Range Queries

Today we will examine at a data structure that quickly answers range queries on one dimensional data. The tree that we construct is a balanced binary tree, where each node represents a subrange of the data. For simplicity, we are going to start with a complex solution to a simple problem, and then we see how the techniques apply to a more difficult problems.

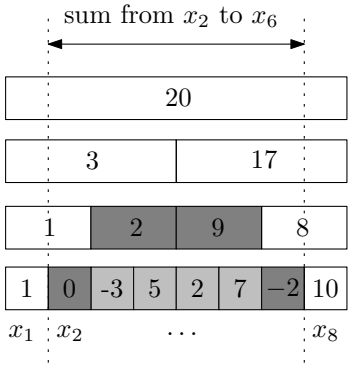
Problem: You are given a sequence of integers $x_1, x_2, \ldots, x_n$. Quickly answer queries of the form "What is the sum of $x_i + x_{i+1} + \ldots + x_j$?", for any $i \leq j$.

Simple Solution: Create an array $S[0 \ldots n]$ where $S[i] = \sum_{k=1}^{i} x_k = S[i-1] + x_i$. Then given a query return $S[j] - S[i-1] = \sum_{k=1}^{j} x_k - \sum_{k=1}^{i-1} x_k = \sum_{k=i}^{j} x_k$. Preprocessing takes $\Theta(n)$ time and space. Queries are answered in $\Theta(1)$ time.

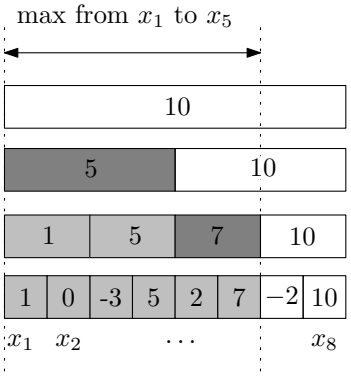
Complex Solution: Partition the range $x_i, x_{i+1}, \ldots, x_j$ into subranges for which we have precomputed sums. Return the sum of the subrange sums.

In this example, we start with the sum of the entire sequence at the top. In each successive level, we split each range in half, until we end up with ranges containing single elements.

When answering a query, we carefully choose how we partition the query range into our subranges: we select subranges contained in the query range (light gray); we only choose such subranges that are not contained in any other such subrange (dark gray).



Why do we use this more complicated solution? Because we can answer other types of queries as well: For example, we can find the maximum of a query range.



How do we represent, construct, and query such a data structure? Recursively! First we start with the representation. Each subrange has a start and end, can be refined into two subranges, and has a representative value.

```
type Range {
   Real value;
   Integer start, end;
   Range left←nil,right←nil;
}
```

We construct the data structure recursively.

```
Algorithm QueryTreeInit(X,start,end)
   Range r
   r.start←start
   r.end←end
   if (start = end) then
      r.value← X[start]
   else
      mid← ⌊(start+end) /2⌋
      r.left ← QueryTreeInit(X, start, mid)
      r.right ← QueryTreeInit(X, mid + 1, right)
      r.value ← r.left.value + r.right.value
   return r
```

What is the runtime of this construction? Let $n = \text{start} - \text{end} + 1 = $ size of range. Then the following recurrence describes the runtime:

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(1) \\ &= 2T(n/2) + \Theta(1) \text{ for powers of 2} \end{aligned}$$

This recurrence is case 1 of the master theorem, so $T(n) \in \Theta(n)$. The same recurrence describes the number of ranges constructed. So the query tree can be constructed in $\Theta(n)$ time and space, which is competitive with the simple solution we outlined earlier.

```
Algorithm QueryTreeSum(T,start,end)
  if (T.end < start) or (T.start > end) then
     return 0
  if (start < T.start) and (T.end < end) then
     return T.value
  mid← ⌊(start+end)/2⌋
  return QueryTreeSum(T.left,start,mid)+
         QueryTreeSum(T.right,mid+1,end)
```

What is the runtime of a query? Rather than using a recurrence relation, we use a counting argument based on a simple observation: in order to spawn a recursive call, the range must intersect the query range but must not be contained in the query range. How many such ranges are there per level? As the two recursive calls are written, one of the query range endpoints (either `mid` or `mid+1`) is a child subrange endpoint. The other query range endpoints (`start` and `end`) are endpoints of an original query. The condition for spawning recursive calls (intersecting but not being contained) is only satisfied if one (but not both) of the original query endpoints is contained in the subrange. There are at most two such intervals per level. Hence, there are at most four recursive calls per level. Using the recursive calls are a barometer, the runtime of this algorithm is $O(h)$, where h is the height of the query tree.

What is the height of the query tree? It satisfies this recurrence:

$$\begin{aligned} h(n) &= \max\{h(\lceil n/2 \rceil), h(\lfloor n/2 \rfloor)\} + 1 \\ &= h(\lceil n/2 \rceil) + 1 \text{ assuming that } h(n) \text{ is non-decreasing} \end{aligned}$$

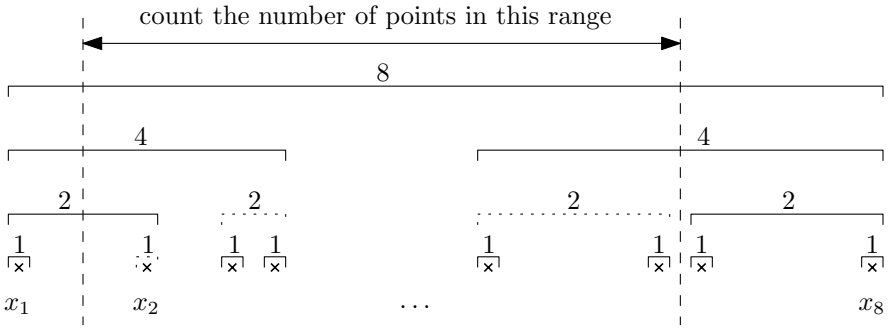
This recurrence satisfies case 2 of the master theorem. So $h(n) \in \Theta(\log n)$. Thus, queries are not as fast as the simple solution that we identified at the beginning but they are close.

We now look at a different problem that can be solved with the same technique:

Problem: Given points $x_1 < x_2 < \ldots < x_n$ answer queries of the form "How many points are contained in the interval $[a, b]$?"

Simple Solution: Binary search for the earliest point x_i such that $a < x_i$. Binary search for the last point x_j such that $x_j < b$. Return $j - i + 1$.

Complex Solution: Same general idea. We create subranges, and for each subrange count the number of points within it. For each query, find a representative set of subranges and add up the totals of the subranges.



The construction is similar.

```
Algorithm QueryTreeInit(X, i, j)
    Range r
    r.start ← X[i]
    r.end ← X[j]
    if (i = j) then
        r.value ← 1
    else
        mid ← ⌊(i + j)/2⌋
        r.left ← QueryTreeInit(X, i, mid)
        r.right ← QueryTreeInit(X, mid + 1, j)
        r.value ← r.left.value + r.right.value
    return r
```

Practice: Write the `QueryTreeCount(`T, a, b`)`.