Fenwick Trees
a.k.a. Binary Indexed Trees, or BITs

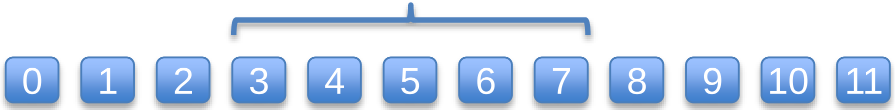
Ahto Truu, Guardtime

The Problem

- Given an array, need to
  - … compute sums of arbitrary segments
  - … and update arbitrary elements
  - … and do both efficiently

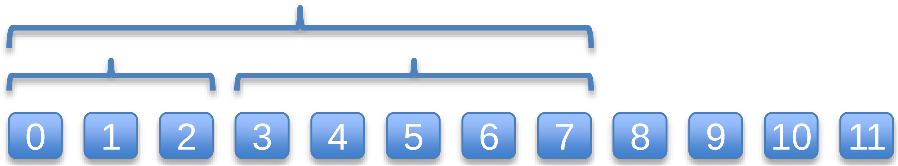
Obvious Solutions

- Keep the original array
  - Updates O(1), sums O(N)



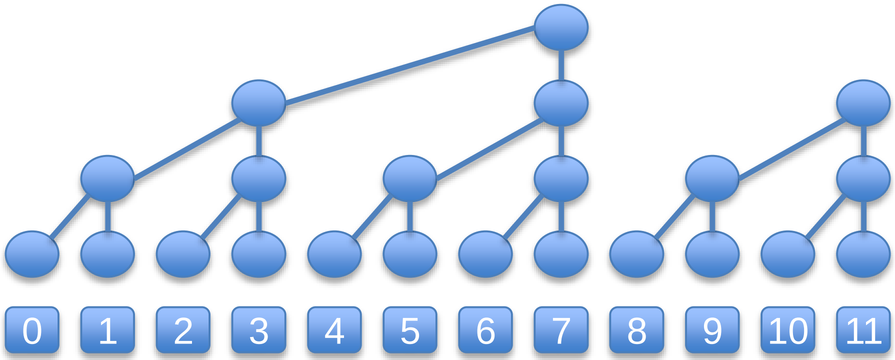
Obvious Solutions

- Keep the original array
  - Updates O(1), sums O(N)
- Use prefix sums
  - Sums O(1), updates O(N)



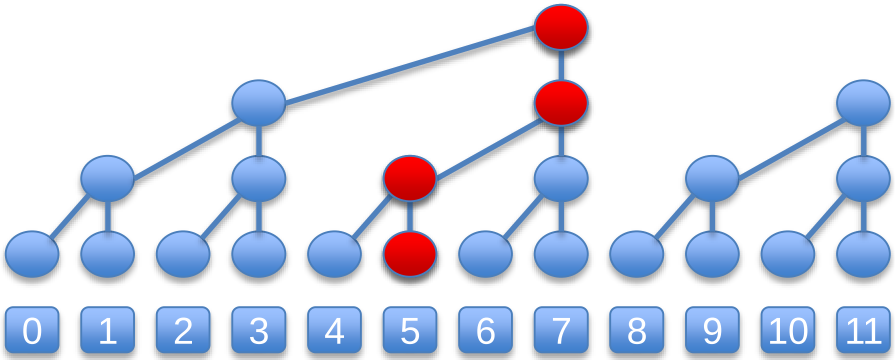
Build an Index

- A binary tree on top of the array
  - Leaves contain original array elements
  - Each parent node is sum of the children



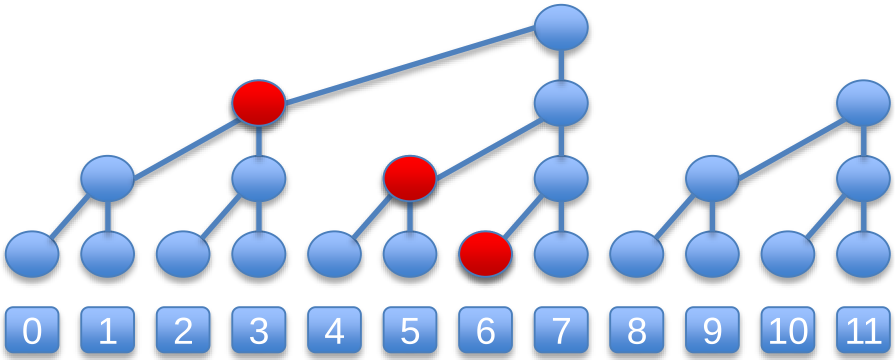
Build an Index

- A binary tree on top of the array
  - Leaves contain original array elements
  - Each parent node is sum of the children
- Updates O(log(N))



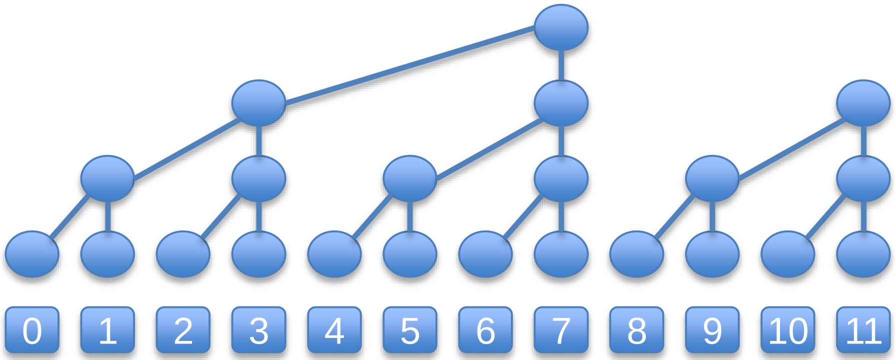
Build an Index

- A binary tree on top of the array
  - Leaves contain original array elements
  - Each parent node is sum of the children
- Updates O(log(N))
- Sums O(log(N))



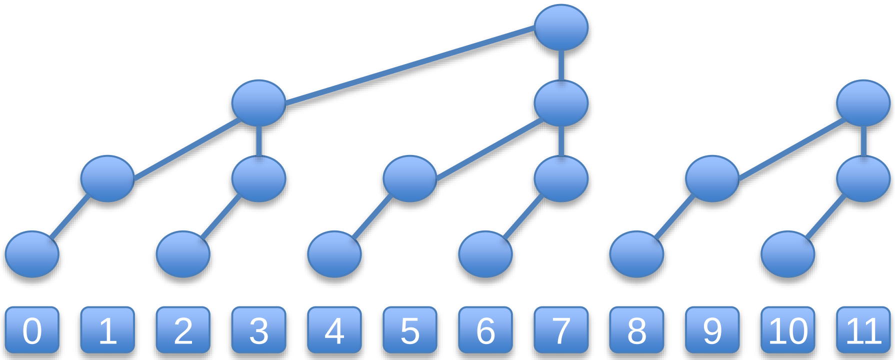
Skimping on Memory

- Each parent is sum of the children



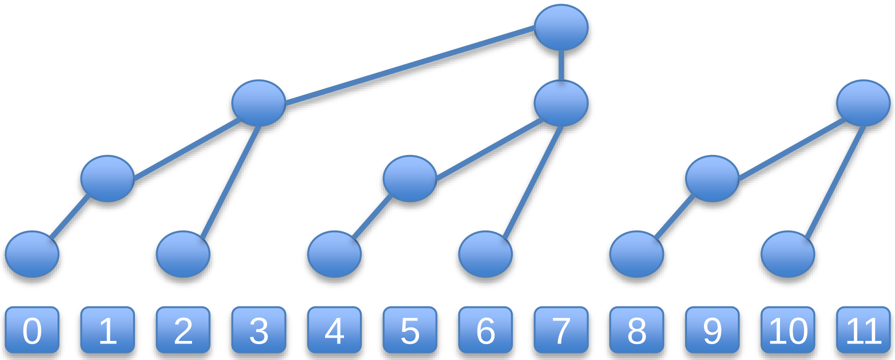
Skimping on Memory

- Each parent is sum of the children
  - … so we only need to keep one child



Skimping on Memory

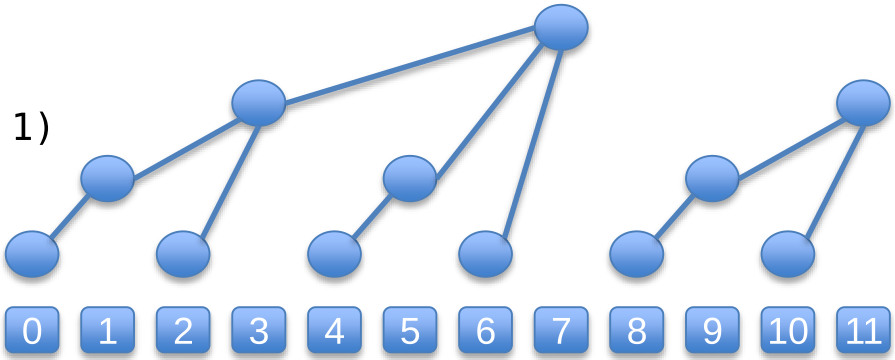
- Each parent is sum of the children
  - … so we only need to keep one child



Skimping on Memory

- Each parent is sum of the children
  - … so we only need to keep one child
  - … so we can keep the tree in the same array

```
void fenwick_init(int a[], int n) {
  for (int i = 0; i < n; ++i)
    for (int m = 1; (i & m) == m; m <<= 1)
      a[i] += a[i - m];
}
```

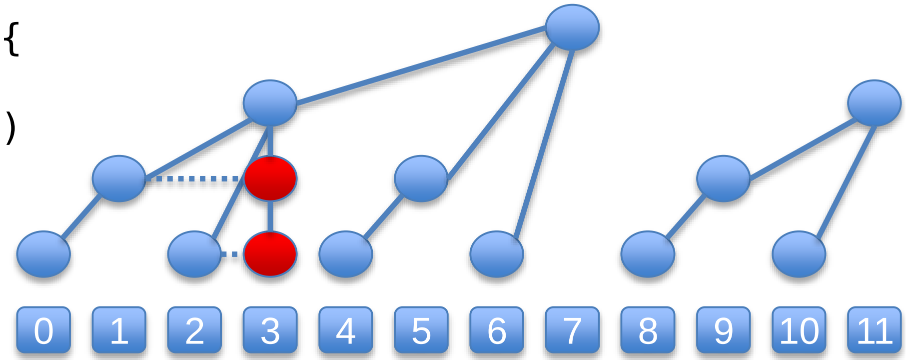


N+N/2+N/4+… ≈ 2N operations to turn the array into tree

Usage: Reads

- Each parent is sum of the children
  - … so we can recover the other child
- Amortized constant time

```
int fenwick_get(int a[], int n, int i) {
  int v = a[i];
  for (int m = 1; (i & m) == m; m <<= 1)
    v -= a[i - m];
  return v;
}
```

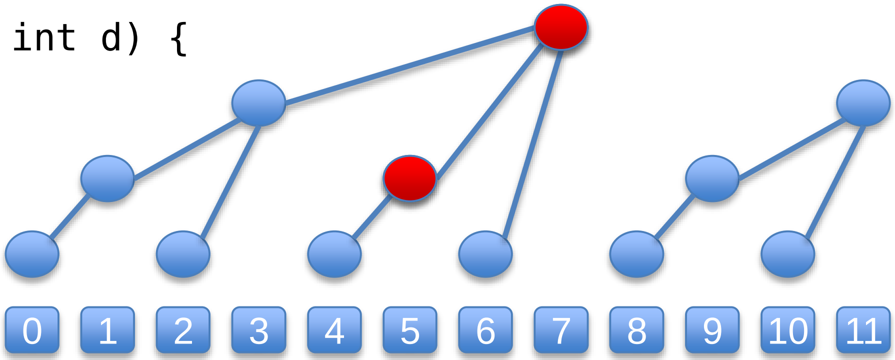


M+M/2+M/4+… ≈ 2M operations for M queries on average

Usage: Updates

- Each parent is sum of the children
  - … so we need to update nodes on the path to root
- This is O(log(N))

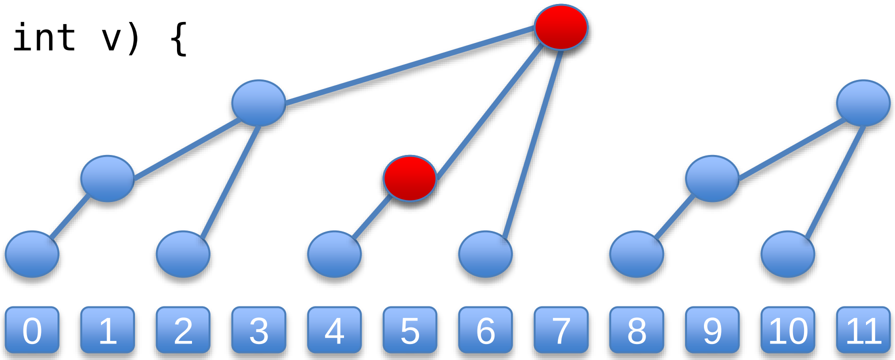
```
void fenwick_inc(int a[], int n, int i, int d) {
  a[i] += d;
  for (int m = 1; i + m < n; m <<= 1)
    if ((i & m) == 0) {
      i += m;
      a[i] += d;
    }
}
```



Usage: Updates

- Each parent is sum of the children
  - … so we need to update nodes on the path to root
- This is O(log(N))

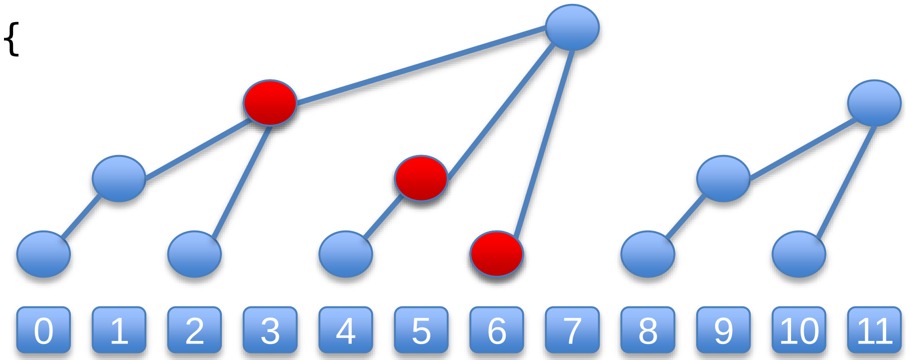
```
void fenwick_set(int a[], int n, int i, int v) {
  int d = v - fenwick_get(a, n, i);
  fenwick_inc(a, n, i, d);
}
```



Usage: Sums

- Each array element is root of a subtree
  - … so we need to just collect the correct ones
- This is O(log(N))

```
int fenwick_sum(int a[], int n, int k) {
  int s = 0;
  for (int m = 1; m <= k; m <<= 1)
    if ((k & m) == 0)
      k += m;
    else
      s += a[k - m];
  return s;
}
```



Fenwick Trees

- Invented by Peter M. Fenwick in 1993
  - Software—Practice and Experience, March 1994
- My code uses slightly different indexing
  - More convenient when array length not a power of 2
- http://github.com/ahtotruu/fenwick/

Questions?

Ahto Truu, Guardtime
ahto.truu@guardtime.com