Sum of the first n terms of the Beatty sequence of $\sqrt{2}$

Zicong Mo

July 18, 2018

The Beatty sequence of an irrational number α is the sequence defined by $a_i = \lfloor i \cdot \alpha \rfloor$. Although a closed form solution for the sum of the first n numbers in the sequence is difficult to find, a recursive algorithmic solution can be obtained and implemented. We first derive the general algorithm to calculate the sum of the first n terms of any Beatty sequence with positive α, and then implement it for the specific case of $\alpha = \sqrt{2}$.

General Case

For shorthand, let $B_\alpha = \lfloor \alpha \rfloor, \lfloor 2\alpha \rfloor, ...$ denote the Beatty sequence of α and $B_\alpha(n) = \lfloor n\alpha \rfloor$. For a given Beatty sequence B_α with $\alpha > 1$, let β be the irrational number satisfying $\alpha^{-1} + \beta^{-1} = 1$. Then B_α and B_β partition the natural numbers (Beatty's Theorem). Let $S_\alpha(n) = \sum_{i=1}^{n} \lfloor i \cdot \alpha \rfloor$ be the sum of the first n elements of B_α. We can compute $S_\alpha(n)$ recursively, based on the values of α and n. For the base case, note that $S_\alpha(0) = 0$. There are three cases to consider for α:

1. $\alpha > 2$. Because $\alpha > 2$, there exist an irrational number α' between 1 and 2 and a positive integer x such that $\alpha = \alpha' + x$. Although the expression $\lfloor x + y \rfloor$ is not equal to $\lfloor x \rfloor + \lfloor y \rfloor$ in general, the formula holds when either x or y is an integer. Therefore,

$$S_\alpha(n) = \sum_{i=1}^{n} \lfloor i \cdot \alpha \rfloor = \sum_{i=1}^{n} \lfloor i(\alpha' + x) \rfloor = \sum_{i=1}^{n} \lfloor i \cdot \alpha' \rfloor + x \sum_{i=1}^{n} i = S_{\alpha'}(n) + x \left(\frac{n(n+1)}{2} \right)$$

   Since by definition $1 < \alpha' < 2$, we reduce the problem to case 2.

2. $1 < \alpha < 2$. We can solve explicity for the complemetary value β:

$$\frac{1}{\alpha} + \frac{1}{\beta} = 1 \Rightarrow \beta = \frac{\alpha}{\alpha - 1}$$

   Let $m = \lfloor n\alpha \rfloor$ and $k = \lfloor m/\beta \rfloor$. Since B_α and B_β partition the natural numbers, the sum of the first m numbers is equal to the sum of the first n numbers in B_α and the sum of the first k numbers in B_β. That is,

$$\sum_{i=1}^{m} i = \sum_{i=1}^{n} B_\alpha(i) + \sum_{i=1}^{k} B_\beta(i) \Rightarrow \sum_{i=1}^{n} B_\alpha(i) = \frac{m(m+1)}{2} - \sum_{i=1}^{k} B_\beta(i)$$

Note that because n is an integer, and each integer from 1 to m is either one of the n values summed by B_α or one of the k values summed by B_β,

$$m = n + k \Rightarrow k = m - n = \lfloor n\alpha \rfloor - \lfloor n \rfloor = \lfloor n(\alpha - 1) \rfloor$$

$$\sum_{i=1}^{n} B_\alpha(i) = \frac{(n+k)(n+k+1)}{2} - \sum_{i=1}^{k} B_\beta(i)$$

Thus, to compute the original sum, we just need to compute the sum of the first k elements of B_β. Note that since $1 < \alpha < 2$, $\beta > 2$, and this second sum falls under case 1. Most importantly, we are only summing k numbers, and since $\alpha - 1 < 1$, $k = \lfloor n(\alpha - 1) \rfloor < n$, so with each step of the algorithm we sum fewer and fewer numbers untill we reach $n = 0$.

3. $0 < \alpha < 1$. Similar to case 1, there exists an irrational number α' between 1 and 2 such that $\alpha = \alpha' - 1$. Therefore,

$$S_\alpha(n) = \sum_{i=1}^{n} \lfloor i \cdot \alpha \rfloor = \sum_{i=1}^{n} \lfloor i(\alpha' - 1) \rfloor = S_{\alpha'}(n) - \frac{n(n+1)}{2}$$

Since by definition $1 < \alpha' < 2$, we reduce the problem to case 2.

The structure of the algorithm can be seen in the recursive nature of the three cases. Since each iteration of case 2 reduces the number of elements we are summing up, the algorithm will eventually terminate, as it is guaranteed to reach the base case of $n = 0$. A Python implementation of the algorithm is below. Although `alpha_2` is computed using the fractional component, which seems to imply that `alpha` is rational, the algorithm is valid as long as the floating point representation is sufficiently precise.

```python
def compute_sum(alpha, n):
    if n == 0:
        return 0
    if alpha > 2:
        # 1 + the fractional part of alpha
        alpha_2 = 1 + alpha - int(alpha)
        x = alpha - alpha_2
        return x*n*(n+1)/2 + compute_sum(alpha_2, n)
    elif 1 < alpha < 2:
        beta = alpha/(alpha - 1)
        m = int(n * alpha)
        k = m - n
        return m*(m+1)/2 - compute_sum(beta, k)
    else:
        alpha_2 = alpha + 1
        return compute_sum(alpha_2, n) - n*(n+1)/2
```

Algorithm for $\alpha = \sqrt{2}$

Although the above algorithm does work for $\alpha = \sqrt{2}$, we can use a nice property of $\sqrt{2}$ to obtain a much cleaner algorithm. Consequently, the algorithm is faster and requires fewer decimal points to obtain correct answers. The general algorithm above requires a large number of decimal points in order to maintain correctness ($d \approx 200$) for an input of 100 digits, while the specific algorithm for $\sqrt{2}$ only requires 101. For $\alpha = \sqrt{2}$, the irrational number passed around in each iteration of the algorithm has a periodic nature. Note that

$$\beta = \frac{\sqrt{2}}{\sqrt{2}-1} = 2 + \sqrt{2}$$

From the above formulas for the $1 < \alpha < 2$ case,

$$S_{\sqrt{2}}(n) = \frac{(n+k)(n+k+1)}{2} - S_{2+\sqrt{2}}(k)$$

Applying the $\alpha > 2$ step to $S_{2+\sqrt{2}}(k)$, we have

$$S_{2+\sqrt{2}}(k) = k(k+1) + S_{\sqrt{2}}(k)$$

Substituting this into the previous equation, we obtain the recurrence relation

$$S_{\sqrt{2}}(n) = \frac{(n+k)(n+k+1)}{2} - k(k+1) - S_{\sqrt{2}}(k)$$

Because each term in this expression is an integer, the only place where floating point approximations are required is in the calculation of k. Therefore, we only need the number of decimal places needed to ensure that the value $\lfloor n(\alpha - 1) \rfloor$ is accurate. If $n \leq 10^d$, then d decimal places are needed to ensure that there are sufficient decimal places. Since $\sqrt{2}$ has one digit before the decimal place, in order to calculate $S_{\sqrt{2}}$ up to $n = 10^d$, we need to calculate $\sqrt{2}$ using a precision of $d+1$ digits. A Python implementation of the algorithm is below.

```python
import decimal

rt_2 = decimal.Decimal(2).sqrt() # Can set this precision higher if needed
def compute_sum(n):
    if n == 0:
        return 0
    k = int((rt_2 - 1) * n)
    return int(((n+k)*(n+k+1))/2) - k*(k+1) - compute_sum(k)
```

Finally, in order to speed up the algorithm very slightly, we can change the recursive algorithm into an iterative one, which eliminates the need for setting up function calls and allocating stack frames. Although the performance benefits of rewriting the algorithm are somewhat insignificant, rewriting the algorithm to an iterative one also allows it to be used for larger numbers, as there would not be a limit on the size of n placed by the recursive depth limit.

```
import decimal

rt_2 = decimal.Decimal(2).sqrt()
def compute_sum(n):
    sum = 0
    temp = n
    scale = 1
    while temp > 0:
        k = int((rt_2 - 1) * temp)
        contrib = int(((temp+k)*(temp+k+1))/2) - k*(k+1)
        sum += contrib * scale
        scale = scale * -1
        temp = k
    return sum
```