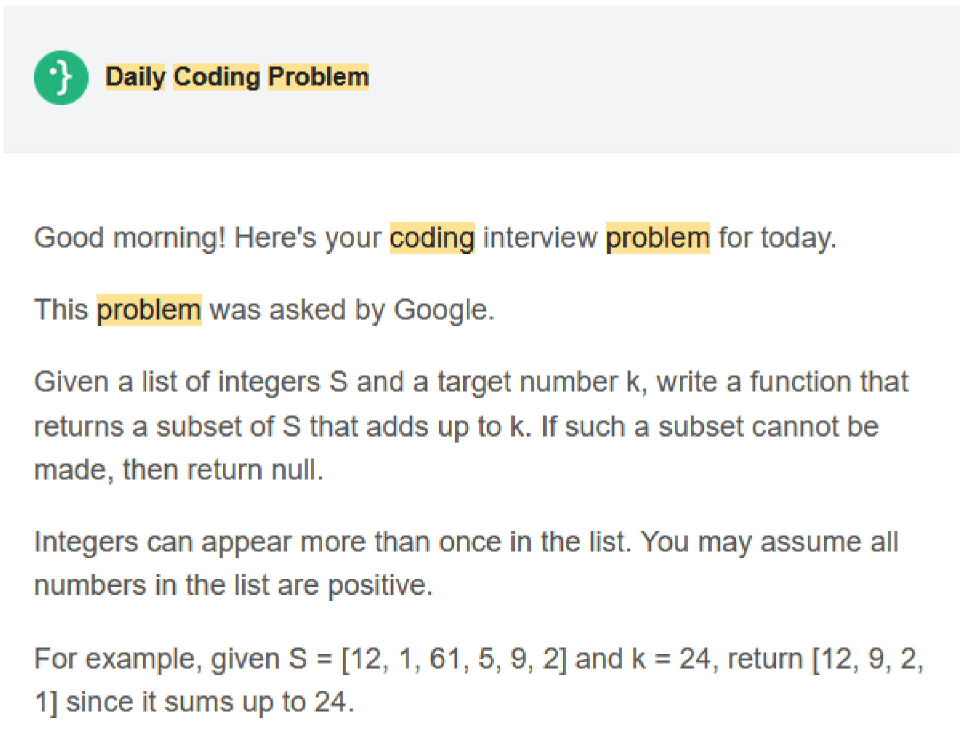

Daily Coding Problem

Good morning! Here's your coding interview problem for today.

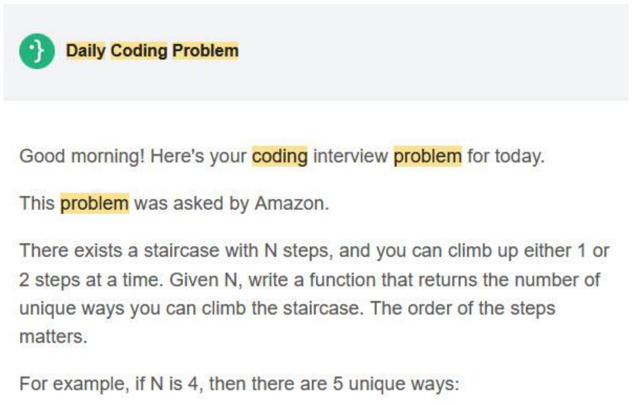
This problem was asked by Google.

Given a list of integers S and a target number k, write a function that returns a subset of S that adds up to k. If such a subset cannot be made, then return null.

Integers can appear more than once in the list. You may assume all numbers in the list are positive.

For example, given S = [12, 1, 61, 5, 9, 2] and k = 24, return [12, 9, 2, 1] since it sums up to 24.

This would be a good example to re-familiarize with concepts. It is exactly same as the Staircase and steps example.... (asked by Amazon)...   completed on 30 October 2024...
There is no harm re-visiting and testing it again....


Daily Coding Problem

Good morning! Here's your coding interview problem for today.

This problem was asked by Amazon.

There exists a staircase with N steps, and you can climb up either 1 or 2 steps at a time. Given N, write a function that returns the number of unique ways you can climb the staircase. The order of the steps matters.

For example, if N is 4, then there are 5 unique ways:

I will track all my changes. I guess this is similar to concept of beginning journey in coding in which sample code is used and changing variable names...
This time it is additional luxury of changing my personal code.....

But with a lengthy code, it can be still be very erroneous.

TRACKING

static int N = 25; // this can be changed by end user, size of the staircase
changed to:
static int k = 24; // values in subset will add up to value k

System.out.println("VALUE OF N: " + N);
changed to:
System.out.println("VALUE OF k: " + k);

if (total==N) // running total equals to N (steps in staircase)
changed to:
if (total==k) // running total equals to defined k

System.out.println("TOTAL HERE: " + total + "   N Value is: " + N);
changed to:
System.out.println("TOTAL HERE: " + total + "   k Value is: " + k);

if (total>N)
changed to:
if (total>k

maxRValue=(Staircase.N)/minimum)
changed to:
maxRValue=(Staircase.k)/minimum;

if (total>N)
changed to:
if (total>k)

System.out.println("Staircase size is: " + Staircase.N);
changed to:
System.out.println("Subset should equal to : " + Staircase.k);
//NOTE FOR NOW, NOT CHANGING THE INSTANCE VARIABLE.. INFACT NO NEED TO CHANGE IT AT ALL....

```
System.out.println("Maximum value of r: " + (maxRValue+ 1) + " has been set using minimum value in set: " + minimum + " , which should be in proximity to N(" + Staircase.N + ")");
```

changed to:

```
System.out.println("Maximum value of r: " + (maxRValue+ 1) + " has been set using minimum value in set: " + minimum + " , which should be in proximity to k(" + Staircase.k + ")");
```

```
int[] X; // this is the array containing the steps
```

changed to:

```
int[] S; // this is the array containing the steps
```

```
this.X=X;
```

to:

```
this.S=S;
```

This now leaves issue of changing the existing Set<String> s   to another variable name:

```
Set<String> s
```

to:

```
Set<String> st
```

```
this.s=s;
```

to:

```
this.st=st;
```

```
Set<String> s; // this will contain the combinations for achieving total N
```

to:

```
Set<String> st; // this will contain the combinations for achieving total k
```

```
currentSetSize=s.size(); //current size of set
```

to:

```
currentSetSize=st.size(); //current size of set
```

```
s.add(sj.toString()); // it will only add it if the conditions are met above (size of r and Total=N)
```

to:

```
st.add(sj.toString()); // it will only add it if the conditions are met above (size of r and Total=k)
```

```
for (String g: s)
```

to:

for (String g: st)

currentSetSize=s.size(); //current size of set
to:
currentSetSize=st.size(); //current size of set

newSetSize = s.size(); // new set size
to:
newSetSize = st.size(); // new set size

At this point, I decided it is sensible to test the code to ensure no logic is compromised.

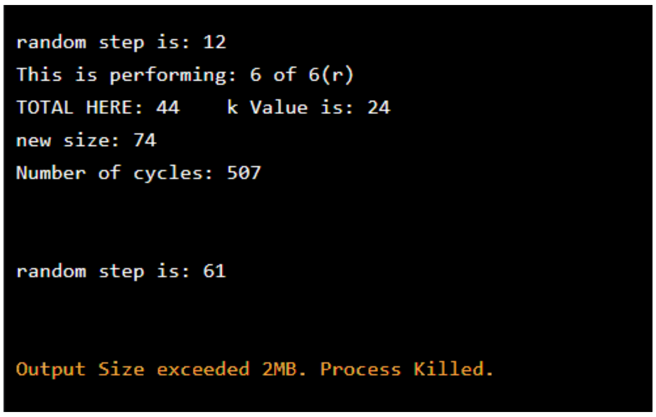
AND FORTUNATELY IT HAS COMPILED....

I decided now before making any further changes to console messages or variable names, to try and actually change the values for k and S as per the requirements.

N= 4 S={1,2}
to:
k= 24 S={12,1,61,5,9,2}

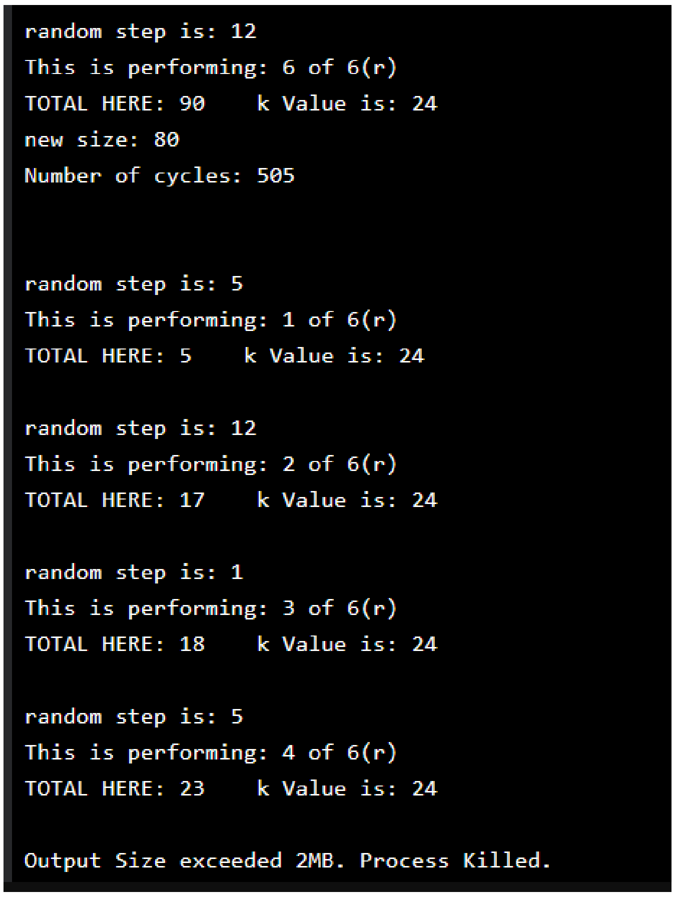
And see if it compiles... This will give a good indication that the logic for steps example is applicable to Microsoft problem.

My issues commenced immediately....
It had hit a high cycle count... of 507 (see end of document).......
It had only hit r=6 of 6 so somewhere around C(6,6)



```
random step is: 12
This is performing: 6 of 6(r)
TOTAL HERE: 44     k Value is: 24
new size: 74
Number of cycles: 507


random step is: 61


Output Size exceeded 2MB. Process Killed.
```

Now it is interesting to see that during execution of new Microsoft challenge, did it pick up any subsets equal to k before timeout. It can be seen it found lots of solutions, but do while loop over executed and utilized available 2MB resources.



```
random step is: 12
This is performing: 6 of 6(r)
TOTAL HERE: 90    k Value is: 24
new size: 80
Number of cycles: 505


random step is: 5
This is performing: 1 of 6(r)
TOTAL HERE: 5    k Value is: 24

random step is: 12
This is performing: 2 of 6(r)
TOTAL HERE: 17    k Value is: 24

random step is: 1
This is performing: 3 of 6(r)
TOTAL HERE: 18    k Value is: 24

random step is: 5
This is performing: 4 of 6(r)
TOTAL HERE: 23    k Value is: 24

Output Size exceeded 2MB. Process Killed.
```

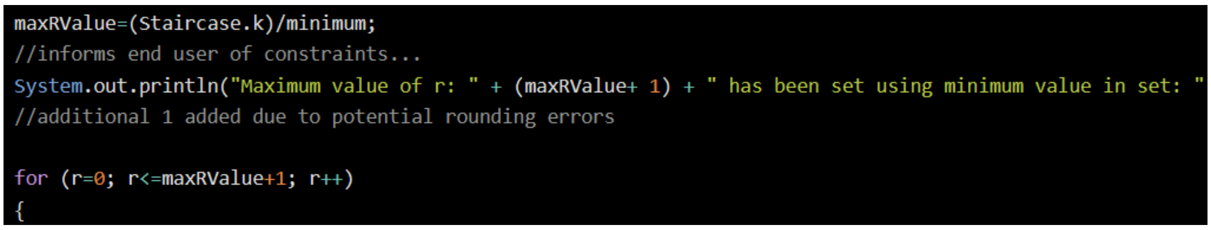
Only 505 cycles... (see end of code)...

First step is to ascertain the number of cycles the code was scheduled to execute...

}while (cycles<combinations*10);
Just as clarification, I have chosen a multiplier since it is not guaranteed, in fact it is close to impossible the required combinations can be executed first time....

I did a rough calculation of the combinations (with replacement) for the different selection scenarios:

So instead, working out combinations (with replacement) would entail C(total objects, r)
We know that it is C(6,0) + C(6,1) ..... C(6, maxRValue = 24/1 = 24)
= 1 C(6,0) + 6 C(6,1) + 21 C(6,2) + 56 C(6,3) + 126 C(6,4) + 252 C(6,5) + 462 C(6,6) + 792 C(6,7) + 1287 C(6,8) + 2002 C(6,9) + 3003 C(6,10) + 4368 C(6,11) + 6188 C(6,12) + 8568 C(6,13) + 11628 C(6,14) + 15504 C(6,15) + 20349 C(6,16) + 26334 C(6,17) + 33649 C(6,18) + 42504 C(6,19) + 53130 C(6,20) + 65780 C(6,21) + 80730 C(6,22) + 98280 C(6,23) + 118755 C(6,24) = **593775**



```
maxRValue=(Staircase.k)/minimum;
//informs end user of constraints...
System.out.println("Maximum value of r: " + (maxRValue+ 1) + " has been set using minimum value in set: "
//additional 1 added due to potential rounding errors

for (r=0; r<=maxRValue+1; r++)
{
```

We know from previous completed exercises such as Suduko grid, that online IDE will handle between 200,000 and 300,000 screen outputs.

Also, the value above would need to be multiplied by 10 (as described above).

Firstly, it has to be expressed again why it stretched to C(6,24)... It is because of the 1 in the subset...and k=24. Perhaps cycles can be skipped taking examination of the set...
For instance, we can examine the next highest number available from the subset.

The subset is:

S = [12, 1, 61, 5, 9, 2]

This has created a nightmare scenario since in essence
24 can be calculated with below... This is just example I have completed manually.

S = [12, 1, 61, 5, 9, 2]

24 x 1			r=24
22 x 1	1 x 2		r=23
20 x 1	2 x 2		r=22
18 x 1	3 x 2		r=21
16 x 1	4 x 2		r=20
14 x 1	5 x 2		r=19
12 x 1	6 x 2		r=18
10 x 1	7 x 2		r=17
8 x 1	8 x 2		r=16
6 x 1	9 x 2		r=15
4 x 1	10 x 2		r=14
2 x 1	11 x 12		r=13
0 x1	12 x 2		r=12
7 x 1	1 x 2	3 x 5	r=11
1 x 12	1 x 5	7 x 1	r=10
1 x 5, 6 x 2	7 x 1		r=9
4 x 2, 2 x 1	1 x 9	1 x 5	r=8
1 x 5, 1 x2	1 x 9, 3 x 1	1x5	r=7
	2 x 2	4 x 5	r=6
12 x 1	1 x 9	3 x 1	r=5
12 x 1, 1x1	1 x 9	2 x 2	r=4
			r=3 (not possible)

		2 x 12	r=2
			r=1 (not possible)

So it can be seen that execution cycles can not be saved whatsoever apart from r=3 or r=1 This only carries under 100 cycles...

Only option is to remove outputs to console and retry execution and only output StringJoiner is if the following condition is satisfied:

newSize>currentSize

*************** CONCLUSION ***************************

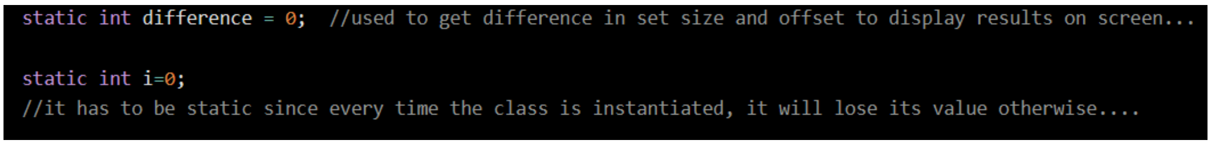
I tried extremely hard with this problem, unfortunately I could only get the longest subset to be approximately 11 numbers. This is quite natural given that its practically impossible selecting 1 each time.....

I also have a feeling that even though there is sufficient proof that the subset number (static variable stored in i) is increasing when I increase

}while (cycles<combinations);

This is a new piece of minor but essential code I can adapt into future similar problems..
Since it gives a true indication on screen that it is progressing through cycles and subset number represents when it has successful hit... Another good visual information would be running total of the cycles undertaken.. Infact this is critical to test the performance of the code.... And make adjustments where necessary.....
Since it has to be remembered that cycle was being reset each time it started a new c(n,r).

Also the difference variable, this is used in outputting from the set...
Again it is important, since if the program finishes abnormally due to memory, there are running outputs of the set displayed to end user on the screen....
It is not storing to any files, so visual screen outputs are imperative.



```
static int difference = 0;  //used to get difference in set size and offset to display results on screen...

static int i=0;
//it has to be static since every time the class is instantiated, it will lose its value otherwise....
```

However I did notice a slight issue with combination calculation during execution.

something appears slightly amiss.. I am unsure if it is affecting the outcomes that can be derived..... These numbers are simply too big..

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,1) = 6! / 1!(5)!
Combinations: 6

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,2) = 7! / 2!(5)!
Combinations: 21

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,3) = 8! / 3!(5)!
Combinations: 56

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,4) = 9! / 4!(5)!
Combinations: 126

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,5) = 10! / 5!(5)!
Combinations: 252

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,6) = 11! / 6!(5)!
Combinations: 462

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,7) = 12! / 7!(5)!
Combinations: 792

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,8) = 13! / 8!(5)!

Combinations: 1287
This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,9) = 14! / 9!(5)!
Combinations: 2002

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,10) = 15! / 10!(5)!
Combinations: 3003

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,11) = 16! / 11!(5)!
Combinations: 4368

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,12) = 17! / 12!(5)!
Combinations: 6188

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,13) = 18! / 13!(5)!
Combinations: 8568

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,14) = 19! / 14!(5)!
Combinations: 11628

This is correct
COMBINATIONS (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,15) = 20! / 15!(5)!
Combinations: 15504

This is incorrect

```
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,16) = 21! / 16!(5)!
Combinations: -1692
```

This is incorrect

```
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,17) = 22! / 17!(5)!
Combinations: -29
```

This is incorrect

```
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,18) = 23! / 18!(5)!
Combinations: 10
FUKCIN VALUE CYCLES: 0
```

This is incorrect

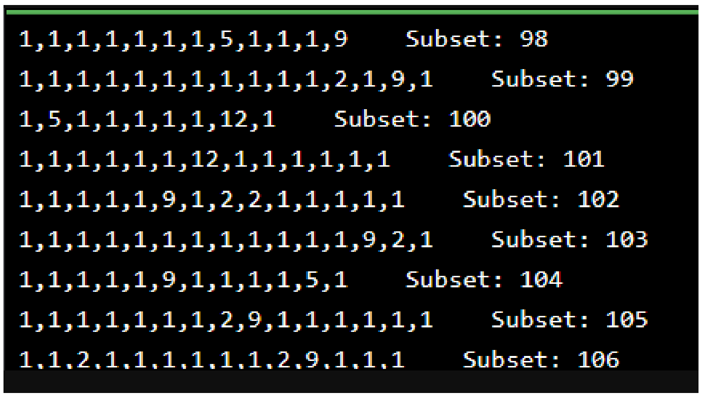
```
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,19) = 24! / 19!(5)!
Combinations: 2
```

UP TO HERE INCORRECT....

```
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(6,25) = 30! / 25!(5)!
Combinations: 1
FUKCIN VALUE CYCLES: 0
```

AS A FINAL TEST, JUST TO ENSURE THAT THE CODE HAS CAPACITY TO PULL LONGER LENGTH OF NUMBERS FROM THE ORIGINAL array X, I have completed a small modification by declaring another X array.
It can be seen the code can reach close to target k with practically all   1's

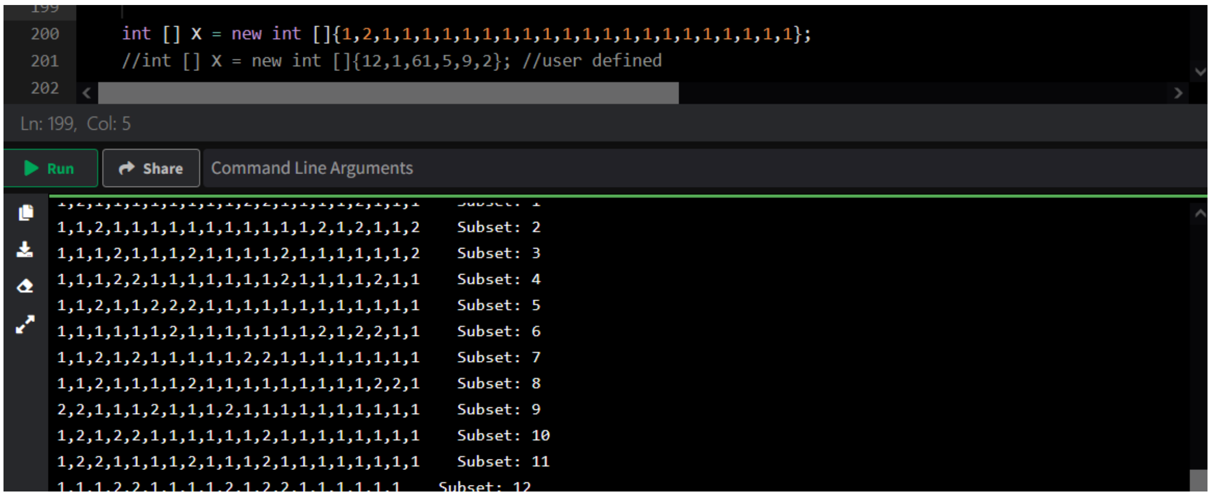
int [] X = new int []{12,1,61,5,9,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};



```
1,1,1,1,1,1,1,5,1,1,1,9     Subset: 98
1,1,1,1,1,1,1,1,1,1,1,2,1,9,1     Subset: 99
1,5,1,1,1,1,1,12,1     Subset: 100
1,1,1,1,1,1,12,1,1,1,1,1,1     Subset: 101
1,1,1,1,1,9,1,2,2,1,1,1,1,1     Subset: 102
1,1,1,1,1,1,1,1,1,1,1,1,9,2,1     Subset: 103
1,1,1,1,1,9,1,1,1,1,5,1     Subset: 104
1,1,1,1,1,1,1,2,9,1,1,1,1,1,1     Subset: 105
1,1,2,1,1,1,1,1,1,2,9,1,1,1     Subset: 106
```

I tried setting the X array to all 1's, but it fails to make a selection of all 1's. I am totally unsure why this has occurred.

When I redefined the set with a number 2 in it, it proceeded.
It appears that the limit is 20 numbers.. Again, just not sure why this arises...



```
    int [] X = new int []{1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
    //int [] X = new int []{12,1,61,5,9,2}; //user defined
```

```
1,1,2,1,1,1,1,1,1,1,1,1,1,1,2,1,2,1,1,2    Subset: 2
1,1,1,2,1,1,1,2,1,1,1,1,2,1,1,1,1,1,1,2    Subset: 3
1,1,1,2,2,1,1,1,1,1,1,1,2,1,1,1,1,2,1,1    Subset: 4
1,1,2,1,1,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1    Subset: 5
1,1,1,1,1,1,2,1,1,1,1,1,1,1,2,1,2,2,1,1    Subset: 6
1,1,2,1,2,1,1,1,1,1,2,2,1,1,1,1,1,1,1,1    Subset: 7
1,1,2,1,1,1,1,2,1,1,1,1,1,1,1,1,1,2,2,1    Subset: 8
2,2,1,1,1,2,1,1,1,2,1,1,1,1,1,1,1,1,1,1    Subset: 9
1,2,1,2,2,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1    Subset: 10
1,2,2,1,1,1,1,2,1,1,1,2,1,1,1,1,1,1,1,1    Subset: 11
1,1,1,2,2,1,1,1,1,2,1,2,2,1,1,1,1,1,1    Subset: 12
```

One last area of exploration would be to manually put in the largest Java value possible into combinations variable and let it cycle..... through r=20 to r=24
And ensure S consists of all 1's.   I will try to see if having a smaller S size will allow the combination of {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

Test case:

These are areas explored:

```
for (r=20; r<=maxRValue; r++)

}while (cycles<700000);

int [] X = new int []{1,1,1,1}; //user defined

static int k = 24; // values in subset will add up to value k
```

*** OUTPUT ************************

```
Note: Combination.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Welcome to Online IDE!! Happy Coding :)


***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(4,20) = 23! / 20!(3)!
Combinations: -2
INITIAL VALUE OF  CYCLES: 0
*************NEW VALUE CYCLES: 700000
*************RUNNING TOTAL CYCLES: 700000
***PROCESSING SET AT INDEX: 1
**ENDING AT INDEX:***** 0


***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(4,21) = 24! / 21!(3)!
Combinations: 1
INITIAL VALUE OF  CYCLES: 0
*************NEW VALUE CYCLES: 700000
*************RUNNING TOTAL CYCLES: 1400000
***PROCESSING SET AT INDEX: 1
**ENDING AT INDEX:***** 0
```

```
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(4,22) = 25! / 22!(3)!
Combinations: 0
INITIAL VALUE OF  CYCLES: 0
*************NEW VALUE CYCLES: 700000
*************RUNNING TOTAL CYCLES: 2100000
***PROCESSING SET AT INDEX: 1
**ENDING AT INDEX:***** 0


***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(4,23) = 26! / 23!(3)!
Combinations: 0
INITIAL VALUE OF  CYCLES: 0
*************NEW VALUE CYCLES: 700000
*************RUNNING TOTAL CYCLES: 2800000
***PROCESSING SET AT INDEX: 1
**ENDING AT INDEX:***** 0


***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(4,24) = 27! / 24!(3)!
Combinations: 0
INITIAL VALUE OF  CYCLES: 0
*************NEW VALUE CYCLES: 700000
*************RUNNING TOTAL CYCLES: 3500000
***PROCESSING SET AT INDEX: 1
**ENDING AT INDEX:***** 1


** Process exited - Return Code: 0 **
```

Test case:

These are areas explored:

```
for (r=0; r<=maxRValue; r++)
```

```
}while (cycles<700000);
```

```
int [] X = new int []{1,1,1,1}; //user defined
```

```
static int k = 10; // values in subset will add up to value k
```

*** OUTPUT **************************

Still no output

Test case:

Same as above but changing to:

```
 int [] X = new int []{1,1,2,1}; //user defined
```

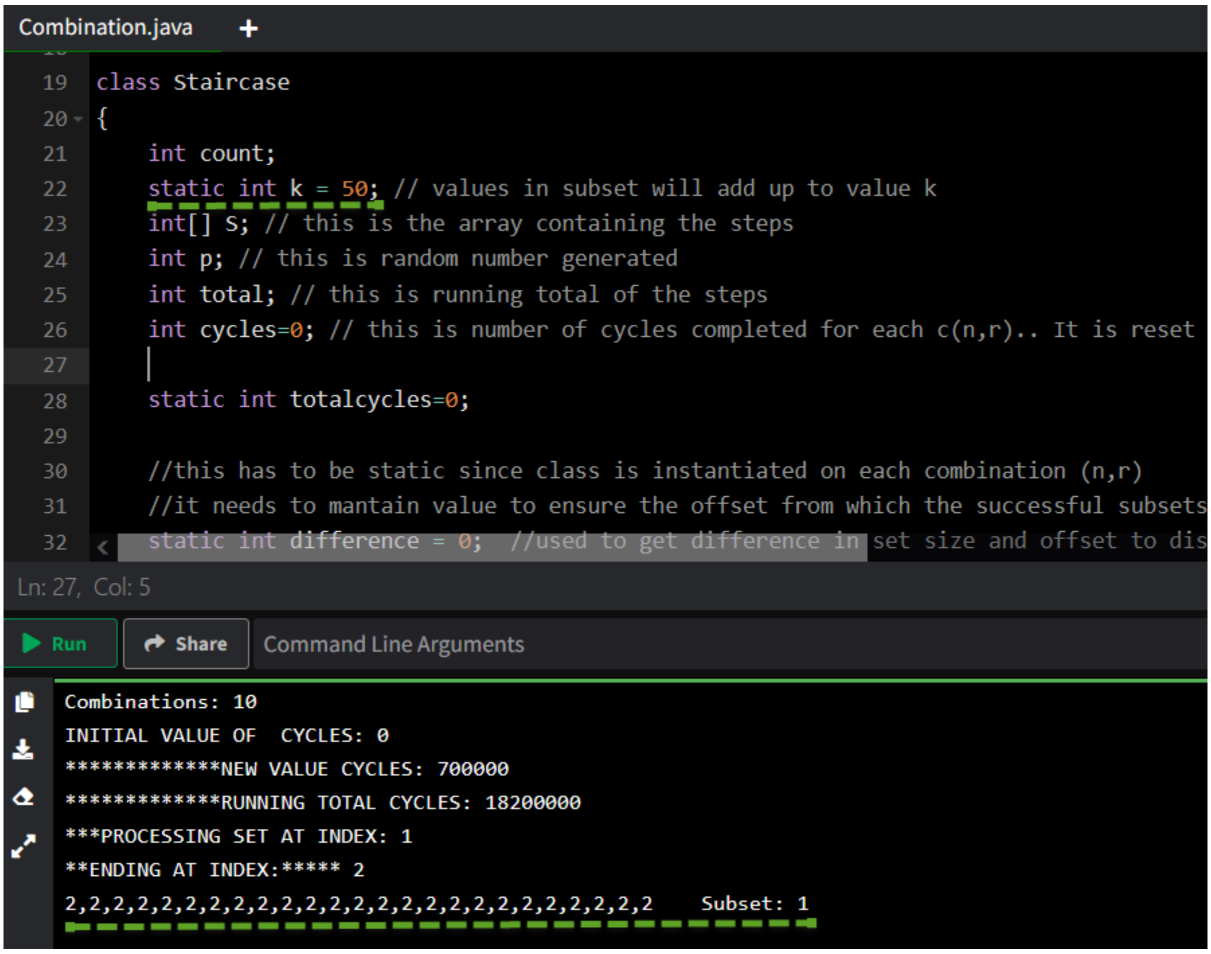
*** OUTPUT **************************

Functions as expected…

So clearly, it seems to have issues with pulling out consecutive 1's and getting the total. Perhaps, its best to include logic in the code that if the entire X consists of 1, it should abandon and just return X to end user....

But it is worth examining if issue is inherent if all numbers in X are 2 for instance...

There are no issues



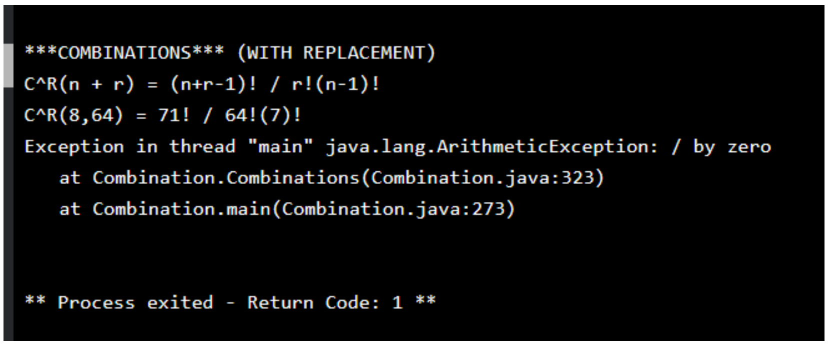
```
class Staircase
{
    int count;
    static int k = 50; // values in subset will add up to value k
    int[] S; // this is the array containing the steps
    int p; // this is random number generated
    int total; // this is running total of the steps
    int cycles=0; // this is number of cycles completed for each c(n,r).. It is reset
    
    static int totalcycles=0;

    //this has to be static since class is instantiated on each combination (n,r)
    //it needs to mantain value to ensure the offset from which the successful subsets
    static int difference = 0; //used to get difference in set size and offset to dis
```

```
Combinations: 10
INITIAL VALUE OF  CYCLES: 0
*************NEW VALUE CYCLES: 700000
*************RUNNING TOTAL CYCLES: 18200000
***PROCESSING SET AT INDEX: 1
**ENDING AT INDEX:***** 2
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2    Subset: 1
```

So now, I tried to increase k even further to 200.
But this time, it ran into issues as follows.
This means at C(8,64) that trying to handle r in this magnitude of r=64 is causing limitation...
So I set the k value back down to 128, in practice r should reach 64 here.
And it crashed again as expected.

So this is the limit... and worth putting this in the code

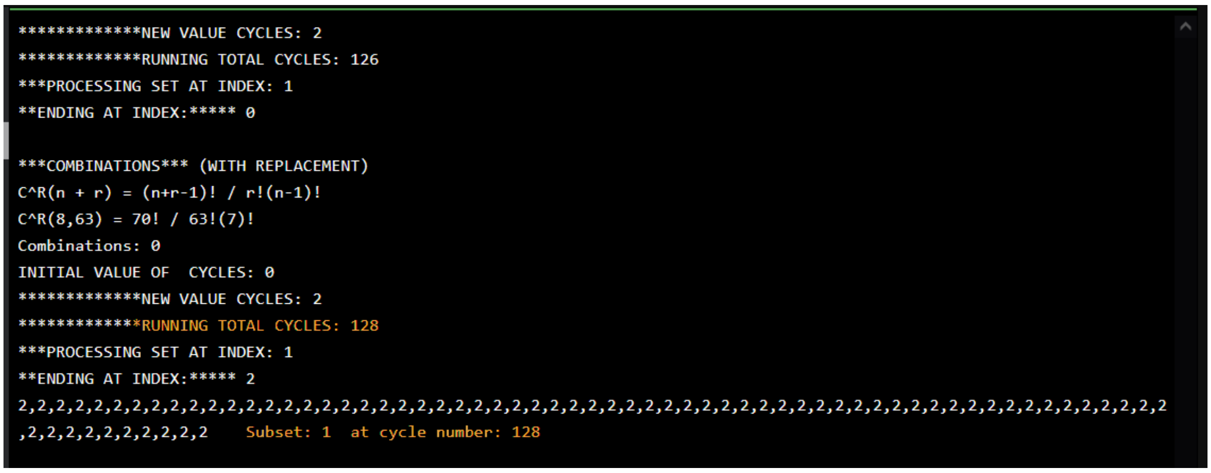
```
//as per the findings in the documentation.....
if (maxRValue>=64)
{
    System.out.println("Value is too high computationally. It will be reduced to: " + (maxRValue-1))
}
```



```
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(8,64) = 71! / 64!(7)!
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Combination.Combinations(Combination.java:323)
    at Combination.main(Combination.java:273)


** Process exited - Return Code: 1 **
```

So I kept r=126 and it had no issues.
Just to clarify (64 x number available selection) = 64 x 2 = 128
126 is less than 128.

Note it can also be seen that it has rendered output on the last execution cycle... In principle, this has wasted lots of compute.. So if all numbers are same, it will be outputted at earliest opportunity... And in regards to selecting all number 1's, this is a separate issue with no solution.



```
*************NEW VALUE CYCLES: 2
*************RUNNING TOTAL CYCLES: 126
***PROCESSING SET AT INDEX: 1
**ENDING AT INDEX:***** 0

***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(8,63) = 70! / 63!(7)!
Combinations: 0
INITIAL VALUE OF  CYCLES: 0
*************NEW VALUE CYCLES: 2
*************RUNNING TOTAL CYCLES: 128
***PROCESSING SET AT INDEX: 1
**ENDING AT INDEX:***** 2
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,2,2,2,2,2,2,2,2,2,2    Subset: 1  at cycle number: 128
```

Now I have almost finished this exercise.
It is just interesting to see if I increased the combinations for the original exercise, the impact it would have on the number of subsets....

TEST CASE

combinations = 100,000 (2,1,1,1,2,1,5,9,2 Subset: 3922 at cycle number: 2600000)
combinations = 200,000 (9,5,2,1,2,2,1,2 Subset: 5274 at cycle number: 5200000
combinations = 300,000 (9,5,2,1,2,2,1,2 Subset: 6191 at cycle number: 7800000
combinations = 1,000,000

It is a bit late to notice this, but there appears to be duplication entries on the screen.
I have had to print both arrays for st   and also   valueSet
(see Excel attachment).
Neither had sign of any duplicate values...
And the screen output is not consistent with order of valuesSet or st.

This tells me straight away that my technique for providing screen output (for offsetting) will be flawed since the set is not populated in any specific manner... And hence subject to repetition.

There are only few techniques available, unfortunately the Set can not be compromised since it governs all unique entries..

I have devised following to ensure a unique set entry is shown to the screen:

Before valuesSet takes all of the set values again, it keeps a copy

The valuesSet then takes values again from set.

do for loop between valuesSet and old valuesSet.

remove those matching entries from valuesSet...

I tried extremely hard, the code is attached in the appendix but I just could not crack the code.

I have finally finished the coding.
This has been a really fun exercise.
Even though on paper it looked like a few tweaks from Amazon one, it required several changes.
I have expressed it in my comments on website numerous times that these permutation, combination exercises require a lot of thought process.......

I am just doing one final check to see if the values outputted to the screen are unique.
This will be stored on the same excel spreadsheet.

Below are a few examples with it exceeding the memory available:

```
random step is: 12
This is performing: 2 of 2(r)
TOTAL HERE: 24   k Value is: 24
This is stringjoiner right now: 12,12
Count:1
current:0
TOTAL: 24
VALUE OF k: 24
COUNT: 1
r: 2
new size: 1
This has been added into the set:
Number of cycles: 28


random step is: 9
This is performing: 4 of 4(r)
TOTAL HERE: 24   k Value is: 24
This is stringjoiner right now: 2,1,12,9
Count:1
current:1
TOTAL: 24
VALUE OF k: 24
COUNT: 1
r: 4
new size: 2
This has been added into the set:
Number of cycles: 51


random step is: 2
This is performing: 4 of 4(r)
TOTAL HERE: 24   k Value is: 24
This is stringjoiner right now: 12,9,1,2
Count:1
current:2
TOTAL: 24
VALUE OF k: 24
COUNT: 1
r: 4
new size: 3
This has been added into the set:
Number of cycles: 77


random step is: 1
```

```
This is performing: 4 of 4(r)
TOTAL HERE: 24    k Value is: 24
This is stringjoiner right now: 12,2,9,1
Count:1
current:3
TOTAL: 24
VALUE OF k: 24
COUNT: 1
r: 4
new size: 4
This has been added into the set:
Number of cycles: 104


random step is: 2
This is performing: 4 of 4(r)
TOTAL HERE: 24    k Value is: 24
This is stringjoiner right now: 12,5,5,2
Count:1
current:4
TOTAL: 24
VALUE OF k: 24
COUNT: 1
r: 4
new size: 5
This has been added into the set:

Number of cycles: 106



random step is: 2
This is performing: 4 of 4(r)
TOTAL HERE: 24    k Value is: 24
This is stringjoiner right now: 1,12,9,2
Count:1
current:5
TOTAL: 24
VALUE OF k: 24
COUNT: 1
r: 4
new size: 6
This has been added into the set:
Number of cycles: 139


random step is: 5
This is performing: 4 of 4(r)
TOTAL HERE: 24    k Value is: 24
```

```
This is stringjoiner right now: 12,2,5,5
Count:1
current:10
TOTAL: 24
VALUE OF k: 24
COUNT: 1
r: 4
new size: 11
This has been added into the set:
Number of cycles: 293
```

UP TO 507   AND MEMORY ISSUES!!

**** APPENDIX ****

Attempting to fix the multiple subset messages during execution... To no avail:

```
//*** CODE ***

/*

Online Java - IDE, Code Editor, Compiler

Online Java is a quick and easy tool that helps you to build, compile, test your programs

online.

*/

// This has been created to ensure I can utilize any random functions more efficiently.

// It is a creation of the combinations (with replacement) calculator.

// It has used techniques I learnt including recursion and also memoization to speed up
execution.

// I will incorporate this into Java applications I created previously..

//TEST CASES

// All done on My Approach


//NOTE THE CODE IS MASSIVELY COMMENTED OUT AND PRESERVED CRITICAL COMMENTS
ON THE SCREEN....


import java.math.*;

import java.util.*;


class Staircase

{

  int count;

  static int k = 24; // values in subset will add up to value k

  int[] S; // this is the array containing the steps

  int p; // this is random number generated
```

```
   int total; // this is running total of the steps

   int cycles=0; // this is number of cycles completed for each c(n,r).. It is reset each time....

   int maxRValue;


   static int totalcycles=0;

   static int subsetIndex=0;


   //this has to be static since class is instantiated on each combination (n,r)

   //it needs to mantain value to ensure the offset from which the successful subsets are
returned are not repeated...

   static int difference = 0;  //used to get difference in set size and offset to display results on
screen...


   //it has to be static since every time the class is instantiated, it will lose its value otherwise....

   //this is now a good indicator to see the cycle on the screen and occurrence of subset totalling
k.

   static int i=0;


   static Set<String> st; // this will contain the combinations for achieving total k

   Random rand = new Random(); // object created of type Random.

   // there is no sense of X getting smaller since its with replacement....

   // need to only add the stringJoiner (converted into string) into the set if the total is equal to N

   // if total goes over, it will continue adding until all cycles for C^R (n,r) has finished

   // Combinations is total of combinations for C^R(n,r)

   // r is the sample size... This can exceed n.


    // this has to be initialised due to cloning process by valuesSetPrevious

    //if any unexpected occurrences in code, consider this.. However unlikely it will cause issue

    //since it is ample size....


   String[] valuesSet= new String[500000];   // this will be used to hold all values from the set...
```

```
  String[] valuesSetPrevious = new String[500000];


  public Staircase(long combinations, int[] X, int r, Set<String> st, int maxRValue)
  {
    int j;
    this.S=S;
    this.st=st;
    this.maxRValue=maxRValue;
    System.out.println("Combinations: " + combinations);
    StringJoiner sj = new StringJoiner(","); // creating StringJoiner for the final output
    StringJoiner sj1 = new StringJoiner(",");  //creating StringJoiner if all numbers same in X []


    int currentSetSize=0; // holds size of set before entry added
    int newSetSize; // holds size of set after entry added
    int count;
    int q; //used in the for loop to iterate through r
    int steps; // it will hold value of step generated randomly from r
    boolean allSameDigit=false;  // this is used to check if X array contains all number 1's....


    String subsetIntToString; //  since it needs to add number back into the StringJoiner,
    //it has to be converted into a String first....


    int pos=0;  // used to index array X. Required so that it can check if previous number holds
same value...
    int previousI; //value of previous number in X array


    int h=0;


    int prevcurrentSetSize=0;


    //As per my documentation, there is a serious issue if all the numbers are same in
selection..
```

```
    // Also it will create a massive hindrance since the r value in combinations will be driven too
high...

    for (int i: X)
    {
      pos++;
      previousI=i;

      subsetIntToString =  Integer.toString(i);

      sj.add(subsetIntToString);

      //only does previous check if not the first item...
      if (pos>0)
      {
      if (i==previousI)
      {
        allSameDigit=true;
      }
      }
    }

    if (!allSameDigit && pos>0)
    {
      System.out.println("This is the solution: " +  sj.toString());
      allSameDigit=false;
      System.exit(0);
    }

    System.out.println("INITIAL VALUE OF  CYCLES: " + cycles);
    do
    {
      count=1;

      //sets count back to 1 when it has finished. This is visual check to ensure that r is not
exceeded

      for (q=0; q<r;q++) // processing up to r only since this is the sample size
      {
        p= rand.nextInt(X.length); // if length 3, it will give index numbers 0-2... This is suitable
        steps = X[p]; // step value
        //System.out.println("\nrandom number from original subset is : " + steps);
        sj.add(Integer.toString(X[p])); //adds the step into StringJoiner

        prevcurrentSetSize=currentSetSize;

        currentSetSize=st.size(); //current size of set




        total=total+steps; //keeps running total
        //System.out.println("This is performing: " + (q+1) +" of " + r + "(r)");

        //reminding end user value N and running total
        //System.out.println("TOTAL HERE: " + total + "    k Value is: " + k);
        //if (total==N /*&& count<=r*/)

        if (total==k) // running total equals to defined k
        {
          //System.out.println("This is stringjoiner right now: " + sj.toString());
          //System.out.println("Count:" + count);

          //System.out.println("current:" + currentSetSize);

          //System.out.println("TOTAL: " + total);

          //System.out.println("VALUE OF k: " + k);

          //System.out.println("COUNT: " + count);

          //System.out.println("r: " + r);

          st.add(sj.toString()); // it will only add it if the conditions are met above (size of r and
Total=N)


          count++;


        }


        if (q==r-1) // if its on the last loop..

        {

          sj = new StringJoiner(",");

          total=0;


        }


      }


      if (total>k)

      {


        total=0; //resets total

        sj = new StringJoiner(","); //creates new instance, resets contents...

      }


      newSetSize = st.size(); // new set size
```

```
        //System.out.println("new size: "+ newSetSize);


        if (newSetSize>currentSetSize) //if larger, i.e a new combination, it will inform end user..

        {

          //System.out.println("This has been added into the set:");

          //System.out.println("Total is " + k +  ":  " + stringWithTotal24);


        }

        cycles++; //increases cycles

        //it is increasing cycle every time its on the end of completing do while loop


        totalcycles++;




        //System.out.println("Number of cycles: " + cycles + "\n");

        // can not set this to a certain set size... since it is giving total combinations (with
replacement) and not

        // related to placing items in certain order in which once combinations are known.

        // There are most likely distributions in statistics which might assist with the cycles...

        // This can only be placed in speculatively like throwing dice (a fixed time) and getting
total....

        //the safest option for the maximum iterations is combinations x 10.

        //but in reality there is no guarantee that it would have covered all combinations at least
once in that duration...




      //}while (cycles<combinations*500);

      }while (cycles<250000);


      // this is now getting the String in the set...
```

```
//initially i will be 0 and it will output entire contents of the set..
//once it has completed it first time, it will continue from its last position.
//this is taken to be difference....


//this is only way to process the set entry in an if loop...
//seems to be inaccurate...


 // as per documentation, it is now good opportunity to remove all items.
//it can not be done from the set, otherwise it will start duplicating entries again....
// the valuesSet are already overwritten each time...
//there is only option available...


//Before valuesSet takes all of the set values again, it keeps a copy  (this is ok)
//The valuesSet then takes values again from set.  (this is ok)
//complete for loop between valuesSet and old valuesSet.
//any matches (i.e those that have been displayed to the screen),
//in ValuesSet, it changes String to "ALREADY PROCESSED".


// fuck,  the valuesSetPrevious is cumulative...
//so it is not a clone!!!!
//the system.arraycopy has to occur on destination starting at its existing length.....


//System.out.println("FDSFDSF: " + lengthvaluesSet);


//String valuesSetPrevious[]=valuesSet.clone();



 //the destPos is a bit tricky...
 //need to figure this out from logic below.....
	//it should be consistent with set size... since zero index, this would be the first spot with null..


	// len requires severe thought.. This refers to the length of data that needs to be copied from
	// valuesSet to valuesSetPrevious
	//surely the length is required to be the difference from a logical mindset  (valuesSet-valuesSetPrevious)
	//since valuesSet is getting bigger alongside Set...
	//but we know both have length inline with the declaration 500,000
	//We require difference in filled content!!!!
	//Length valuesset would just be the same as the set size.....
	//Length of valuesSetPrevious  would be 0 initially.
	//it would then take the value of the oldsetsize.... this variable already defined... (currentSetSize);
	//so length would be  newSetSize-currentSetSize....



	//valuesSet is the exact contents of the set...
	//newSetSize is the difference between  set size from the last occurence..... this has to be the destPos...



	System.out.println("FUCKIN NEW SET SIZE: " + newSetSize);
	System.out.println("FUCKIN CURRENT SET SIZE: " + currentSetSize);



	System.out.println("NSS:" + newSetSize);
	//System.out.println(lengthvaluesSet);


	//sourcepos surely must be destpos + length?
	int sourcePos = newSetSize + (newSetSize-prevcurrentSetSize);
	//prevcurrentSetSize
 //System.arraycopy(source_arr, sourcePos, dest_arr,  destPos, len);

System.arraycopy(valuesSet, sourcePos, valuesSetPrevious, newSetSize, (newSetSize-
prevcurrentSetSize));

System.out.println("This is length content filled into valuesSetPrevious: " + (newSetSize-
currentSetSize));




System.out.println("confirm clone");


for(String m: valuesSetPrevious)

{


  if (valuesSetPrevious[h]!=null)

  {

  System.out.println(m);

  }

  h++;

}




valuesSet = st.toArray(new String[st.size()]);


System.out.println("length of previous set: " + valuesSetPrevious.length);


for (int n=0; n<valuesSetPrevious.length; n++)

{

  for (int p=0; p<valuesSet.length; p++)

  {

    //System.out.println("Current value: " + valuesSet[p]);
```

```
          if (valuesSetPrevious!=null)
          {
          if (valuesSetPrevious[n]==valuesSet[p])
          {
            System.out.println(valuesSet[p]);
            valuesSet[p]="ALREADY PROCESSED";

          }
        }

        }
      }



      System.out.println("*************NEW VALUE CYCLES: " + cycles);
      System.out.println("*************RUNNING TOTAL CYCLES: " + totalcycles);

      System.out.println("***PROCESSING SET AT INDEX: " + (difference));
      System.out.println("**ENDING AT INDEX:***** " + st.size());



      for (int i=0; i<valuesSet.length; i++)
      {
        if (valuesSet[i]!="ALREADY PROCESSED")
        {
          subsetIndex++;

        System.out.println(valuesSet[i] + "    Subset: " + subsetIndex  + "  at cycle number: " +
totalcycles);
```

```
        }
      }

      System.out.println("Size of valueSet: " + valuesSet.length);
      System.out.println("Size of set: " + newSetSize);

      //on first time it reaches above,    currentSetSize will be 0....
      //next time, currentsetSize will be previously newSetSize....
      //so it will perform incremental difference...

      //Set is zero index,  so it would have completed 5 items at  s.get(4)
      //s.size() is actual length...
      //if set size was 5 after  first  C(n,r) ,  it would have processed 0,1,2,3,4
      //difference would  be  5-0 = 5
      //so this is correct start point.....

      //difference = newSetSize;

      }  //end of constructor...
}
public class Combination
{
  public static void main(String[] args)
{
  System.out.println("Welcome to Online IDE!! Happy Coding :)");
  int originalNumber=0; // for moment it is declared a 0 to keep next line content...

  int n=originalNumber;

  int r; // this does not need be in for loop. user defined
```

```
Set <String> s = new HashSet<>(); // it will contain all combinations...



int [] X = new int []{12,1,61,5,9,2}; //user defined
//int [] X = new int []{1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
//int [] X = new int []{12,1,61,5,9,2}; //user defined

int minimum; // the smallest value in X. This will assist with generating r value...

int maxRValue; //upper limit for r in the loop to process 0<r<=maxRValue+1

Staircase sc=null; //object of type Staircase
//need to understand that r can be greater than n in replacement.......
//r is taken from n objects
// in this example, r can get larger....

Map <Integer, Long> m; // this can stay outside loop since its not impacted by size of n and r

n=X.length; //length of the array X of steps

//System.out.println("Staircase size is:  " + Staircase.k);
//System.out.println("Steps are: " + Arrays.toString(X));

originalNumber=n; // this will remain constant during a full iteration..

m= new HashMap<>(); //new instance is required to start process again...
//r can be estimated as follows based on:
//Maximum length of a combination is approximately (N/ minimum value in X)

minimum=X[0];
```

```
  for (int i=0; i<X.length;i++)

  {

    if (X[i]<minimum && X[i]!=0)

    {

      minimum=X[i];


    }


  }

  maxRValue=(Staircase.k)/minimum;

  //informs end user of constraints...

  //System.out.println("Maximum value of r: " + (maxRValue+ 1) + " has been set using minimum
value in set: " + minimum + " , which should be in proximity to N(" + Staircase.k + ")");

  //additional 1 added due to potential rounding errors


  //as per the findings in the documentation.....

  if (maxRValue>=64)

  {

    System.out.println("Value is too high computationally. It will be reduced to: " + (maxRValue-
1));

  }



  for (r=0; r<=maxRValue+1; r++)

  {

    System.out.println("\n***COMBINATIONS*** (WITH REPLACEMENT)");


    System.out.println("C^R(n + r) = " + "(n+r-1)! / r!(n-1)!");


    System.out.println("C^R(" + n+","+r+") = " + (n+r-1)+ "!" + " / " + r+"!"+"("+(n-1)+")!");


    //creates instance of Staircase and main execution of code...
```

```
      sc = new Staircase (Combinations (n,r,originalNumber, m), X, r,s, maxRValue);


    }



    Iterator iterator = Staircase.st.iterator();


    while (iterator.hasNext())
    {
      System.out.println(iterator.next());
    }


    System.out.println("Value set");


    for (int j=0; j<sc.valuesSet.length; j++)
    {
      System.out.println(sc.valuesSet[j]);


    }




}


public static long Combinations (int n, int r, int originalNumber, Map factorialResults)
{
    long result=0;
    int denominator1; //denominator split two parts since there are two factorial calculations
    int denominator2; //denominator split two parts since there are two factorial calculations
    int Numerator=n+r-1; // Numerator
    int zero=0;
```

```
  long zeroFactorial = 1;

  // if no sample or objects, there are no outcomes...

  if (originalNumber==0 && r==0)

  {

    System.out.println("n and r can not both be equal to zero");

    //System.exit(0);


    return 0;


  }

  //this situation would occur if n is 0 only and r is any positive number accept 0 (if statement
above)

  //for instance (C^R (n,r)) = (0,3) 0+3-1 = 2 2<3


  if (originalNumber==0 && originalNumber+r-1<r)

  {

    System.out.println("n+r-1 must be > or = to r");

    //System.exit(0);

    return 0;


  }


  if (Numerator>=1)

  {

    result = ((n+r-1)* (Combinations (n-1, r,originalNumber, factorialResults)));

    // this completes factorial for numerator

    factorialResults.put(Numerator,result); //result stored in the Map

    //factorialResults.put(n-1,result); //result stored in the Map

    //System.out.println("getting result back out numerator: " + (Numerator) + " " +
factorialResults.get(n+r-1));


    if (n==originalNumber) // this will occur once
```

```
      {

        denominator1 = r;

        denominator2 = originalNumber-1;

        factorialResults.put(zero,zeroFactorial); //0! is equal to 1


        if (factorialResults.containsKey(denominator1) &&
factorialResults.containsKey(denominator2))

        {

          long returnValue = result / ((long)factorialResults.get(denominator1)
*(long)factorialResults.get(denominator2));

          return returnValue;


        }


      }

      return result;


    }

    return 1; // it will reach here only when condition not met (Numerator>=1)

    }


}
```