Lecture 12: Range Query

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Range Query

- Query a quantity in the given range of an 1D array
  - What is the minimum in A[i…j]?
  - What is the sum of A[i…j]?
- **Static array + Millions of queries**
  - Totally N^2 queries in an array of N elements
  - Q queries lead to O(N*Q) complexity
    - Simply scan the range per query
    - O(N) time per query
    - Can we do better?
- **Problem can extend to 2D, 3D, … ND cases**

Example

- **A[10] = [1, 7, 8 ,2, 4, 0, 8, 1, 2, 65]**
  - Query 1: find the minimum in A[0:9]
    - Answer: 0
  - Query 2: find the minimum in A[6:7]
    - Answer: 1
  - Query 3: find the minimum in A[4:6]
    - Answer: 0
  - Query 4: find the minimum in A[3:8]
    - Answer: 0
  - … million queries to follow

Sparse Table (Maximum Domain)

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: $j, j+1, j+2, \ldots, j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0								
1								
2								
3								

Sparse Table (Maximum Domain)

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: j, j+1, j+2, …, $j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0								
1								
2								
3								

entry[i][j] represent the maximum value **in the range [j : j +2^i)**

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: j, j+1, j+2, …, $j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2							
1								
2								
3								

entry[0][1] = max {1~1}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: j, j+1, j+2, …, $j+2^i$ - 1

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4						
1								
2								
3								

entry[0][2] = max {2~2}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: j, j+1, j+2, …, $j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5					
1								
2								
3								

entry[0][3] = max {3~3}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: j, j+1, j+2, …, $j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1				
1								
2								
3								

entry[0][4] = max {4~4}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: j, j+1, j+2, …, $j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5			
1								
2								
3								

entry[0][5] = max {5~5}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: $j, j+1, j+2, \ldots, j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8		
1								
2								
3								

entry[0][6] = max {6~6}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: $j, j+1, j+2, \ldots, j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	
1								
2								
3								

entry[0][7] = max {7~7}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point
  - Covered range: j, j+1, j+2, …, $j+2^i - 1$

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1								
2								
3								

entry[0][8] = max {8~8}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1								
2								
3								

entry[1][1] = max{entry[0][1], entry[0][1+1]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1								
2								
3								

entry[1][1] = max{entry[0][1], entry[0][1+1]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4							
2								
3								

entry[1][2] = max{entry[0][2], entry[0][2+1]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5						
2								
3								

entry[1][3] = max{entry[0][3], entry[0][3+1]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5					
2								
3								

entry[1][4] = max{entry[0][4], entry[0][4+1]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1				
2								
3								

entry[1][5] = max{entry[0][5], entry[0][5+1]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8			
2								
3								

entry[1][6] = max{entry[0][6], entry[0][6+1]}

Sparse Table

- ❑ Sparse Table
  - ❑ row i: the expanded size 2^i
  - ❑ column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	
2								
3								

…

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	
2								
3								

entry[2][1] = max{entry[1][1], entry[1][1+2]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	
2								
3								

entry[2][1] = max{entry[1][1], entry[1][1+2]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	
2	5							
3								

entry[2][2] = max{entry[1][2], entry[1][2+2]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	6
2	5	5						
3								

entry[2][3] = max{entry[1][3], entry[1][3+2]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	6
2	5	5	8	11				
3								

entry[2][4] = max{entry[1][4], entry[1][4+2]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	6
2	5	5	8	11	11			
3								

…

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	6
2	5	5	8	11	11			
3								

entry[3][1] = max{entry[2][1], entry[2][1+4]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	6
2	5	5	8	11	11			
3								

entry[3][1] = max{entry[2][1], entry[2][1+4]}

Sparse Table

- Sparse Table
  - row i: the expanded size 2^i
  - column j: the starting point

entry[i][j] =
max {entry[i-1][j], entry[i-1][j+mid]}

For entry[i][j] with i>0, we define mid = 2^{i-1}

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	6
2	5	5	8	11	11			
3	11							

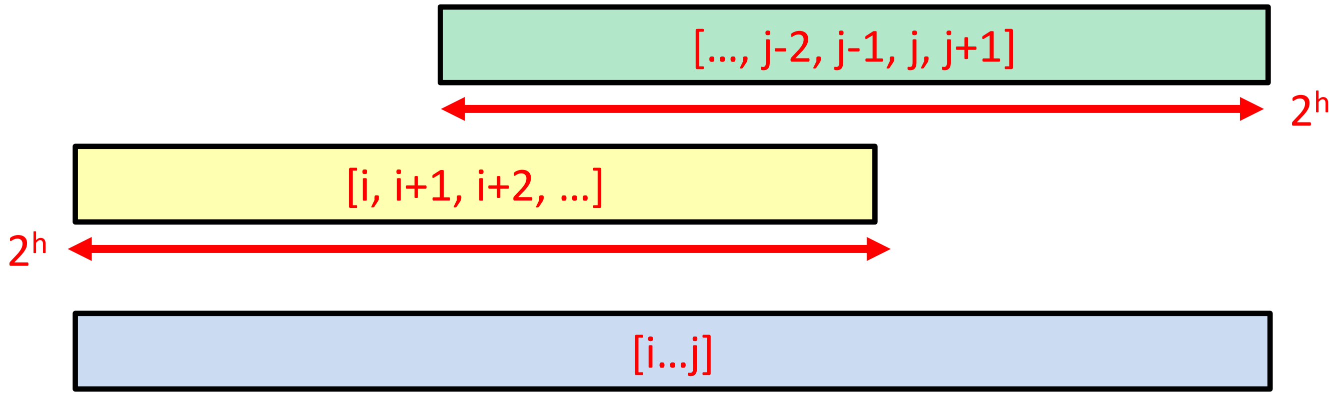
Final table…

Sparse Table

- Sparse Table
  - Constructed with O(NlogN)
- How to query the maximum one in an interval [i…j] ?
  - two sub-interval must be overlapped
  - define $h = \log_2(j+1-i)$
  - $\mathrm{Min}(\mathrm{sparse\_table}[h, i], \mathrm{sparse\_table}[h, j-2^h])$

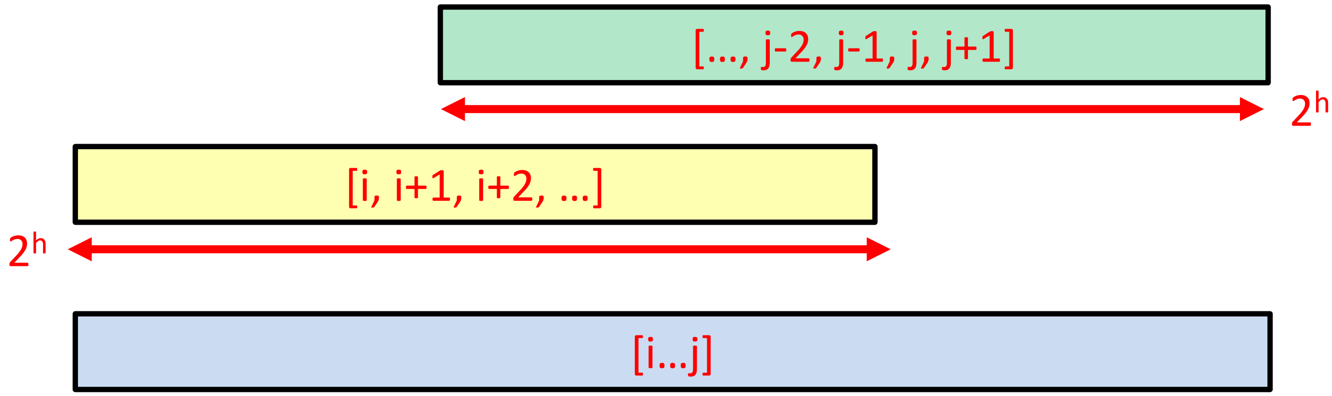
Sparse Table

- Sparse Table
  - Constructed with O(NlogN)
- How to query the maximum one in an interval [i…j] ?
  - two sub-interval must be overlapped
  - define $h = \log_2(j+1-i)$
  - Min(sparse_table[h, i], sparse_table[h, j-2^h])



Sparse Table

- Sparse Table
  - Constructed with O(NlogN)
- How to query the maximum one in an interval [i…j] ?
  - two sub-interval must be overlapped
  - define $h = \log_2(j+1-i)$
  - Min(sparse_table[h, i], sparse_table[h, j-2^h])



We can answer each query with constant time!

Time Complexity of Query

- **Given an array of N elements, we have Q queries**
- **Naïve method**
  - Construction time: O(1)
  - Query time: O(Q*N)
- **Sparse table method**
  - Construction time: O(NlogN)
  - Query time: O(Q)
- **What if N is too large … ?**

Practice 1

- **Construct the sparse table in the minimum domain**
  - Need to quickly probe the max value of A[i, j]

	1 **(2)**	2 **(4)**	3 **(5)**	4 **(1)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0								
1								
2								
3								

Update the Value of an Entry?

- Sometimes, we need to update the entry
  - For instance, change the 4th element to 100

	1 **(2)**	2 **(4)**	3 **(5)**	4 ~~**(1)**~~ **(100)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	1	-5	8	11	6
1	4	5	5	1	8	11	11	6
2	5	5	8	11	11			
3	11							

Update the Value of an Entry?

- Sometimes, we need to update the entry
  - For instance, change the 4th element to 100

	1 **(2)**	2 **(4)**	3 **(5)**	4 ~~(1)~~ **(100)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	**100**	-5	8	11	6
1	4	5	5	1	8	11	11	6
2	5	5	8	11	11			
3	11							

Update the Value of an Entry?

- Sometimes, we need to update the entry
  - For instance, change the 4th element to 100

	1 **(2)**	2 **(4)**	3 **(5)**	4 **~~(1)~~** **(100)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	**100**	-5	8	11	6
1	4	5	**100**	**100**	8	11	11	6
2	5	5	8	11	11			
3	11							

Update the Value of an Entry?

- Sometimes, we need to update the entry
  - For instance, change the 4th element to 100

	1 **(2)**	2 **(4)**	3 **(5)**	4 ~~**(1)**~~ **(100)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	**100**	-5	8	11	6
1	4	5	**100**	**100**	8	11	11	6
2	**100**	**100**	**100**	**100**	11			
3	11							

Update the Value of an Entry?

- Sometimes, we need to update the entry
  - For instance, change the 4th element to 100

	1 **(2)**	2 **(4)**	3 **(5)**	4 ~~**(1)**~~ **(100)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	**100**	-5	8	11	6
1	4	5	**100**	**100**	8	11	11	6
2	**100**	**100**	**100**	**100**	11			
3	**100**							

Update the Value of an Entry?

- Sometimes, we need to update the entry
  - For instance, change the 4th element to 100

	1 **(2)**	2 **(4)**	3 **(5)**	4 ~~**(1)**~~ **(100)**	5 **(-5)**	6 **(8)**	7 **(11)**	8 **(6)**
0	2	4	5	**100**	-5	8	11	6
1	4	5	**100**	**100**	8	11	11	6
2	**100**	**100**	**100**	**100**	11			
3	**100**							

What is the complexity of updating an entry?

Updating a Sparse Table

- **O(N) time to update a sparse table**
  - Number of operations: $1 + 2 + 4 + 8 + 16 + \ldots + 2^h$
  - What is the value of h?

Range Query with Update Operation

- **Operations**
  1. Increase/decrease the value of an element
  2. Query the quantity within the interval [i…j]
     1. We will use summation for demonstration
  3. Update and Query can interleave
- **How to solve this problem efficiently?**
  - If there are many update operations
    - Sparse table may be slow

Low Bit

- **The least significant bit**
  - 000010 -> low bit at the 2nd bit
  - 000111 -> low bit at the 1st bit
  - 001101 -> low bit at the 1st bit
  - 011000 -> low bit at the 4th bit
- **How do we mask out other bits to have low bit only?**
  - 000010 -> 000010
  - 011101 -> 000001

Low Bit

- **The least significant bit**
  - 000010 -> low bit at the 2^{nd} bit
  - 000111 -> low bit at the 1^{st} bit
  - 001101 -> low bit at the 1^{st} bit
  - 011000 -> low bit at the 4^{th} bit
- **How do we mask out other bits to have low bit only?**
  - 000010 -> 2’s complement -> 111110
    - 000010 & 111110 = 000010
  - 011101 -> 2’s complement -> 100011
    - 011101 & 100011 = 000001

Low Bit

- The least significant bit
  - 000010 -> low value 1
  - 000111 -> low value 0
  - 001101 -> low value 0
  - 011000 -> low value 3
- How do we mask out other bits to have low bit only?
  - 000010 -> 2’s complement -> 111110
    - 000010 & 111110 = 000010
  - 011101 -> 2’s complement -> 100011
    - 011101 & 100011 = 000001
- So we can do (v & -v) to find the low bit

C++ Code to Find the Low Bit

```
int lowbit (int in) {
   return in&(-in);
}
```

Practice 2

- **Layout the binary representation of the following:**
  - 5 & -5
  - 4 & -4
  - 18 & -18

What is the point?

- **Observe**
  - 6: 00110
  - -6: 11010
  - 6 & -6 = 00010
  - 6 + (6 & -6) = 00110 + 00010 = 01000 = 8

What is the point?

- **Observe**
  - 10: 01010
  - -10: (01001)’ = 10110
  - 10 & -10 = 00010
  - 10 + (10 & -10) = 01010 + 00010 = 01100 = 12
- **Iterative the above operation, i.e., $a_{i+1} = a_i$ & $-a_i$**
  - 12: 01100
  - -12: (01011)’ = 10100
  - 12 & -12 = 00100
  - 12 + (12 & -12) = 01100 + 00100 = 10000 = 16

What is the point?

- Observe
  - 10: 01010
  - -10: (01001)’ = 10110
  - 10 & -10 = 00010
  - 10 + (10 & -10) = 01010 + 00010 = 01100 = 12
- Iterative the above operation, i.e., $a_{i+1} = a_i$ & $-a_i$
  - 12: 01100
  - -12: (01011)’ = 10100
  - 12 & -12 = 00100
  - 12 + (12 & -12) = 01100 + 00100 = 10000 = 16

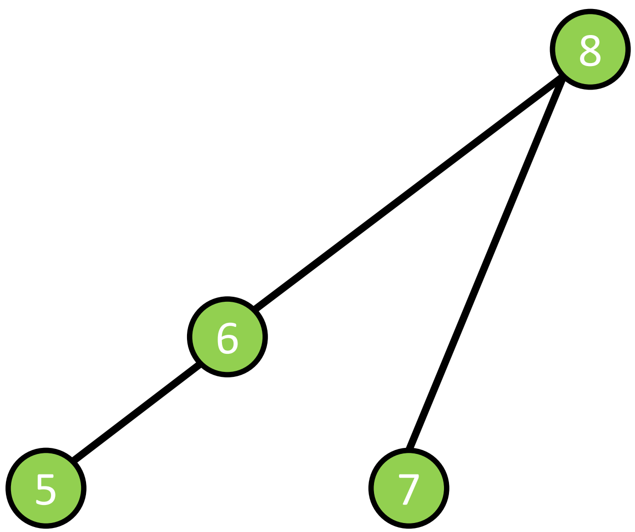
Eventually the number becomes a power of 2 and so forth

Practice 3

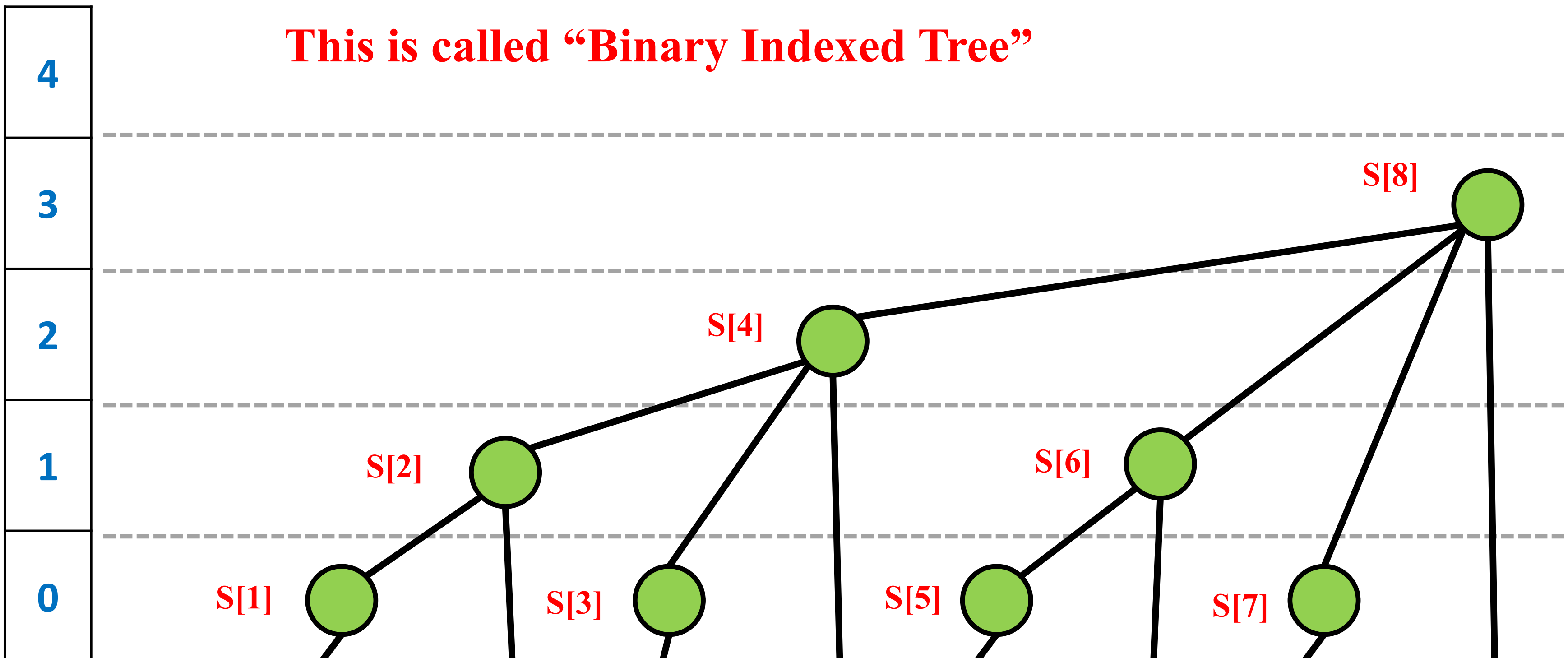
- **Try running $a_{i+1} = a_i$ & $-a_i$ on 21 (10101'b)**
  - How many iterations you end up with?

Binary Indexed Tree

- **For a given number "N", parent it to "N + (N &-N)"**
  - 5's parent is 5 + (5 & -5) = 6
  - 6's parent is 6 + (6 & -6) = 8
  - 7's parent is 7 + (7 & -7) = 8



Binary Indexed Tree



V[i]	1	2	3	4	5	6	7	8
bit	0001 (0)	0010 (1)	0011 (0)	0100 (2)	0101 (0)	0110 (1)	0111 (0)	1000 (3)

Range Each i Covers

$s[1] = v[1]$

$s[2] = v[2] + \mathbf{s[1]}$

$s[3] = v[3]$

$s[4] = v[4] + \mathbf{s[3] + s[2]}$

$s[5] = v[5]$

$s[6] = v[6] + \mathbf{s[5]}$

$s[7] = v[7]$

$s[8] = v[8] + \mathbf{s[7] + s[6] + s[4]}$

…

Recap the Lowbit Function

Define:

```
int lowbit (int in)
{
  return in&(-in);
}
```

ex:

lowbit(1) = 1

lowbit(2) = 2

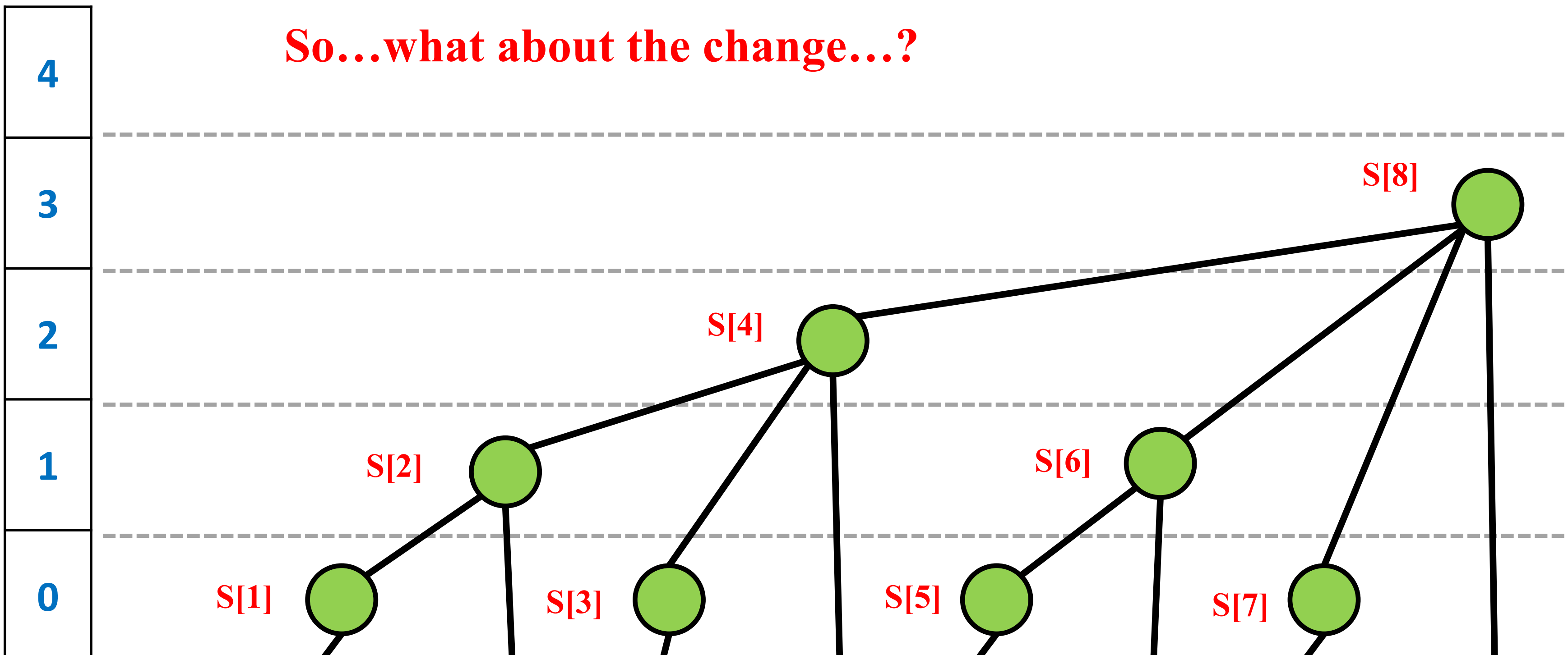
lowbit(3) = 1

lowbit(4) = 4

…

Binary Indexed Tree

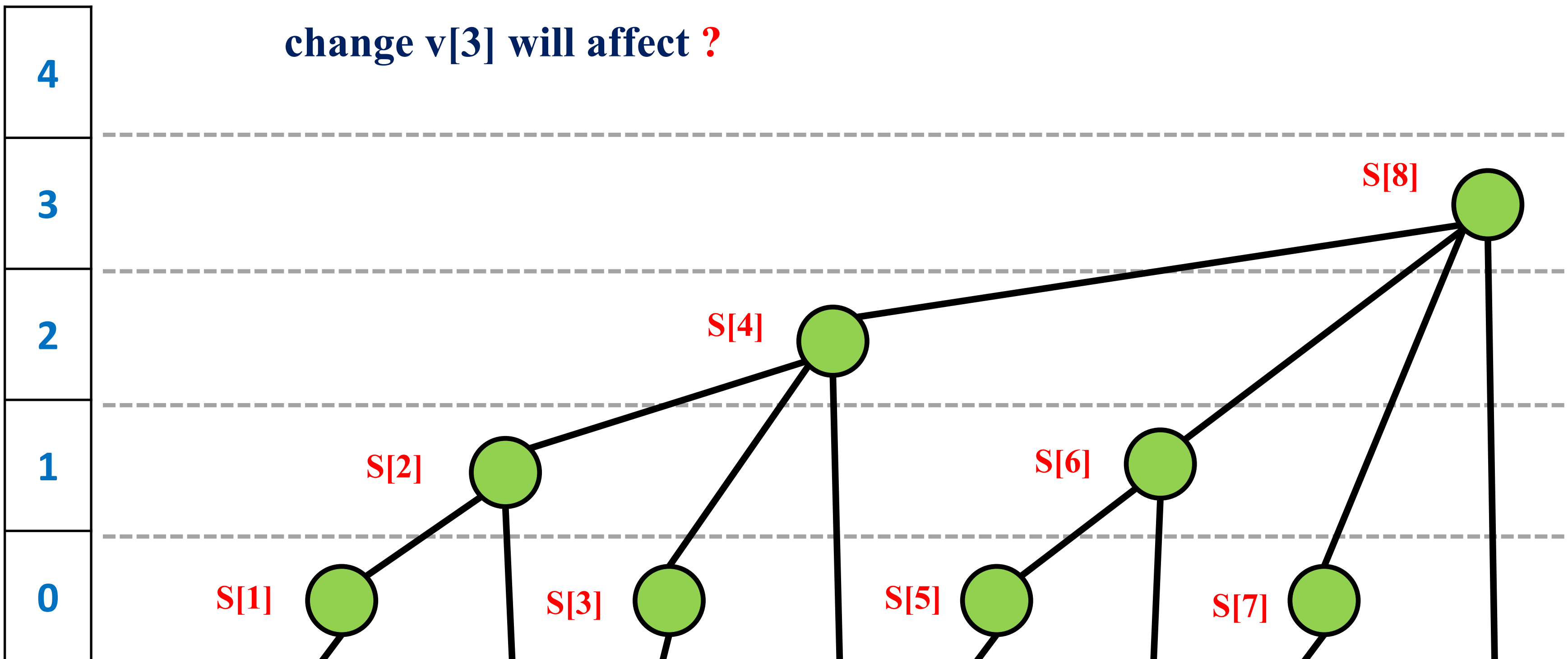
So…what about the change…?



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8

Binary Indexed Tree

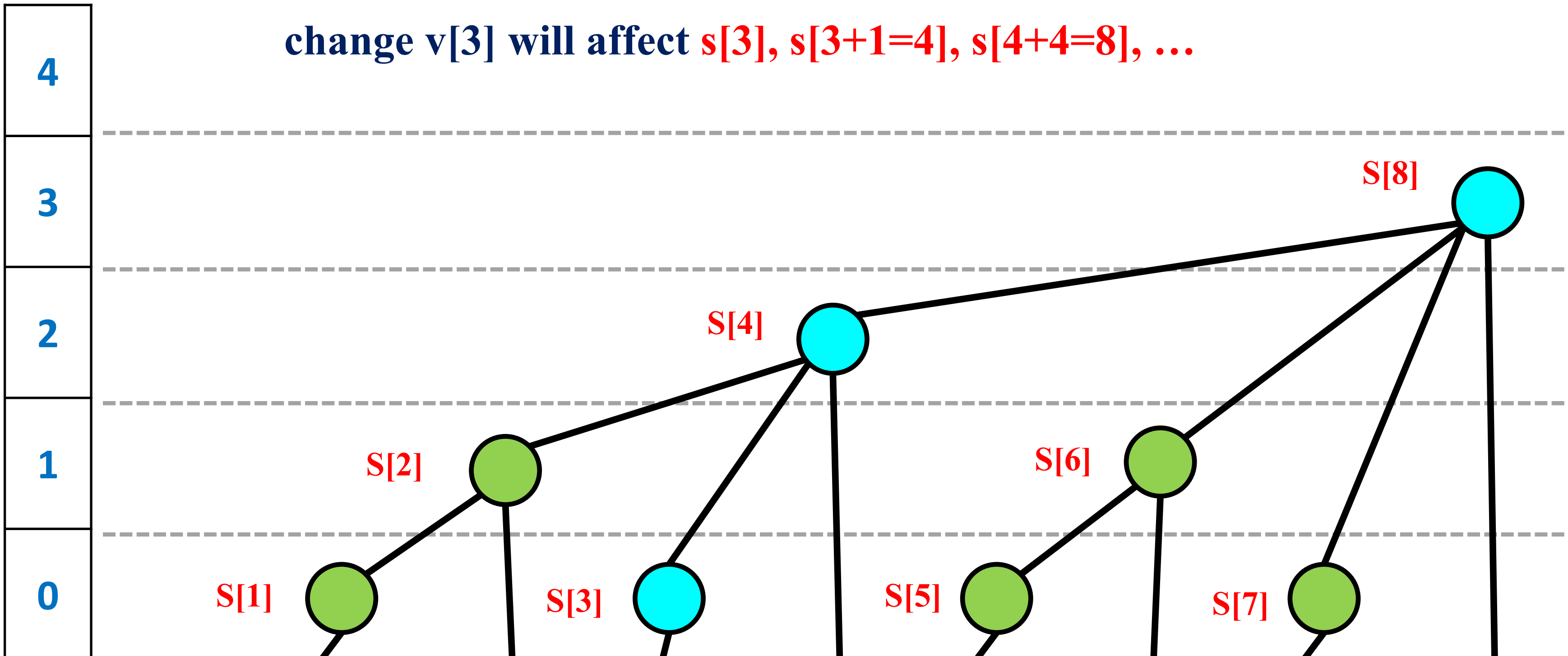
change v[3] will affect ?



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8

Binary Indexed Tree

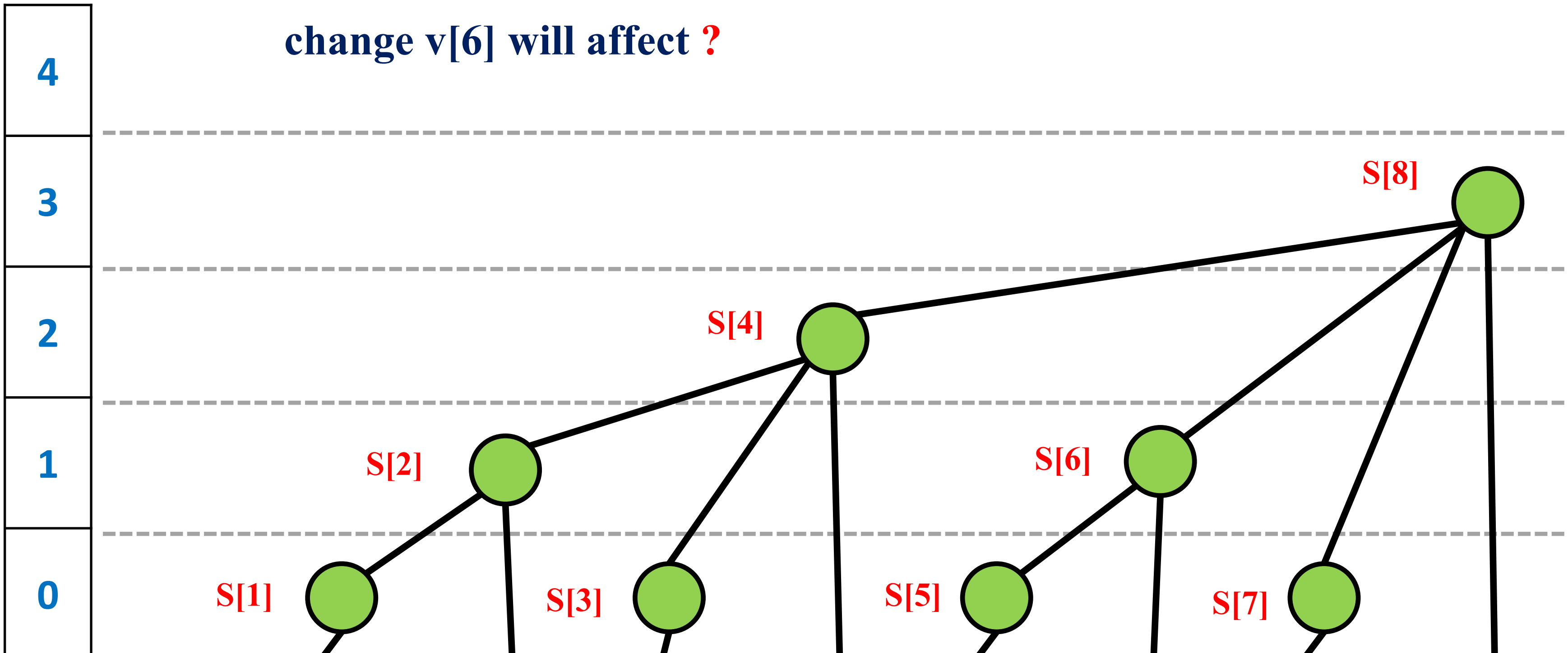
change v[3] will affect s[3], s[3+1=4], s[4+4=8], …



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8

Binary Indexed Tree

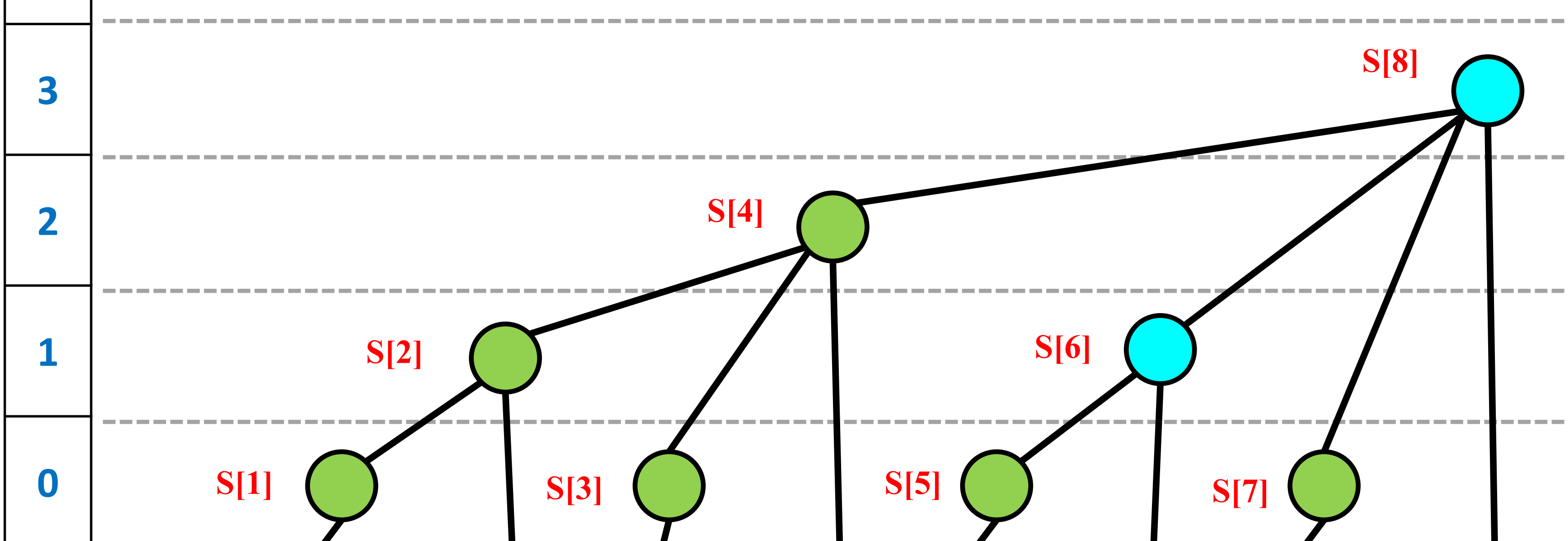
change v[6] will affect ?



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8

Binary Indexed Tree

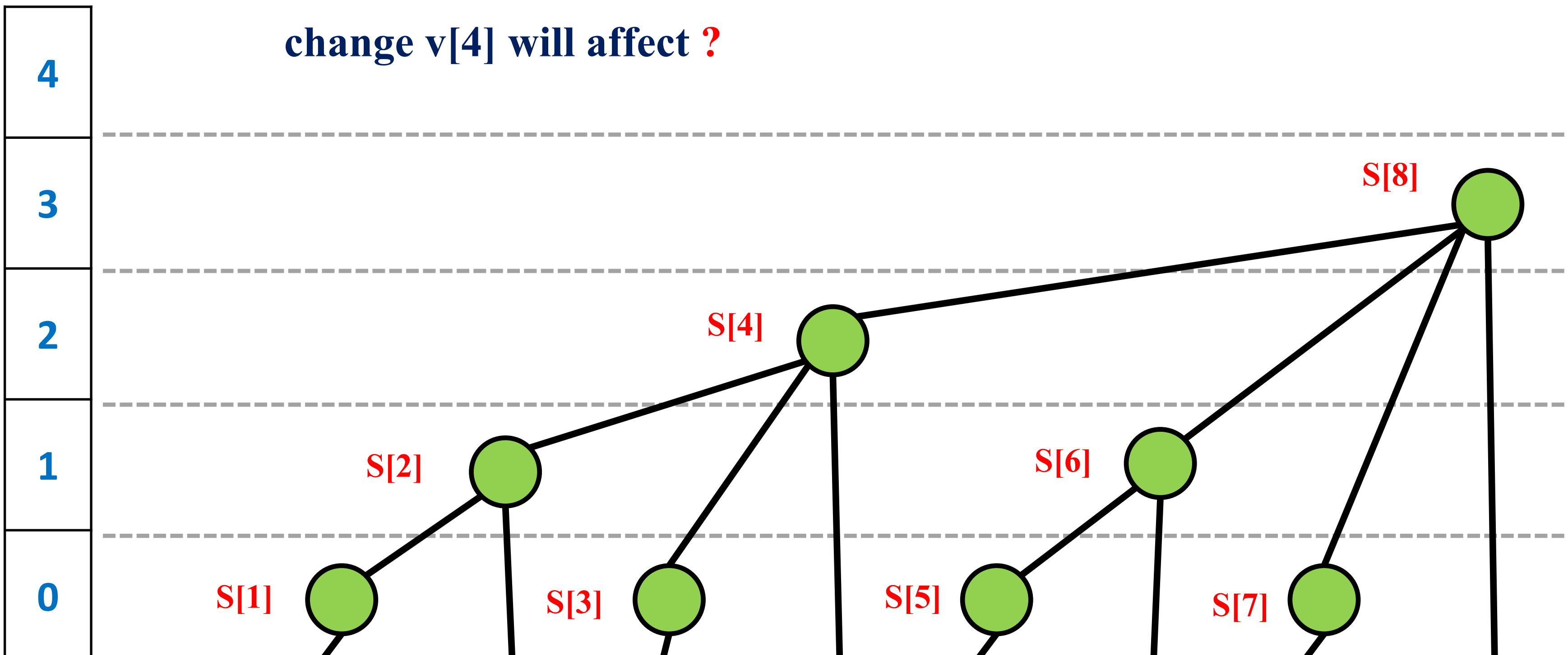
change v[6] will affect s[6], s[6+2=8], …



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8

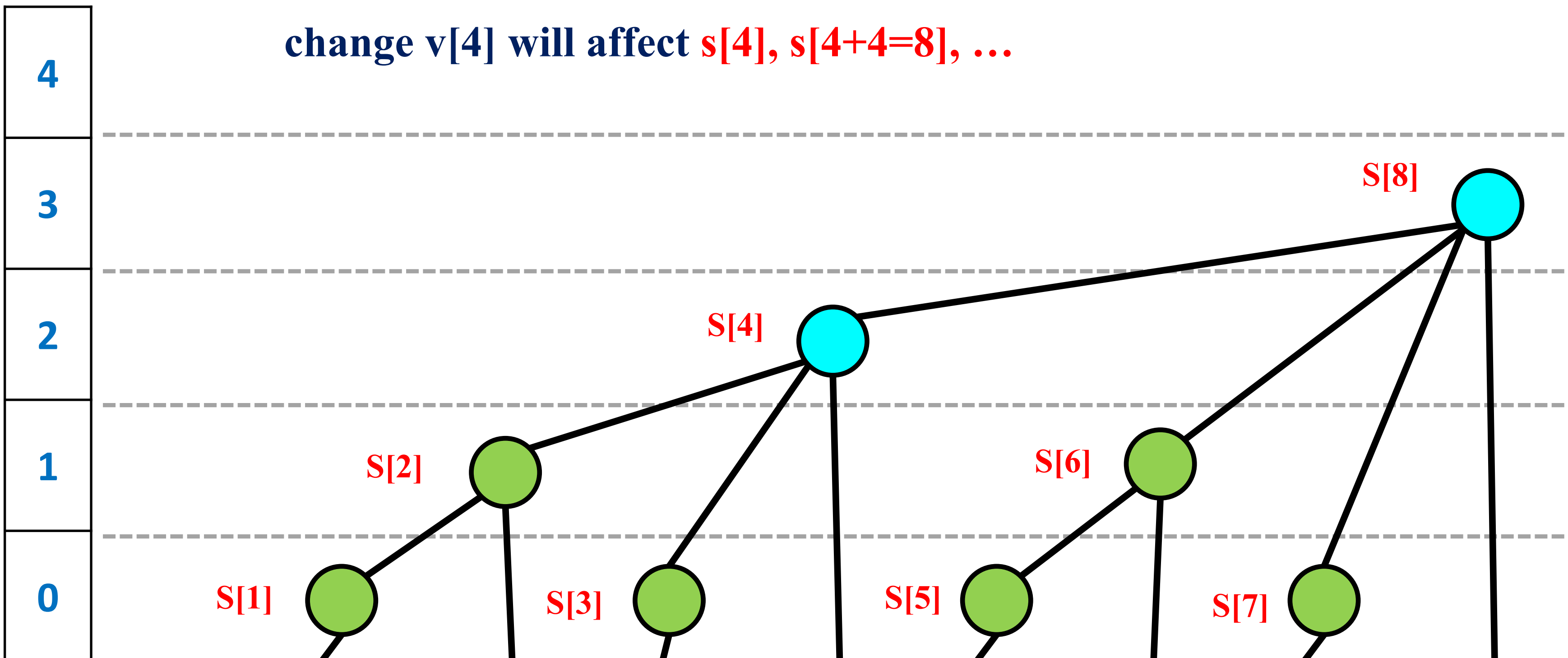
Binary Indexed Tree

change v[4] will affect ?



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8

Binary Indexed Tree



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8

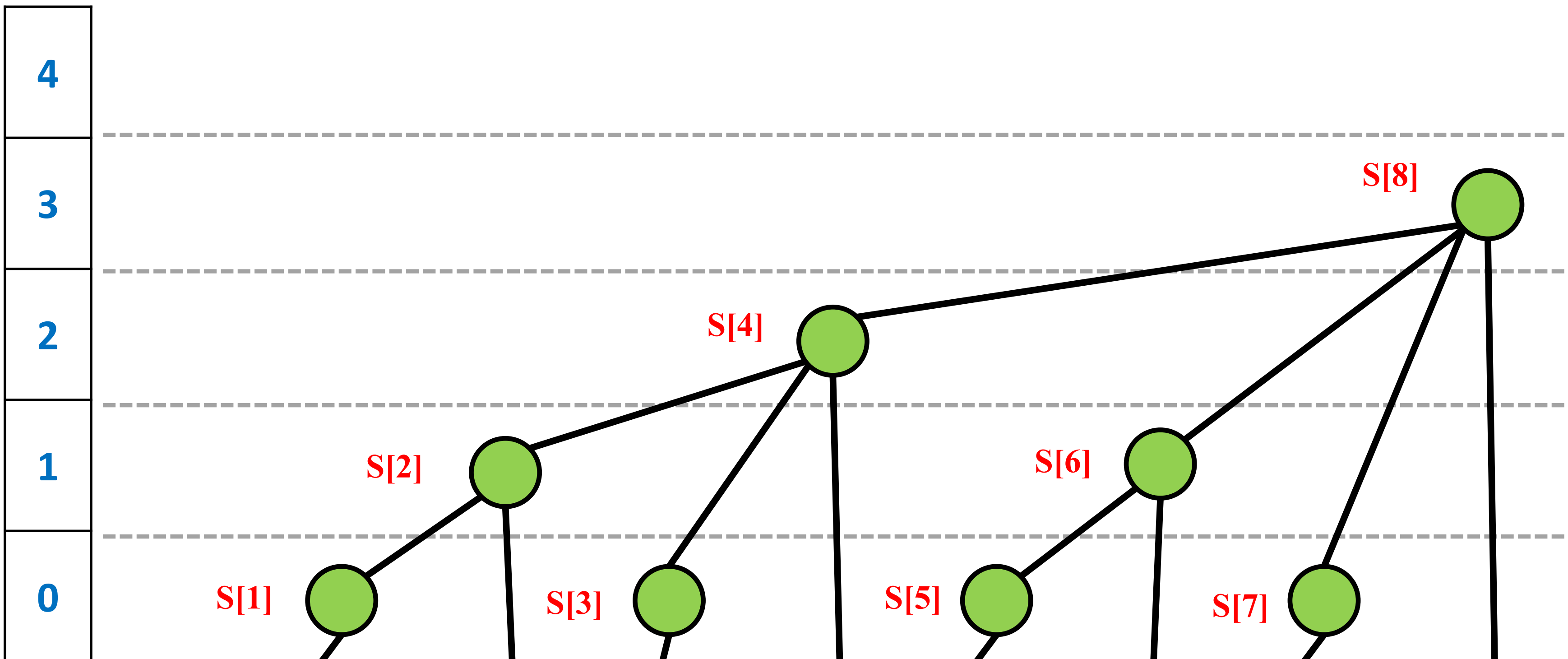
Binary Indexed Tree

Define:

```
int change (int i, int delta) {
    while(i<=maxsize) {
        s[i] += delta;          // can adapt to different types of operations
        i += lowbit(i);
    }
}
```

We use the function "`change`" to initialize the binary indexed tree!

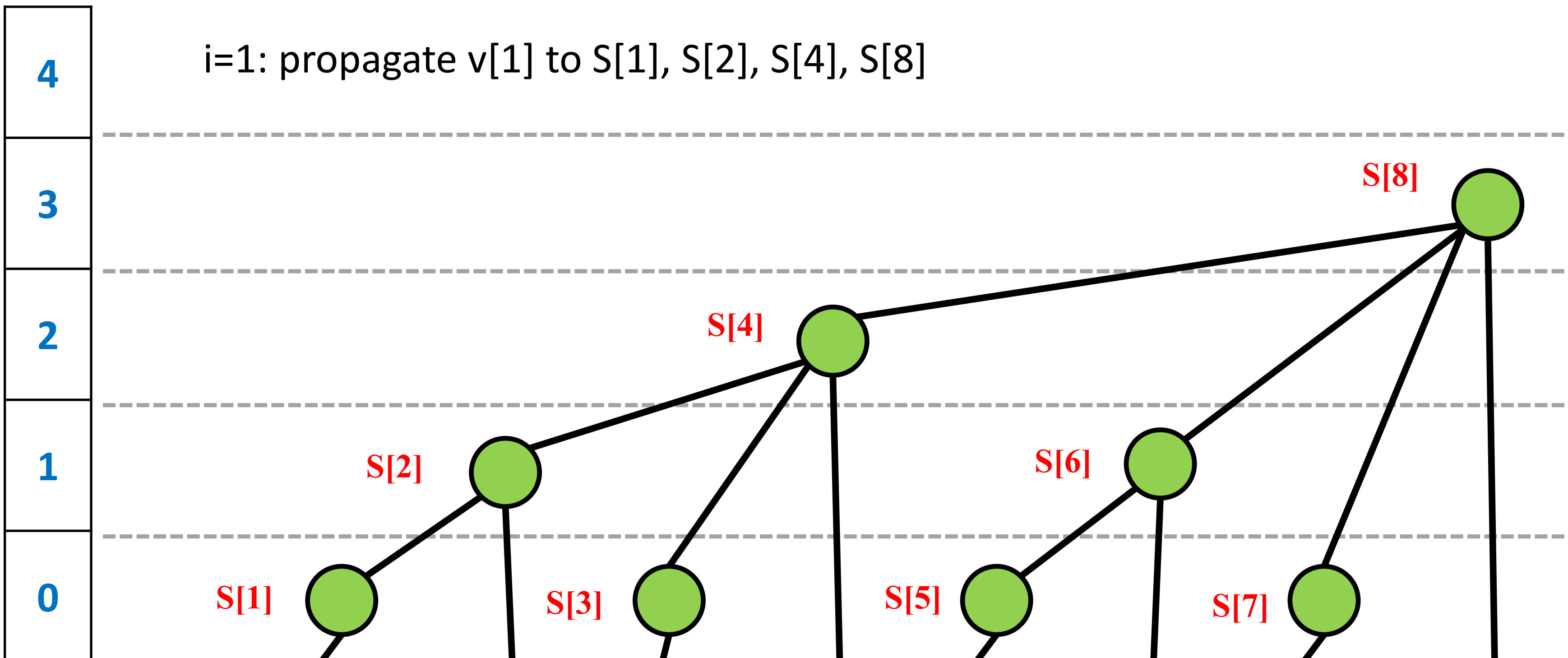
Binary Indexed Tree



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8
S[i]								

Binary Indexed Tree

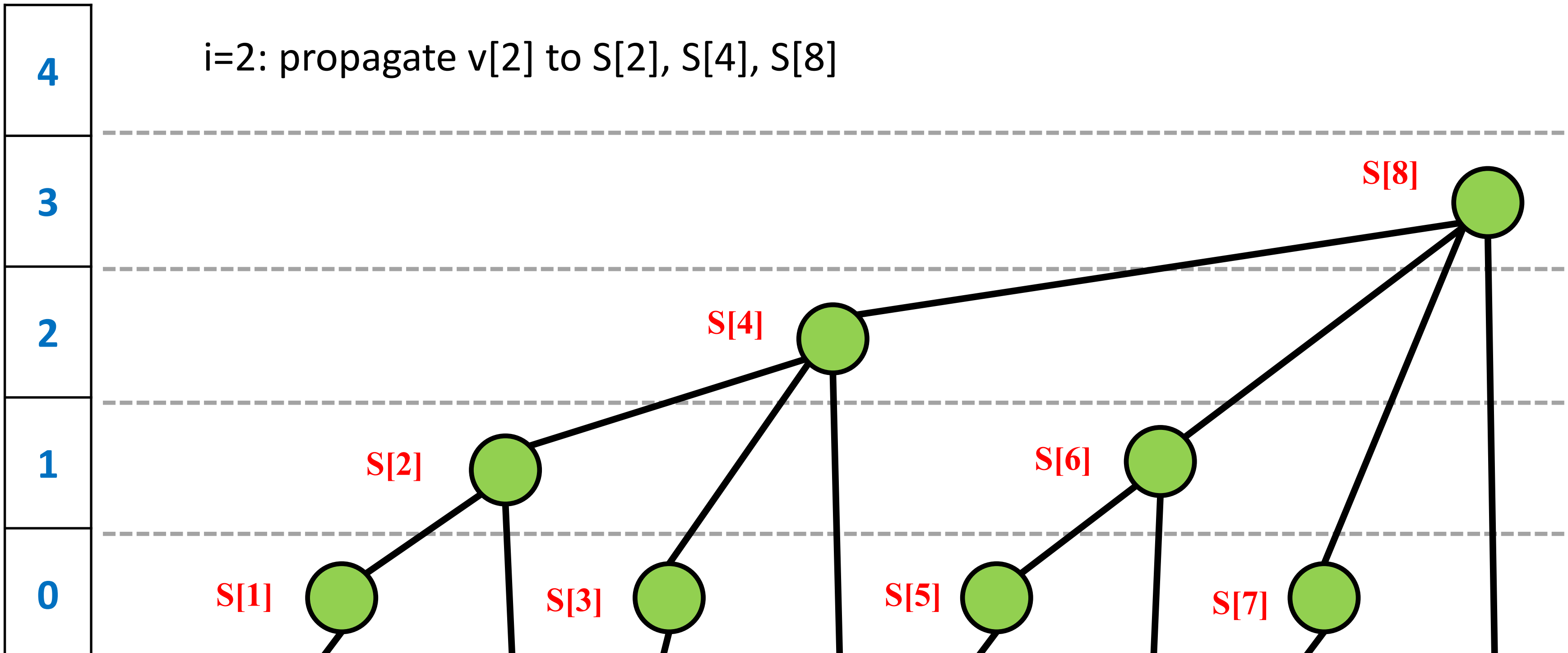
i=1: propagate v[1] to S[1], S[2], S[4], S[8]



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8
S[i]	1	1		1				1

Binary Indexed Tree

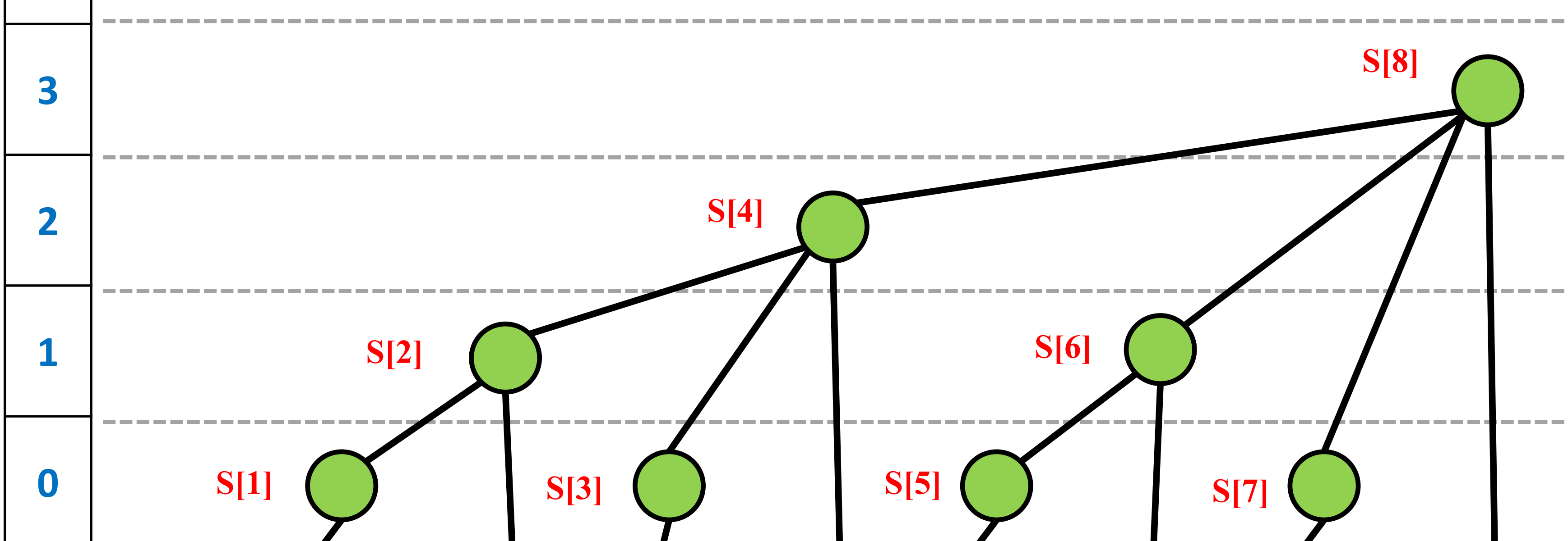
i=2: propagate v[2] to S[2], S[4], S[8]



V[i]	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
lowbit	**1**	**2**	**1**	**4**	**1**	**2**	**1**	**8**
S[i]	**1**	**3**		**3**				**3**

Binary Indexed Tree

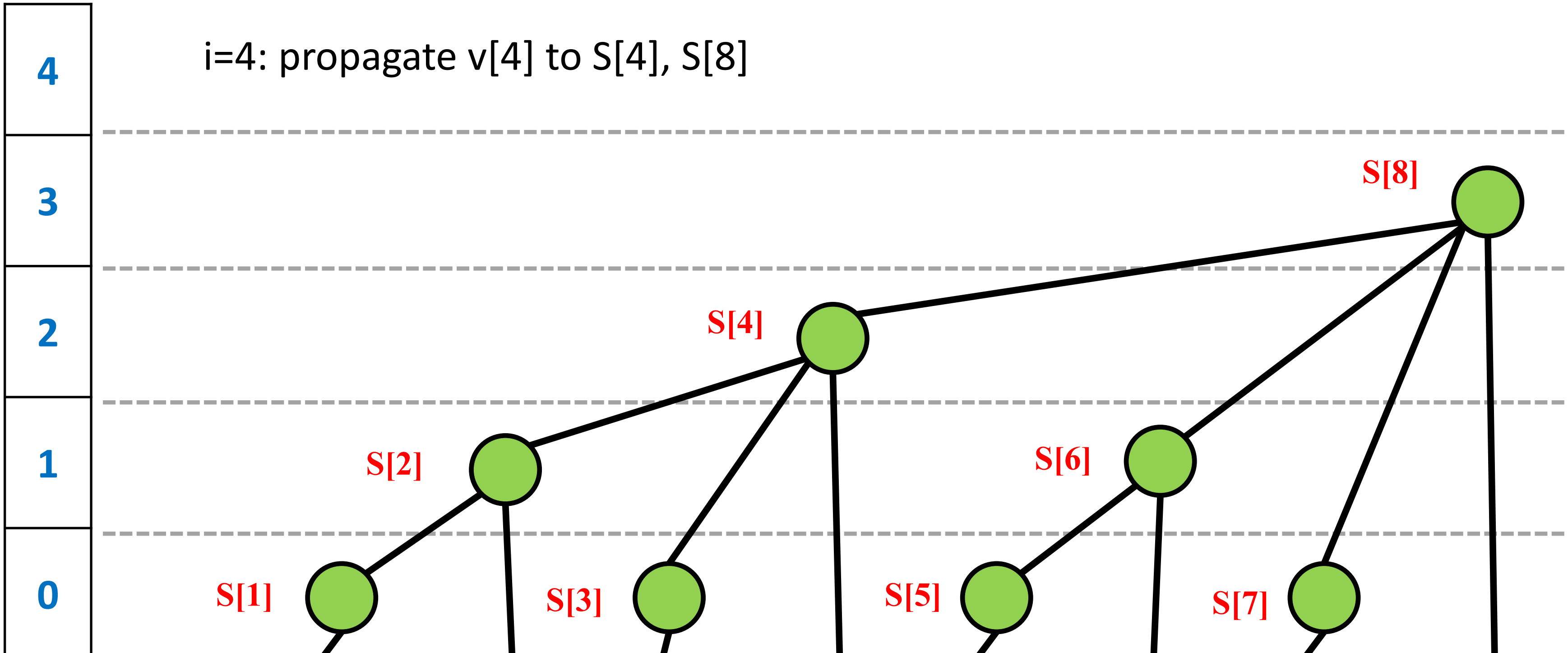
i=3: propagate v[3] to S[3], S[4], S[8]



V[i]	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
lowbit	**1**	**2**	**1**	**4**	**1**	**2**	**1**	**8**
S[i]	**1**	**3**	**3**	**6**				**6**

Binary Indexed Tree

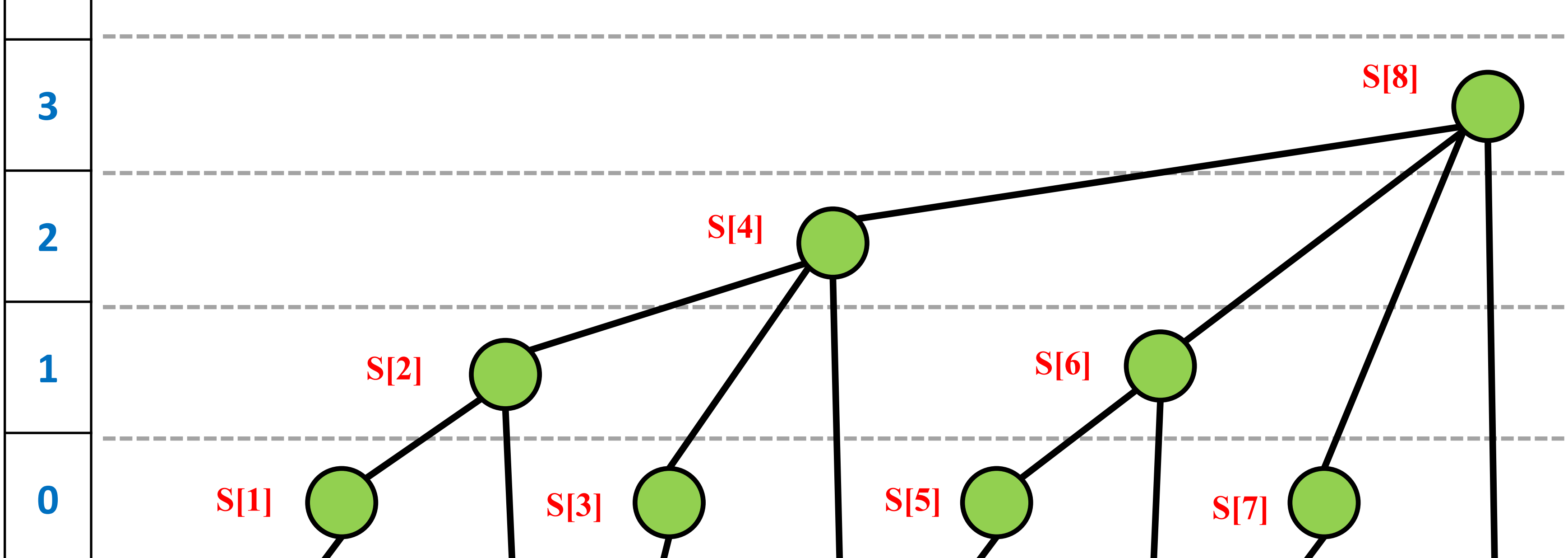
i=4: propagate v[4] to S[4], S[8]



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8
S[i]	1	3	3	10				10

Binary Indexed Tree

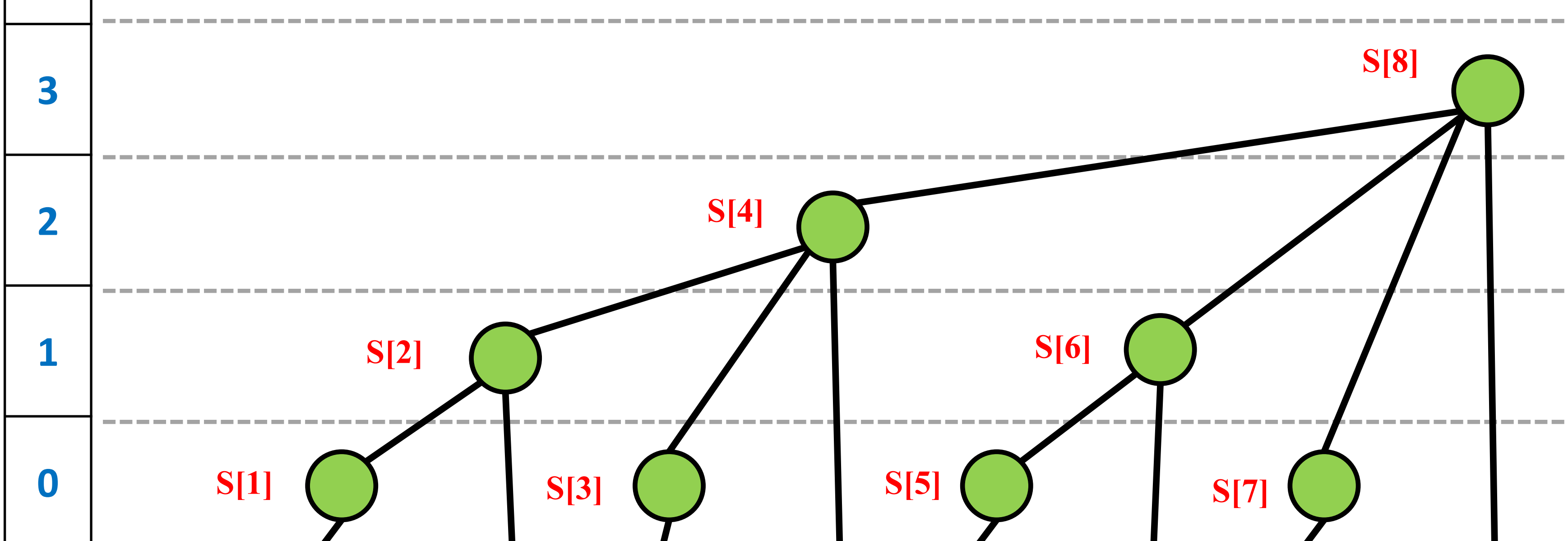
i=5: propagate v[5] to S[5], S[6], S[8]



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8
S[i]	1	3	3	10	5	5		15

Binary Indexed Tree

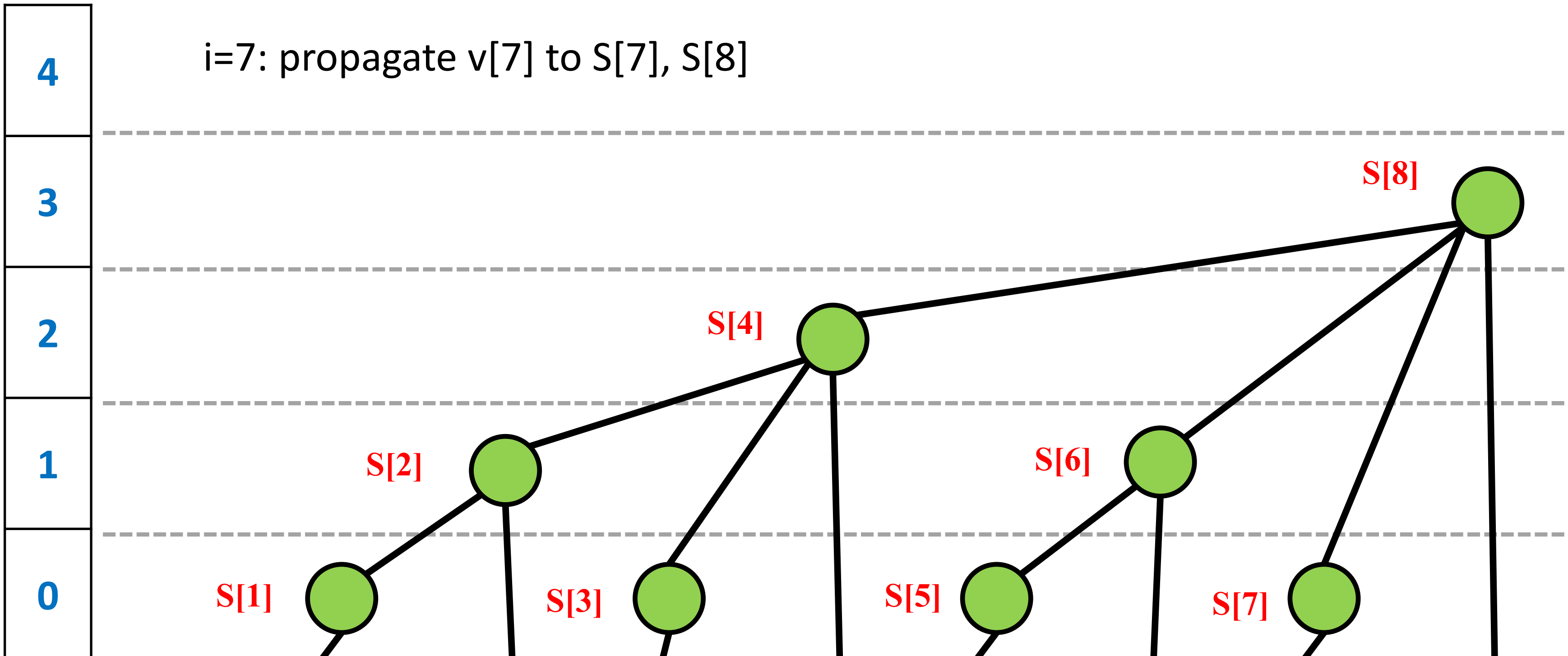
i=6: propagate v[6] to S[6], S[8]



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8
S[i]	1	3	3	10	5	11		21

Binary Indexed Tree

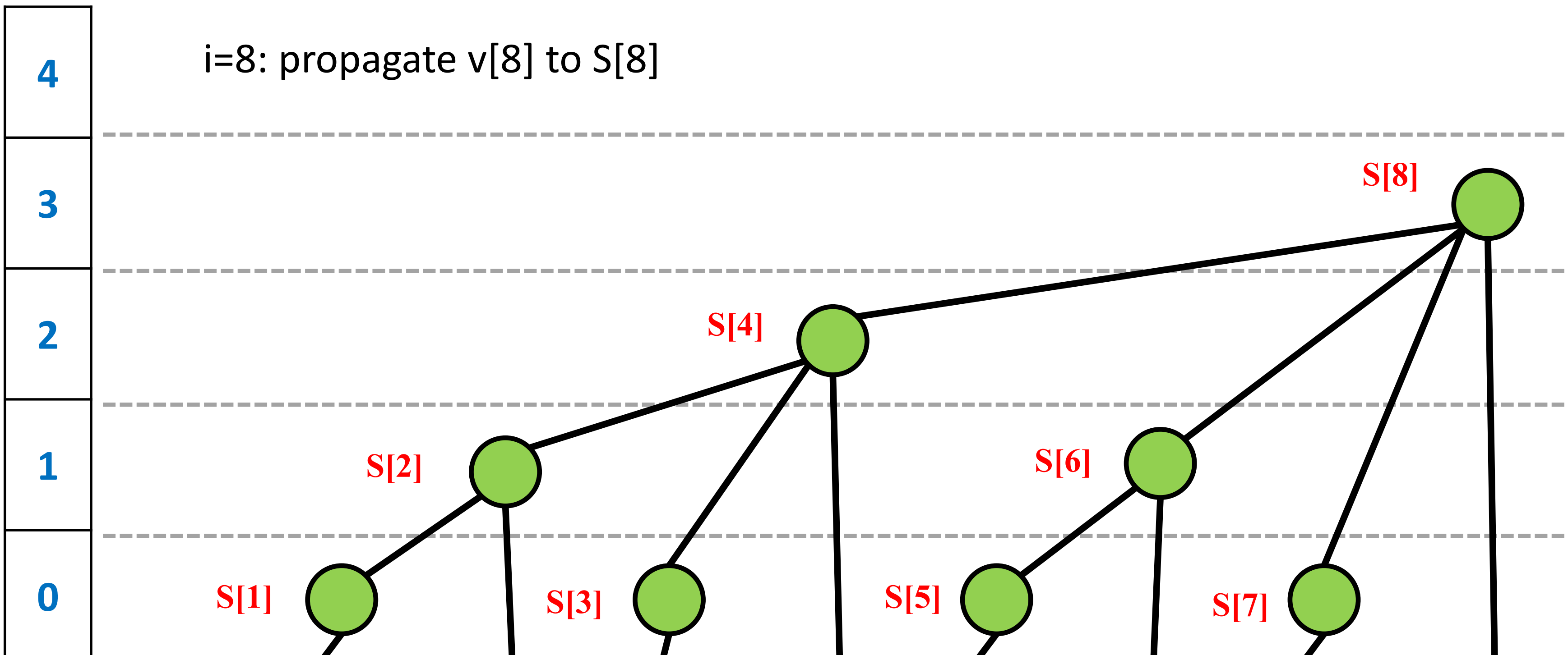
i=7: propagate v[7] to S[7], S[8]



V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8
S[i]	1	3	3	10	5	11	7	28

Binary Indexed Tree

i=8: propagate v[8] to S[8]



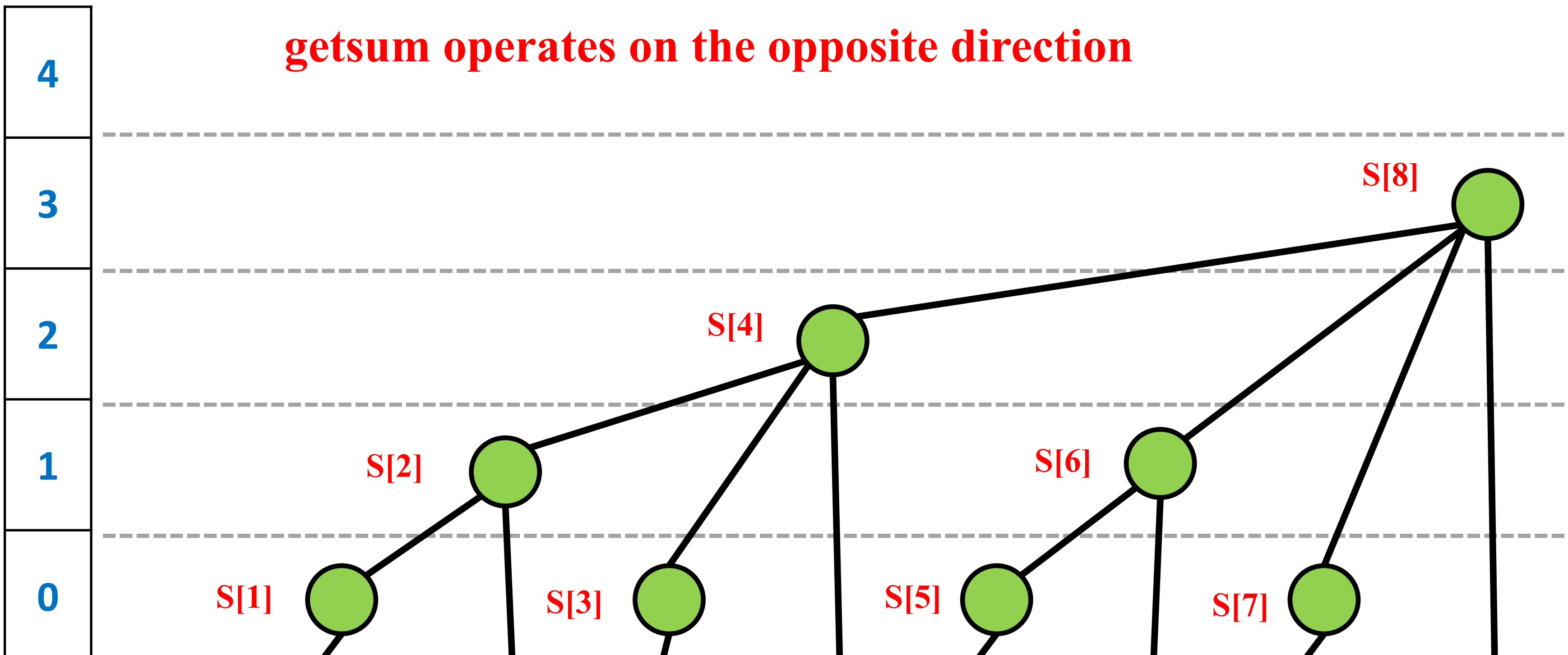
V[i]	1	2	3	4	5	6	7	8
lowbit	1	2	1	4	1	2	1	8
S[i]	1	3	3	10	5	11	7	36

Time Complexity?

- Each entry i runs up to O(log(N)) times
- Totally O(NlogN)

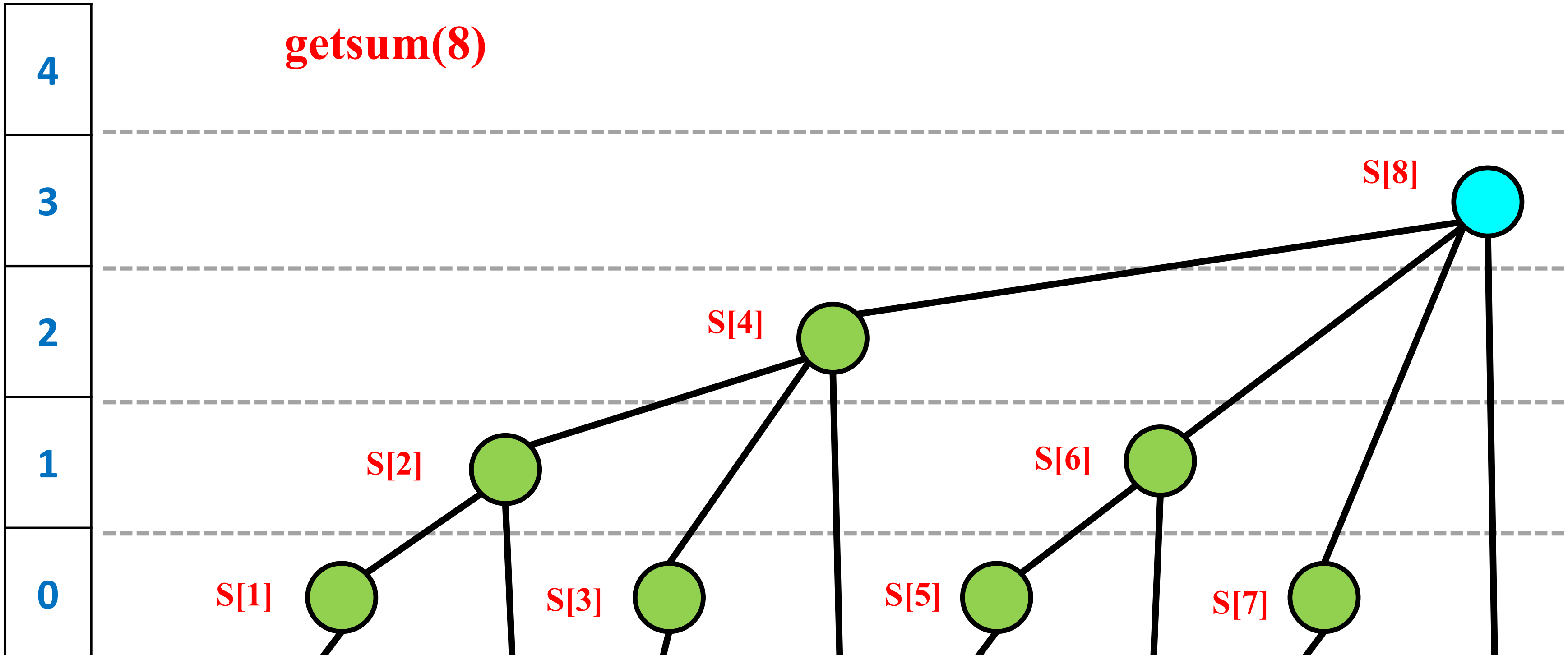
Binary Indexed Tree

getsum operates on the opposite direction



V[i]	1	2	3	4	5	6	7	8
bit	0001 (0)	0010 (1)	0011 (0)	0100 (2)	0101 (0)	0110 (1)	0111 (0)	1000 (3)
S[i]	1	3	3	10	5	11	7	36

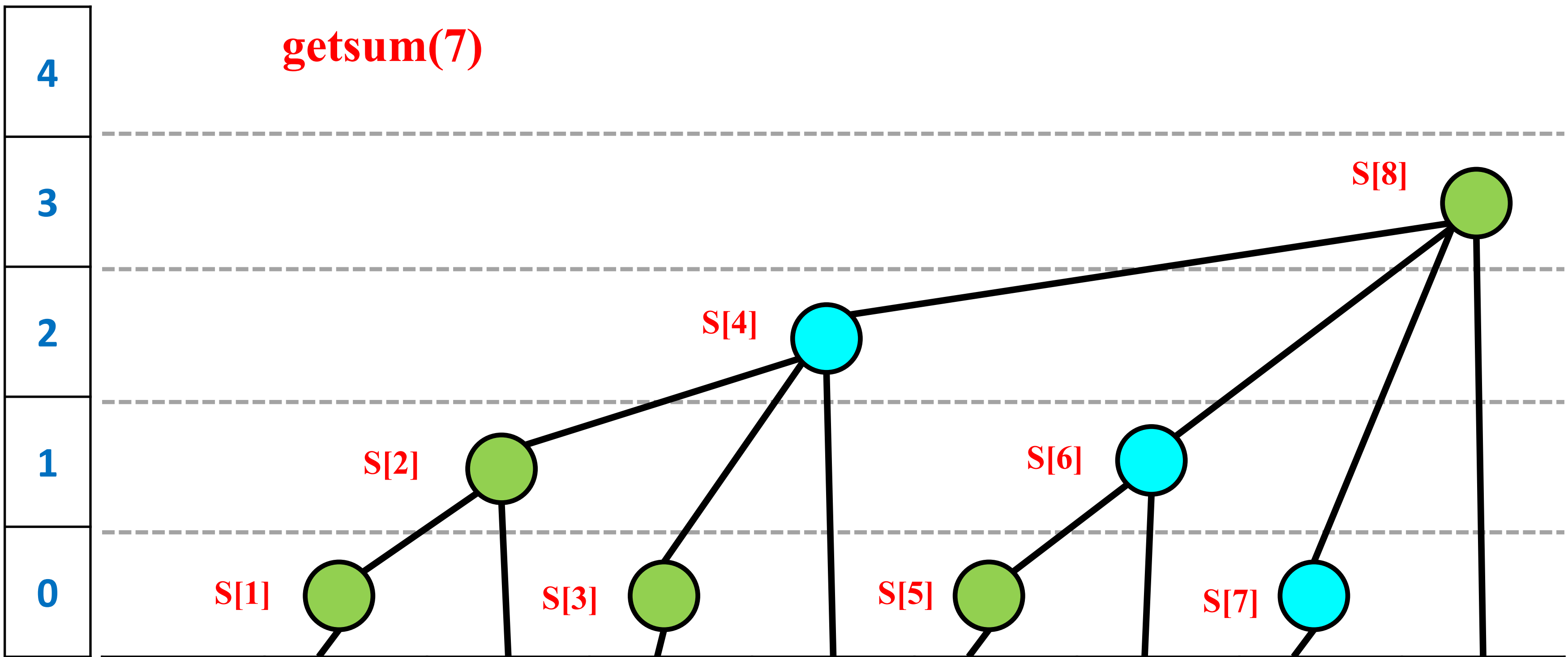
Binary Indexed Tree



V[i]	1	2	3	4	5	6	7	8
bit	0001 (0)	0010 (1)	0011 (0)	0100 (2)	0101 (0)	0110 (1)	0111 (0)	1000 (3)
S[i]	1	3	3	10	5	11	7	36

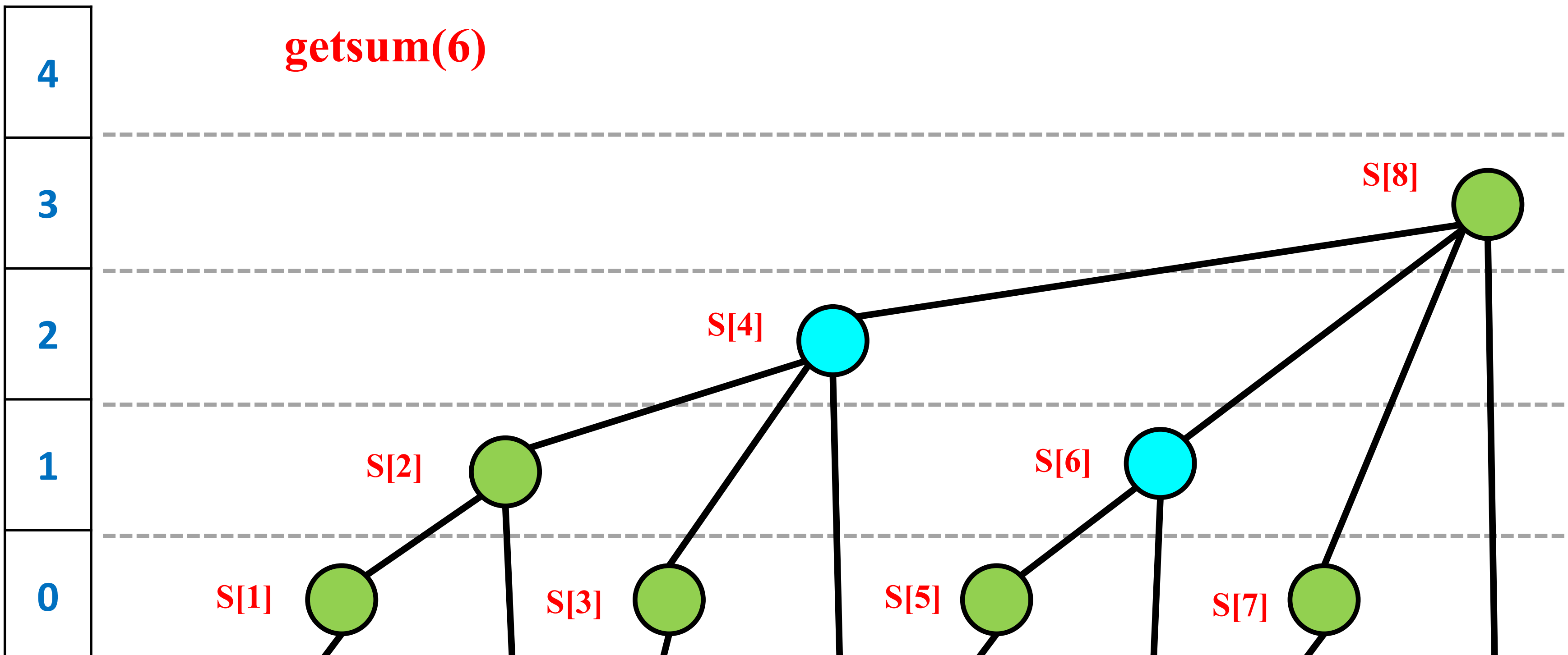
Binary Indexed Tree

getsum(7)



V[i]	1	2	3	4	5	6	7	8
bit	0001 (0)	0010 (1)	0011 (0)	0100 (2)	0101 (0)	0110 (1)	0111 (0)	1000 (3)
S[i]	1	3	3	10	5	11	7	36

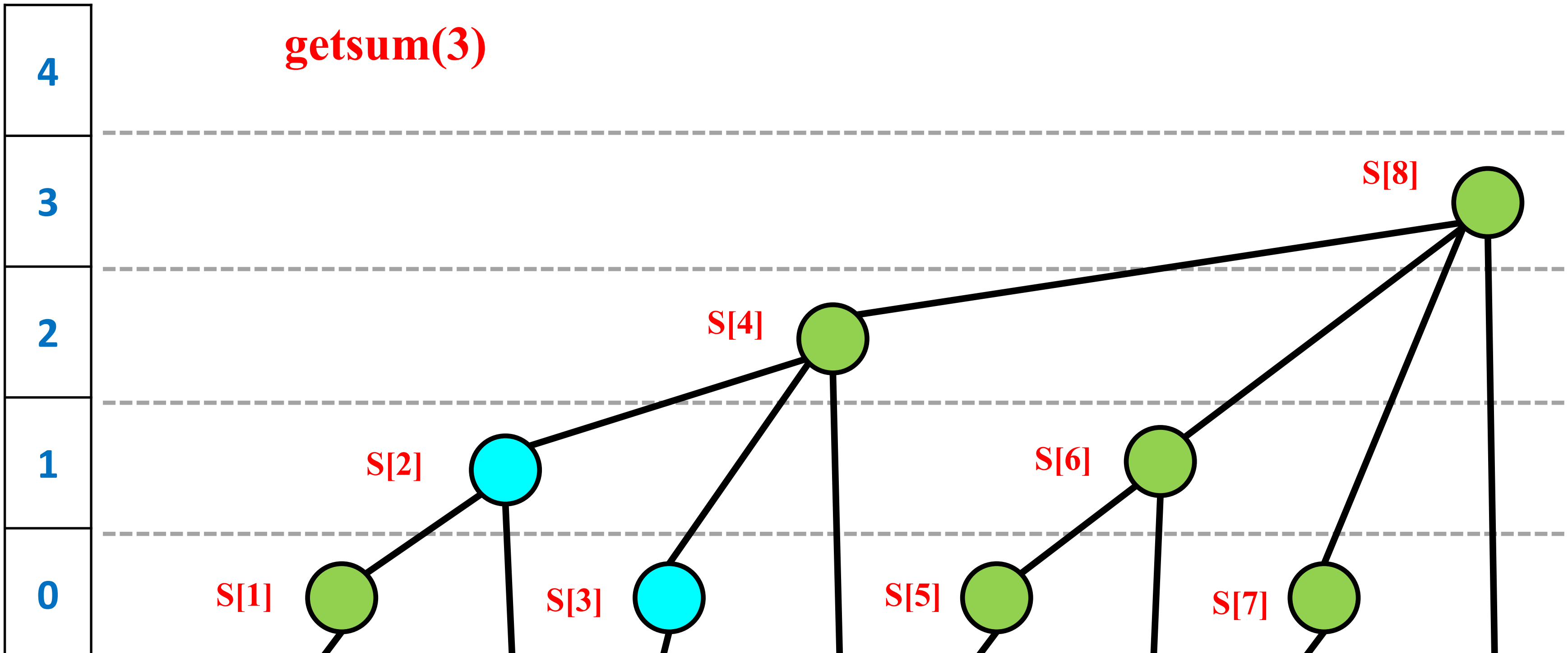
Binary Indexed Tree



V[i]	1	2	3	4	5	6	7	8
bit	0001 (0)	0010 (1)	0011 (0)	0100 (2)	0101 (0)	0110 (1)	0111 (0)	1000 (3)
S[i]	1	3	3	10	5	11	7	36

Binary Indexed Tree

getsum(3)



V[i]	1	2	3	4	5	6	7	8
bit	0001 (0)	0010 (1)	0011 (0)	0100 (2)	0101 (0)	0110 (1)	0111 (0)	1000 (3)
S[i]	1	3	3	10	5	11	7	36

Binary Indexed Tree

Define:

```
int getsum (int end)
{
     int ans = 0;
     while(end>0)
     {
       ans += s[end];
       end -= lowbit(end);
     }
}
```

Binary Indexed Tree

- How to find the summation between interval [i…j] ?
  - call the subroutine **"getsum[j] – getsum[i-1]"**
  - **Two logN operations**
- You can expand this easily to 2D as well

Complexity

- **Binary Indexed Tree**
  - Construction: O(NlogN)
  - Update on one entry: O(logN)
  - Query an interval: O(logN)
  - Space: O(N)
- **Sparse Table**
  - Construction: O(NlogN)
  - Update on one entry: O(N)
  - Query an interval: O(1)
  - Space: O(NlogN)