Technical Slide

1. Lesson 3: Segment Tree
   - Video 3.1: Segment sree structure
   - Video 3.2: Summing a segment
   - Video 3.3: Modifying an element

Technical Slide

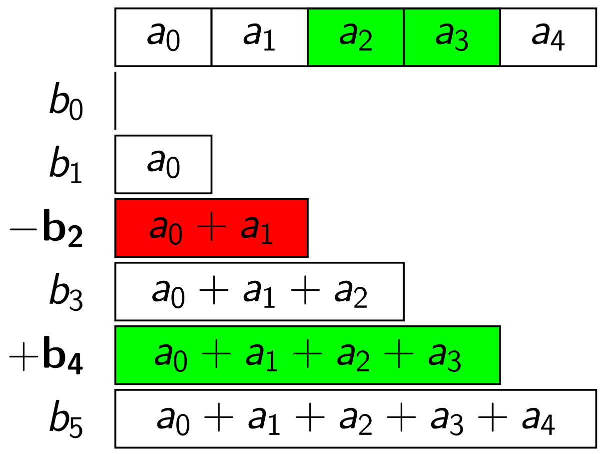
1. Lesson 3: Segment Tree
   - Video 3.1: Segment tree structure
   - Video 3.2: Summing a segment
   - Video 3.3: Modifying an element

Range Sum Query

- It is often required to answer to a larger number of queries like “Find a sum of all numbers of a subarray of a given array”.

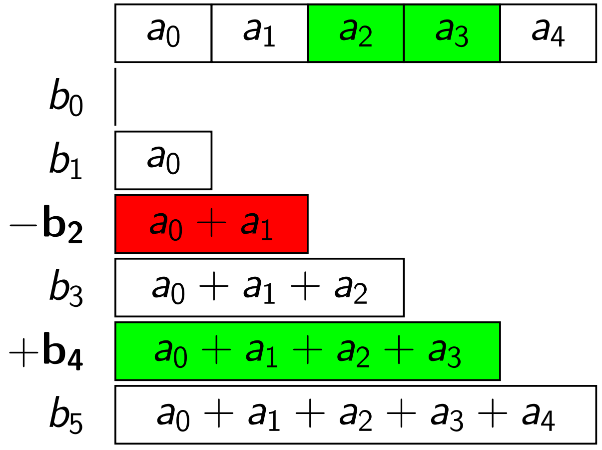
Range Sum Query

- It is often required to answer to a larger number of queries like "Find a sum of all numbers of a subarray of a given array".
- If the given array is constant, it is done by pre-evalualating array of so-called prefix sums: $b_i = \sum_{j=0}^{i-1} a_j$.



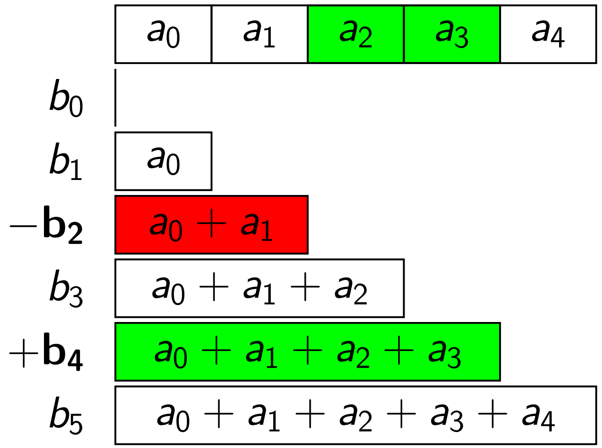
Range Sum Query

- It is often required to answer to a larger number of queries like "Find a sum of all numbers of a subarray of a given array".
- If the given array is constant, it is done by pre-evaluating array of so-called prefix sums: $b_i = \sum_{j=0}^{i-1} a_j$.
- In this case, the sum on a segment $a_k + a_{k+1} + \cdots + a_l$ could be calculated as $b_{l+1} - b_k$.



Range Sum Query

- It is often required to answer to a larger number of queries like “Find a sum of all numbers of a subarray of a given array”.
- If the given array is constant, it is done by pre-evalualating array of so-called prefix sums: $b_i = \sum_{j=0}^{i-1} a_j$.
- In this case, the sum on a segment $a_k + a_{k+1} + \cdots + a_l$ could be calculated as $b_{l+1} - b_k$.
- But what if the given array is modified dynamically?



Segment Tree

- The segment tree also pre-calculates sums on some subarrays of the array, but not only prefixes are selected.

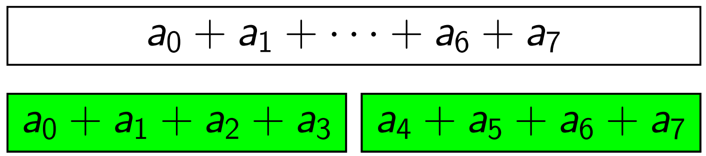
Segment Tree

- The segment tree also pre-calculates sums on some subarrays of the array, but not only prefixes are selected.
- Let's assume the length of an array is some $N = 2^k$ (if it is not, expand the array with zeros to the right).

$$a_0 + a_1 + \cdots + a_6 + a_7$$

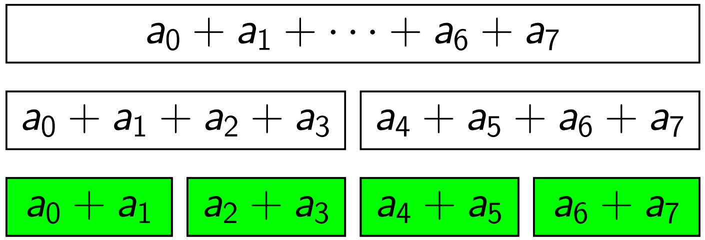
Segment Tree

- The segment tree also pre-calculates sums on some subarrays of the array, but not only prefixes are selected.
- Let's assume the length of an array is some $N = 2^k$ (if it is not, expand the array with zeros to the right).
- The whole segment $[1..2^k]$ is divided on two equal parts: $[1..2^{k-1}]$ and $[2^{k-1} + 1..2^k]$.



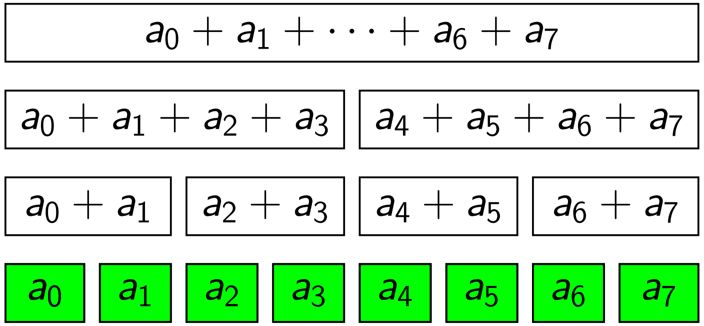
Segment Tree

- The segment tree also pre-calculates sums on some subarrays of the array, but not only prefixes are selected.
- Let's assume the length of an array is some $N = 2^k$ (if it is not, expand the array with zeros to the right).
- The whole segment $[1..2^k]$ is divided on two equal parts: $[1..2^{k-1}]$ and $[2^{k-1} + 1..2^k]$.
- Each of these parts is also divided on two equal parts and so on.



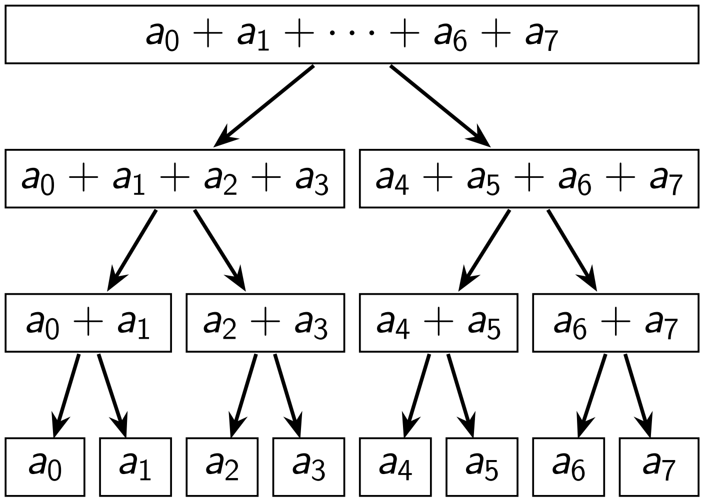
Segment Tree

- The segment tree also pre-calculates sums on some subarrays of the array, but not only prefixes are selected.
- Let's assume the length of an array is some $N = 2^k$ (if it is not, expand the array with zeros to the right).
- The whole segment $[1..2^k]$ is divided on two equal parts: $[1..2^{k-1}]$ and $[2^{k-1}+1..2^k]$.
- Each of these parts is also divided on two equal parts and so on.
- This process is completed when each of the parts consists of exactly one element.



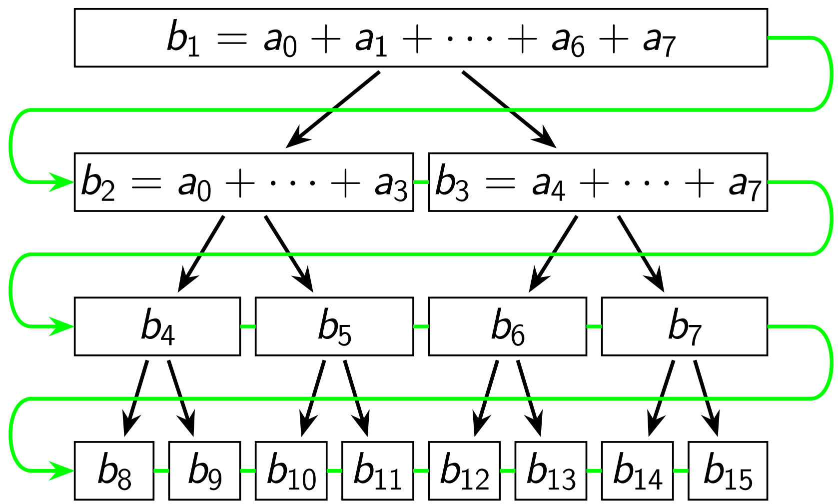
Graphical Representation

It is called a *Segment Tree* because each segment has two children: two smaller segments.



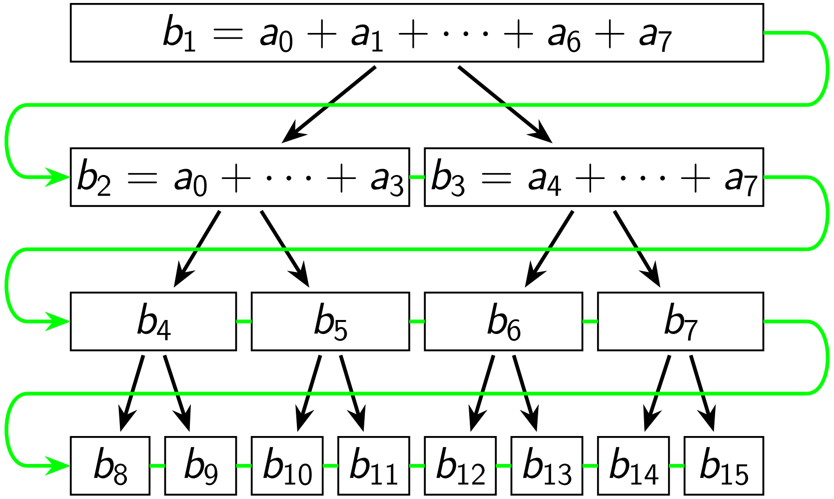
Storing Segment Tree

- The segment tree can be easily stored in an one-dimensional array of length $2N$.



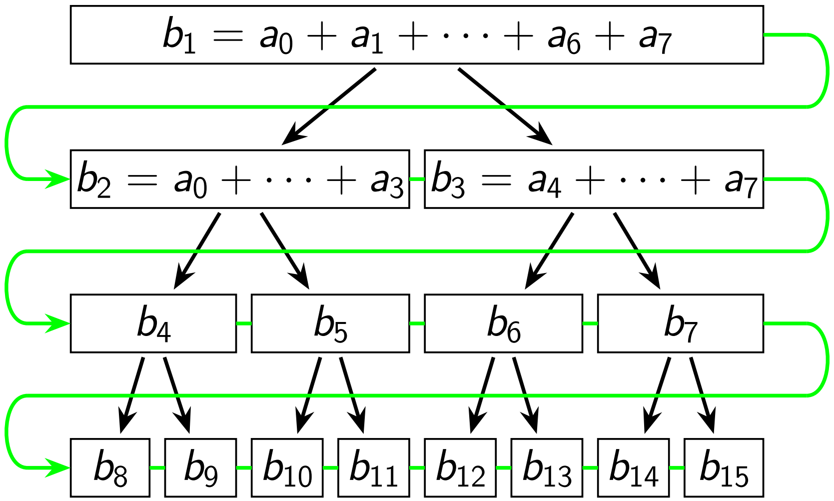
Storing Segment Tree

- The segment tree can be easily stored in an one-dimensional array of length $2N$.
- The root (corresponding to the whole segment) is stored in a cell 1.



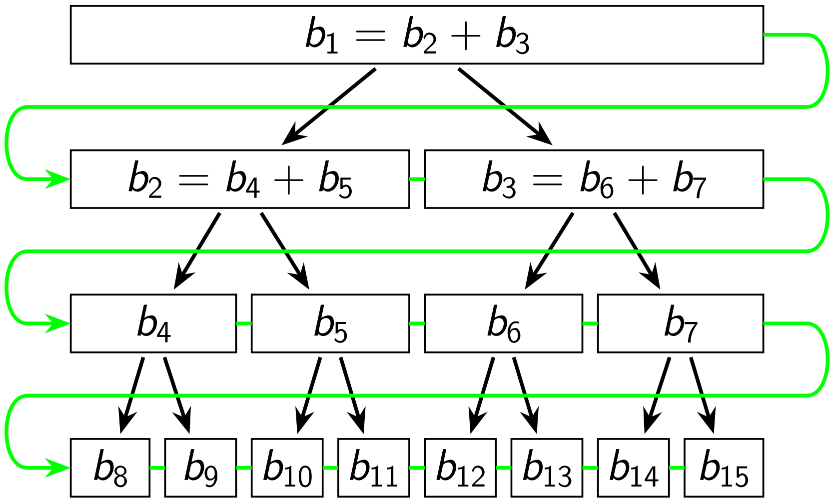
Storing Segment Tree

- The segment tree can be easily stored in an one-dimensional array of length $2N$.
- The root (corresponding to the whole segment) is stored in a cell 1.
- The chidren of any segment stored in some cell i are stored in cells $2i$ and $2i+1$.



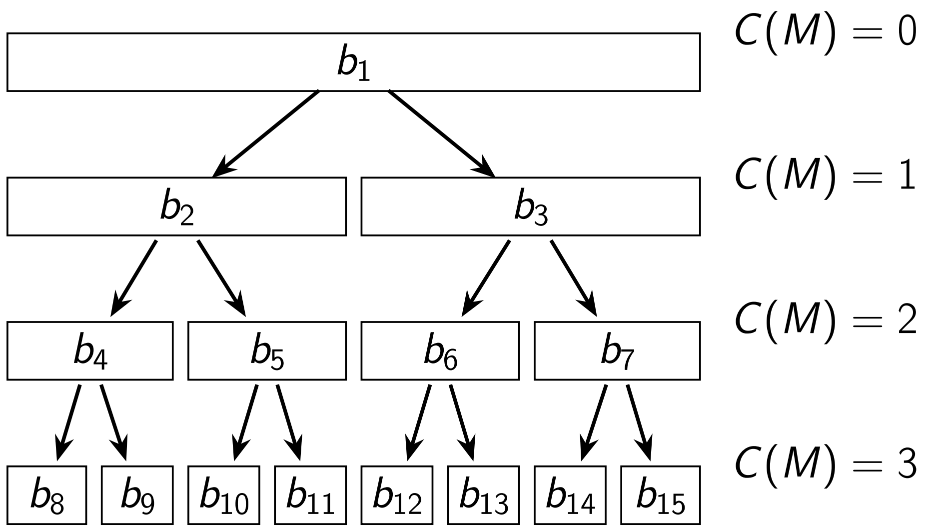
Storing Segment Tree

- The segment tree can be easily stored in an one-dimensional array of length $2N$.
- The root (corresponding to the whole segment) is stored in a cell 1.
- The chidren of any segment stored in some cell i are stored in cells $2i$ and $2i+1$.
- The number in the cell is just the sum of the numbers in its children.



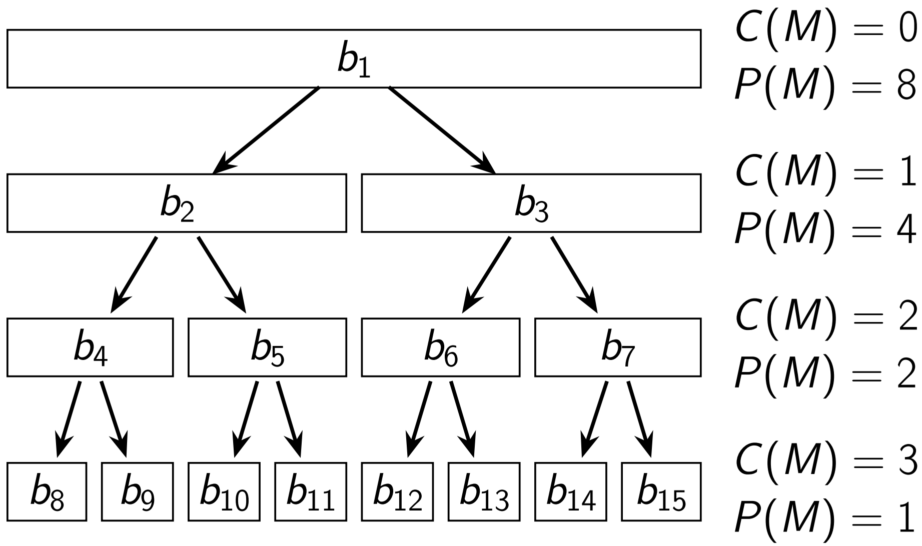
Some Useful Formulae

- Let's denote $C(M) = \lfloor \log_2 M \rfloor$.



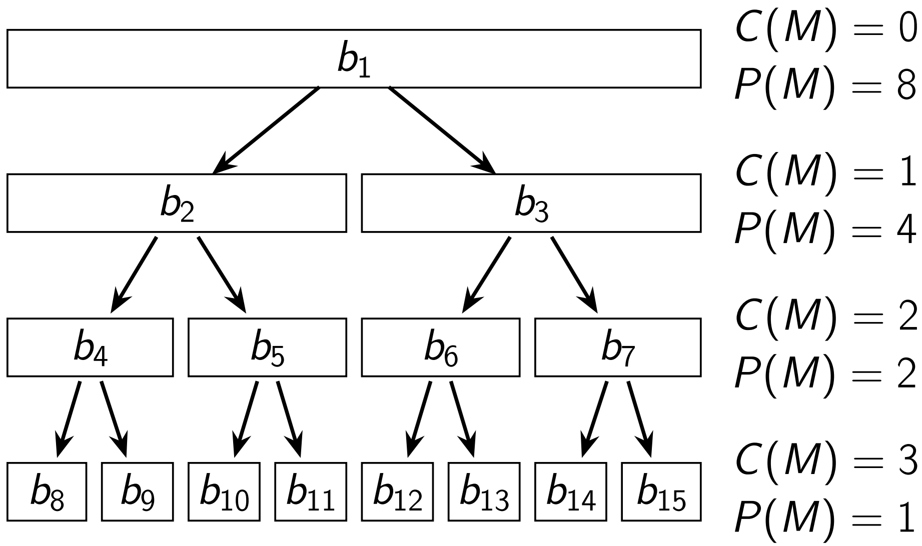
Some Useful Formulae

- Let's denote $C(M) = \lfloor \log_2 M \rfloor$.
- Note that segment corresponding to tree cell b_M has length of exactly $P(M) = 2^{k-C(M)}$.



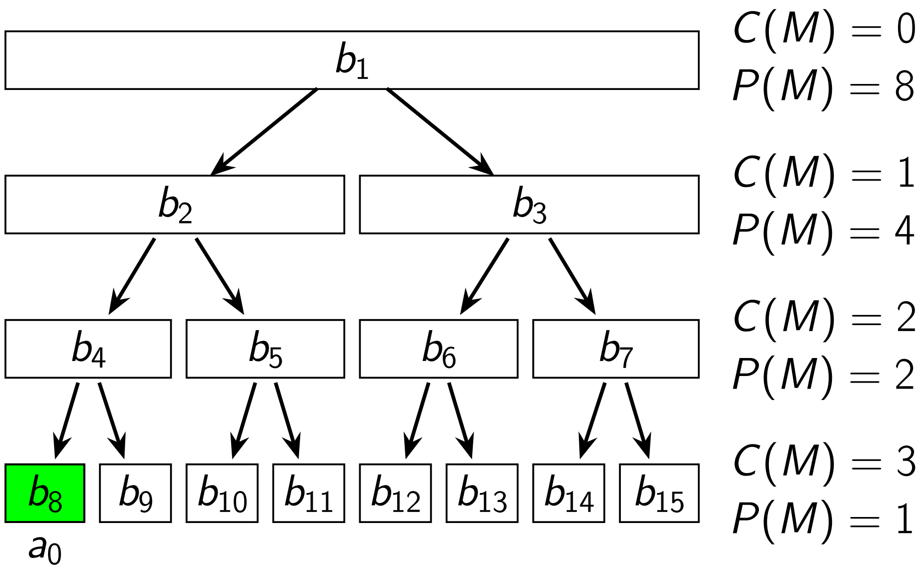
Some Useful Formulae

- Let's denote $C(M) = \lfloor \log_2 M \rfloor$.
- Note that segment corresponding to tree cell b_M has length of exactly $P(M) = 2^{k-C(M)}$.
- One can easily evaluate left and right boundaries of a segment corresponding to cell b_M: its first cell is $P(M) \cdot (M - 2^{C(M)})$, its last cell is $P(M) \cdot (M - 2^{C(M)} + 1) - 1$.



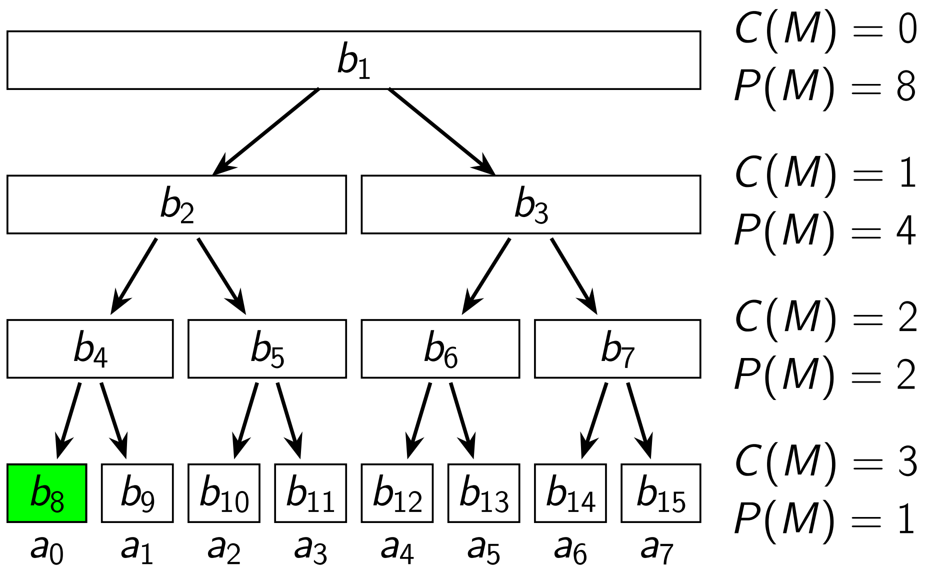
Some Useful Formulae

- Let's denote $C(M) = \lfloor \log_2 M \rfloor$.
- Note that segment corresponding to tree cell b_M has length of exactly $P(M) = 2^{k-C(M)}$.
- One can easily evaluate left and right boundaries of a segment corresponding to cell b_M: its first cell is $P(M) \cdot (M - 2^{C(M)})$, its last cell is $P(M) \cdot (M - 2^{C(M)} + 1) - 1$.
- The leaves of the tree begin from the cell with index $N = 2^k$.



Some Useful Formulae

- Let's denote $C(M) = \lfloor \log_2 M \rfloor$.
- Note that segment corresponding to tree cell b_M has length of exactly $P(M) = 2^{k-C(M)}$.
- One can easily evaluate left and right boundaries of a segment corresponding to cell b_M: its first cell is $P(M) \cdot (M - 2^{C(M)})$, its last cell is $P(M) \cdot (M - 2^{C(M)} + 1) - 1$.
- The leaves of the tree begin from the cell with index $N = 2^k$.
- The leaf corresponding to the element a_i of the original array is stored in the cell b_{N+i}.



Technical Slide

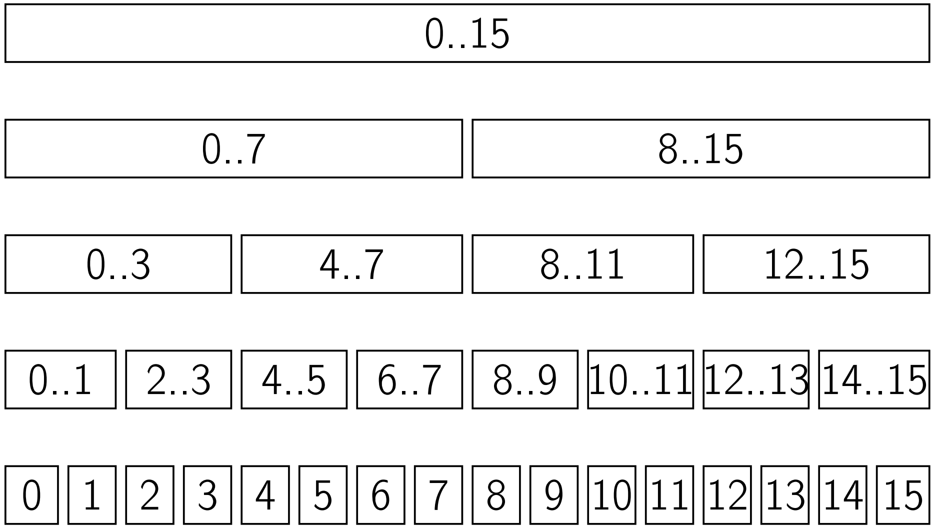
1. Lesson 3: Segment Tree
   - Video 3.1: Segment sree structure
   - Video 3.2: Summing a segment
   - Video 3.3: Modifying an element

Technical Slide

1. Lesson 3: Segment Tree
   - Video 3.1: Segment tree structure
   - Video 3.2: Summing a segment
   - Video 3.3: Modifying an element

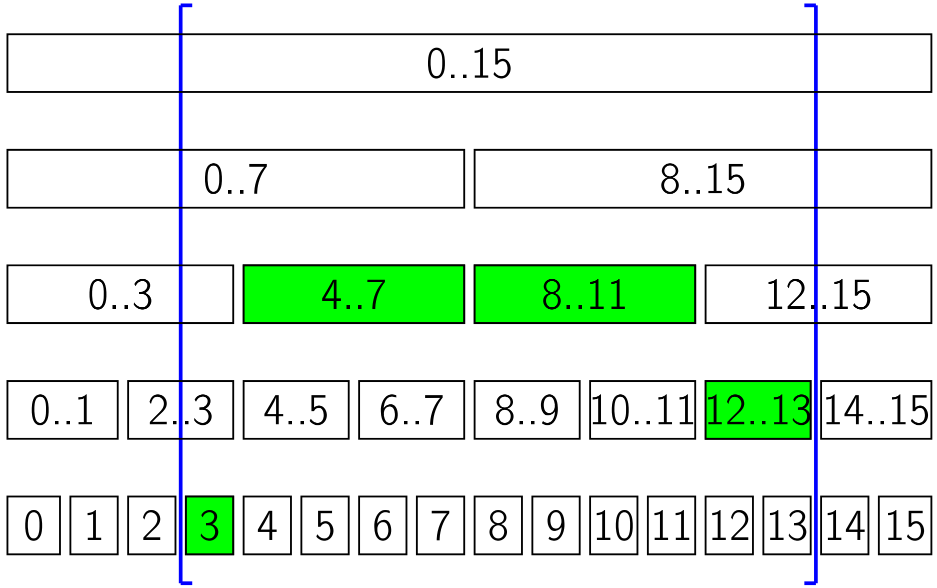
How To Calculate Sum on a Segment

- It is obvious that segment tree contains cells evaluated for all segments of length of some 2^t, starting with an element with index which is a multiple of 2^t.



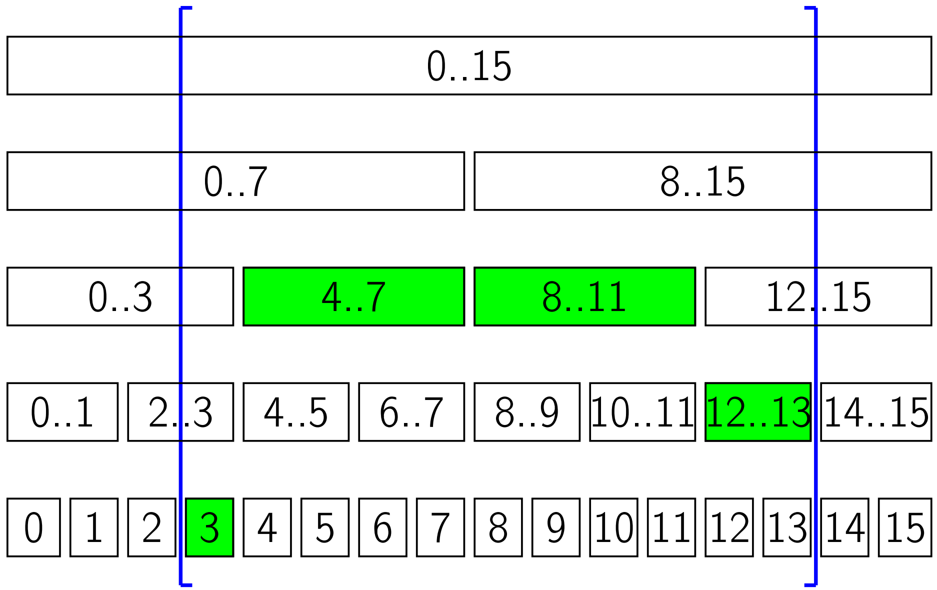
How To Calculate Sum on a Segment

- It is obvious that segment tree contains cells evaluated for all segments of length of some 2^t, starting with an element with index which is a multiple of 2^t.
- So if you partition the segment of query to the segments of such kind, you could just sum the corresponding cells of a segment tree.



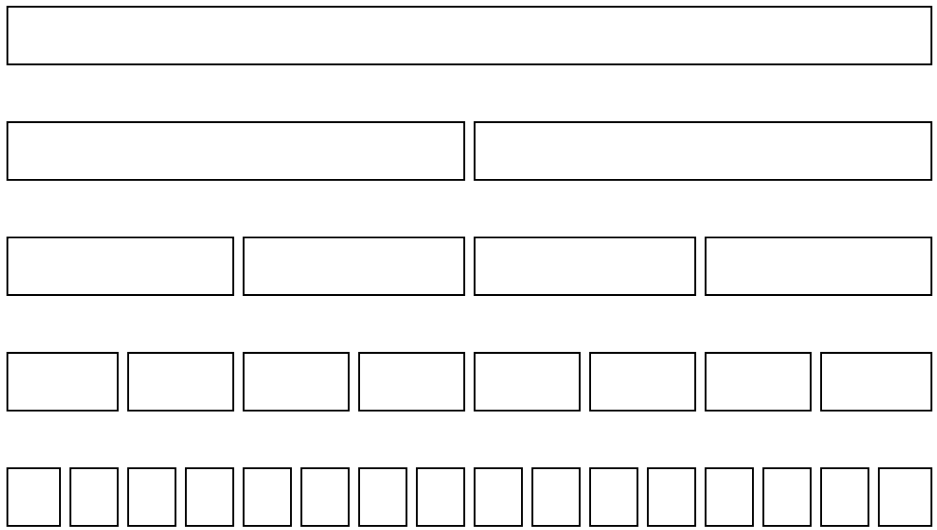
How To Calculate Sum on a Segment

- It is obvious that segment tree contains cells evaluated for all segments of length of some 2^t, starting with an element with index which is a multiple of 2^t.
- So if you partition the segment of query to the segments of such kind, you could just sum the corresponding cells of a segment tree.
- There are two ways of doing this: going upwards (ascending the tree from leaves to its root) and downwards (descending the tree from root to its leaves).



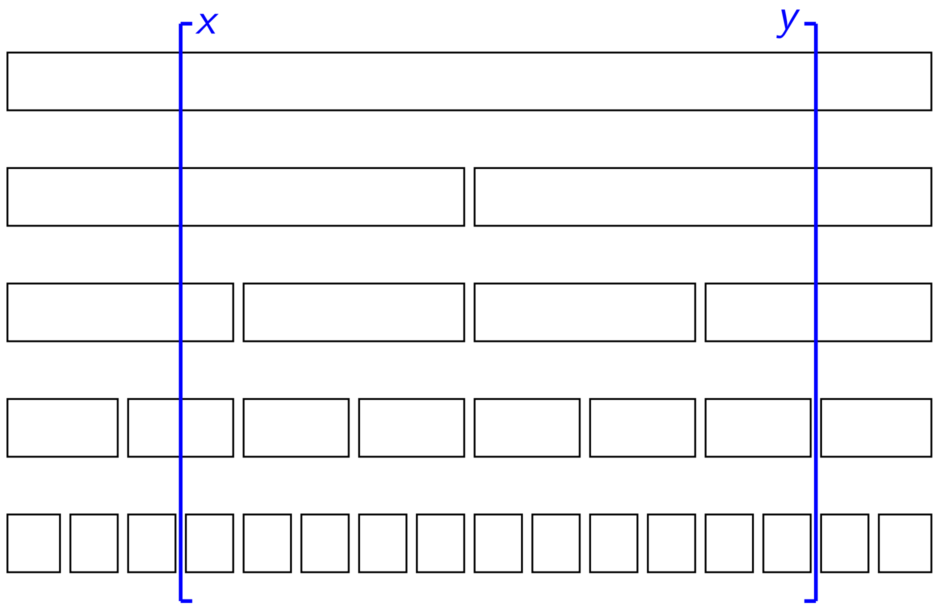
Going Upwards

- It's the fastest way, but less universal.



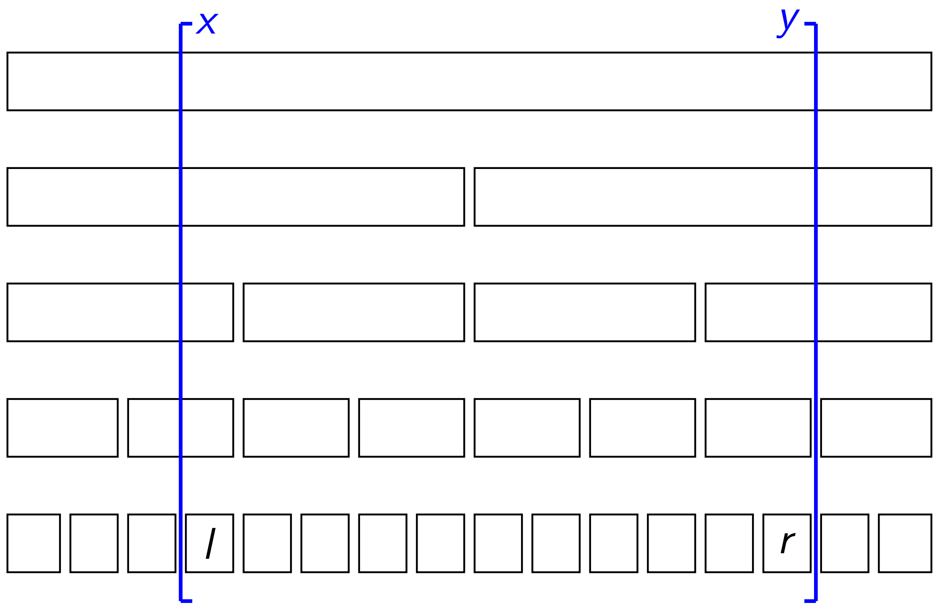
Going Upwards

- It's the fastest way, but less universal.
- Let's consider a query for the segment from x to y, inclusive.



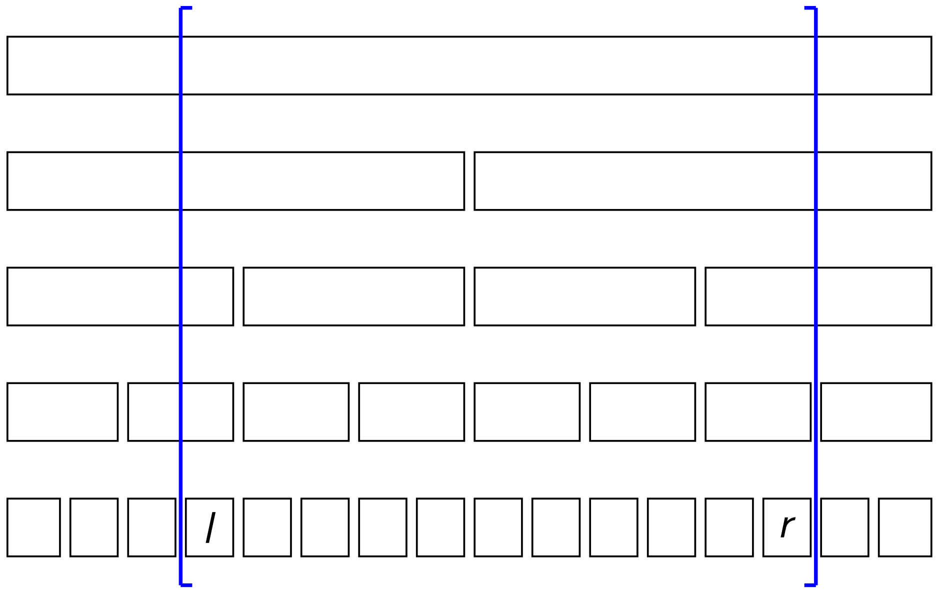
Going Upwards

- It's the fastest way, but less universal.
- Let's consider a query for the segment from x to y, inclusive.
- You can imagine that the query is given not for the original array, but for the tree. Just add $N = 2^k$ to its boundaries: $l = x + N$, $r = y + N$.



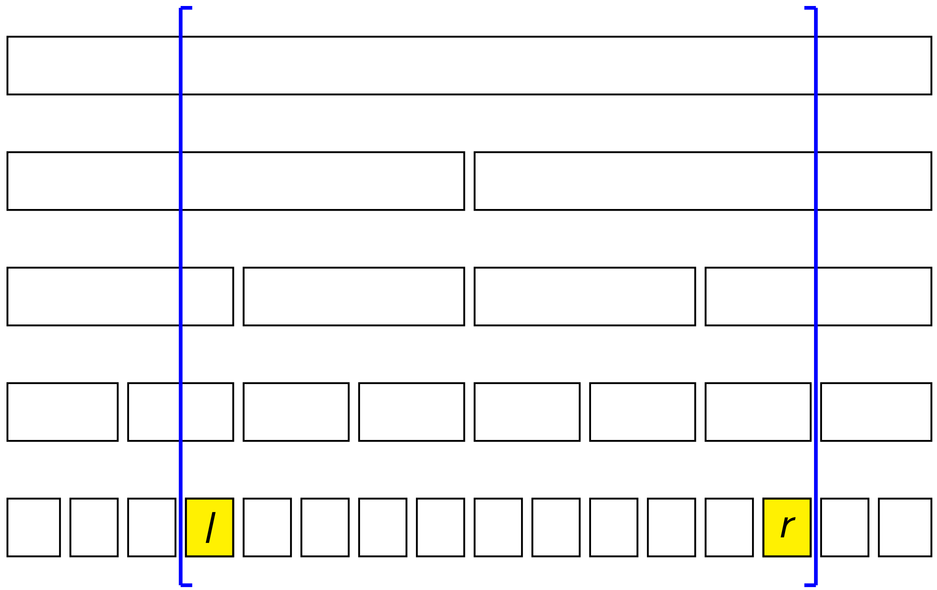
Going Upwards

- It's the fastest way, but less universal.
- Let's consider a query for the segment from x to y, inclusive.
- You can imagine that the query is given not for the original array, but for the tree. Just add $N = 2^k$ to its boundaries: $l = x + N$, $r = y + N$.
- Now we have a new task: calculate the sum for the segment of the *tree array*, from l to r, inclusive. How to solve it?



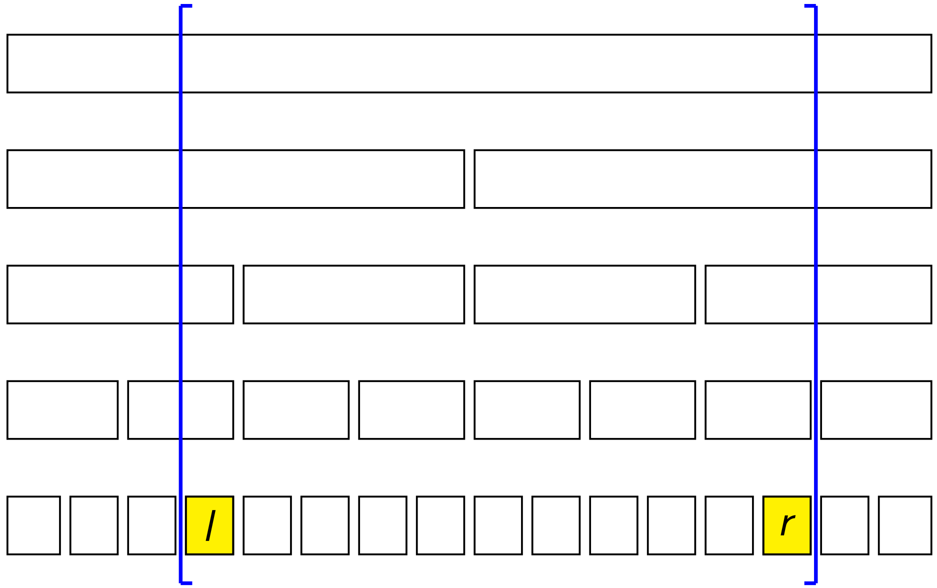
Transition One Level Up - 1

- We are given l and r in the segment tree. How to calculate the sum of stored cells of the tree between l and r, inclusive?



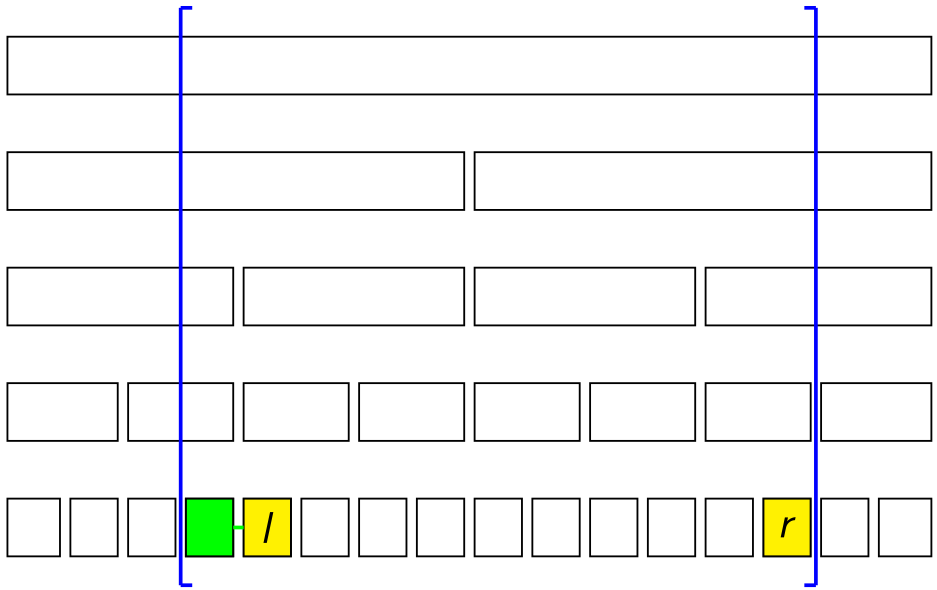
Transition One Level Up - 1

- We are given l and r in the segment tree. How to calculate the sum of stored cells of the tree between l and r, inclusive?
- If $l > r$, the answer is 0. If $l = r$, the answer is $b_l = b_r$.



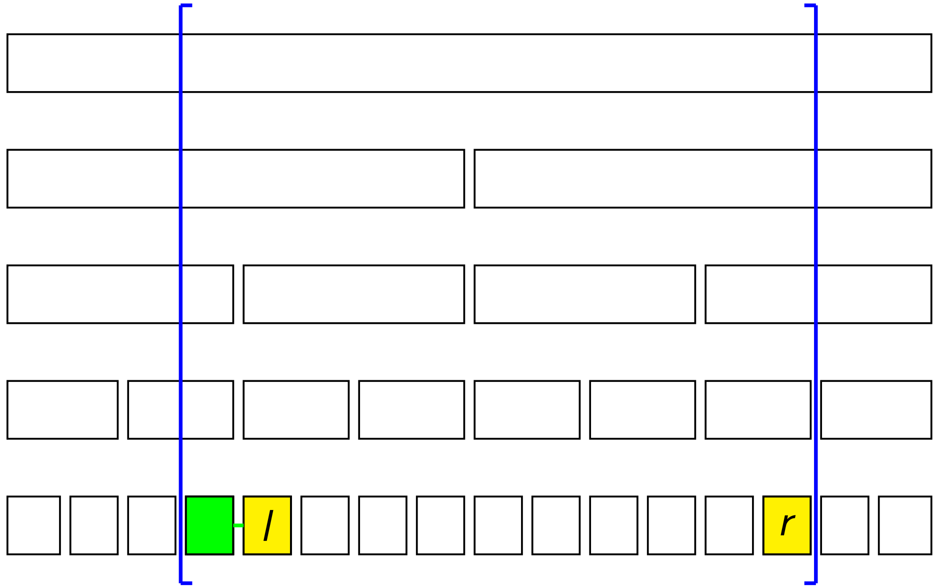
Transition One Level Up - 1

- We are given l and r in the segment tree. How to calculate the sum of stored cells of the tree between l and r, inclusive?
- If $l > r$, the answer is 0. If $l = r$, the answer is $b_l = b_r$.
- If there are at least two elements in the segment, let's look to cell l. If l is even, both b_l and b_{l+1} are children of $b_{l/2}$. Otherwise let's add b_l to some cumulative result variable S and increment l. Now l is even.



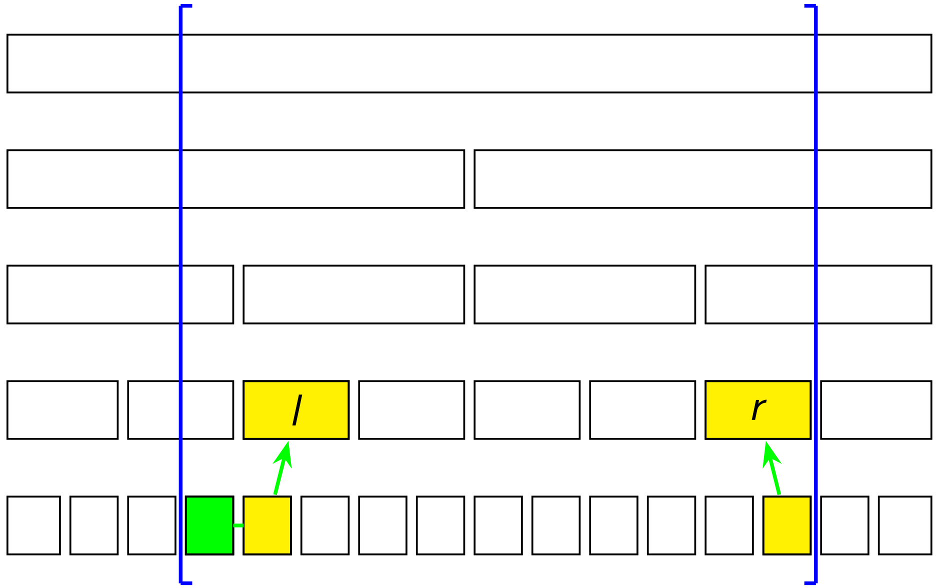
Transition One Level Up - 1

- We are given l and r in the segment tree. How to calculate the sum of stored cells of the tree between l and r, inclusive?
- If $l > r$, the answer is 0. If $l = r$, the answer is $b_l = b_r$.
- If there are at least two elements in the segment, let's look to cell l. If l is even, both b_l and b_{l+1} are children of $b_{l/2}$. Otherwise let's add b_l to some cumulative result variable S and increment l. Now l is even.
- You can also do the same thing with r. If r is odd, b_{r-1} and b_r are children of $b_{\lfloor r/2 \rfloor}$. Otherwise let's add b_r to S and decrement r. Now r is odd.



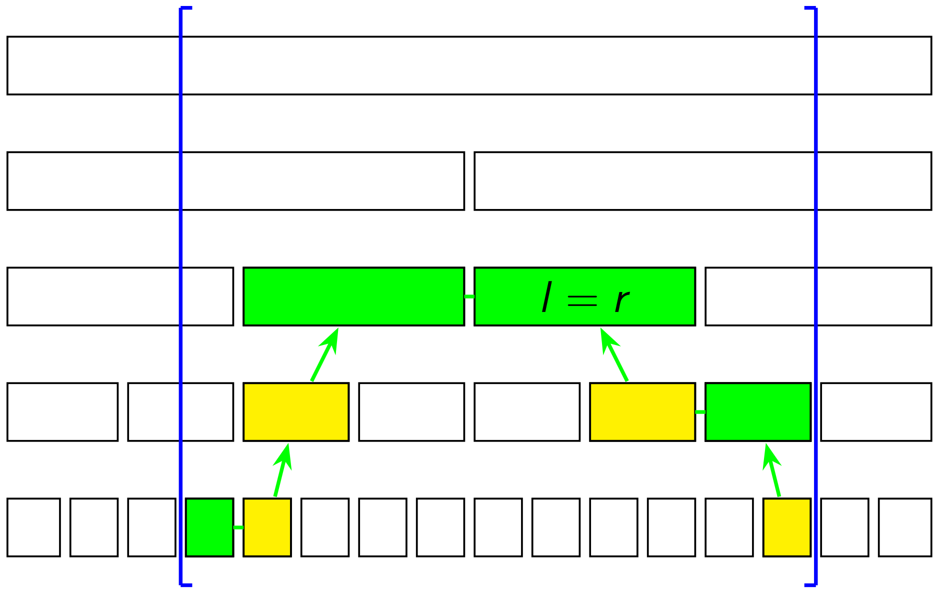
Transition One Level Up - 2

- Now we can easily move one level up. Just divide l and r by 2 (as integers). It's the same sum in terms of original array.



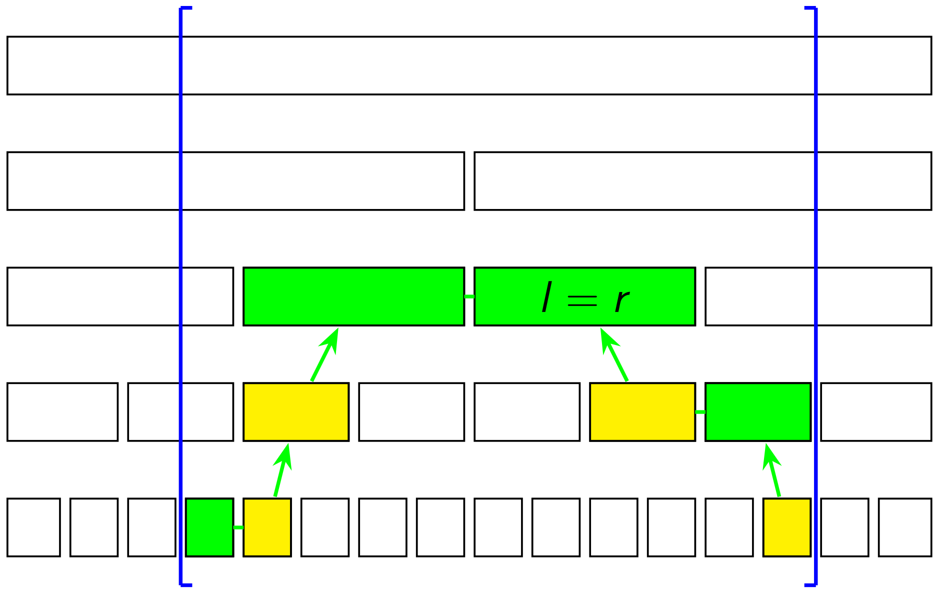
Transition One Level Up - 2

- Now we can easily move one level up. Just divide l and r by 2 (as integers). It's the same sum in terms of original array.
- After the process is terminated, S will contain the answer to the query.



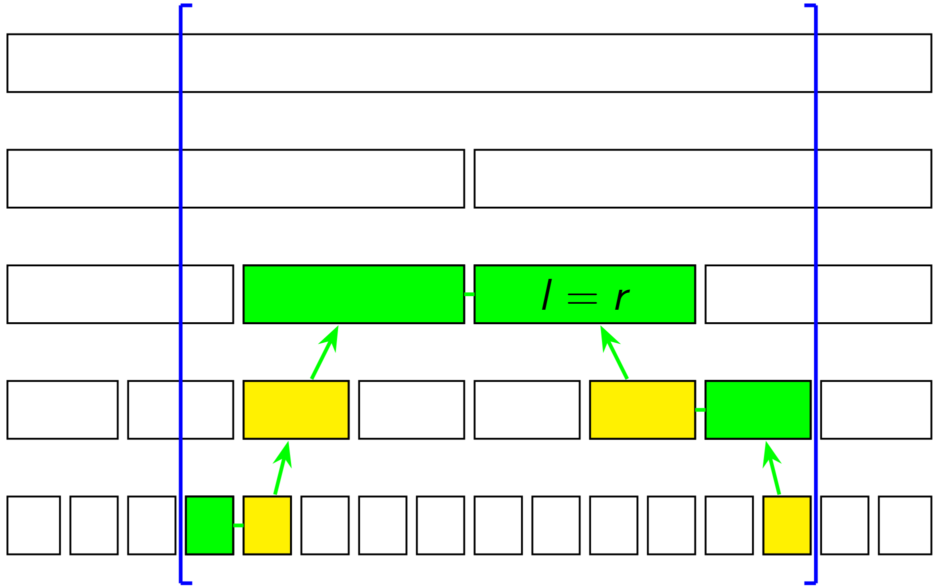
Transition One Level Up - 2

- Now we can easily move one level up. Just divide l and r by 2 (as integers). It's the same sum in terms of original array.
- After the process is terminated, S will contain the answer to the query.
- The complexity is obviously $O(k) = O(\log N)$ (there are $O(k)$ levels).



Transition One Level Up - 2

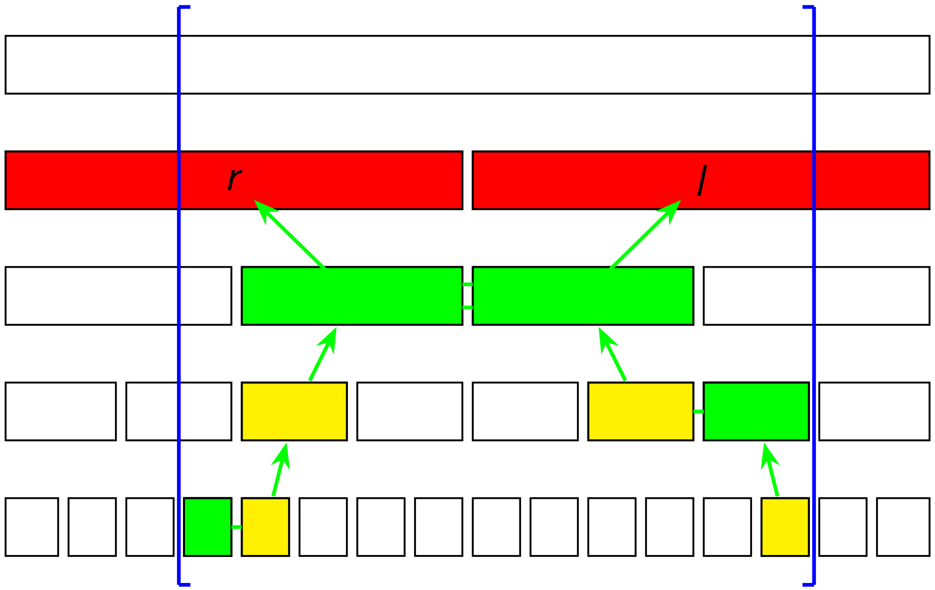
- Now we can easily move one level up. Just divide l and r by 2 (as integers). It's the same sum in terms of original array.
- After the process is terminated, S will contain the answer to the query.
- The complexity is obviously $O(k) = O(\log N)$ (there are $O(k)$ levels).
- Excercise: you can prove that the case $l = r$ can be processed as the general case (the base is only $l > r$).



Code Example: Sum Upwards

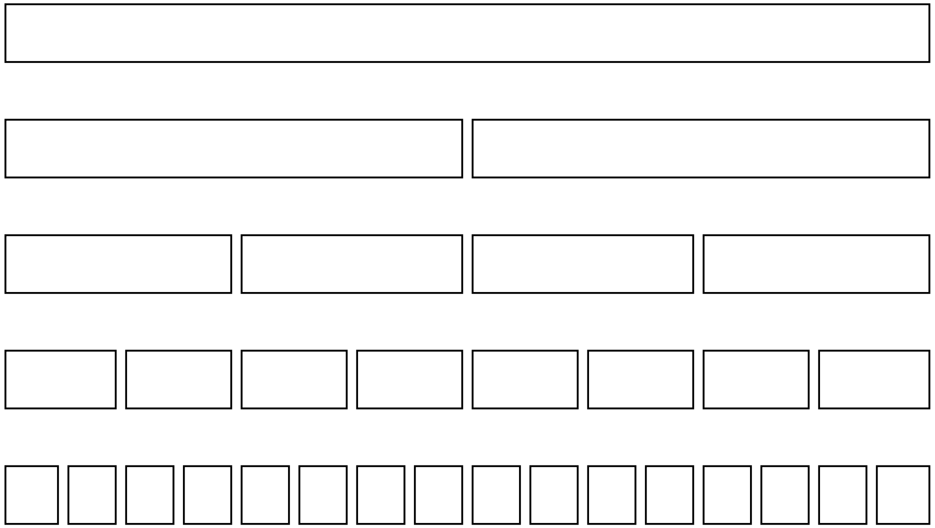
```
l = x + N;
r = y + N;
S = 0;
while (l <= r) {
  if (l % 2 != 0) S += b[l++];
  if (r % 2 == 0) S -= b[r--];
  l /= 2;
  r /= 2;
}
```

Here is a Picture



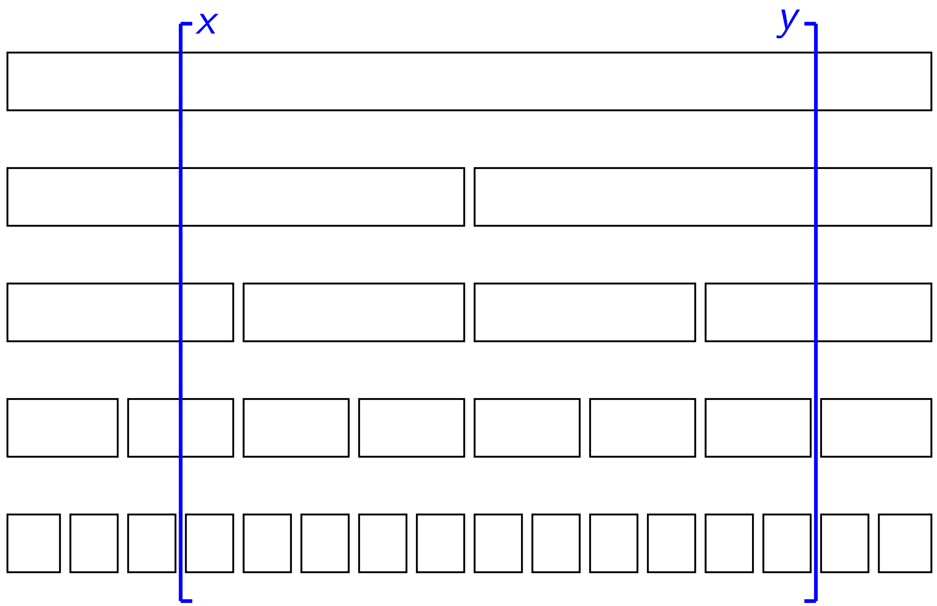
Going Downwards

- It's more universal way. When you go downwards, different operations with segment tree look almost the same.



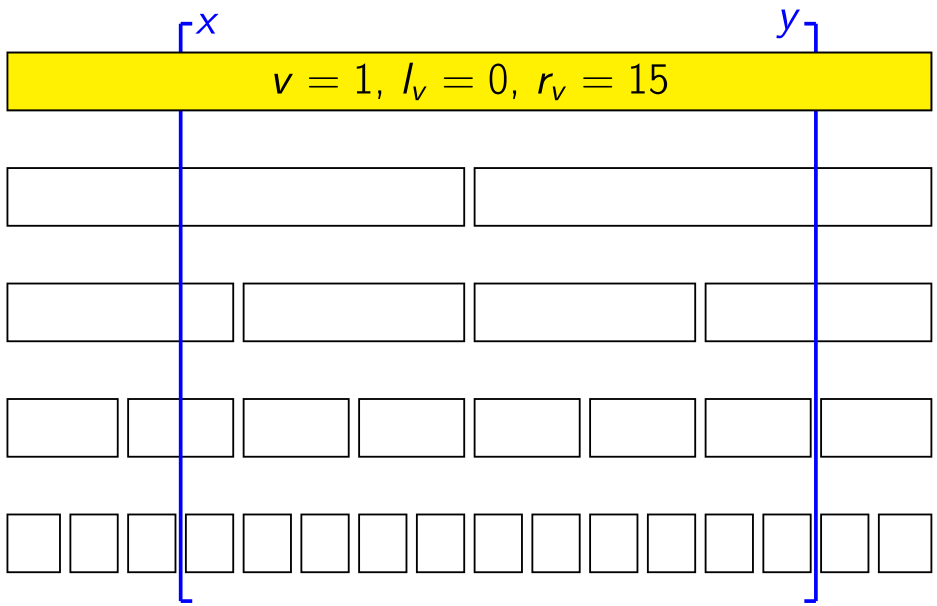
Going Downwards

- It's more universal way. When you go downwards, different operations with segment tree look almost the same.
- Let's consider a query for the segment from x to y, inclusive.



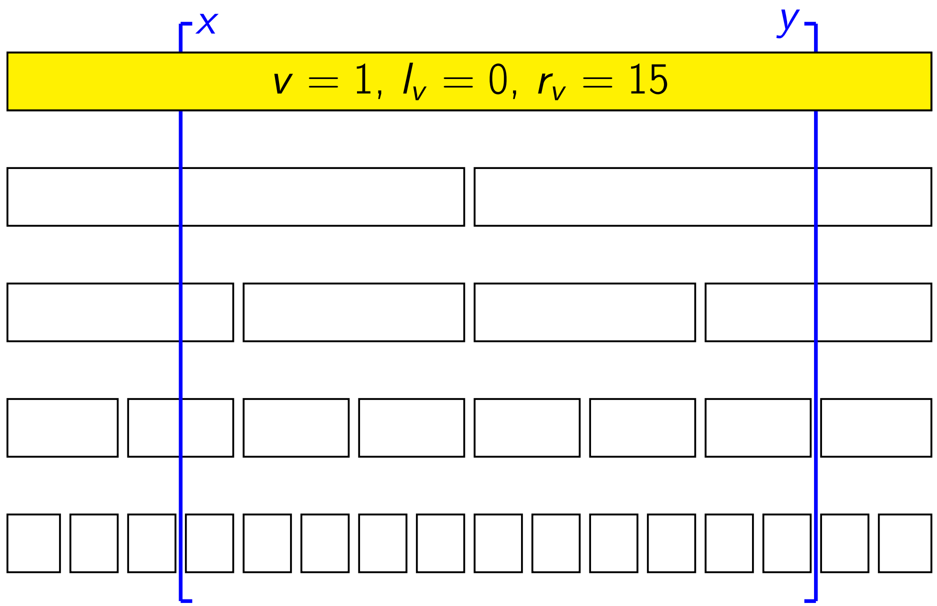
Going Downwards

- It's more universal way. When you go downwards, different operations with segment tree look almost the same.
- Let's consider a query for the segment from x to y, inclusive.
- Let's support the number of current node v and two boundaries of the segment described by this node: l_v and r_v.



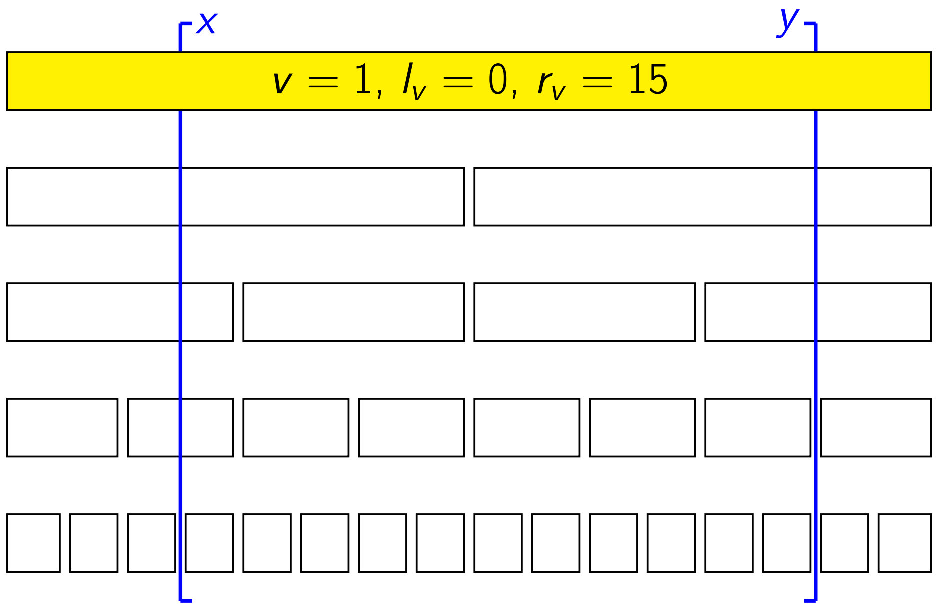
Going Downwards

- It's more universal way. When you go downwards, different operations with segment tree look almost the same.
- Let's consider a query for the segment from x to y, inclusive.
- Let's support the number of current node v and two boundaries of the segment described by this node: l_v and r_v.
- Let's generalize the task: find the sum of intersection of two segments of the original array: the query (from x to y) and the current node (from l_v and r_v).



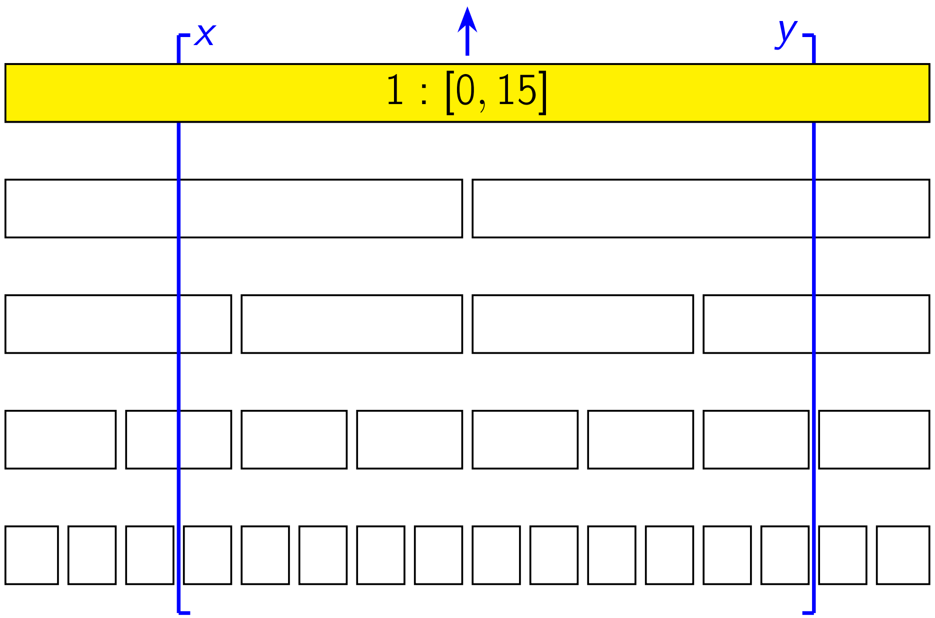
Going Downwards

- It's more universal way. When you go downwards, different operations with segment tree look almost the same.
- Let's consider a query for the segment from x to y, inclusive.
- Let's support the number of current node v and two boundaries of the segment described by this node: l_v and r_v.
- Let's generalize the task: find the sum of intersection of two segments of the original array: the query (from x to y) and the current node (from l_v and r_v).
- The original task is now rewritten as finding the sum from x to y at node 1.



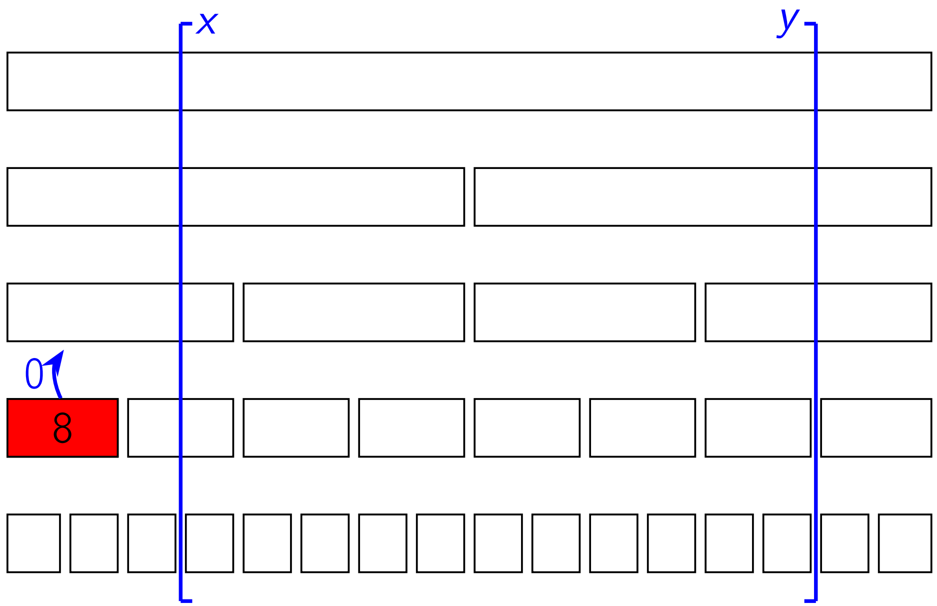
Transition One Level Down - 1

- We are given x and y of the original query. How to calculate sum on intersection of this segment with segment from l_v to r_v corresponding to node v?



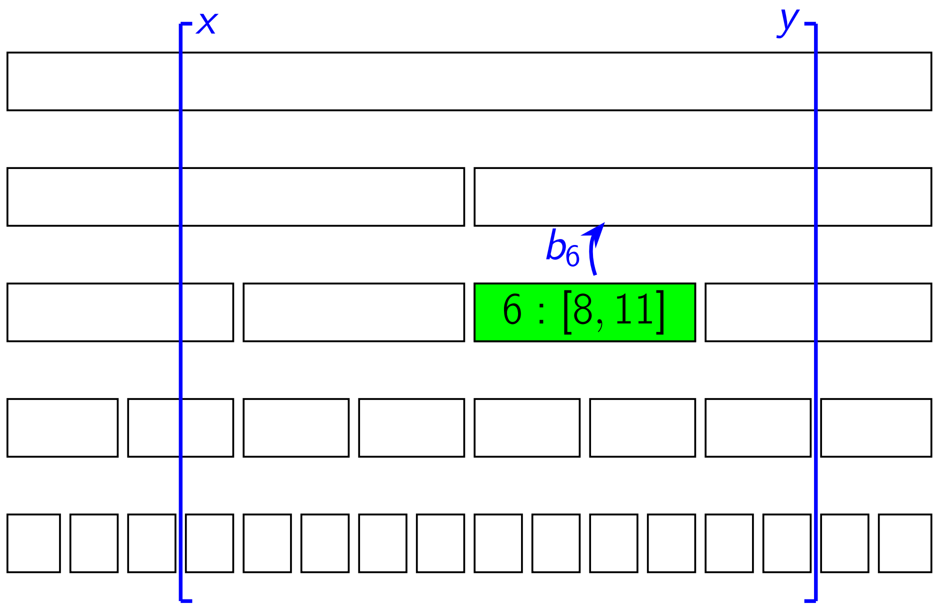
Transition One Level Down - 1

- We are given x and y of the original query. How to calculate sum on intersection of this segment with segment from l_v to r_v corresponding to node v?
- If the intersection is empty ($r_v < x$ or $y < l_v$), the sum is zero.



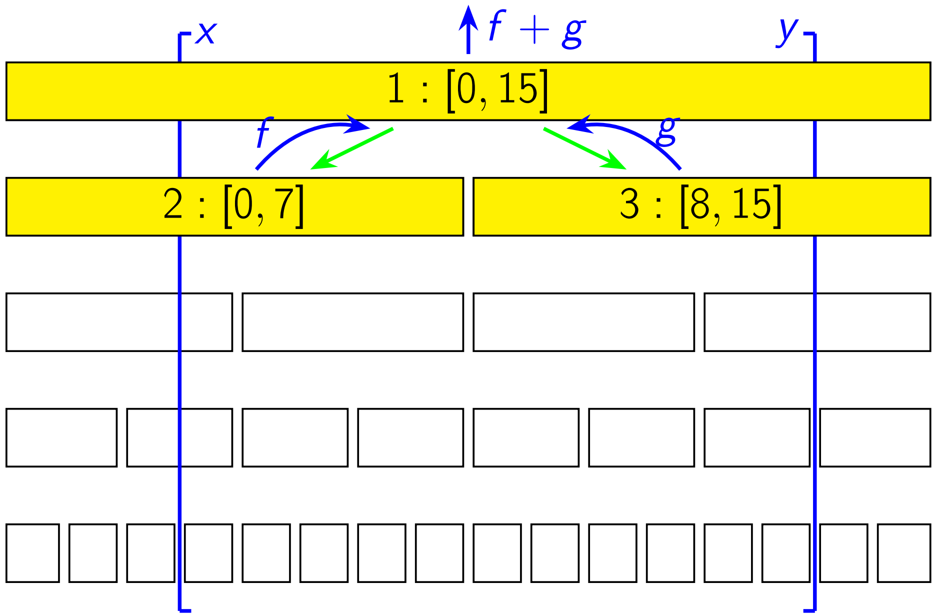
Transition One Level Down - 1

- We are given x and y of the original query. How to calculate sum on intersection of this segment with segment from l_v to r_v corresponding to node v?
- If the intersection is empty ($r_v < x$ or $y < l_v$), the sum is zero.
- If the intersection is the whole segment corresponding to node v ($x \leq l_v$ and $r_v \leq y$), the sum is exactly b_v.



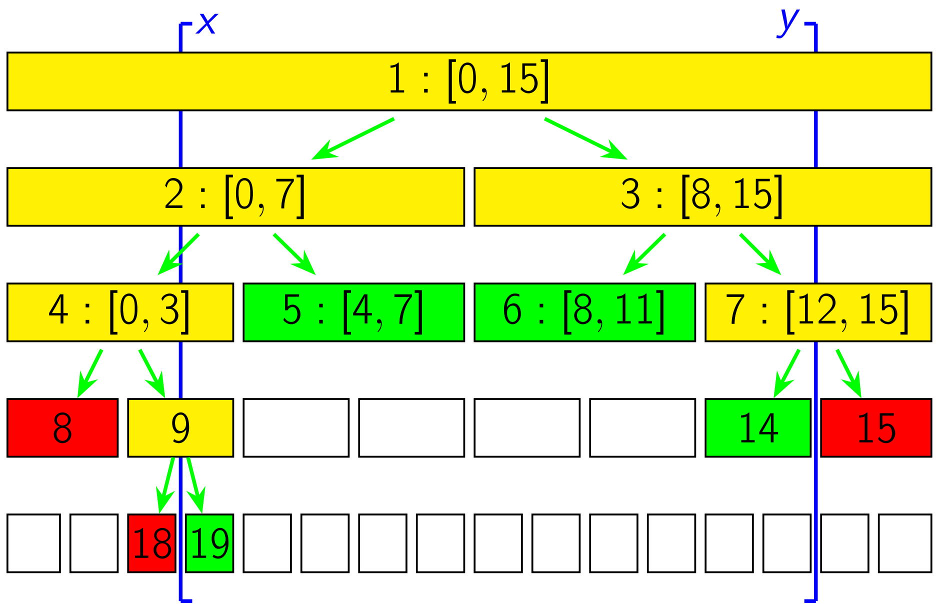
Transition One Level Down - 2

- If we are not in one of two basic cases, the sum can be easily calculated by solving the same task descending one level down: let's solve the task for nodes $2v$ and $2v+1$ and sum the answers.



Transition One Level Down - 2

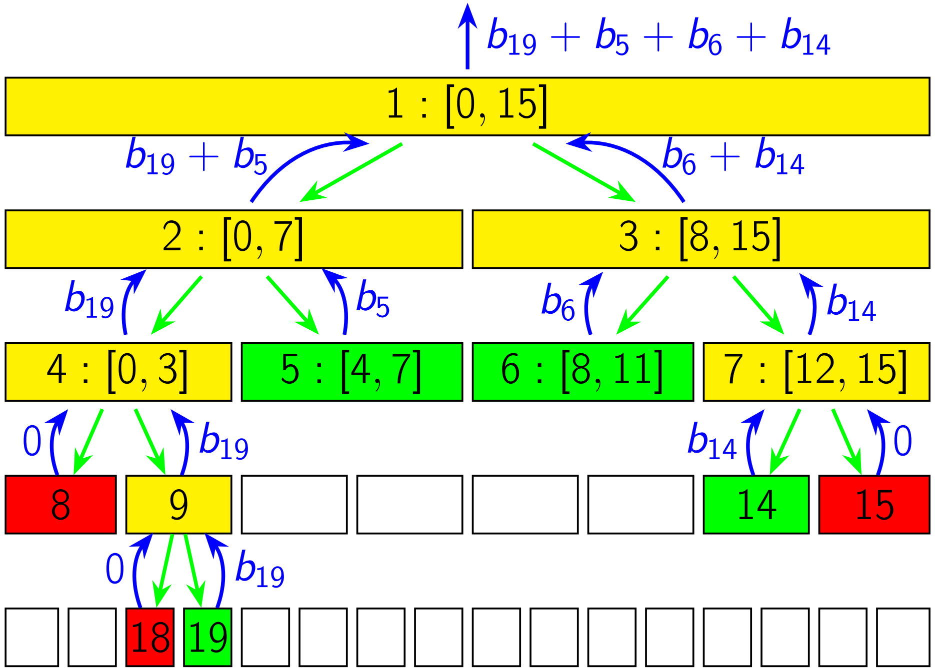
- If we are not in one of two basic cases, the sum can be easily calculated by solving the same task descending one level down: let's solve the task for nodes $2v$ and $2v + 1$ and sum the answers.
- The complexity is also $O(\log N)$. To prove this, one can notice that there are no more than two segments of each of three kinds on each level. Each segment of the third kind produces no more than one segment of the third kind and no more than one segment of the first or second kind.



Code Example: Sum Downwards

```
int sum (int x, int y, int l, int r, int v) {
  if (r < x || y < l) return 0;
  if (x <= l && r <= y) return b[v];
  int m = (l + r) / 2;
  return sum (x, y, l, m, 2 * v) +
         sum (x, y, m + 1, r, 2 * v + 1);
}
```

Here is a Picture



Technical Slide

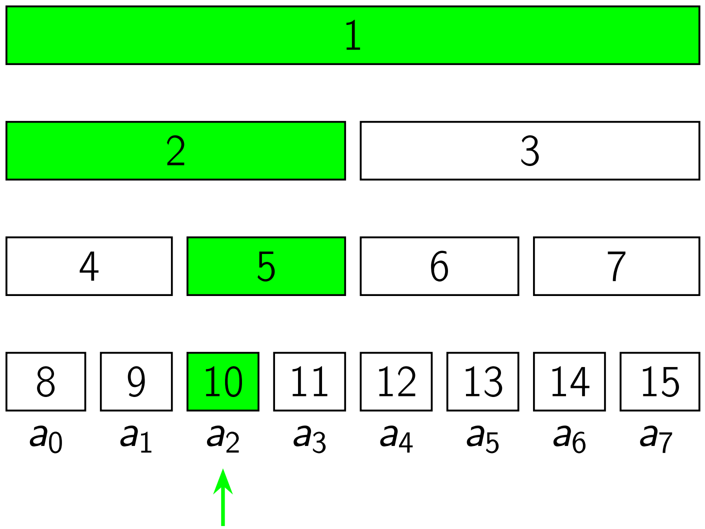
1. Lesson 3: Segment Tree
   - Video 3.1: Segment sree structure
   - Video 3.2: Summing a segment
   - Video 3.3: Modifying an element

Technical Slide

1. Lesson 3: Segment Tree
   - Video 3.1: Segment tree structure
   - Video 3.2: Summing a segment
   - Video 3.3: Modifying an element

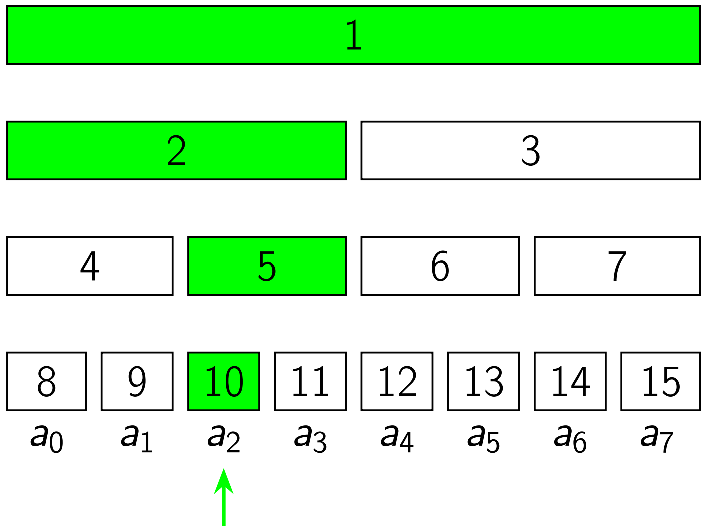
Modification Of An Element

- But what about modification? To modify an element, one needs to modify the values of all segments that contain this element.



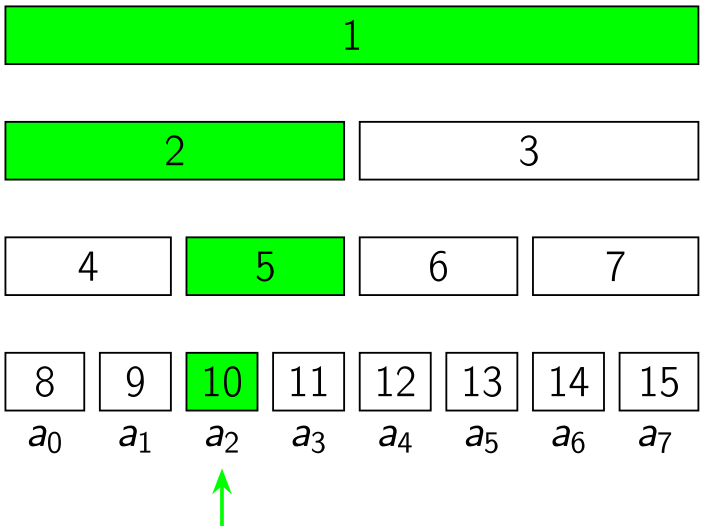
Modification Of An Element

- But what about modification? To modify an element, one needs to modify the values of all segments that contain this element.
- Let's consider the case that we need to add the value p to some element x. If we need just to assign some value q to the element, let's calculate the difference between new value and the old value and assume $p = q - a_x$.



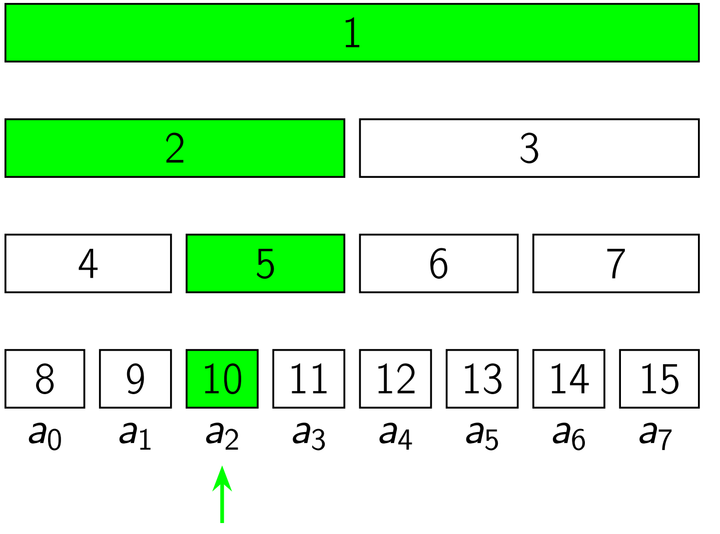
Modification Of An Element

- But what about modification? To modify an element, one needs to modify the values of all segments that contain this element.
- Let's consider the case that we need to add the value p to some element x. If we need just to assign some value q to the element, let's calculate the difference between new value and the old value and assume $p = q - a_x$.
- Modification can be easily done in two ways: upwards and downwards.



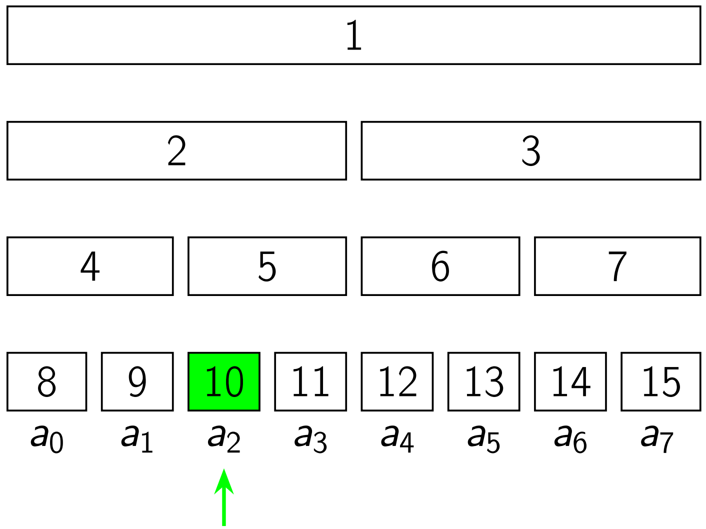
Modification Of An Element

- But what about modification? To modify an element, one needs to modify the values of all segments that contain this element.
- Let's consider the case that we need to add the value p to some element x. If we need just to assign some value q to the element, let's calculate the difference between new value and the old value and assume $p = q - a_x$.
- Modification can be easily done in two ways: upwards and downwards.
- To modify an element upwards, one needs to start from the corresponding leaf and then process the path of cells from this leaf to the root.



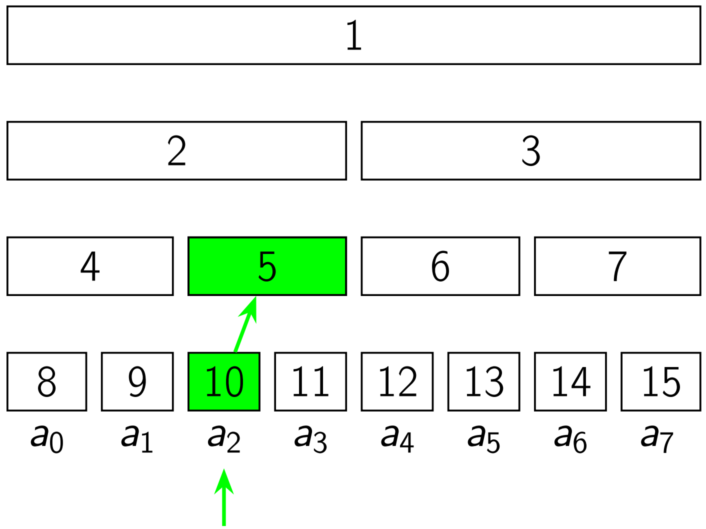
Modifying Element Upwards

- The process starts from the cell $v = x + N$ which is the leaf corresponding to the modified element.



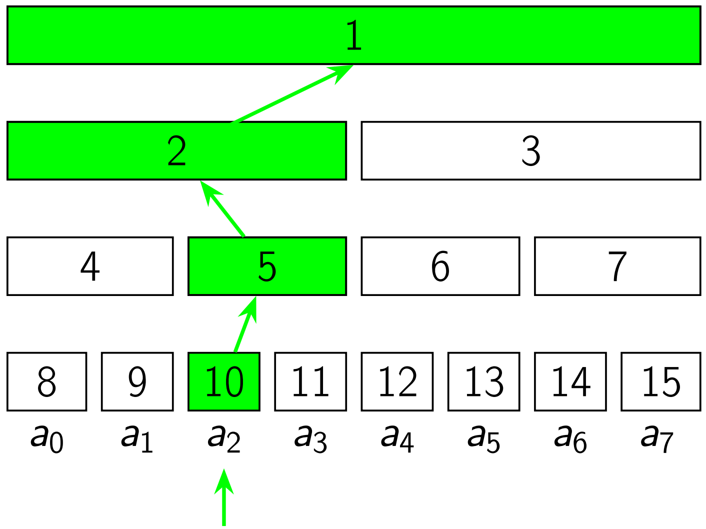
Modifying Element Upwards

- The process starts from the cell $v = x + N$ which is the leaf corresponding to the modified element.
- On each step, you double the size of the segment by moving $v \to \lceil v/2 \rceil$. There is exactly one segment on each level that contains the given element.



Modifying Element Upwards

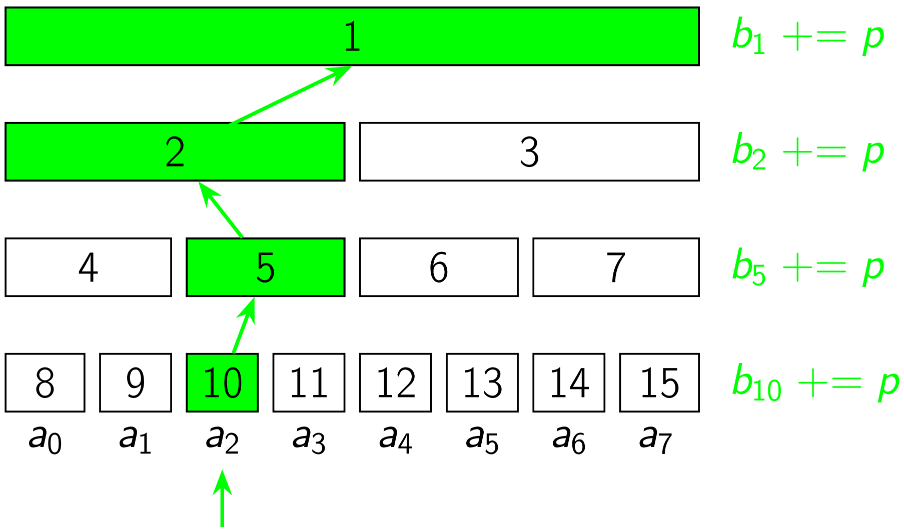
- The process starts from the cell $v = x + N$ which is the leaf corresponding to the modified element.
- On each step, you double the size of the segment by moving $v \to \lceil v/2 \rceil$. There is exactly one segment on each level that contains the given element.
- You must stop after processing the root of the tree.



Here is Code

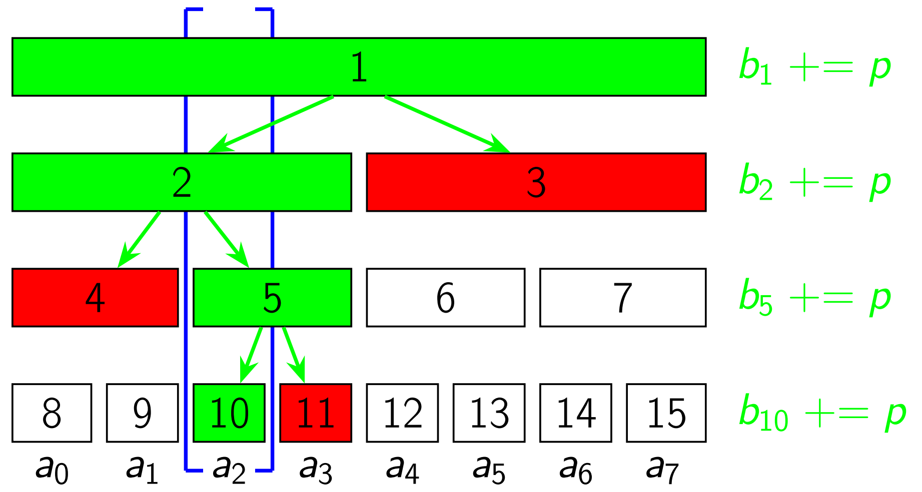
```
while (x > 0) {
  b[x] += p;
  x /= 2;
}
```

Here is a Picture (path of modifying)



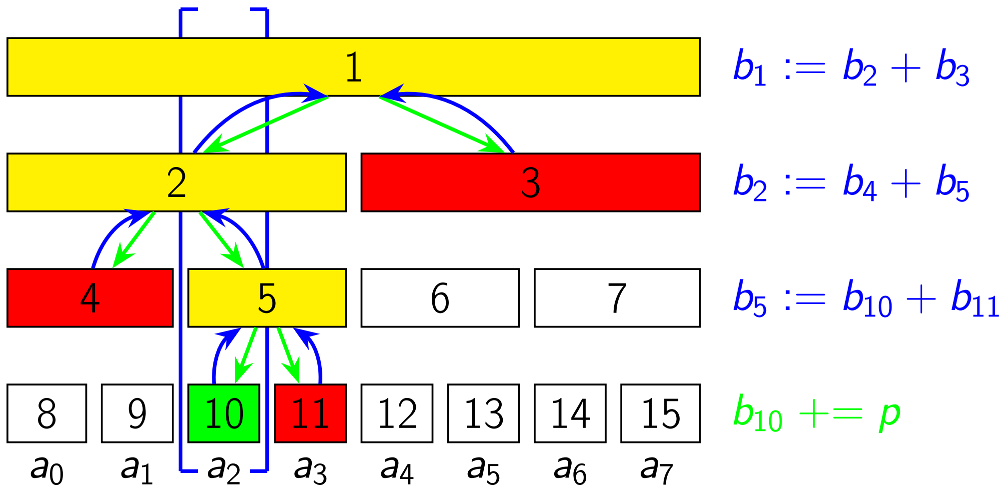
Modifying Element Downwards

- To modify an element downwards, one needs to do the same as if you calculate the sum of the segment. But now, if the intersection is non-empty, you should add the value p to the cell.



Modifying Element Downwards

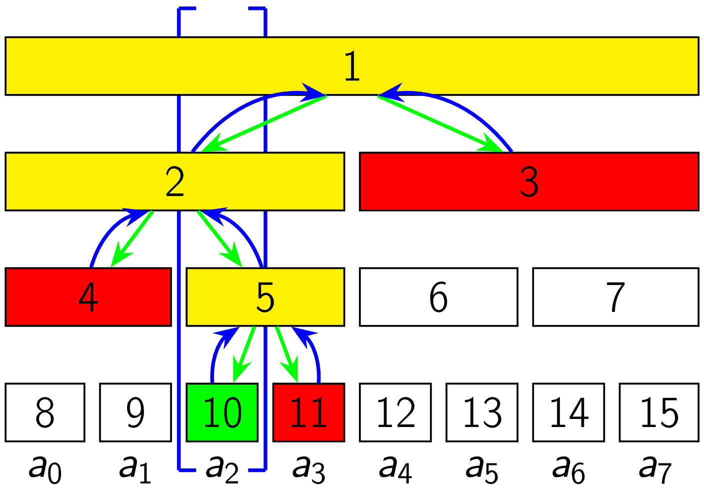
- To modify an element downwards, one needs to do the same as if you calculate the sum of the segment. But now, if the intersection is non-empty, you should add the value p to the cell.
- The similar method will be to add the value p to the cell only if the segment is fully covered (i.e. only in a leaf). If we are in general case (neither empty intersection nor fully covered segment), we just restore the sum in cell v by evaluating $b_v \leftarrow b_{2v} + b_{2v+1}$. In this case you can do assignment directly.



Here is Code

```
int assign (int x, int y, int l, int r,
    int v, int q) {
  if (r < x || y < l) return 0;
  if (x <= l && r <= y) {
    b[v] = q;
    return;
  }
  int m = (l + r) / 2;
  assign (x, y, l, m, 2 * v, q);
  assign (x, y, m + 1, r, 2 * v + 1, q);
  b[v] = b[2 * v] + b[2 * v + 1];
}
```

Here is a Picture (path of modifying)



$b_1 := b_2 + b_3$

$b_2 := b_4 + b_5$

$b_5 := b_{10} + b_{11}$

$b_{10} := q$

Summary

- We built a segment tree and now we can modify elements in array and calculate sum of any segment in logarithmic time.

Summary

- We built a segment tree and now we can modify elements in array and calculate sum of any segment in logarithmic time.
- There are two ways of traversing the tree: upwards and downwards.

Summary

- We built a segment tree and now we can modify elements in array and calculate sum of any segment in logarithmic time.
- There are two ways of traversing the tree: upwards and downwards.
- We only need the tree if you have some modification operations (otherwise use prefix sums).

Summary

- We built a segment tree and now we can modify elements in array and calculate sum of any segment in logarithmic time.
- There are two ways of traversing the tree: upwards and downwards.
- We only need the tree if you have some modification operations (otherwise use prefix sums).
- Exercise: The sum can be replaced by minimum or maximum or some other operations with a few tricks.