Fenwick Trees

AKA Binary Index Trees

Overview

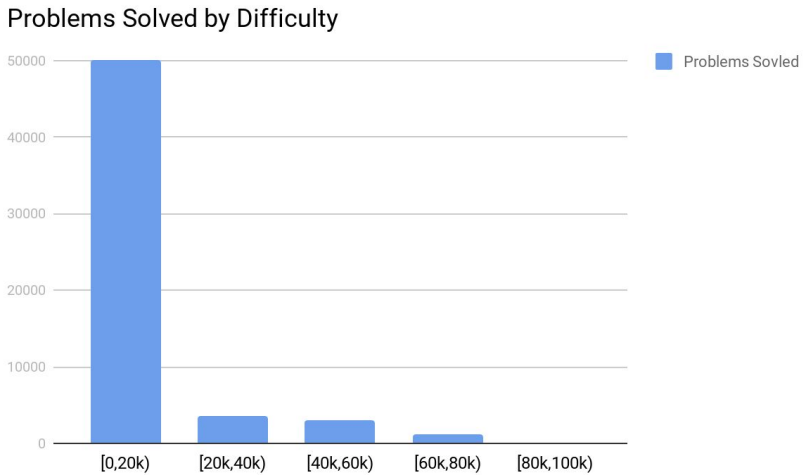
- The Problem
- The Ugly
- The Bad
- The Good

Motivation

- Competitive programming
- We are making a data structure to store problems we solved
- Each problem has a difficulty of 0 to 100k

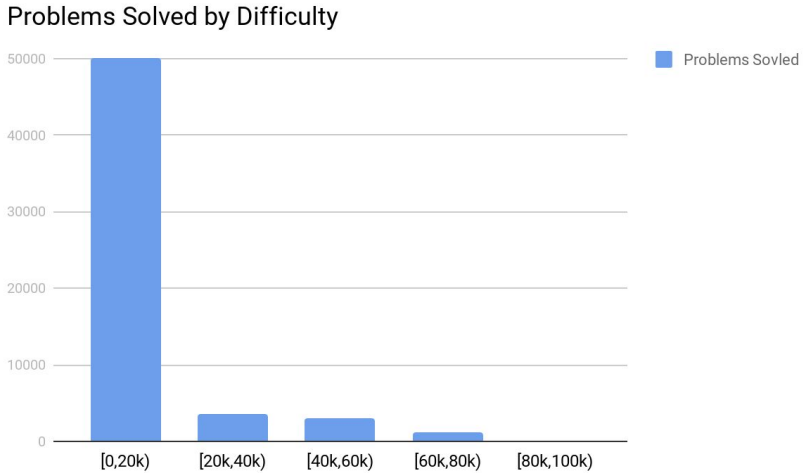
Motivation

- Competitive programming
- We are making a data structure to store problems we solved
- Each problem has a difficulty of 0 to 100k



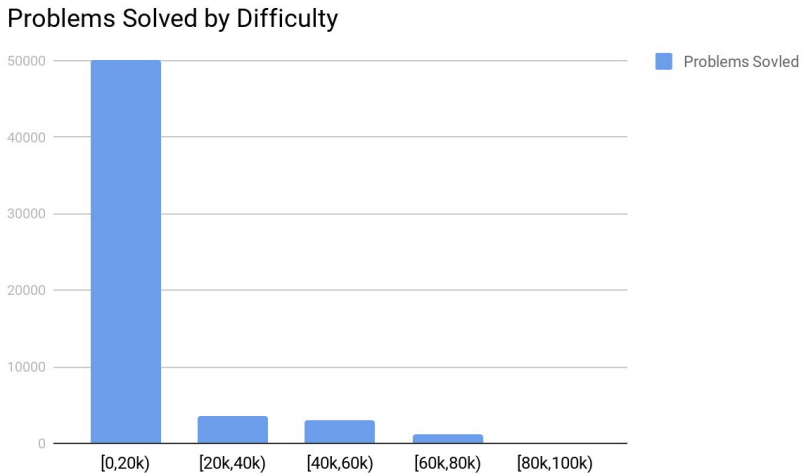
Motivation

- Competitive programming
- We are making a data structure to store problems we solved
- Each problem has a difficulty of 0 to 100k
- Obviously 0 is hardest



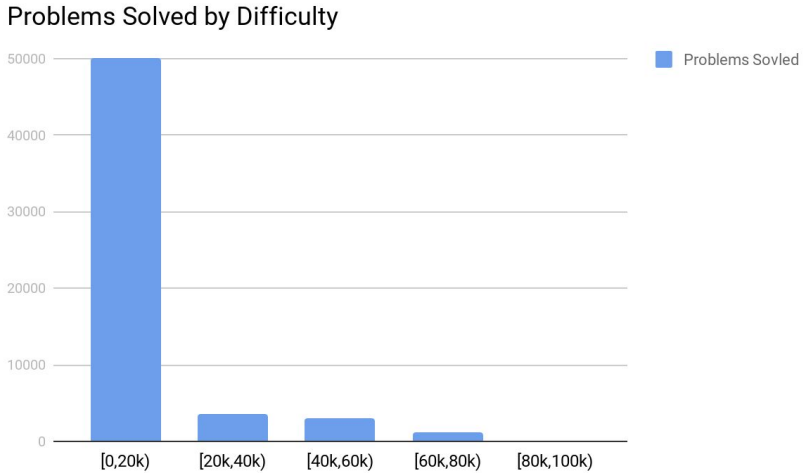
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges



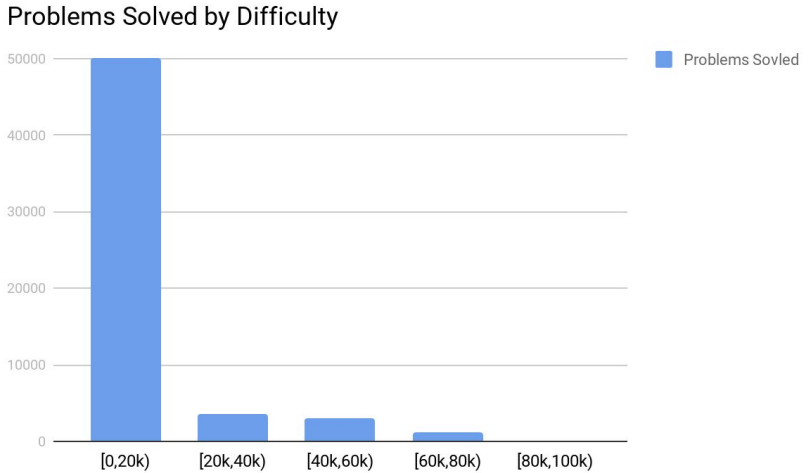
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea 1
  - Loop through the array of problems and update a counter if the problem is of the right difficulty



Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea 1
  - Loop through the array of problems and update a counter if the problem is of the right difficulty

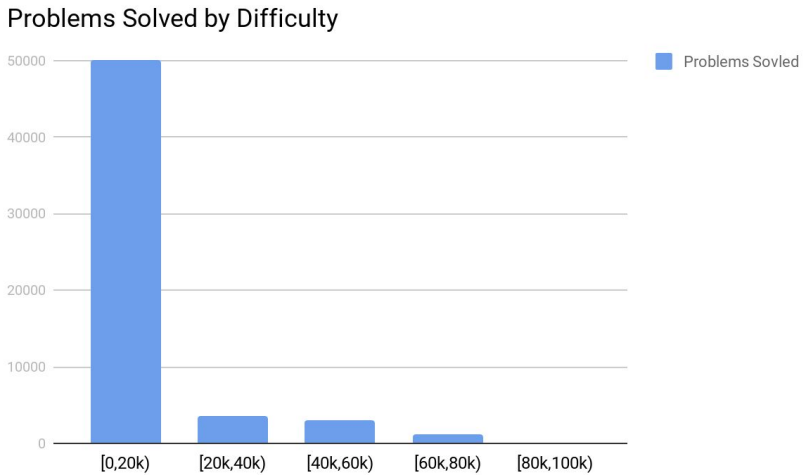


O(N) where N is number of problems

Motivation (cont.)

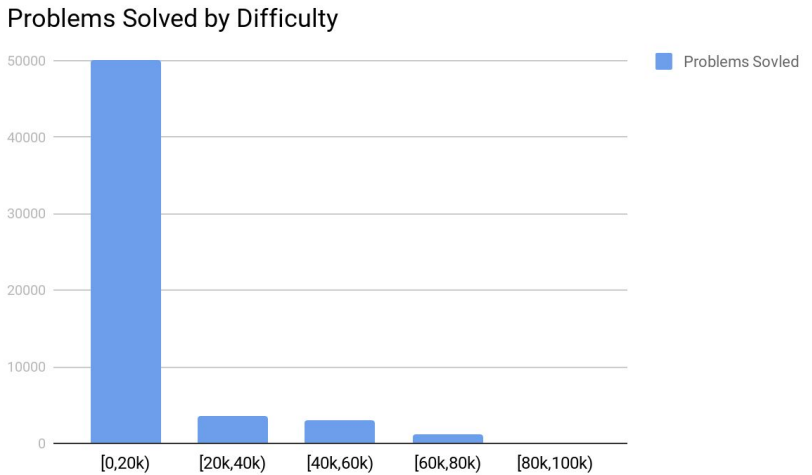
- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea 1
  - Loop through the array of problems and update a counter if the problem is of the right difficulty
- But Arup asks a LOT of questions

O(N) where N is number of problems



Motivation (cont.)

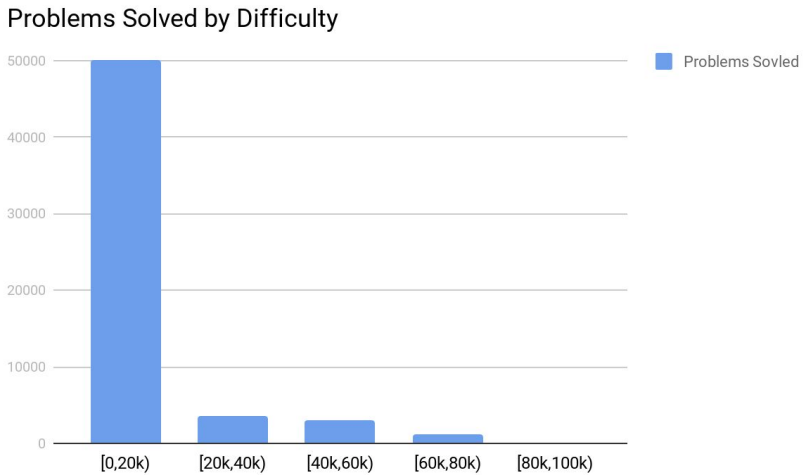
- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea 1
  - Loop through the array of problems and update a counter if the problem is of the right difficulty
- But Arup asks a LOT of questions
- 100k+ queries



O(N) where N is number of problems

Motivation (cont.)

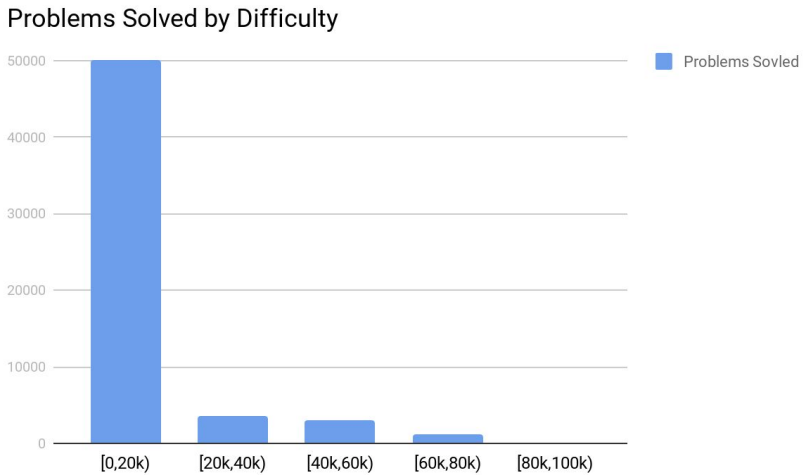
- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea 1
  - Loop through the array of problems and update a counter if the problem is of the right difficulty
- But Arup asks a LOT of questions
- 100k+ queries



O(NQ) where N is number of problems and Q is number of queries
10^{10}

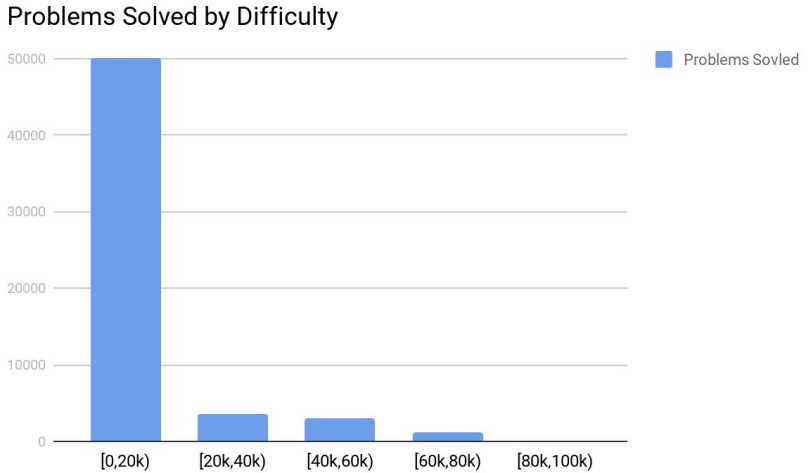
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea



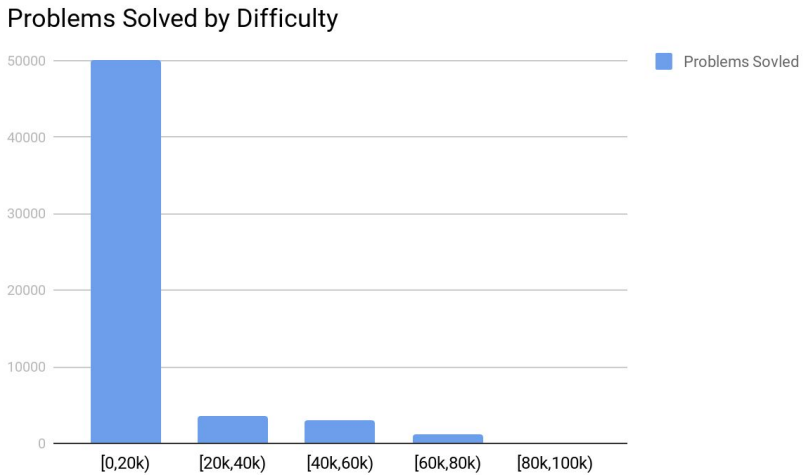
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea
  - Precompute the sums from 0 to i for all i [prefix sum]



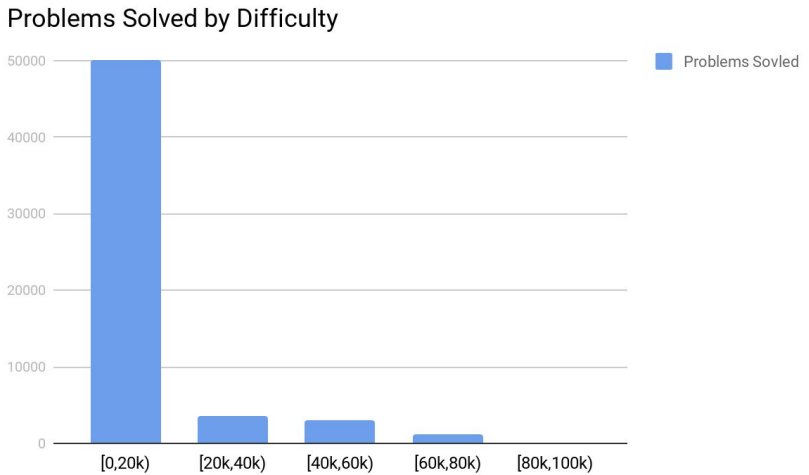
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea
  - Precompute the sums from 0 to i for all i [prefix sum]
- Each query takes O(1) time



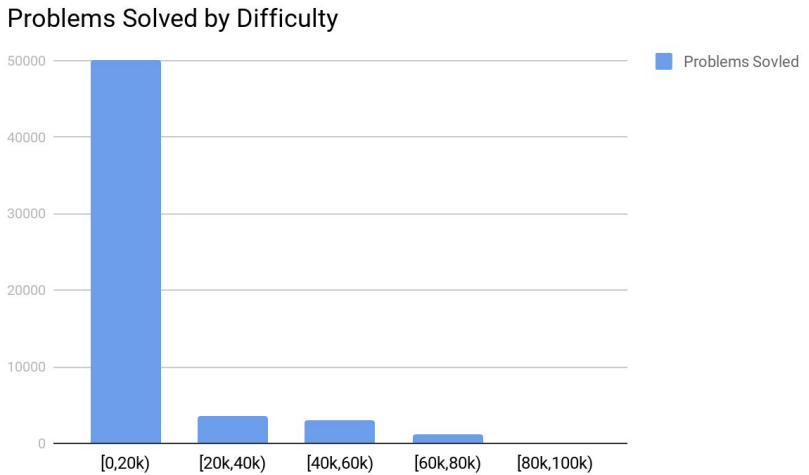
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea
  - Precompute the sums from 0 to i for all i [prefix sum]
- Each query takes O(1) time
  - Inclusion exclusion



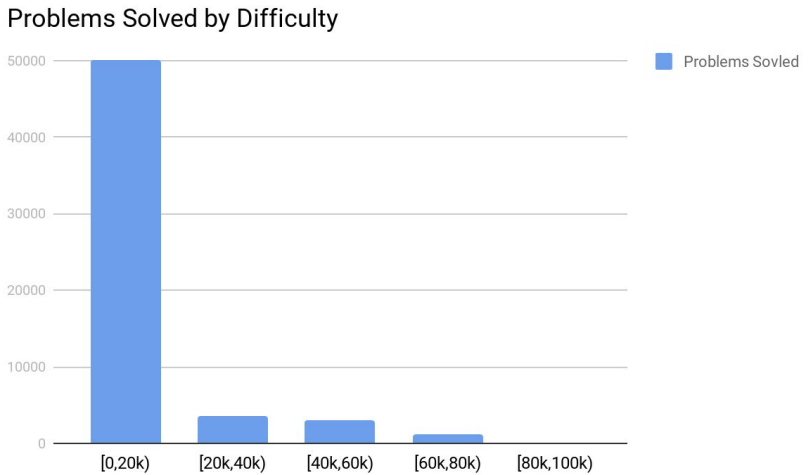
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea
  - Precompute the sums from 0 to i for all i [prefix sum]
- Each query takes O(1) time
  - Inclusion exclusion
  - Add up to upper bound subtract lower



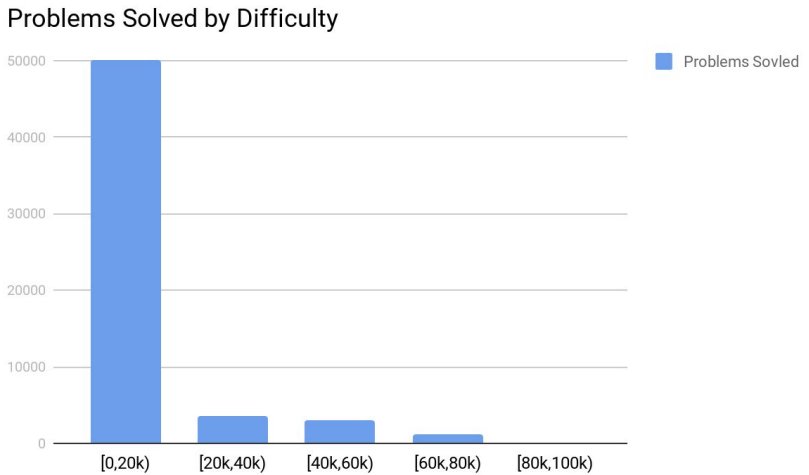
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea
  - Precompute the sums from 0 to i for all i [prefix sum]
- Each query takes O(1) time
  - Inclusion exclusion
  - Add up to upper bound subtract lower
- BUT I get better at coding...
  - A problem's difficulty can change



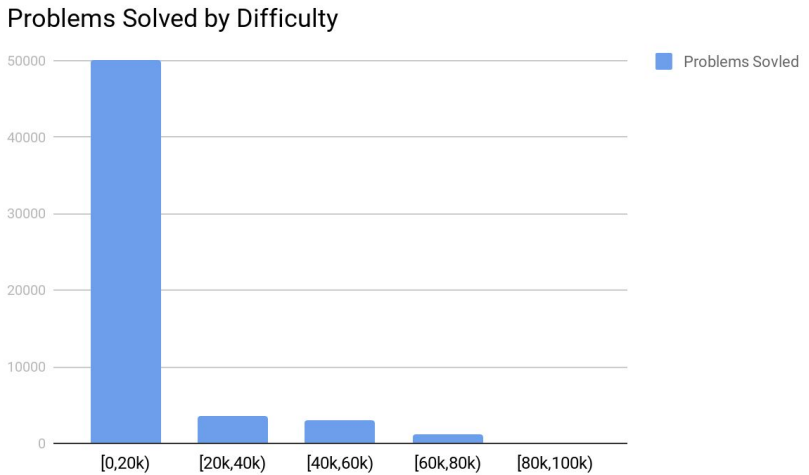
Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea
  - Precompute the sums from 0 to i for all i [prefix sum]
- Each query takes O(1) time
  - Inclusion exclusion
  - Add up to upper bound subtract lower
- BUT I get better at coding...
  - A problem's difficulty can change
- Changing a problem's difficulty by d
  - => Changing d prefix sums



Motivation (cont.)

- My friend, Arup, wants to know how many problems I solved in MANY different ranges
- Idea
  - Precompute the sums from 0 to i for all i [prefix sum]
- Each query takes O(1) time
  - Inclusion exclusion
  - Add up to upper bound subtract lower
- BUT I get better at coding...
  - A problem's difficulty can change
- Changing a problem's difficulty by d
  - => Changing d prefix sums
- IF d is around 100k
  - 100k changes is 10^{10}



The Ugly

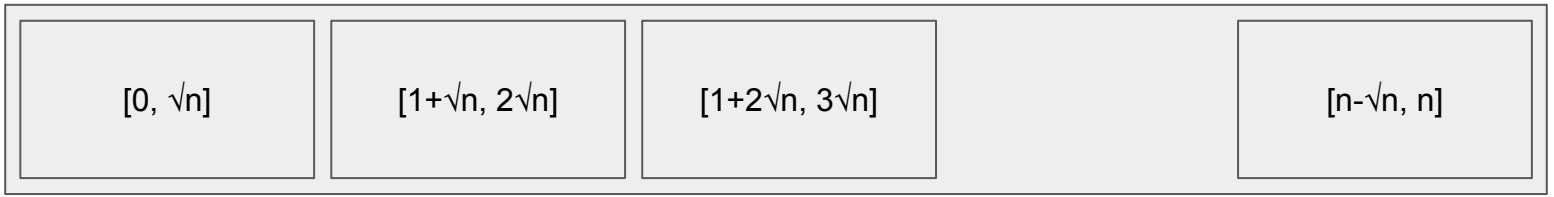
- Many solutions seem promising but fall short
  - Modify prefix sum
  - loop over all problems
  - binary search + resort array each change

The Ugly

- Many solutions seem promising but fall short
  - Modify prefix sum
  - loop over all problems
  - binary search + resort array each change
- Don’t do these please
  - Your (and possibly my) computer will cry

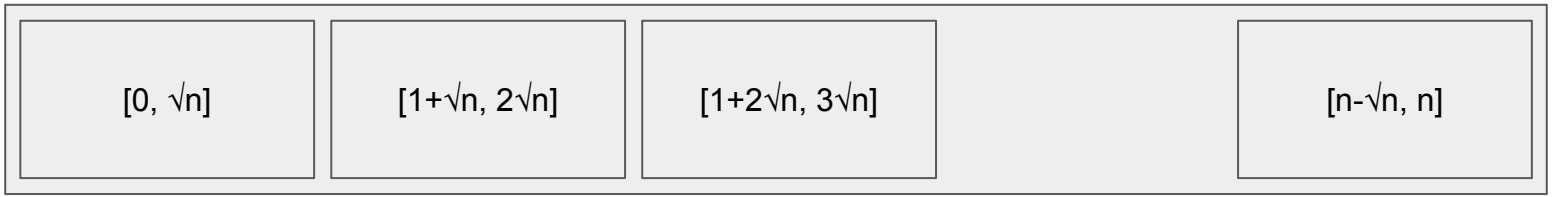
The Bad

- A bit complex, but we can break the difficulty into buckets of sqrt N size



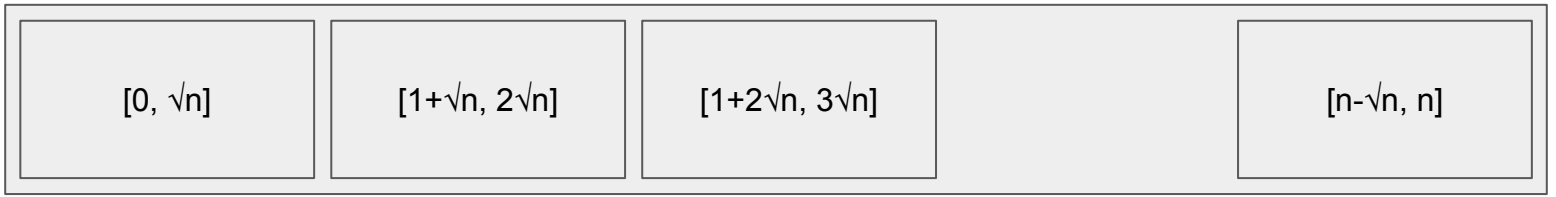
The Bad

- A bit complex, but we can break the difficulty into buckets of sqrt N size
- When updating…
  - We can update the buckets we need (source+dest)
  - Update the individual location as well



The Bad

- A bit complex, but we can break the difficulty into buckets of sqrt N size
- When updating…
  - We can update the buckets we need (source+dest)
  - Update the individual location as well
- To sum a range…
  - We add all the buckets completely covered (at most root n)
  - We the values manually for those in the ending buckets



The Bad (Example)

- Consider a range of size 16

The Bad (Example)

- Consider a range of size 16

5	3	10	2	4	3	6	2	1	3	0	1	0	0	1	1
20				15				5				2			

The Bad (Example)

- Consider a range of size 16

5	3	10	2	4	3	6	2	1	3	0	1	0	0	1	1
20				15				5				2			

Let’s sum from 1 to 12

The Bad (Example)

- Consider a range of size 16

5	3	10	2	4	3	6	2	1	3	0	1	0	0	1	1
20				15				5				2			

Let’s sum from 1 to 12 (0 indexed of course)

The Bad (Example)

- Consider a range of size 16

5	3	10	2	4	3	6	2	1	3	0	1	0	0	1	1
20				15				5				2			

Let’s sum from 1 to 12 (0 indexed of course)

The sum is 3 + 10 + 2 + **15** + **5** + 0 = 35

The Bad (Analysis)

- Updates
  - A constant number of operations
  - Does not depend on the max value

The Bad (Analysis)

- Updates
  - A constant number of operations
  - Does not depend on the max value
- Summation
  - At most [2 times square root max] summations **in** a bucket
  - At most [square root max] summations **of** buckets
  - Requires some constant times square root number of additions in the worst case

The Bad (Analysis)

- Updates
  - A constant number of operations
  - Does not depend on the max value
- Summation
  - At most [2 times square root max] summations **in** a bucket
  - At most [square root max] summations **of** buckets
  - Requires some constant times square root number of additions in the worst case
- We can do better

The Good

- We can “move effort” into the updates to make the summations faster
- Not all buckets need to be the same size

The Good

- We can “move effort” into the updates to make the summations faster
- Not all buckets need to be the same size
- We will make larger and small buckets
    - Buckets will grow in size exponentially!

The Good

- We can “move effort” into the updates to make the summations faster
- Not all buckets need to be the same size
- We will make larger and small buckets
  - Buckets will grow in size exponentially!

5	3	10	2	4	3	6	2	1	3	0	1	0	0	1	1
8		12		7		8		4		1		0		2	
20				15				5				2			
35								7							
42															

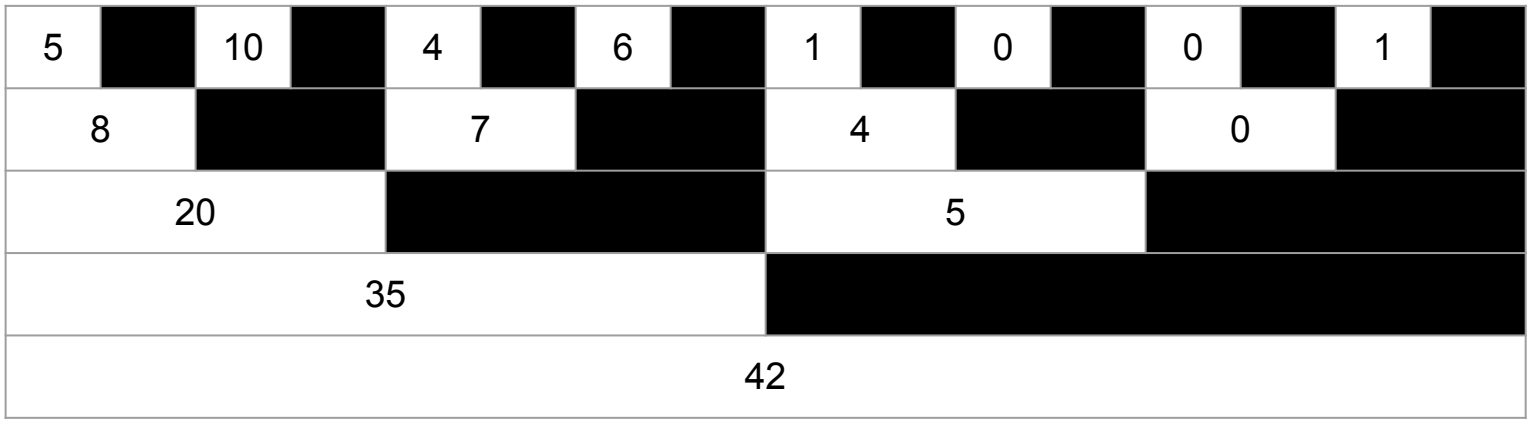
The Good

- We can “move effort” into the updates to make the summations faster
- Not all buckets need to be the same size
- We will make larger and small buckets
  - Buckets will grow in size exponentially!
- But this is not a fenwick tree
- We will eliminate some buckets

5	3	10	2	4	3	6	2	1	3	0	1	0	0	1	1
8		12		7		8		4		1		0		2	
20				15				5				2			
35								7							
42															

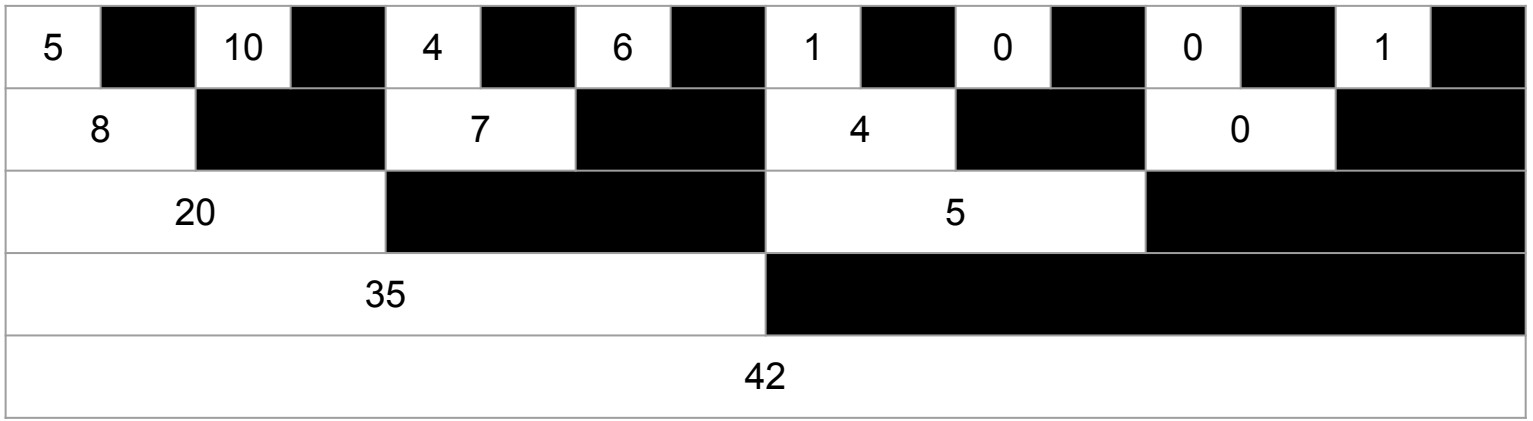
The Good

- We can “move effort” into the updates to make the summations faster
- Not all buckets need to be the same size
- We will make larger and small buckets
  - Buckets will grow in size exponentially!
- But this is not a fenwick tree
- We will eliminate some buckets



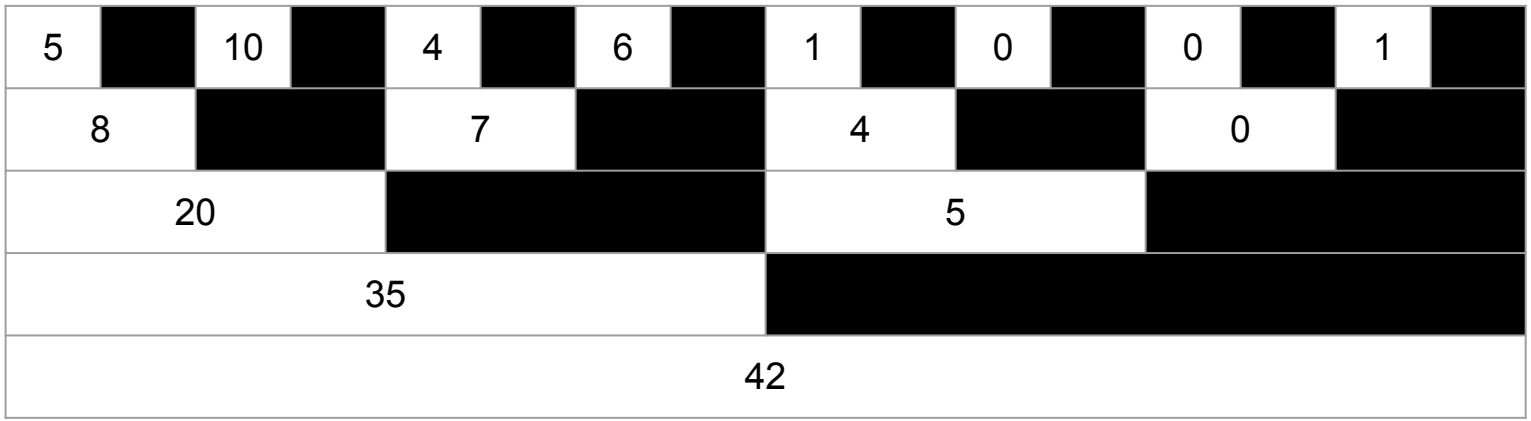
The Good (cont.)

- What's going on?



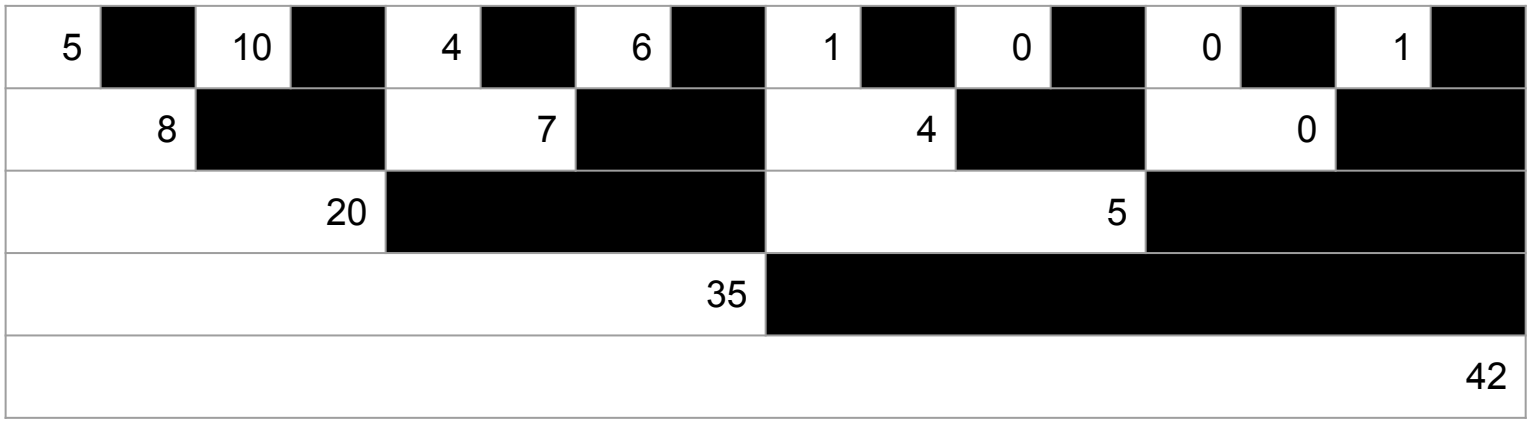
The Good (cont.)

- What's going on?
  - We reduced the number of buckets and we get a nice property



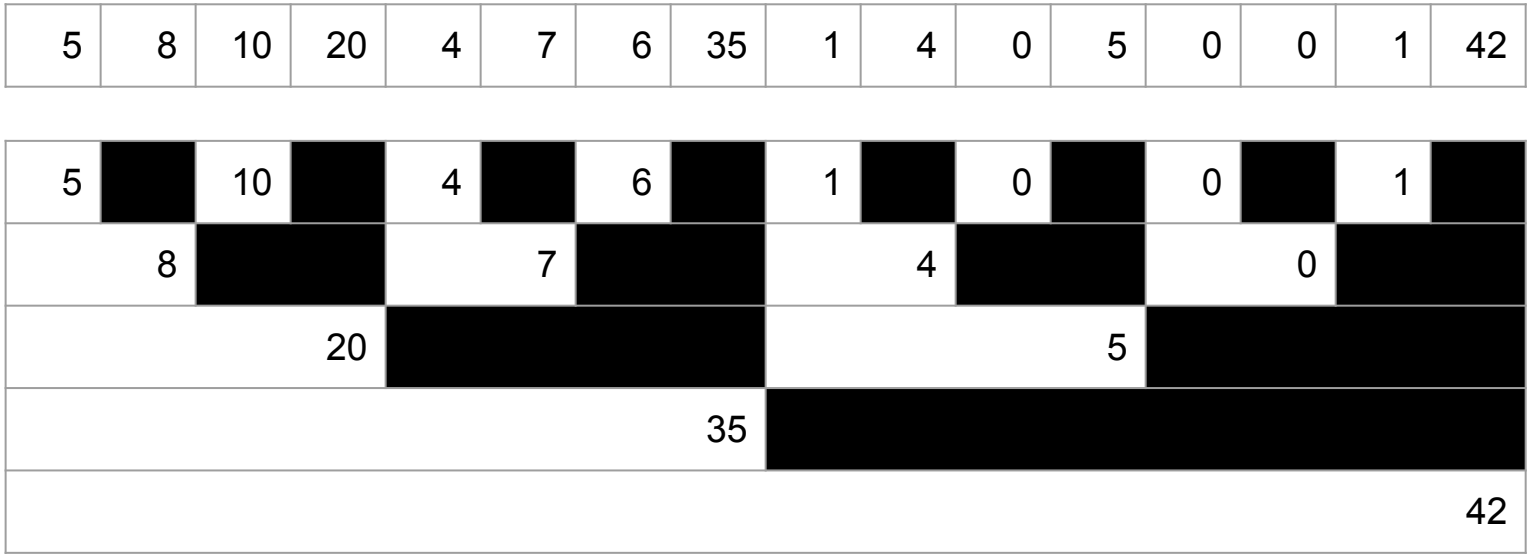
The Good (cont.)

- What's going on?
  - We reduced the number of buckets and we get a nice property



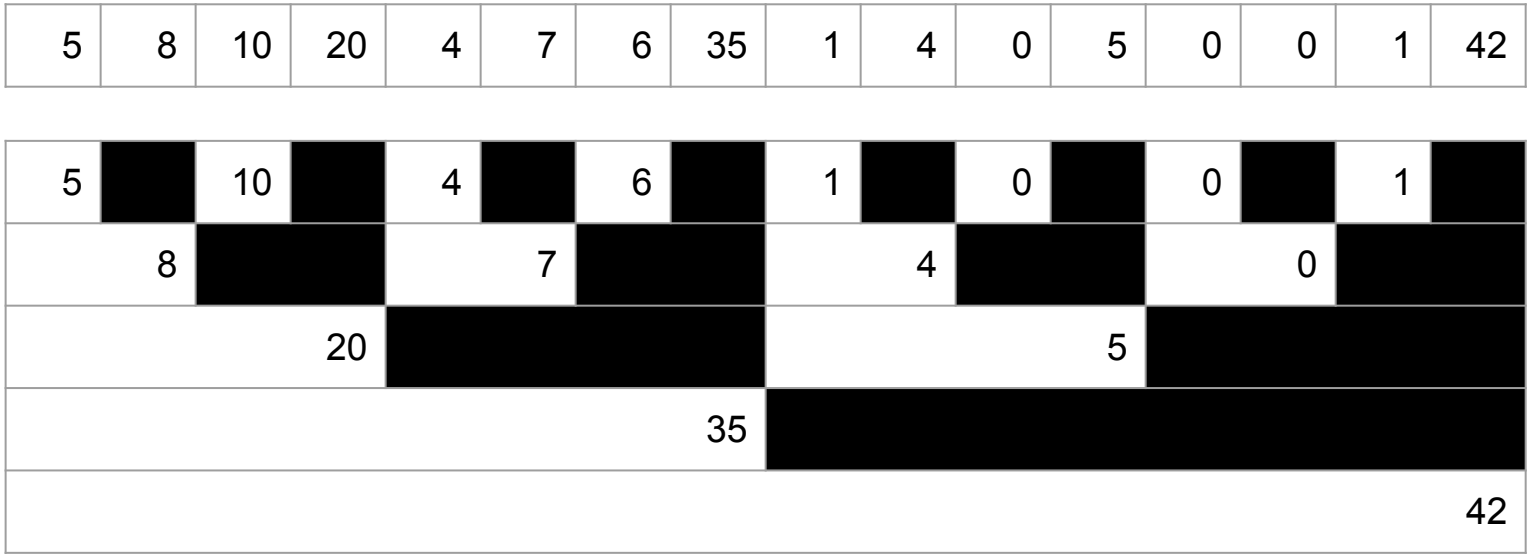
The Good (cont.)

- What's going on?
  - We reduced the number of buckets and we get a nice property
- Only N buckets!



The Good (cont.)

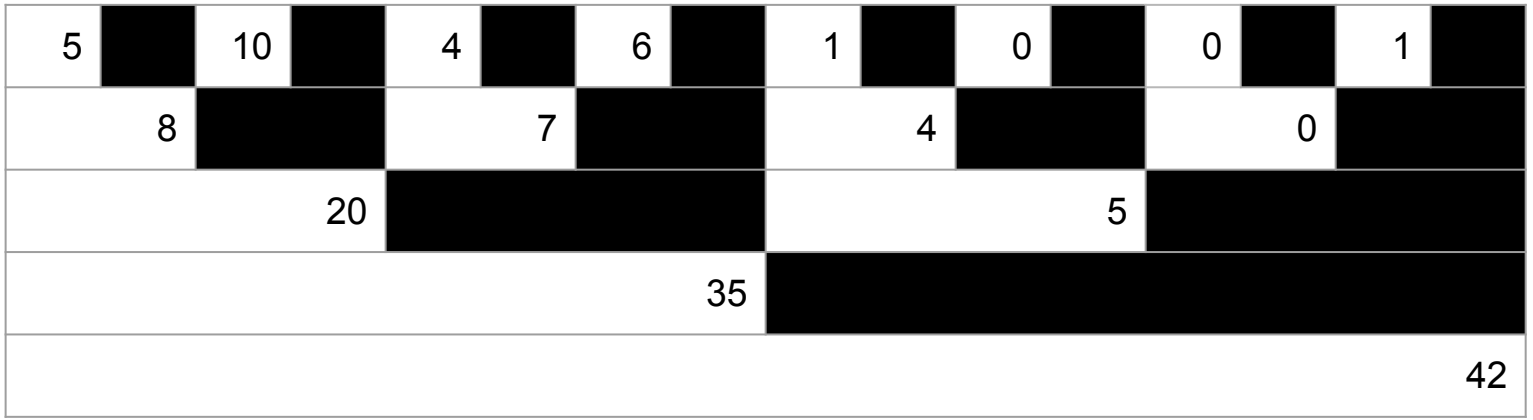
- Addt. We can recover any range from 0 to n (a prefix sum)



The Good (cont.)

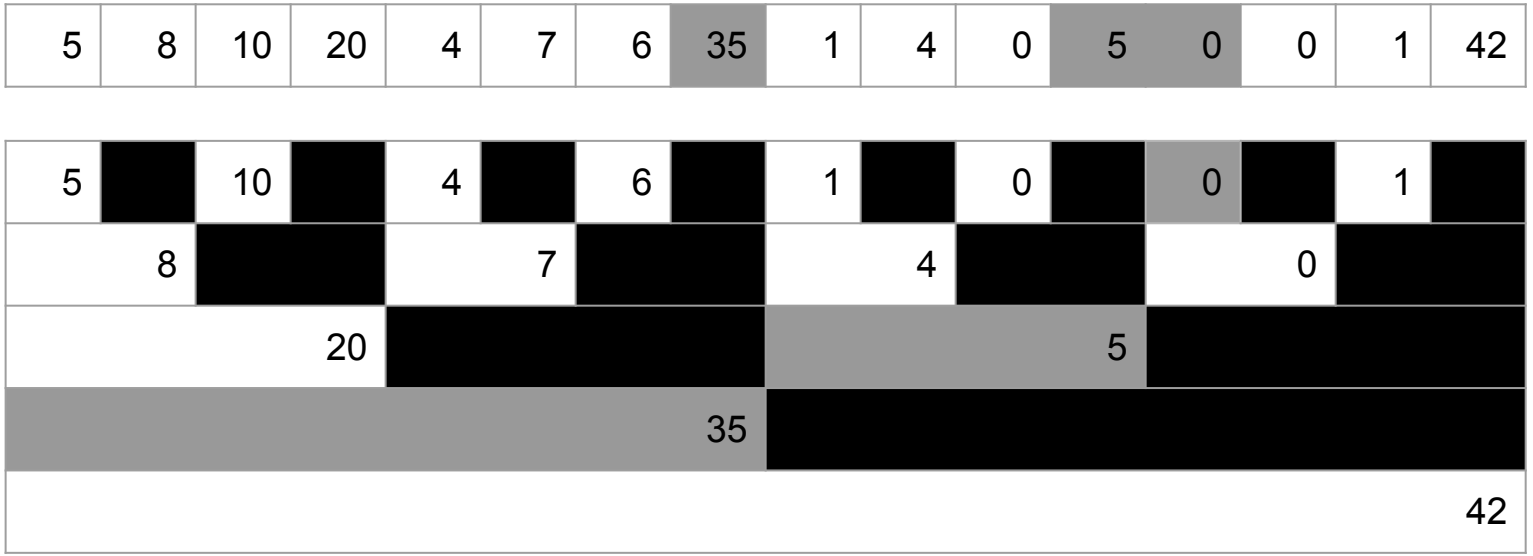
- Addt. We can recover any range from 0 to n (a prefix sum)
- Example 0 to 12

5	8	10	20	4	7	6	35	1	4	0	5	0	0	1	42



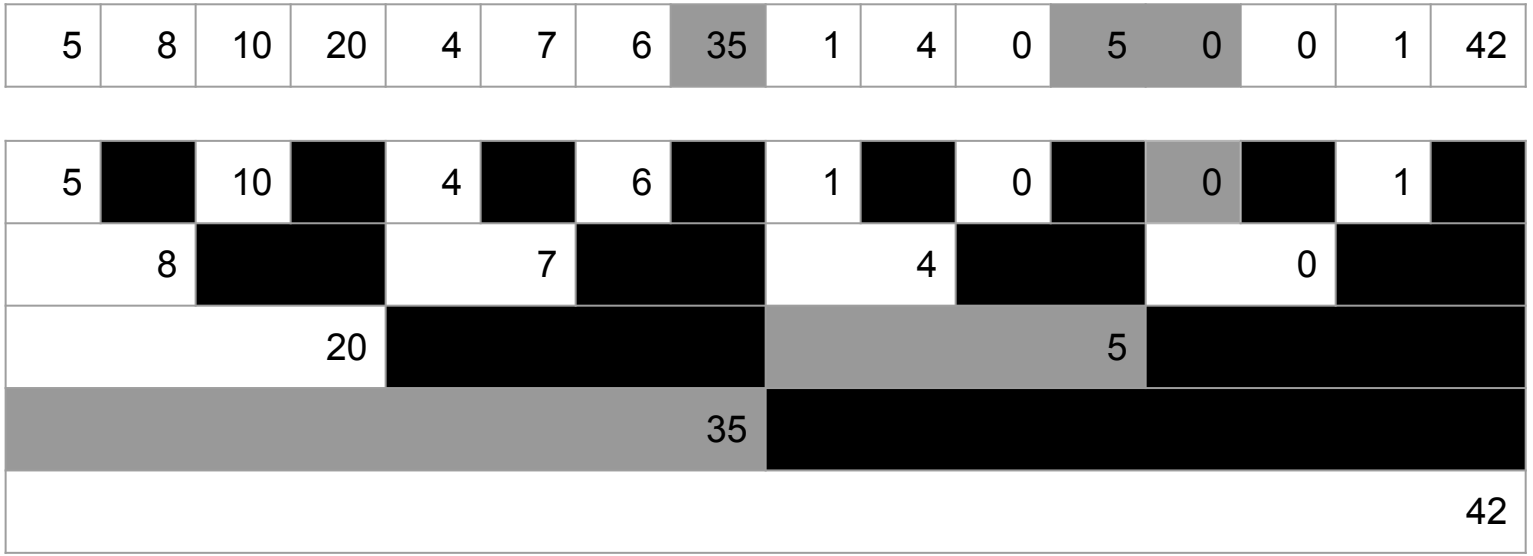
The Good (cont.)

- Addt. We can recover any range from 0 to n (a prefix sum)
- Example 0 to 12



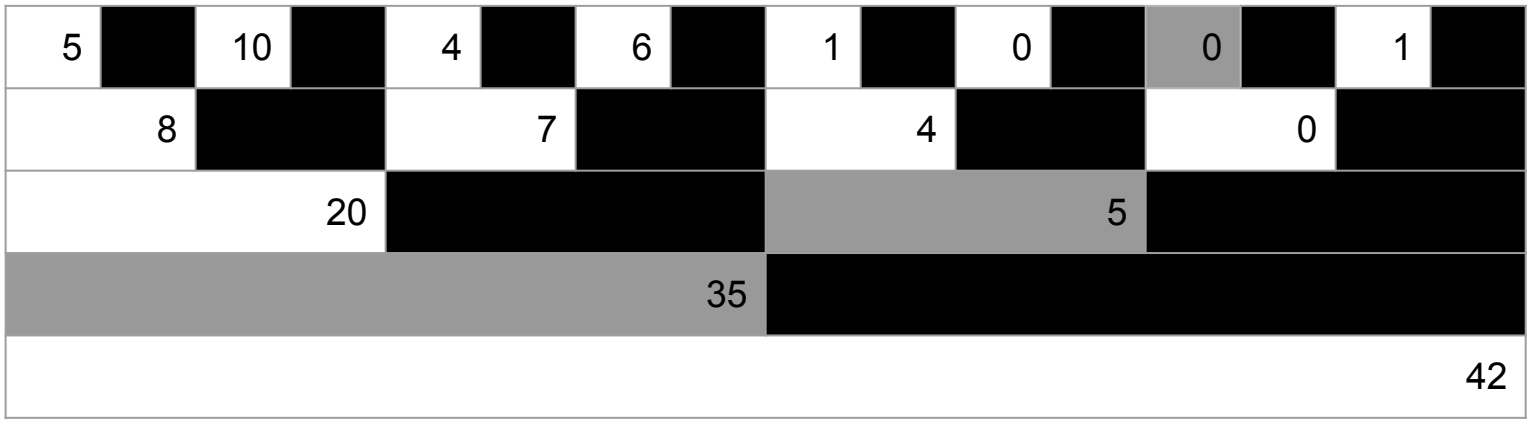
The Good (cont.)

- How do we know which buckets?



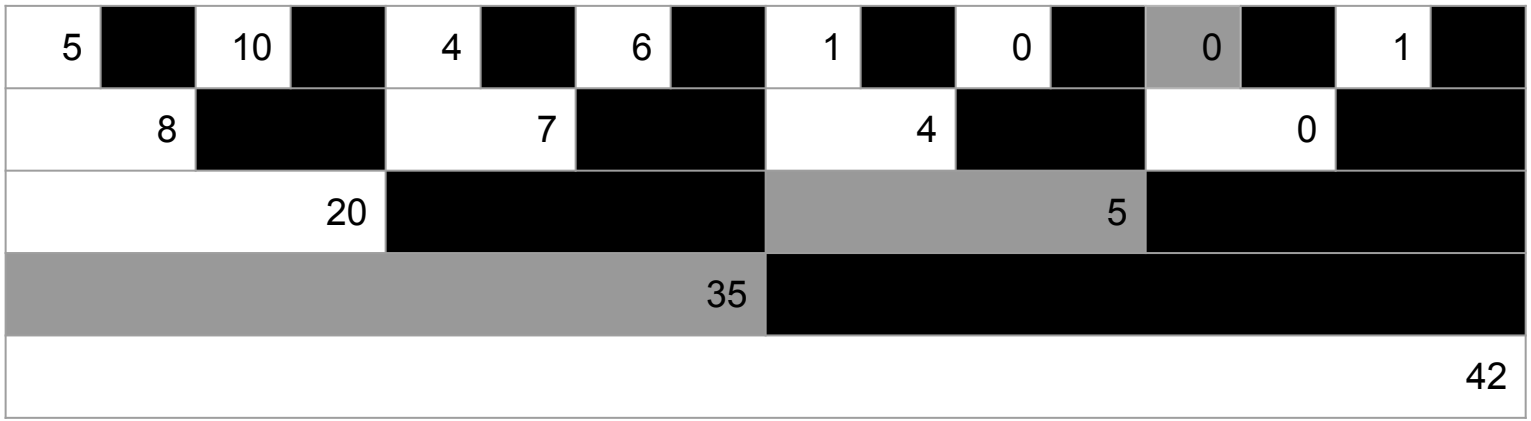
The Good (cont.)

- How do we know which buckets?
  - It’s easier if you 1 index…
  - 12 => 13 = $(1101)_2$



The Good (cont.)

- How do we know which buckets?
  - It's easier if you 1 index…
  - 12 => 13 = $(1101)_2$
  - We have the 2^0 bit on, the 2^2 bit on, and the 2^3 bit on...
  - This tells me we have a 2^0 bucket, a 2^2 bucket, and a 2^3 bucket.
  - The buckets are 13 $(1101)_2$, 12 $(1100)_2$, and 8 $(1000)_2$



The Good Summation

- Sum up the buckets for the bits on in the corresponding buckets

The Good Summation

- Sum up the buckets for the bits on in the corresponding buckets
- Remove the lowest power of 2 and add the bucket until the “index” is 0

The Good Summation

- Sum up the buckets for the bits on in the corresponding buckets
- Remove the lowest power of 2 and add the bucket until the “index” is 0

`x&-x gets the lowest 1 bit of an integer thanks to how 2’s complement works with negative numbers.`

The Good Summation

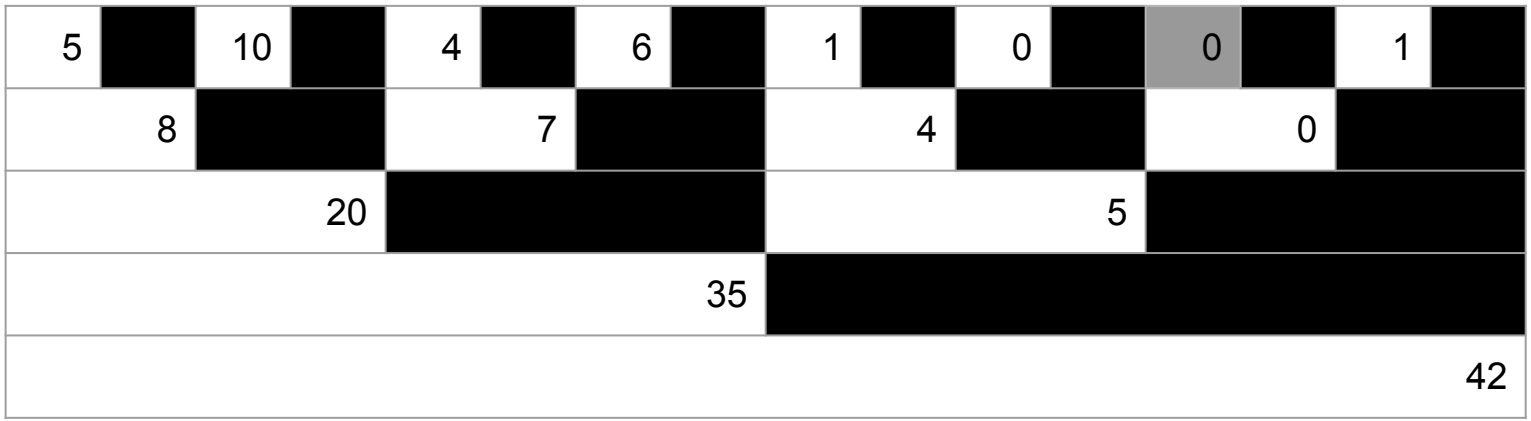
- Sum up the buckets for the bits on in the corresponding buckets
- Remove the lowest power of 2 and add the bucket until the “index” is 0

`x&-x` gets the lowest 1 bit of an integer thanks to how 2’s complement works with negative numbers.

```
int sum (int x) {
   int ans = bucket[++x]; // 1-index and initial bucket
   while (x != 0) ans += bucket[x^=(x&-x)];
   return ans;
}
```

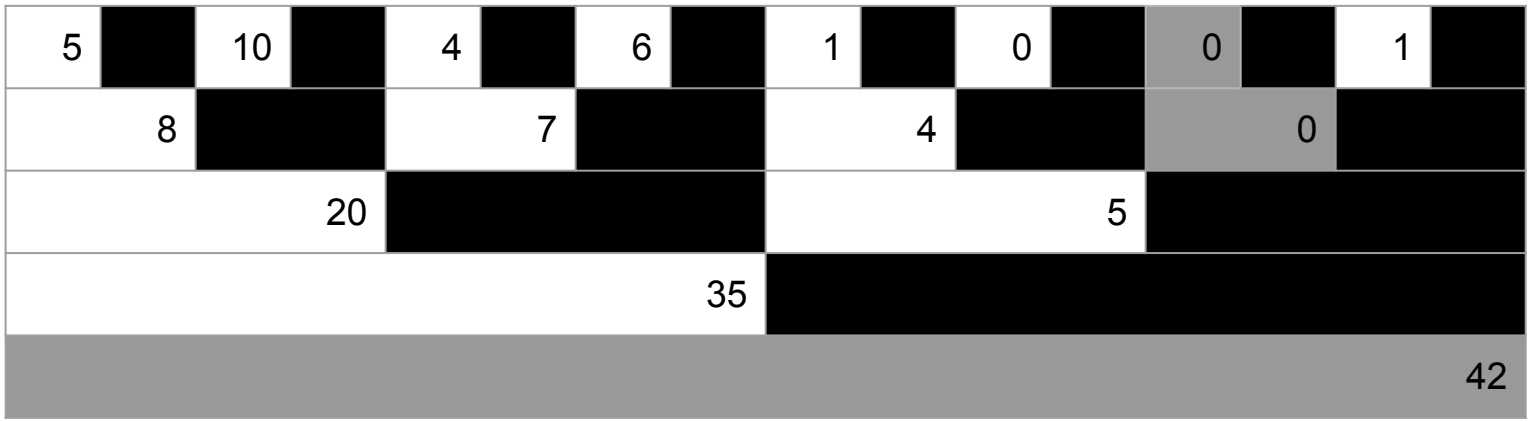
The Good Update

- Example 12



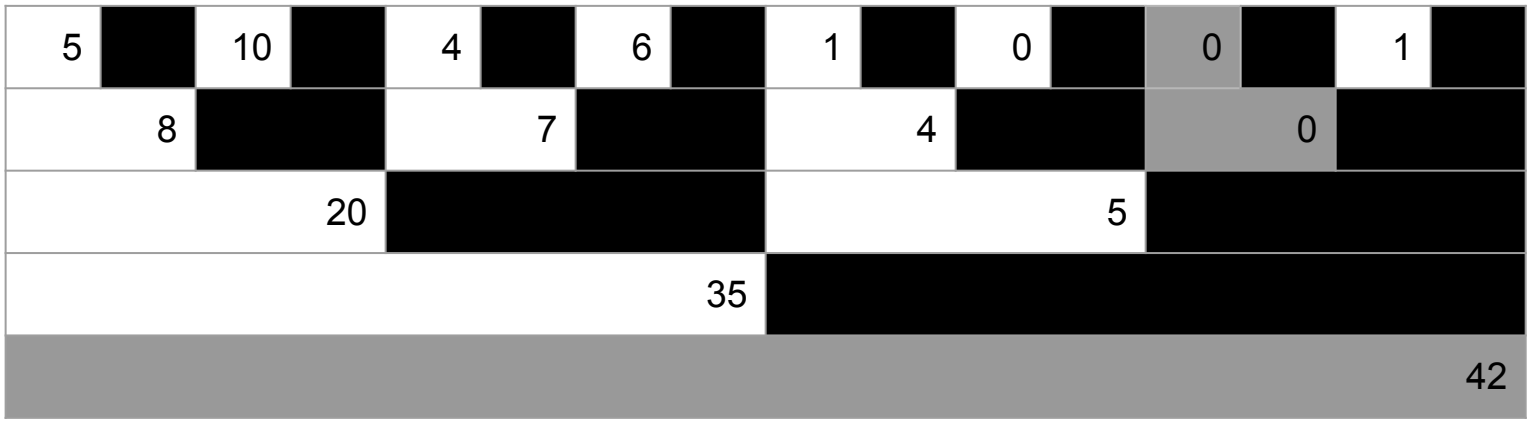
The Good Update

- Example 12
- Update the overlapping buckets



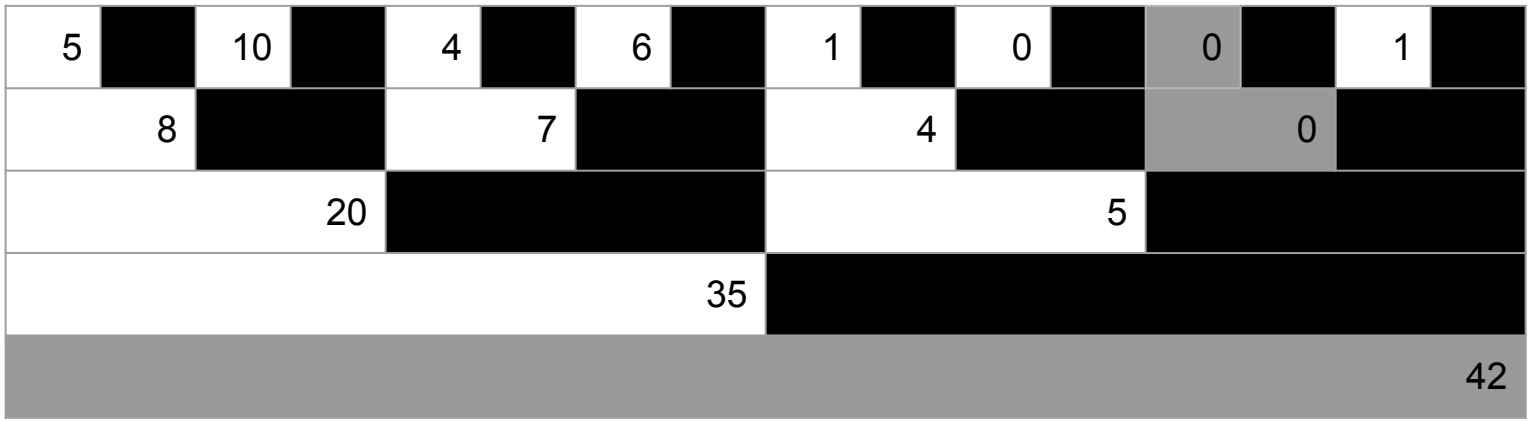
The Good Update

- Example 12
- Update the overlapping buckets
- Again 1 index
  - 13 $(1101)_2$, 14$(1110)_2$, and 16 $(10000)_2$



The Good Update

- Example 12
- Update the overlapping buckets
- Again 1 index
  - 13 $(1101)_2$, 14$(1110)_2$, and 16 $(10000)_2$
- Not as obvious, but we are ADDING the lowest 1 bit.



The Good Update

- Update the buckets until we go out of our array

The Good Update

- Update the buckets until we go out of our array

```
void add(int x, int val) {
   x++; // 1-index
   while (x<max) {
      bucket[x] += val;
      x+=(x&-x)
   }
}
```

VS Point Query Range Update

- What was discussed was a point update and range query

VS Point Query Range Update

- What was discussed was a point update and range query
- A different thought process but the same data structure can be used for range update and point query

VS Point Query Range Update

- What was discussed was a point update and range query
- A different thought process but the same data structure can be used for range update and point query
- Problem
  - I can multi-task. I have many friends I chat online with at the same time. Each friend will talk to me for a continuous segment of time (an interval). I want to know at several distinct points how many people I was talking to. My memory is fuzzy so I might add or remove intervals.

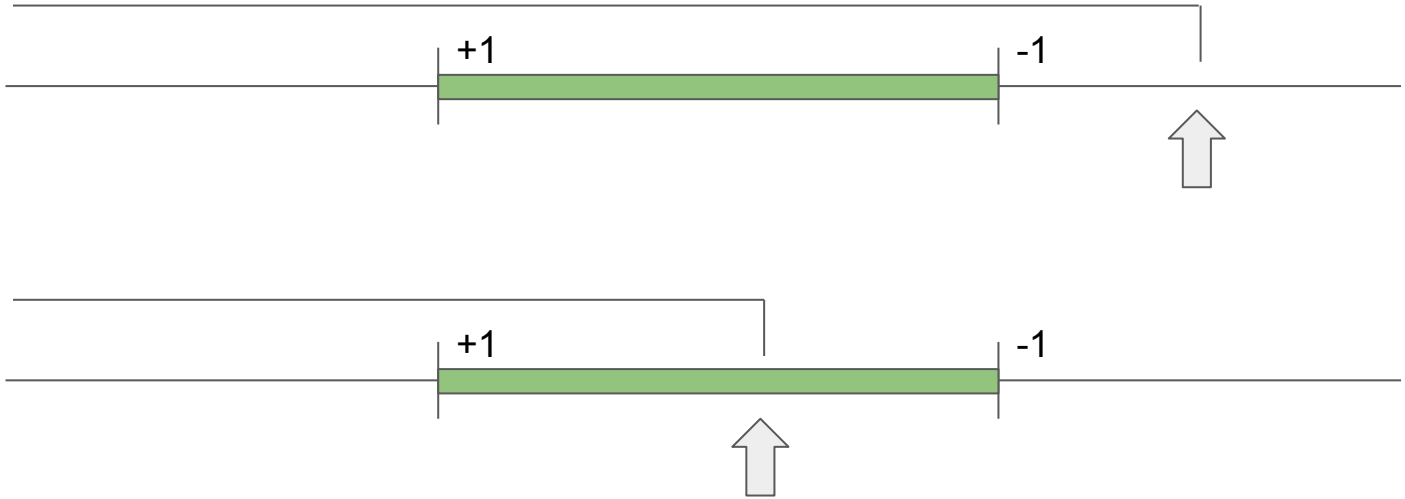
VS Point Query Range Update

- What was discussed was a point update and range query
- A different thought process but the same data structure can be used for range update and point query
- Problem
  - I can multi-task. I have many friends I chat online with at the same time. Each friend will talk to me for a continuous segment of time (an interval). I want to know at several distinct points how many people I was talking to. My memory is fuzzy so I might add or remove intervals.
- I can add 1 at the start of an interval and subtract 1 at the end of an interval

VS Point Query Range Update

- What was discussed was a point update and range query
- A different thought process but the same data structure can be used for range update and point query
- Problem
  - I can multi-task. I have many friends I chat online with at the same time. Each friend will talk to me for a continuous segment of time (an interval). I want to know at several distinct points how many people I was talking to. My memory is fuzzy so I might add or remove intervals.
- I can add 1 at the start of an interval and subtract 1 at the end of an interval
- This means
  - if the query point is before the interval the interval is not counted
  - If the query point is within the interval only the 1 is added to the query
  - If the query point is after the interval (outside) the 1 and -1 cancel out

VS Point Query Range Update



Types

- Point Query Range Update
- Point Update Range Query
- XOR stuff