Antoine Zoghbi

Fast Algorithm AZ1 Optimization of Algorithm ZS1

Integer Partitions Standard Representation

Algorithms (ZS1/JK1/AZ1) – (ZS2/AZ2) – (AZ3)

Table of Contents

Contents

1. Introduction

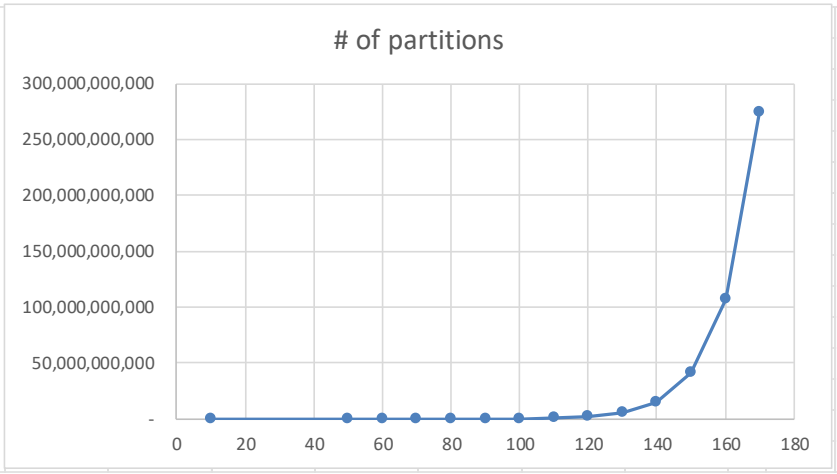
My thesis “Algorithms for generating integer partitions” was published in 1993 followed by my paper “Fast algorithms for generating integer partitions” (Zoghbi, Stojmenovic, International Journal for Computing Math 1998, vol 70, pp 319-332).

This paper provides further analysis to ZS1/ZS2 algorithms. It introduces 2 new algorithms AZ1 and AZ2 respectively in the anti-lexicographically and lexicographically standard representation. AZ1 is an enhancement of ZS1 and AZ2 is an enhancement of ZS2. Furthermore, it provides fairly empirical comparison between ZS1/JK1/AZ1 algorithms. JK1 algorithm was introduced by Jerome Kelleher 2010.

In this paper we focus on partitions in the standard representation form. For anti-lexicographic representation It has been known that algorithm ZS1 is very popular and adopted by most institutes and research centers and considered as state of the art. see JD Opdyke – A Unified Approach to Algorithms Generating Unrestricted and Restricted Integer Compositions and integer Partitions – 2010 Journal of Mathematical Modeling and Algorithms.

It is well known that the number of partitions is growing exponentially. Thus, there is a need to have very fast algorithms to generate integer partitions.

Integer	# of partitions
10	42
50	204,226
60	966,467
70	4,087,968
80	15,796,476
90	56,634,173
100	190,569,292
110	607,163,746
120	1,844,349,560
130	5,371,315,400
140	15,065,878,135
150	40,853,235,313
160	107,438,159,466
170	274,768,617,130

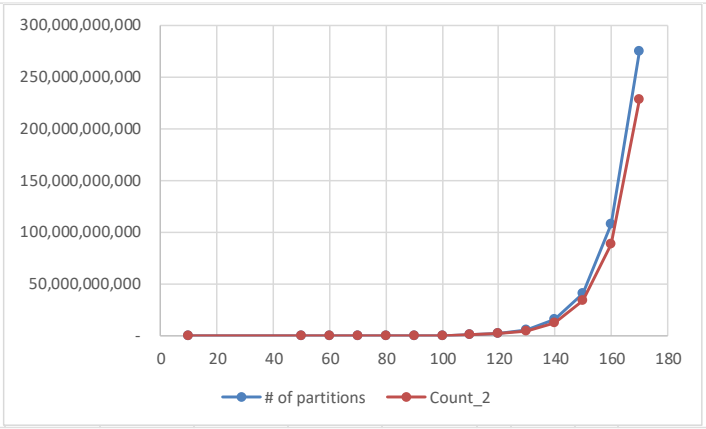


2. Algorithm ZS1

```
For I ←0 to n do a[i] ←1;
A[1] ←n; h←1; m←1; x←0;
While(a[1]>1)  do {
        If (a[h] = 2) then {
                m←m+1, a[h] ←1; h←h-1;}
        else {
          t ← m – h + 1; a[h] ← a[h] – 1; r ←a[h];
```

```
        while (t >= r) do { h ← h+1; a[h] ← r; t ← t – r;}
        if  (t > 0) then {
            m ← h + 1;
            if (t > 1) then {h ← h + 1; a[h] ← t;}
            }
       else { m ←h }
      for  i ← 1 to m do output a[i];
}
```

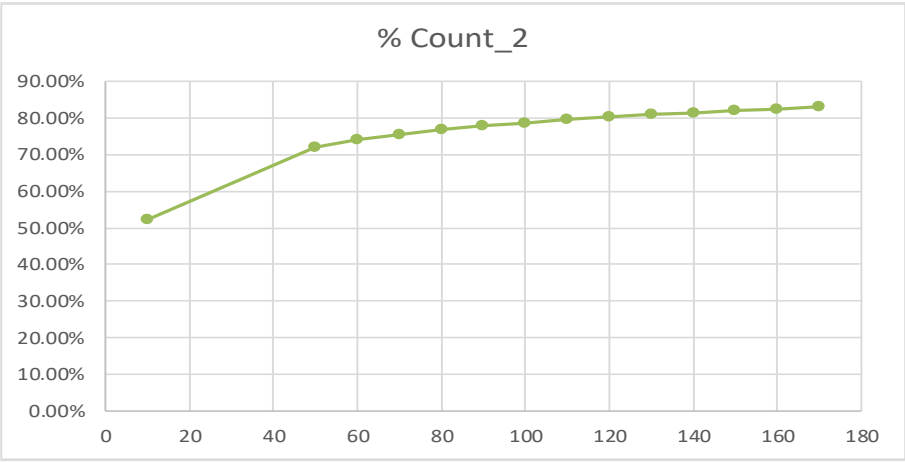
Integer	# of partitions	Count_2	% Count_2
10	42	22	52.38%
50	204,226	147,273	72.11%
60	966,467	715,220	74.00%
70	4,087,968	3,087,735	75.53%
80	15,796,476	12,132,164	76.80%
90	56,634,173	44,108,109	77.88%
100	190,569,292	150,198,136	78.82%
110	607,163,746	483,502,844	79.63%
120	1,844,349,560	1,482,074,143	80.36%
130	5,371,315,400	4,351,078,600	81.01%
140	15,065,878,135	12,292,341,831	81.59%
150	40,853,235,313	33,549,419,497	82.12%
160	107,438,159,466	88,751,778,802	82.61%
170	274,768,617,130	228,204,732,751	83.05%



Algorithm ZS1 treats partitions ending with 2 followed by 1's if any in a very dynamic and fast way. By few instructions the new partition is generated. This resulted to prove that algorithm ZS1 has constant average delay property. As shown in this chart, the more the number is higher, the more the percentage of the partitions that contains 2 is increased. For example, for integer 50, 72.11% of the partitions are having 2 followed by 1's if any. For integer 170, 83.05% of the partitions are having 2 followed by 1's if any at the right most part of the partition. The more the n is higher, the more the percentage of 2 partitions is higher.

Algorithm ZS1 - Count 2 cases

Integer	% Count_2
10	52.38%
50	72.11%
60	74.00%
70	75.53%
80	76.80%
90	77.88%
100	78.82%
110	79.63%
120	80.36%
130	81.01%
140	81.59%
150	82.12%
160	82.61%
170	83.05%



3. Algorithm AZ1

We introduce algorithm AZ1 as an enhancement and optimization to algorithm ZS1. This algorithm treats the partitions that have at the rightmost 3 preceding the 1's if any straight forward without any iteration in a very fast way. In fact, it generates all the partitions of 3 until parts 3 and parts 2 are becoming 1. For example, in the algorithm below, for n=10 partition 4,3,3 is treated under case x = 0 and for the rightmost 3 will generate the following partitions: 4,3,2,2 / 4,3,2,1,1 / 4,3,1,1,1. Since 3 is shown again at the rightmost before the 1's, then in the second iteration partition 4,3,1,1,1 is treaded under case x=3 and will generate straight forward the following partitions 4,2,2,2 / 4,2,2,1,1 / 4,2,1,1,1,1 / 4,1,1,1,1,1,1.

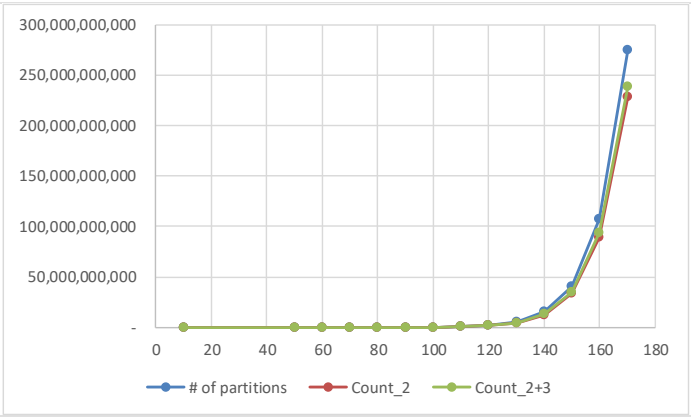
Algorithm AZ1

```
For (I =0, I <= n, i←i+1)  a[i] ←1;
A[1] ←n, h←1, m←1, x←0;
While(a[1]>1) do {
        While(a[h]=2) do {
                m←m+1, a[h] ←1; h←h-1;
                for  i ← 1 to m do output a[i];}
        if (a[h] = 3) then {
            x ← m – h; a[h] ←2;
            if (x > 3) then {
                while (x > 0) do {h←h+1; a[h] ← 2; x ← x – 2;}
                 if     (x=0) then {m ← h + 1;}
                 else {m ← h;}
                 }
        Else if ( x = 3) then {
                 A[h] ← 2; m ← m – 1; a[m] ← 2;
                 for  i ← 1 to m do output a[i];
                 a[m] ← 1; m ← m + 1;
                 for  i ← 1 to m do output a[i];
                 a[h+1] ← 1; m ← m + 1;
                 for  i ← 1 to m do output a[i];
                 a[h] ← 1; h ← h – 1; m ← m + 1;}
        else if ( x = 1) then {
                  a[m] ← 2;
                  for  i ← 1 to m do output a[i];
                  a[m] ← 1; m ← m + 1;
                  for  i ← 1 to m do output a[i];
                  a[h] ← 1; h ← h – 1; m ← m + 1;}
         else if (x =2) then {
                  a[h+1] ← 2;
                  for  i ← 1 to m do output a[i];
```

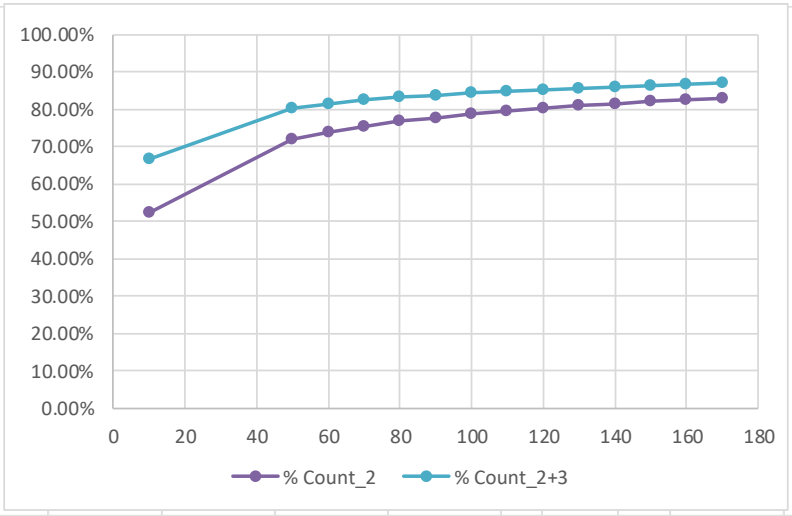
```
            a[h+1] ← 1; m ← m + 1;
            for  i ← 1 to m do output a[i];
            a[h] ← 1; h ← h – 1; m ← m + 1;}
        else  { /* x = 0 and m=h */
            m ← m + 1;
            for  i ← 1 to m do output a[i];
            a[h] ← 1; h ← h – 1; m ← m + 1;
            }
          // for cases x=0, x=1, x=2, x=3
          for  i ← 1 to m do output a[i];
       } /* a[h] =3 */
else if (a[h] > 3) then {
       t ← m – h + 1; a[h] ← a[h] – 1; r ←a[h]
       while (t >= r) do { h ← h+1; a[h] ← r; t ← t – r}
       if (t > 0) then {
          m ← h + 1;
          if (t > 1) then {h ← h + 1; a[h] ← t}
          }
       else { m ←h}
       for  i ← 1 to m do output a[i];
       } /* a[h] > 3 */
} /* while a[1] > 1 */
```

Algorithm AZ1 has increased the speed of treating partitions that contain 2 or 3 followed by the 1 if any in a partition. For n = 170, there are 207,202,977,471 partitions that contain 2 before the 1's if any, and 31,862,447,624 partitions that contain 3 before the 1's if any. However; for ZS1 there are 228,204,732,751 partitions that contain 2 before the 1 if any. The difference 228,204,732,751 - 207,202,977,471 is treated as part of the batch when number 3 is encountered and processed as described in the example of partition 4,3,3 above. The sequence of the cases x=3, x=1, x=2 and x=0 is based on the count of each case. Number of cases x=3 is higher than number of case x=1 and so on.

Integer	# of partitions	Count_2	Count_2+3
10	42	22	28
50	204,226	147,273	164,215
60	966,467	715,220	788,110
70	4,087,968	3,087,735	3,370,872
80	15,796,476	12,132,164	13,145,279
90	56,634,173	44,108,109	47,494,283
100	190,569,292	150,198,136	160,879,829
110	607,163,746	483,502,844	515,558,700
120	1,844,349,560	1,482,074,143	1,574,166,343
130	5,371,315,400	4,351,078,600	4,605,625,353
140	15,065,878,135	12,292,341,831	12,972,060,763
150	40,853,235,313	33,549,419,497	35,308,939,917
160	107,438,159,466	88,751,778,802	93,179,849,732
170	274,768,617,130	228,204,732,751	239,065,425,095



Integer	% Count_2	% Count_2+3
10	52.38%	66.67%
50	72.11%	80.41%
60	74.00%	81.55%
70	75.53%	82.46%
80	76.80%	83.22%
90	77.88%	83.86%
100	78.82%	84.42%
110	79.63%	84.91%
120	80.36%	85.35%
130	81.01%	85.74%
140	81.59%	86.10%
150	82.12%	86.43%
160	82.61%	86.73%
170	83.05%	87.01%



For example, for integer 50, 80.41% of the partitions are having 2 or 3 followed by 1's if any. For integer 170, 87.01% of the partitions are having 2 or 3 followed by 1's if any at the right most part of the partition. The more the n is higher, the more the percentage of 2 or 3 partitions is higher. For n = 170 AZ1 algorithm increased the treatment of the special cases 2 or 3 from 83.05% to 87.01%.

4. Algorithm JK1

Algorithm JK1 was introduced by Jerome Kelleher

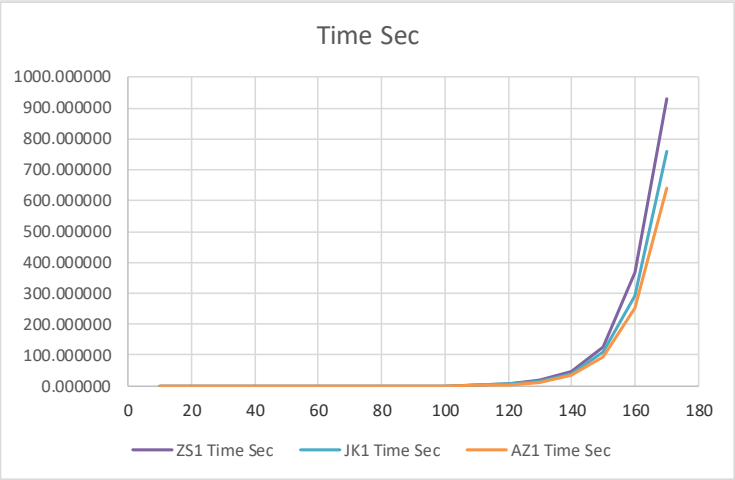
Algorithm JK1

```
For (I =0, I <= n, i←i+1) a[i] ←1;
A[1] ← 0;  k←2;  y←n - 1; x  ← 0; m ← 0;
while (k != 1) do {
        k ←k-1;
        x ← a[k] + 1;
        while ((2*x) <= y) do {
                a[k] ←= x;
                y ← y - x;
                k ← k +1;
                }
        m ← k + 1;
        while (x <= y) do {
                a[k] ← x;
                a[m] ←y;
                x ← x+1;
                y ←y- 1;
                for  i ← 1 to m do output a[i];
               }
        y ← y + x - 1;
       a[k] ← y + 1;
```

```
for  i ← 1 to k do output a[i];
}
```

Using C-free compiler on AMD Ryzen 7 5800HS,8 cores, 16 processors, 16 RAM, AZ1 was ranked the first, then JK1 the second, then ZS1 the third. For n=150 ZS1 took 127.610001 sec, JK1 took 108.864999 sec and AZ1 took 92.912003 sec. AZ1 saved 27% of the time compared to ZS1, and 15% of the time compared to JK1.

Integer	ZS1 Time Sec	JK1 Time Sec	AZ1 Time Sec
10	0.000000	0.000000	0.000000
50	0.000000	0.000000	0.000000
60	0.000000	0.000000	0.000000
70	0.015000	0.015000	0.008000
80	0.047000	0.033000	0.031000
90	0.182000	0.148000	0.125000
100	0.607000	0.517000	0.451000
110	1.953000	1.654000	1.422000
120	5.877000	5.025000	4.280000
130	17.037000	14.451000	12.367000
140	47.401001	40.383999	34.370998
150	127.610001	108.864999	92.912003
160	365.397003	290.608002	252.632000
170	928.369019	760.140991	639.046021



Using codeblock compiler, on AMD Ryzen 7 5800HS,8 cores, 16 processors, 16 RAM, AZ1 was ranked the first, then JK1 the second, then ZS1 as the third. For n=150, AZ1 took 118.500000 sec, JK1 took 135.964005 sec, ZS1 took 156.615005 sec. As a result, AZ1 saved 13% of the time compared to JK1, and 24% of the time compared to ZS1.

Using C-free compiler on Intel TM-i7-12700H 2.3Ghz, JK1 was ranked the first, then both AZ1 and ZS1 the second. For n=150 JK1 took 85.724000 sec, AZ1 took 111.524 sec and ZS1 111.429 sec. JK1 saved 23% of the time compared to ZS1, and AZ1. No remarkable improvement or significant change between ZS1 and AZ1.

INTEL TM-i7-12700H 2.3Ghz

Integer	ZS1 Time Sec	JK1 Time Sec	AZ1 Time Sec
10	0.000000	0.000000	0.000000
50	0.000000	0.000000	0.000000
60	0.011000	0.000000	0.000000
70	0.016000	0.000000	0.000000
80	0.048000	0.047000	0.046000
90	0.148000	0.125000	0.140000
100	0.520000	0.391000	0.517000
110	1.637000	1.226000	1.608000
120	4.940000	3.763000	4.962000
130	14.401000	11.289000	14.545000
140	40.679000	31.474000	41.146000
150	111.429000	85.724000	111.539000

5. Algorithm ZS2 / AZ2

Algorithm ZS2 generates unrestricted partitions in a lexicographic order standard form. This is the only algorithm in this category. I made some enhancement to this algorithm.

Algorithm ZS2

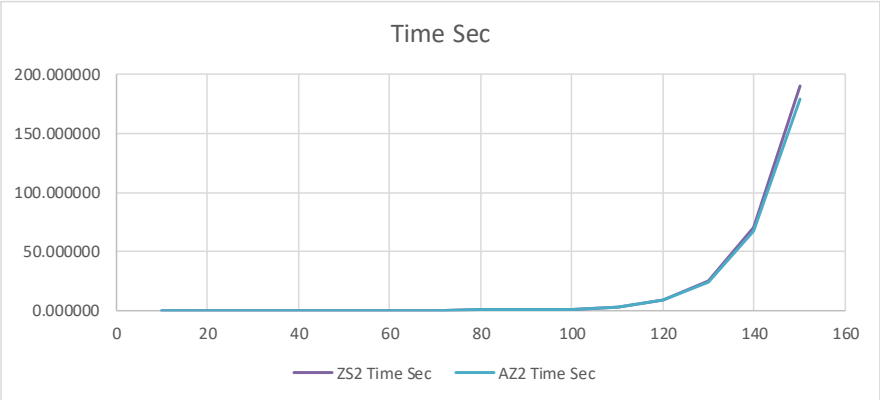
```
For (I =1, I <= n, i←i+1)  a[i] ←1;
for  i ← 1 to m do output a[i];
a[0] ← -1; a[1] ←2; h←1; m ← n – 1;
for  i ← 1 to m do output a[i];
while (a[1] != n) do {
        if (m – h > 1) then { h←h+1; a[h] ←2; m←m-1;}
        else {
              j←m-2; k←m-1; s←a[k]; r←a[m];
              while (a[j] = s) do {a[j] ←1; r←r + s; j← j – 1;}
              h← J + 1; a[h] ← s + 1; a[m] ← 1;
             if (k != h) then {a[k] ←1;}
             m←h + r – 1;
             }
       for  i ← 1 to m do output a[i];
     }
```

Algorithm AZ2

```
For (I =1, I <= n, i←i+1)  a[i] ←1;
for  i ← 1 to m do output a[i];
a[0] ← -1; a[1] ←2; h←1; m ← n – 1;
for  i ← 1 to m do output a[i];
while (a[1] != n) do {
        while  (m – h > 1) do {
                 h←h+1;  a[h] ←2; m←m-1;
                 for  i ← 1 to m do output a[i];
                }
        j←m-2; k←m-1; s←a[k]; r←a[m];
        while (a[j] = s) do {a[j] ←1; r←r + s; j← j – 1;}
        h← J + 1; a[h] ← s + 1; a[m] ← 1;
        if (k != h) then {a[k] ←1;}
        m←h + r – 1;
       for  i ← 1 to m do output a[i];
     }
```

As it is shown above, we simply replaced the if statement with while statement. Using compiler Codeblock the test was done on AMD Ryzen 7 5800HS,8 cores, 16 processors, 16 RAM. The more the integer is higher the more algorithm AZ2 is performing better. For n=150, AZ2 performed 6% better than ZS2.

Integer	ZS2 Time Sec	AZ2 Time Sec
10	0.000000	0.000000
50	0.000000	0.000000
60	0.000000	0.000000
70	0.016000	0.016000
80	0.780000	0.780000
90	0.266000	0.261000
100	0.899000	0.876000
110	2.882000	2.751000
120	8.664000	8.373000
130	25.090000	24.212000
140	70.445999	66.836998
150	190.085007	179.453003



Using compiler C-Free compiler the test was done on AMD Ryzen 7 5800HS,8 cores, 16 processors, 16 RAM. The more the integer is higher the more algorithm AZ2 is performing better. For n=150, ZS2 took 281.330994 sec while AZ2 took 273.997986 sec. AZ2 performed 3% better than ZS2.

However; using C-Free compiler the test was done on Intel TM-i7-12700H 2.3Ghz processor, no significant change was noticed on the performance of both algorithms AZ2 and SZ2.

INTEL TM-i7-12700H 2.3Ghz

Integer	ZS2 Time Sec	AZ2 Time Sec
10	0.000000	0.000000
50	0.000000	0.000000
60	0.000000	0.000000
70	0.015000	0.015000
80	0.044000	0.046000
90	0.172000	0.172000
100	0.549000	0.546000
110	1.734000	1.754000
120	5.251000	5.222000
130	15.358000	15.360000
140	43.088000	43.032000
150	117.834000	116.962000

6. Algorithm AZ3

Input integer -> n
Input lowest value of leftmost part -> x1
Input highest value of leftmost part -> x2
Input lowest value of number of parts -> y1
Input highest value of number of parts -> y2

```
Input lowest value of rightmost part > 1  -> z1
Input highest value of rightmost part > 1 -> z2
Input exclude partitions with 1's 0 means NO, 1 means YES -> z3
For I ←0 to n do a[i] ←1;
A[1] ←x2+1; h←1; m←n – x2; x←0;
While(a[1]>= x1)  do {
        If (a[h] = 2) then {
            m←m+1, a[h] ←1; h←h-1;}
        else {
           t ← m – h + 1; a[h] ← a[h] – 1; r ←a[h];
          while (t >= r) do { h ← h+1; a[h] ← r; t ← t – r;}
          if  (t > 0) then {
              m ← h + 1;
              if (t > 1) then {h ← h + 1; a[h] ← t;}
              }
        else { m ←h }

        if (y1 <= m & m <= y2) then {
            if (z1 <= a[h] & a[h] <= z2) then {
              if ((z3 == 1 && m == h) || (z3 == 0)) {
                 if (a[1] >= x1) then {
                     for  i ← 1 to m do output a[i];
                 }

              }

          }

      }

}
```

x1 and x2 determine the boundaries of the partitions exclusively generated by the algorithm. The anti-lexicographic algorithm ZS1 is an ideal solution since a[1] is set to (x2-1), and algorithm stops when a[1] is less than x1. The other partitions out of the boundaries are not generated. y1, y2 filter the generated partitions to select the desired number of parts per partition. z1, z2 is another filtering to select only the lowest part preceding the 1's in a partition. Finaly z3 determines if partitions with 1's are excluded or not. y1,y2,z1,z2,z3 are checked in the output of the partition.

Algorithm could be modified to include other type of filtering according to the needs of the user, or to remove some fileters if not relevant to the user. Here are some examples of AZ3.

For n=10, x1=4, x2=5, y1=3, y2=6, z1=3, z2=5, z3=0 the following partitions were generated:

5 4 1
5 3 1 1
5 1 1 1 1 1
4 4 1 1
4 3 3
4 3 1 1 1

Similarly, For n=10, x1=4, x2=5, y1=3, y2=3, z1=3, z2=5, z3=0 the following partitions were generated:

5 4 1
4 3 3

Similarly, For n=10, x1=4, x2=5, y1=3, y2=6, z1=3, z2=3, z3=0 the following partitions were generated:

5 3 1 1
4 3 3
4 3 1 1 1

Similarly, For n=10, x1=4, x2=5, y1=3, y2=6, z1=3, z2=3, z3=1 the following partitions were generated:

4 3 3

Another example If partitions starting with 4 and 7 are to be selected, then this could be achieved by 2 runs.

7. Conclusions

- Algorithms ZS1, JK1, AZ1, ZS2, AZ2 demonstrated to be fast algorithms. They all have constant average delay property.
- On AMD Ryzen 7 5800HS,8 cores, 16 processors, 16 RAM, AZ1 showed better performance than JK1 by about 15% saving time. Also, AZ1 showed better performance than ZS1 by about up to 30% saving time.
- On Intel TM-i7-12700H 2.3Ghz processor, JK1 showed better performance than both AZ1 and ZS1 about 23% saving time.
- On AMD Ryzen 7 5800HS,8 cores, 16 processors, 16 RAM, AZ2 showed better performance than ZS2 up to 3% saving time.
- On Intel TM-i7-12700H 2.3Ghz processor, no significant changes between AZ1 and ZS1 saving time. Similarly, no significant changes between AZ2 and ZS2.
- For n is less or equal to 100, there is no significant difference in time between the algorithms of the same category. The time saving starts to be shown significantly for n greater than 100 gradually.
- Algorithm ZS1 seems to be very good base to start with and modify it for generating restricted partitions. See AZ3.

8. Standard representation partitions

For n = 10, the Standard representation of the Partitions are:

	Antilexicografic order										Lexicographic order										JK1									
Integer	ZS1 / AZ1										ZS2 / AZ2										JK1									
10																														
	10										1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	9	1									2	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	2	
	8	2									2	2	1	1	1	1	1	1			1	1	1	1	1	1	1	3		
	8	1	1								2	2	2	1	1	1	1				1	1	1	1	1	1	2	2		
	7	3									2	2	2	2	1	1					1	1	1	1	1	1	4			
	7	2	1								2	2	2	2	2						1	1	1	1	1	2	3			
	7	1	1	1							3	1	1	1	1	1	1	1			1	1	1	1	1	5				
	6	4									3	2	1	1	1	1	1				1	1	1	1	2	2	2			
	6	3	1								3	2	2	1	1	1					1	1	1	1	2	4				
	6	2	2								3	2	2	2	2						1	1	1	1	3	3				
	6	2	1	1							3	3	1	1	1	1					1	1	1	1	6					
	6	1	1	1	1						3	3	2	1	1						1	1	1	2	2	3				
	5	5									3	3	2	2							1	1	1	2	5					
	5	4	1								3	3	3	1							1	1	1	3	4					
	5	3	2								4	1	1	1	1	1	1				1	1	1	7						
	5	3	1	1							4	2	1	1	1	1					1	1	2	2	2	2				
	5	2	2	1							4	2	2	1	1						1	1	2	2	4					
	5	2	1	1	1						4	2	2	2							1	1	2	3	3					
	5	1	1	1	1	1					4	3	1	1	1						1	1	2	6						
	4	4	2								4	3	2	1							1	1	3	5						
	4	4	1	1							4	3	3								1	1	4	4						
	4	3	3								4	4	1	1							1	1	8							
	4	3	2	1							4	4	2								1	2	2	2	3					
	4	3	1	1	1						5	1	1	1	1	1					1	2	2	5						
	4	2	2	2							5	2	1	1	1						1	2	3	4						
	4	2	2	1	1						5	2	2	1							1	2	7							
	4	2	1	1	1	1					5	3	1	1							1	3	3	3						
	4	1	1	1	1	1	1				5	3	2								1	3	6							
	3	3	3	1							5	4	1								1	4	5							
	3	3	2	2							5	5									1	9								
	3	3	2	1	1						6	1	1	1	1						2	2	2	2	2					
	3	3	1	1	1	1					6	2	1	1							2	2	2	4						
	3	2	2	2	1						6	2	2								2	2	3	3						
	3	2	2	1	1	1					6	3	1								2	2	6							
	3	2	1	1	1	1	1				6	4									2	3	5							
	3	1	1	1	1	1	1	1			7	1	1	1							2	4	4							
	2	2	2	2	2						7	2	1								2	8								
	2	2	2	2	1	1					7	3									3	3	4							
	2	2	2	1	1	1	1				8	1	1								3	7								
	2	2	1	1	1	1	1	1			8	2									4	6								
	2	1	1	1	1	1	1	1	1		9	1									5	5								
	1	1	1	1	1	1	1	1	1	1	10										10									

9. Algorithms – C codes

ZS1 - C code

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int m,n,i,h,t,r;
clock_t clock1, clock2;
time_t tstart, tfinish;
float cpu_time_used;

int main(void) {
   printf("\n Algorithm ZS1 \n \n \n ");
   printf("\n please enter the value of integer N : ");
   scanf ("%d", &n);
   printf("\n Generating Standard Antilexicographic Partitions for integer %d \n \n", n);
   int a[n+1];
   for (i=0; i<=n; ++i) a[i]=1;
   a[1] = n; h = 1; m = 1;
   time(&tstart);
   clock1 = clock();

   while (a[1] > 1){
            if (a[h] == 2) {m += 1; a[h] = 1; --h; }
          else {
                t = m-h+1; a[h] -=1; r = a[h];
                while (t >= r) {h +=1; a[h] = r; t -=r;}
                if (t > 0) {
                     m = h + 1;
                     if (t > 1 ) { h +=1; a[h] = t;}
                    }
                else { m = h; }
               }
       for(i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
   }

   clock2 = clock();
    cpu_time_used = ( (float) (clock2-clock1))/CLOCKS_PER_SEC;
    printf("\n cpu_time_used : %lf seconds \n", cpu_time_used);
    printf("  ( CLOCKS_PER_SEC:%lld) \n",CLOCKS_PER_SEC);
    printf("\n Start  : %s", ctime(&tstart));
    time(&tfinish);
   printf("\n Finish : %s", ctime(&tfinish));
   scanf ("%d", &i);
   return 0;
}
```

AZ1 – C code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int m,n,i,h,t,r,x;
float cpu_time_used;
clock_t clock1, clock2;
 time_t tstart, tfinish;

int main(void)
{
  printf("Algorithm AZ1  \n \n \n ");
  printf("\n please enter the value of integer N : ");
  scanf ("%d", &n);
  int a[n+1];
  for (i=0; i<=n; ++i) a[i]=1;
   a[1] = n; h = 1; m = 1; x = 0;
   printf("\n Generating Standard Antilexicographic partitions for integer: %d \n",a[1]);
   time(&tstart);
   clock1 = clock();

   while (a[1] > 1){
          while (a[h] == 2){
                 ++m; a[h] = 1; --h;
                  for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                 }
                if (a[h] == 3){
                  x = m - h; a[h] = 2;
                  if (x > 3){
                     while (x > 0){++h; a[h] = 2; x -=2;}
                     if    (x == 0){m = h + 1;}
                    else  {m = h;}
                    }
            else if (x == 3){
                    a[h+1] = 2; --m; a[m] = 2;
                    for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                    a[m] = 1; ++m;
                    for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                    a[h+1] = 1; ++m;
                    for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                    a[h] = 1; --h; ++m;
                    }
             else if (x == 1){
                    a[m] = 2;
                    for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                    a[m] = 1; ++m;
                    for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                    a[h] = 1; --h; ++m;
                    }
             else if (x == 2){
                    a[h+1] = 2;
                    for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                    a[h+1] = 1; ++m;
                    for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
```

```
                    a[h] = 1; --h; ++m;
                    }
                else { //x = 0  m = h
                    ++m;
                    for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                    a[h] = 1; --h; ++m;
                    }
                for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                } // a[h] == 3)
        else if (a[h] > 3){
                 t = m-h+1; a[h] -=1; r = a[h];
                 while (t >= r){++h; a[h] = r; t -=r;}
                 if (t > 0){
                    m = h + 1;
                    if (t > 1) {++h; a[h] = t;}
                    }
              else {m = h;}
              for (i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
              } // (a[h] > 3)
    } // while a[1] > 1

    clock2 = clock();
    cpu_time_used = ( (float) (clock2-clock1))/CLOCKS_PER_SEC;
    printf("\n cpu_time_used : %lf seconds \n", cpu_time_used);
    printf("  ( CLOCKS_PER_SEC:%lld) \n",CLOCKS_PER_SEC);
    printf("\n Start  : %s", ctime(&tstart));
    time(&tfinish);
    printf("\n Finish : %s", ctime(&tfinish));

     scanf ("%d", &i);
     return 0;
}
```

JK1 – C code

```
// jerome kelleher

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int m,n,i,h,t,r,k,x,y;
clock_t clock1, clock2;
time_t tstart, tfinish;
float cpu_time_used;

int main(void) {

printf("\n Algorithm JK1 \n \n \n ");
printf("\n please enter the value of integer N : ");
scanf ("%d", &n);
printf("\n Generating Partitions for integer %d \n", n);

int a[n+1];
for (i=1; i<=n; ++i) a[i]=0;
 a[1] = 0;
 k = 2; y = n - 1; x = 0; m = 0;

 time(&tstart);
 clock1 = clock();

  while (k != 1) {
            k -=1; x = a[k] + 1;
            while ((2*x) <= y) {
                     a[k] = x; y = y - x; k += 1;
                    }
            m = k + 1;
            while (x <= y) {
                     a[k] = x; a[m] = y; x += 1;y -= 1;
                    for (i = 1; i <= m; ++i) {(printf("%d ", a[i]));} printf("\n");
                   }
          y = y + x - 1; a[k] = y + 1;
         for (i = 1; i <= k; ++i) {(printf("%d ", a[i]));} printf("\n");
    }

    clock2 = clock();
    cpu_time_used = ( (float) (clock2-clock1))/CLOCKS_PER_SEC;
    printf("\n cpu_time_used : %lf seconds \n", cpu_time_used);
    printf("  ( CLOCKS_PER_SEC:%lld) \n",CLOCKS_PER_SEC);
    time(&tstart);
    printf("\n Start  : %s", ctime(&tstart));
    time(&tfinish);
    printf("\n Finish : %s", ctime(&tfinish));
    scanf ("%d", &i);
    return 0;
```

ZS2 – C code

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int m,n,i,j,h,k,t,r,s;
clock_t clock1, clock2;
time_t tstart, tfinish;
float cpu_time_used;

int main(void) {

printf("\n Algorithm ZS2 \n \n \n ");
printf("\n please enter the value of integer N : ");
scanf ("%d", &n);
printf("\n Generating Standard Lexicographic Partitions for integer %d \n \n", n);
int a[n+1];
for (i=1; i<=n; ++i) a[i]=1;
for(i = 1; i <= n; ++i) (printf(" %d", a[i])); printf("\n");
a[0] = -1; a[1] = 2;  h = 1; m = n - 1;
 for(i = 1; i <= m; ++i) (printf(" %d", a[i])); printf("\n");

 time(&tstart);
 clock1 = clock();

 while (a[1] != n){
         if   (m - h > 1)
              {
               ++h; a[h] = 2; --m;
              }
       else
             {
               j = m - 2; k = m - 1; s = a[k]; r = a[m];
               while (a[j] == s){a[j]=1; r += s; --j;}
                h = j + 1; a[h] = s + 1; a[m] = 1;
                if (k != h){a[k] = 1;}
          m = h + r - 1;
         }

     for(i = 1; i <= m; ++i) (printf(" %d", a[i])); printf(" \n");
    }

   clock2 = clock();
   cpu_time_used = ( (float) (clock2-clock1))/CLOCKS_PER_SEC;
   printf("\n cpu_time_used : %lf seconds \n", cpu_time_used);
   printf("  ( CLOCKS_PER_SEC:%lld) \n",CLOCKS_PER_SEC);
   printf("\n Start  : %s", ctime(&tstart));
   time(&tfinish);
   printf("\n Finish : %s", ctime(&tfinish));
   scanf ("%d", &i);
   return 0;
}
```

AZ2 – C code

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int m,n,i,j,h,k,t,r,s;
clock_t clock1, clock2;
time_t tstart, tfinish;
float cpu_time_used;

int main(void) {
   printf("\n Algorithm AZ2 \n \n \n ");
   printf("\n please enter the value of integer N : ");
   scanf ("%d", &n);
   printf("\n Generating Standard Lexicographic Partitions for integer %d \n \n", n);
   int a[n+1];
   for (i=1; i<=n; ++i) a[i]=1;
   for(i = 1; i <= n; ++i) (printf(" %d", a[i])); printf("\n");
   a[0] = -1; a[1] = 2;  h = 1; m = n - 1; t = m; s=1;
   for(i = 1; i <= m; ++i) (printf(" %d", a[i])); printf("\n");
   time(&tstart);
   clock1 = clock();

   while (a[1] != n){
          while (m - h > 1)
                 {
                  ++h; a[h] = 2; --m;
                 for(i = 1; i <= m; ++i) (printf(" %d", a[i])); printf(" <--=2 \n");
                 }
         j = m - 2; k = m - 1; s = a[k]; r = a[m];
         while (a[j] == s) {a[j]=1; r += s; --j;}
         h = j + 1; a[h] = s + 1; a[m] = 1;
         if (k != h) {a[k] = 1;}
         m = h + r - 1;
        for(i = 1; i <= m; ++i) (printf(" %d", a[i])); printf(" \n");
   }

    clock2 = clock();
    cpu_time_used = ( (float) (clock2-clock1))/CLOCKS_PER_SEC;
    printf("\n cpu_time_used : %lf seconds \n", cpu_time_used);
    printf("  ( CLOCKS_PER_SEC:%lld) \n",CLOCKS_PER_SEC);
    printf("\n Start  : %s", ctime(&tstart));
    time(&tfinish);
    printf("\n Finish : %s", ctime(&tfinish));
    scanf ("%d", &i);
    return 0;
}
```

AZ3 – C code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int m,n,i,h,t,r,x1,x2,y1,y2,z1,z2,z3;
clock_t clock1, clock2;
time_t tstart, tfinish;
float cpu_time_used;

int main(void) {

   printf("\n Algorithm AZ3 - Restricted \n \n \n ");
   printf("\n please enter the value of integer N : ");
   scanf ("%d", &n);
   printf("\n please enter the lowest  value of the leftmost part: ");
   scanf ("%d", &x1);
   printf("\n please enter the highest value of the leftmost part: ");
   scanf ("%d", &x2);
   printf("\n please enter the lowest  value of the number of parts: ");
   scanf ("%d", &y1);
   printf("\n please enter the highest value of the number of parts: ");
   scanf ("%d", &y2);
   printf("\n please enter the lowest  value of the rightmost part > 1: ");
   scanf ("%d", &z1);
   printf("\n please enter the highest value of the rightmost part > 1: ");
   scanf ("%d", &z2);
   printf("\n Do you want to exclude partitions with 1's? 0 means NO, 1 means YES: ");
   scanf ("%d", &z3);
   printf("\n \n \n Generating Standard Anti-lexicographic Partitions for integer %d \n \n", n);

   int a[n+1];
   for (i=0; i<=n; ++i) a[i]=1;
   a[1] = x2 + 1; h = 1; m = n - x2;

   time(&tstart);
  clock1 = clock();

   while (a[1] >= x1){
           if (a[h] == 2) { m += 1; a[h] = 1; --h; }
          else {
                t = m-h+1; a[h] -=1; r = a[h];
                while (t >= r) {h +=1; a[h] = r; t -=r; }
                if (t > 0) {
                            m = h + 1;
                            if (t > 1 ) { h +=1; a[h] = t; }
                           }
               else {m = h;}
              }
        if (y1 <= m & m <= y2){
           if (z1 <= a[h] & a[h] <= z2){
              if ((z3 == 1 && m == h) || (z3 == 0)){
                if (a[1] >= x1){
                   for(i = 1; i <= m; ++i) (printf("%d ", a[i])); printf("\n");
                }
              }
           }
```

```
        }
    }

    clock2 = clock();
    cpu_time_used = ( (float) (clock2-clock1))/CLOCKS_PER_SEC;
    printf("\n cpu_time_used : %lf seconds \n", cpu_time_used);
    printf("  ( CLOCKS_PER_SEC:%lld) \n",CLOCKS_PER_SEC);
    printf("\n Start  : %s", ctime(&tstart));
    time(&tfinish);
    printf("\n Finish : %s", ctime(&tfinish));
    scanf ("%d", &i);
    return 0;
}
```

10. References:

Using google search “zoghbi integer partitions”, here are some of the interesting web sites:

1. Book – The Art Of Computer Programming (TAOCP) Volume 4 Fascicle 3 – Donald E. Knuth
2. Book – Handbook of Applied Algorithms – Amina Nayak and Ivan Stojmenovic
3. Book – Multiagent Coordination Enabling Autonomous Logistics – Arne Schults
4. Book – Future Urban Energy System for Buildings: The Pathway Towards Flexibility – Xingxing Zhang
5. Zoghbi Stojmenovic – Fast algorithms for generating integer partitions - International Journal for Computing Math 1998, vol 70, pp 319-332
6. JA Unified Approach to Algorithms Generating Unrestricted and Restricted Integer Compositions and integer Partitions – 2010 Journal of Mathematical Modeling and Algorithms – JD Opdyke
7. Encoding Partitions As Ascending Compositions. Department of Computer Science, University College Cork – December 2005 – Jerome Kelleher
8. Georgia Institute of Technology – Layout-Aware Mixture Preparation Of Biochemical Fluids
9. National Institute of Health (NIH) – Efficient Algorithms for Calculating Epistatic Geomic…
10. Boston College – Julie Documentation
11. Sporadic.stanford.edu – William Stein and Jonathan Bober
12. University of Toronto – Space & aerospace – Efficient Homotopy Continuation algorithms with …
13. University of Toronto – Efficient Numerical Differentiation of Implicitly-Defined Curves for Sparce Systems – David A. Brown and David W. Zingg
14. SCISPACE – NASA
15. Uncit – H2F: a hierarchical Hadoop framework to process Big Data…
16. MDPI – The integer Nucleolus of Direct Simple Games
17. University of Auckland – ResearchSpace – Cristian S. Calude
18. Hal-Inria- On the cross-sectional distribution of portfolio returns
19. MetaCpan – David Landgren
20. www.arthurcarabott.com – music
21. Application in Plant Science (agEcon) – sjart_st0142.pdf
22. CHATGPT artificial Intelligence
23. A NONPARAMETRIC INDEPENDENCE TEST USING RANDOM PERMUTATIONS - By Jes´us E. Garc´ıa †, ‡ and Ver´onica A. Gonz´alez-L´opez Universidade Estadual de Campinas (ZS2)
24. Developments from Programming the Partitions Method for a Power Series Expension – Victor Kowalenko
25. Fluidigm2PURC: automated processing and haplotype inference for double-barcoded PCR amplicons1 - Paul D. Blischak2,9, Maribeth Latvis3, Diego F. Morales-Briones4, Jens C. Johnson5,Verónica S. Di Stilio5, Andrea D. Wolfe2, and David C. Tank6,7,8

Other web sites could be found as well, mainly after this paper was prepared.