Uva 1513

Movie Collection

Time: 3 seconds

Problem Descriptions (1/2)

- **Mr. K. I. has a very big movie collection. He has organized his collection in a big stack. Whenever he wants to watch one of the movies, he locates the movie in this stack and removes it carefully, ensuring that the stack doesn't fall over. After he finishes watching the movie, he places it at the top of the stack. Since the stack of movies is so big, he needs to keep track of the position of each movie.**

Problem Descriptions (2/2)

- It is sufficient to know for each movie how many movies are placed above it, since, with this information, its position in the stack can be calculated. Each movie is identified by a number printed on the movie box.
- Your task is to implement a program which will keep track of the position of each movie. In particular, each time Mr. K. I. removes a movie box from the stack, your program should print the number of movies that were placed above it before it was removed.

Input (1/2)

- On the first line a positive integer: the number of test cases, at most 100.
- After that per test case:
  - One line with two integers n and m ($1 \leq n, m \leq 100000$): the number of movies in the stack and the number of locate requests.
  - One line with m integers $a_1, \ldots, a_m$ ($1 \leq a_i \leq n$) representing the identification numbers of movies that Mr. K. I. wants to watch.

Input (2/2)

- For simplicity, assume the initial stack contains the movies with identification numbers 1, 2, . . . , n in increasing order, where the movie box with label 1 is the top-most box.

Output

- **Per test case:**
  - **One line with m integers, where the i-th integer gives the number of movie boxes above the box with label a_i , immediately before this box is removed from the stack.**
  - **Note that after each locate request a_i , the movie box with label a_i is placed at the top of the stack.**

I/O Example

Input

number of test cases

```
2
```

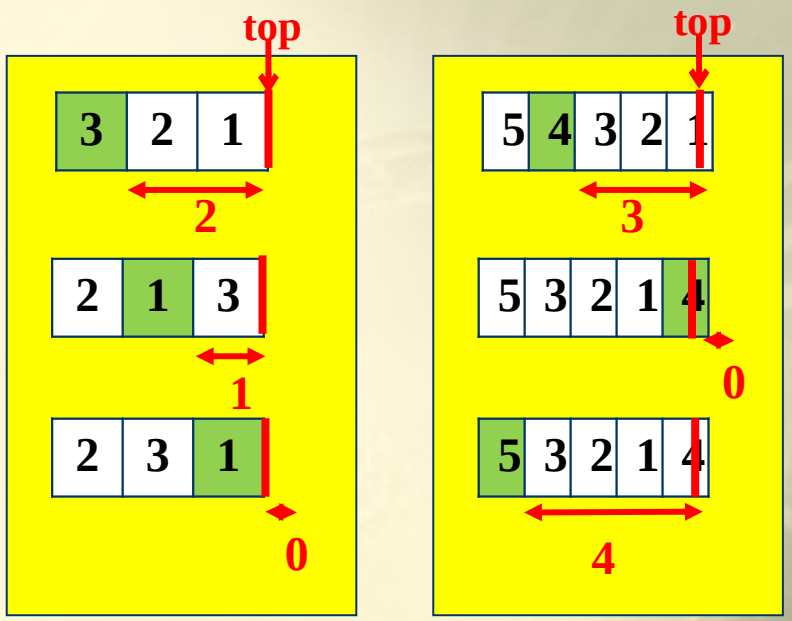
Number of locate requests

number of movies

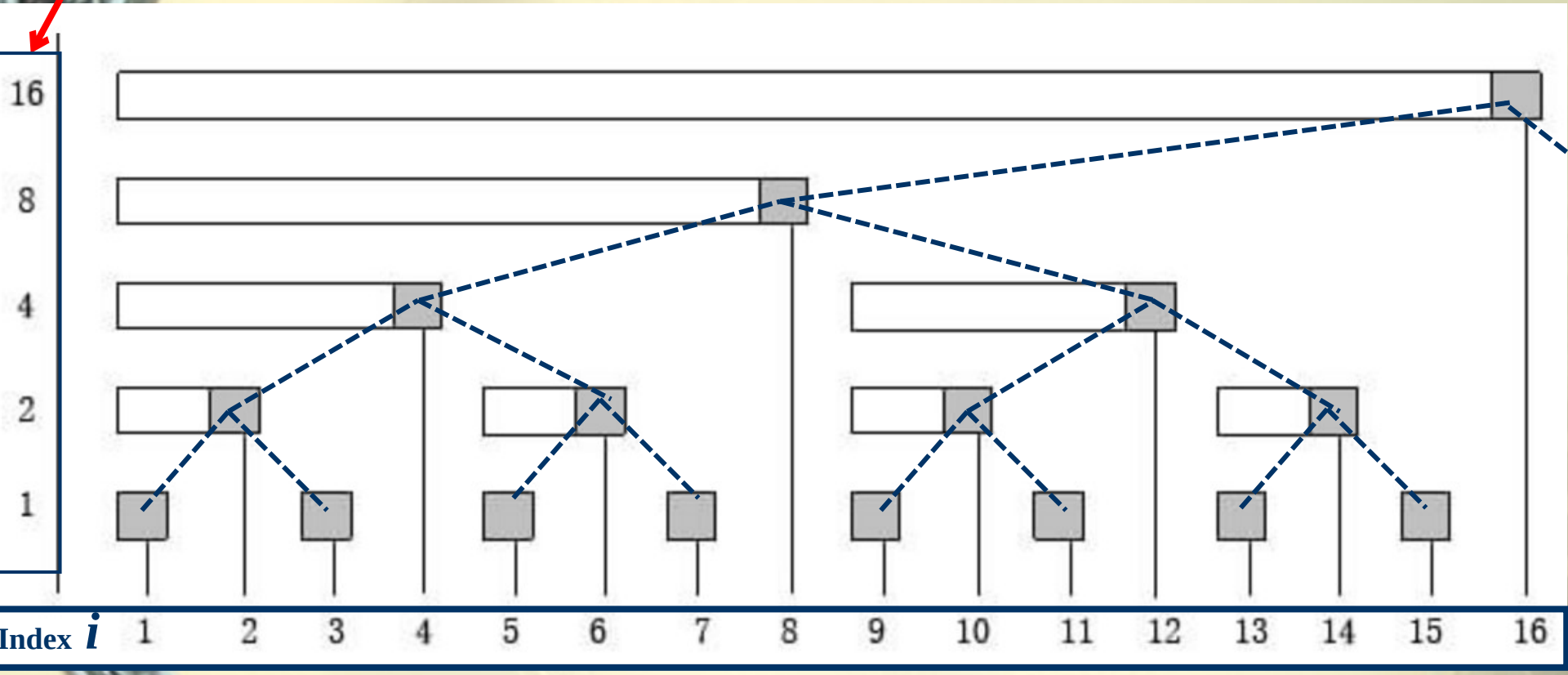
```
3 3
3 1 1
5 3
4 4 5
```

Output

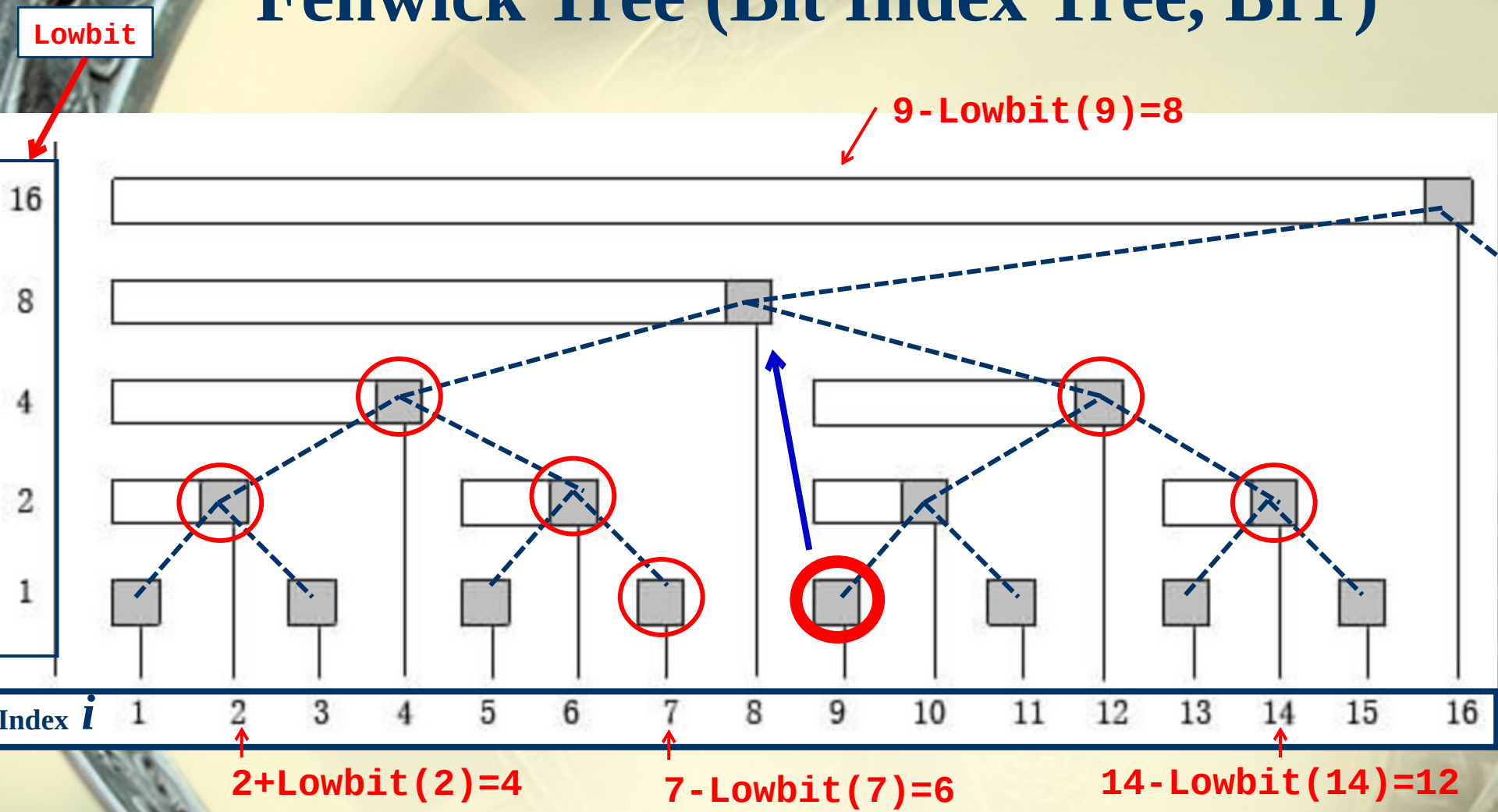
```
2 1 0
3 0 4
```



Fenwick Tree (Bit Index Tree, BIT)



Fenwick Tree (Bit Index Tree, BIT)



Parent of index $i_{(right)} \rightarrow i-Lowbit(i)$

Parent of index $i_{(left\ )} \rightarrow i+Lowbit(i)$

BIT, add( ), sum( ), rsum( )

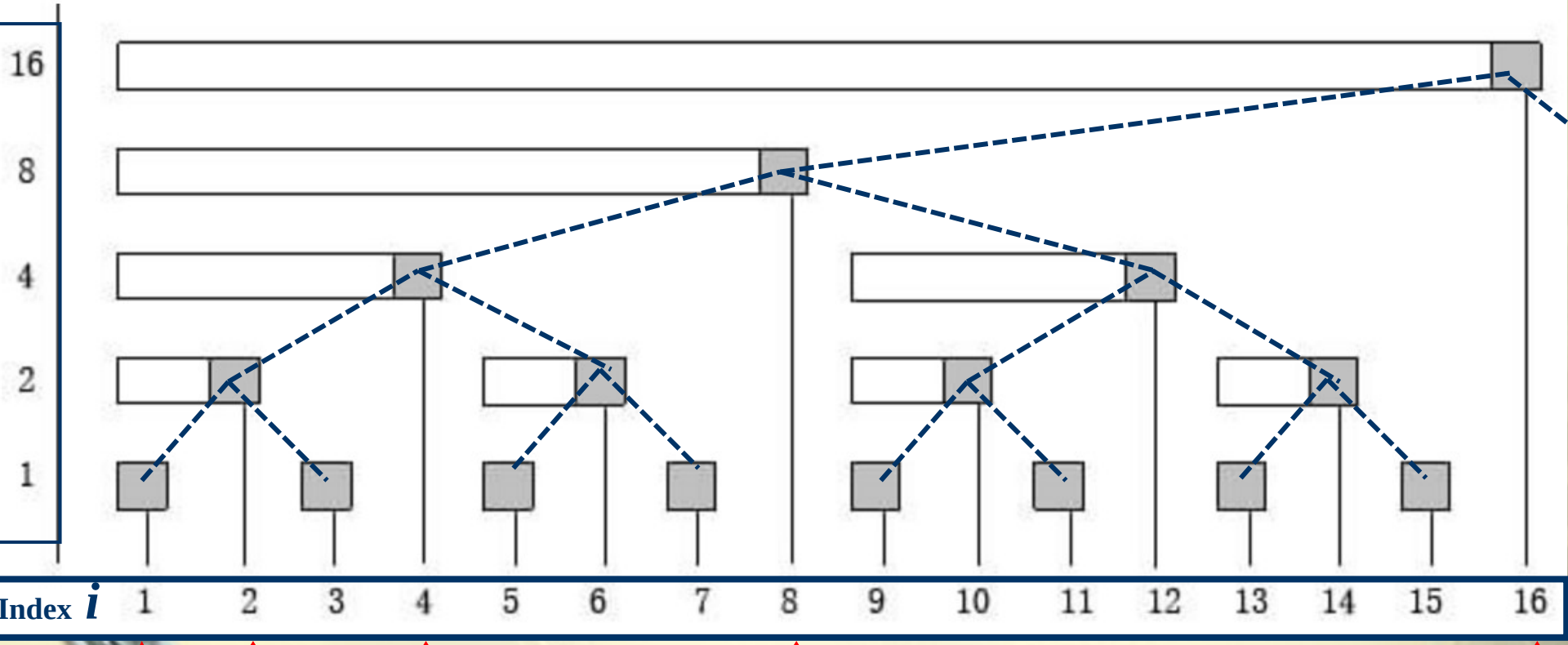
```
void add (int x, int d)
{   while (x<=n)
    { c[x]+=d; x+=lowbit(x);}
}

int sum(int x)
{   int ret=0;
    while (x>0)
    { ret+=c[x]; x-=lowbit(x);}
}

int rsum(int x, int y)
{   return( sum(y)-sum(x-1) );
}
```

BIT, add(1, d)

k+=lowbit(k)



k=0001

k=0010

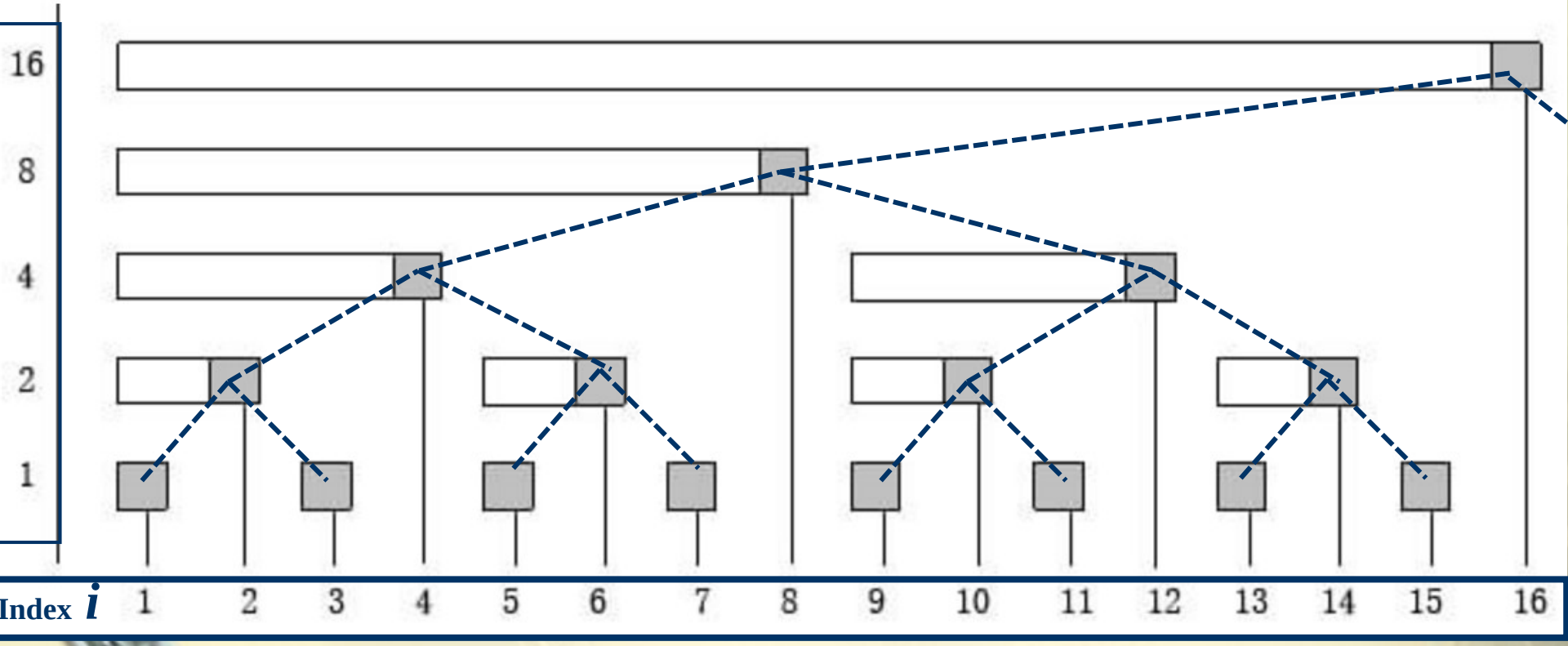
k=0100

k=1000

k=10000

BIT, add(2, d)

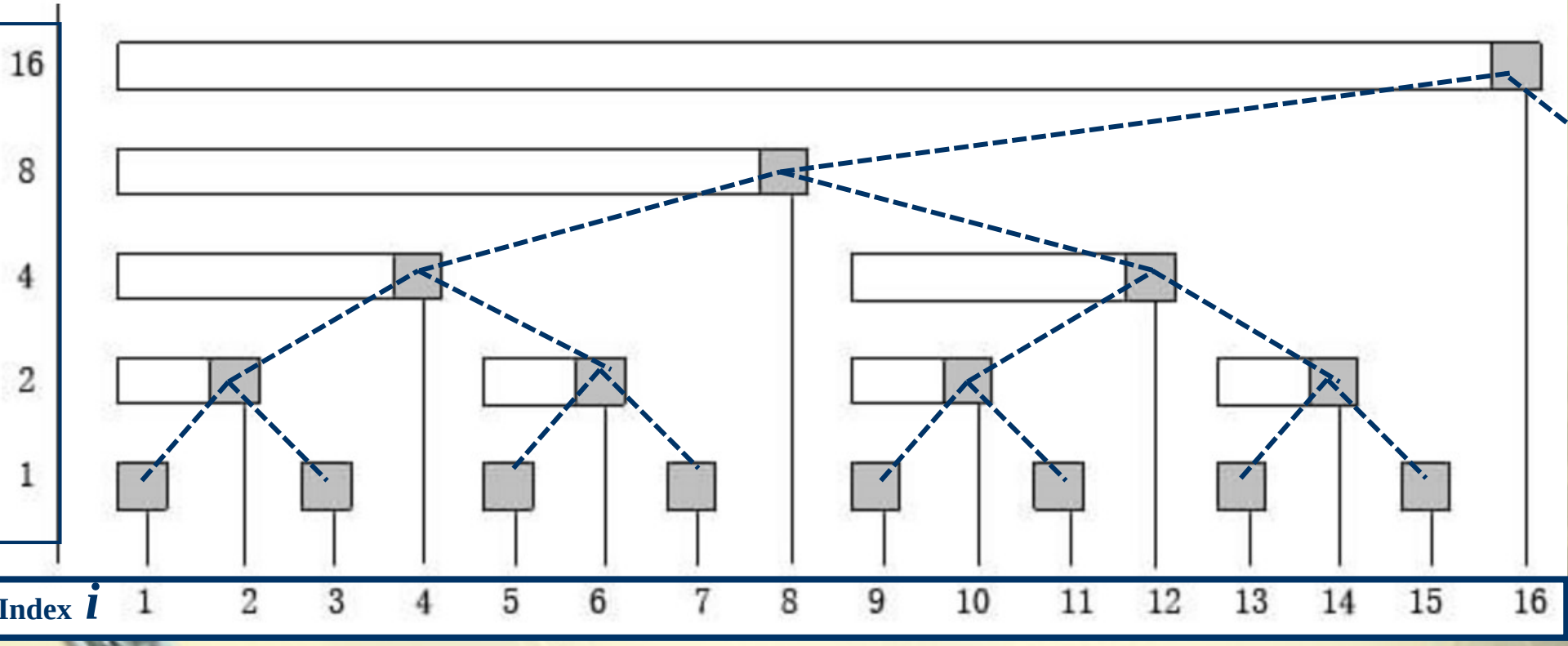
k+=lowbit(k)



k=0010 k=0100 k=1000 k=10000

BIT, add(3, d)

k+=lowbit(k)



k=0011

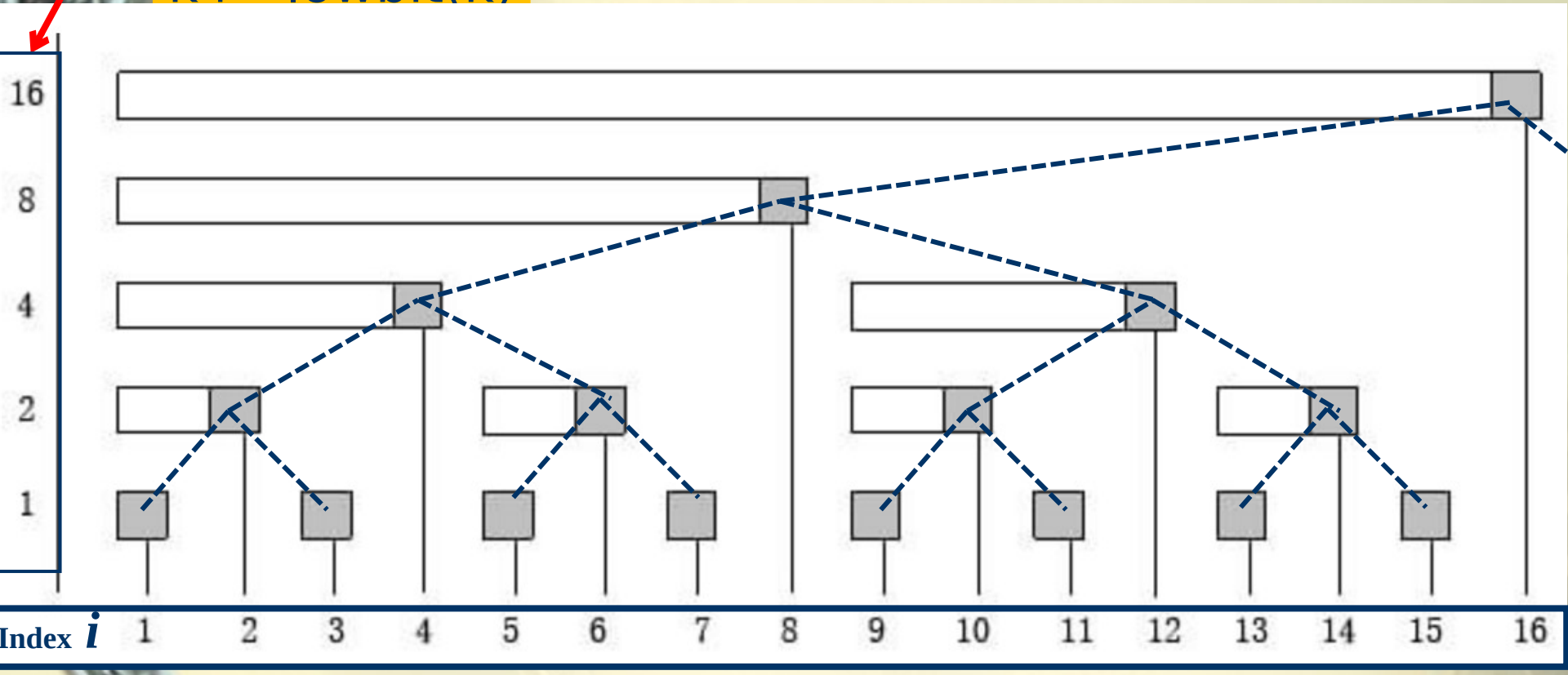
k=0100

k=1000

k=10000

BIT, add(4, d)

k+=lowbit(k)



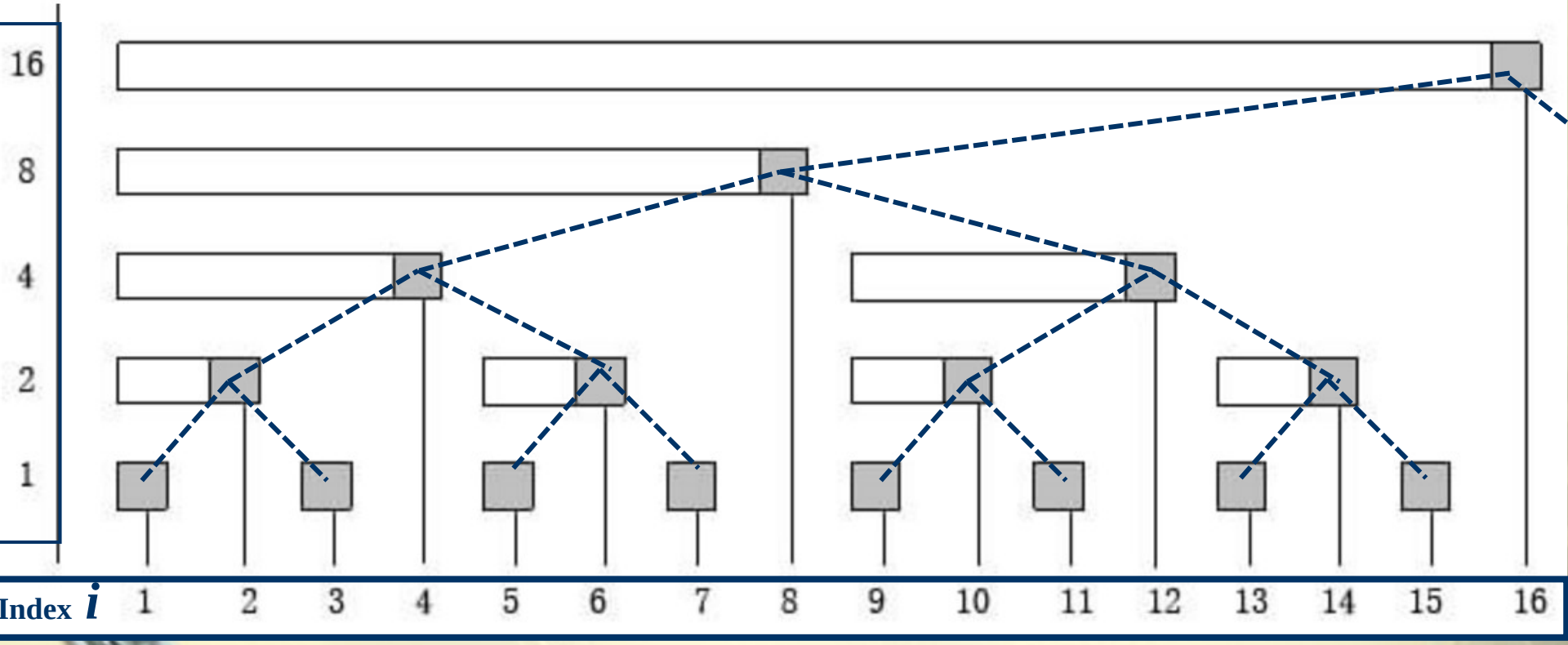
k=0100

k=1000

k=10000

BIT, add(5, d)

k+=lowbit(k)



k=0101

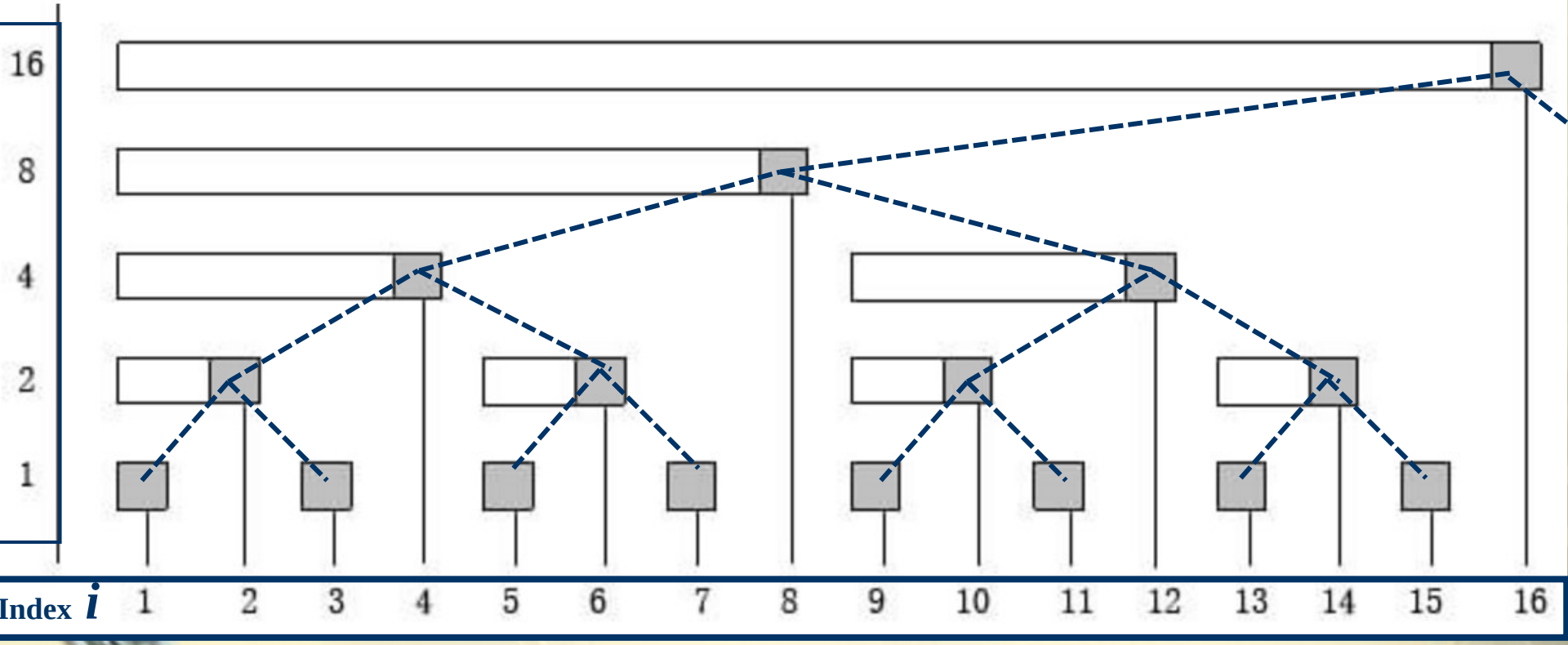
k=0110

k=1000

k=10000

BIT, add(6, d)

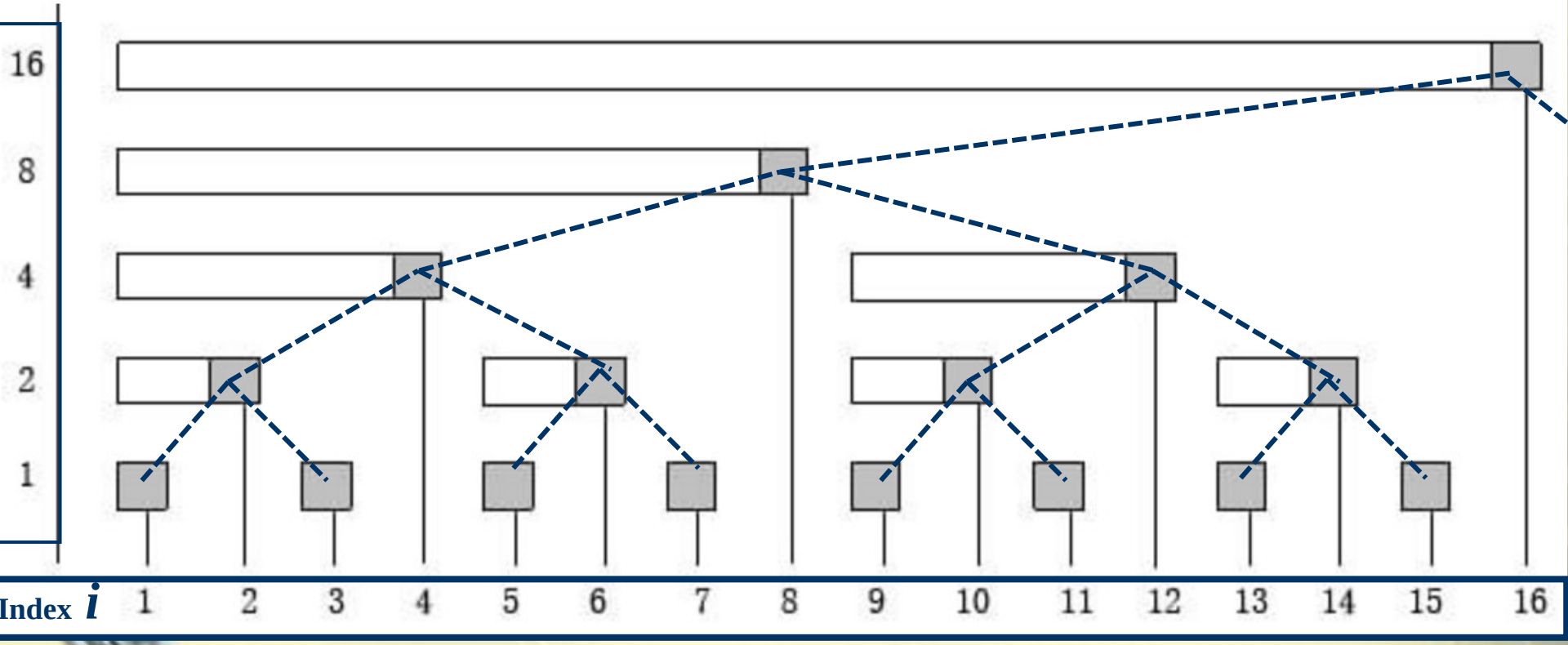
k+=lowbit(k)



k=0110 k=1000 k=10000

BIT, add(7, d)

k+=lowbit(k)



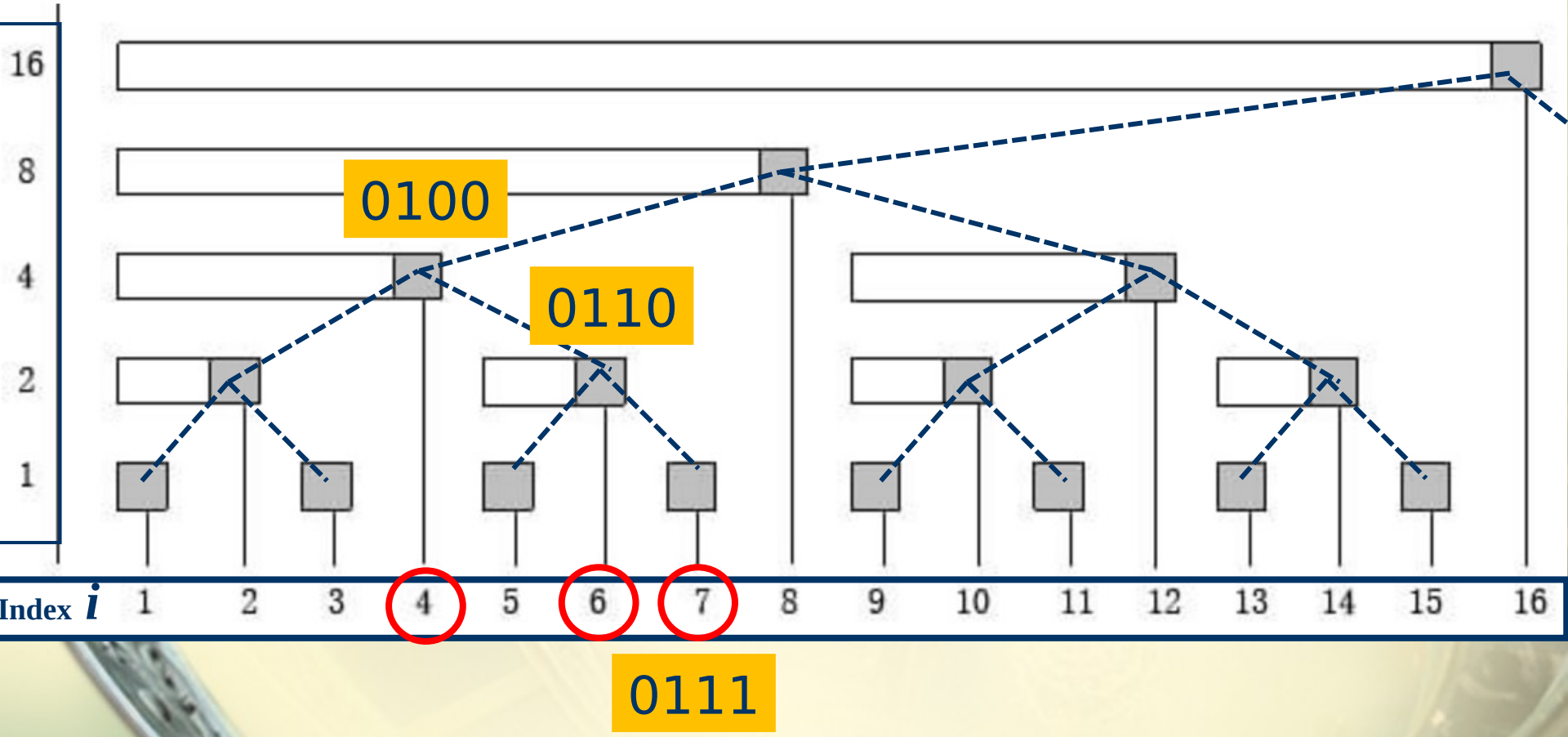
k=0111

k=1000

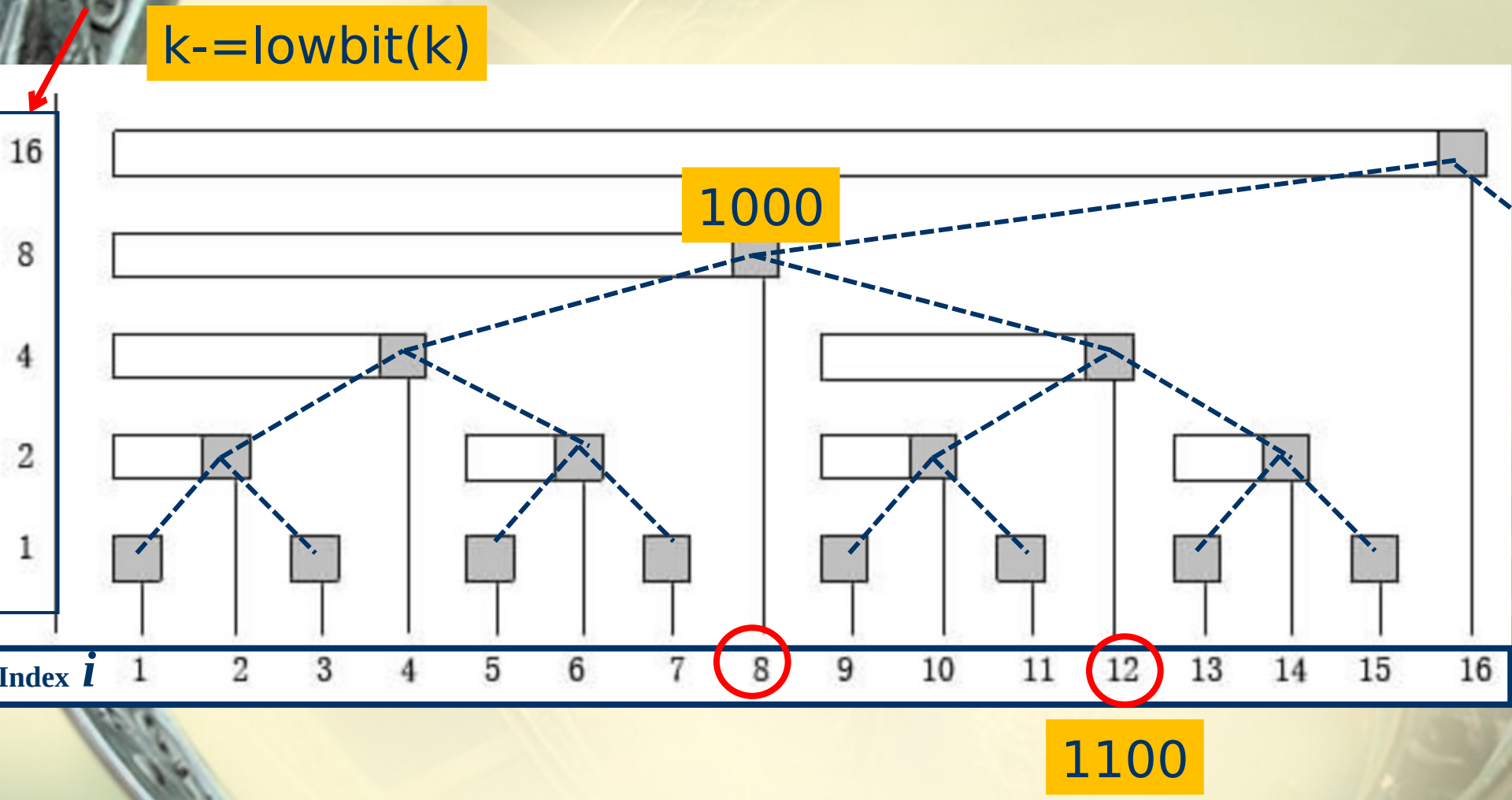
k=10000

BIT, sum (7)

k-=lowbit(k)

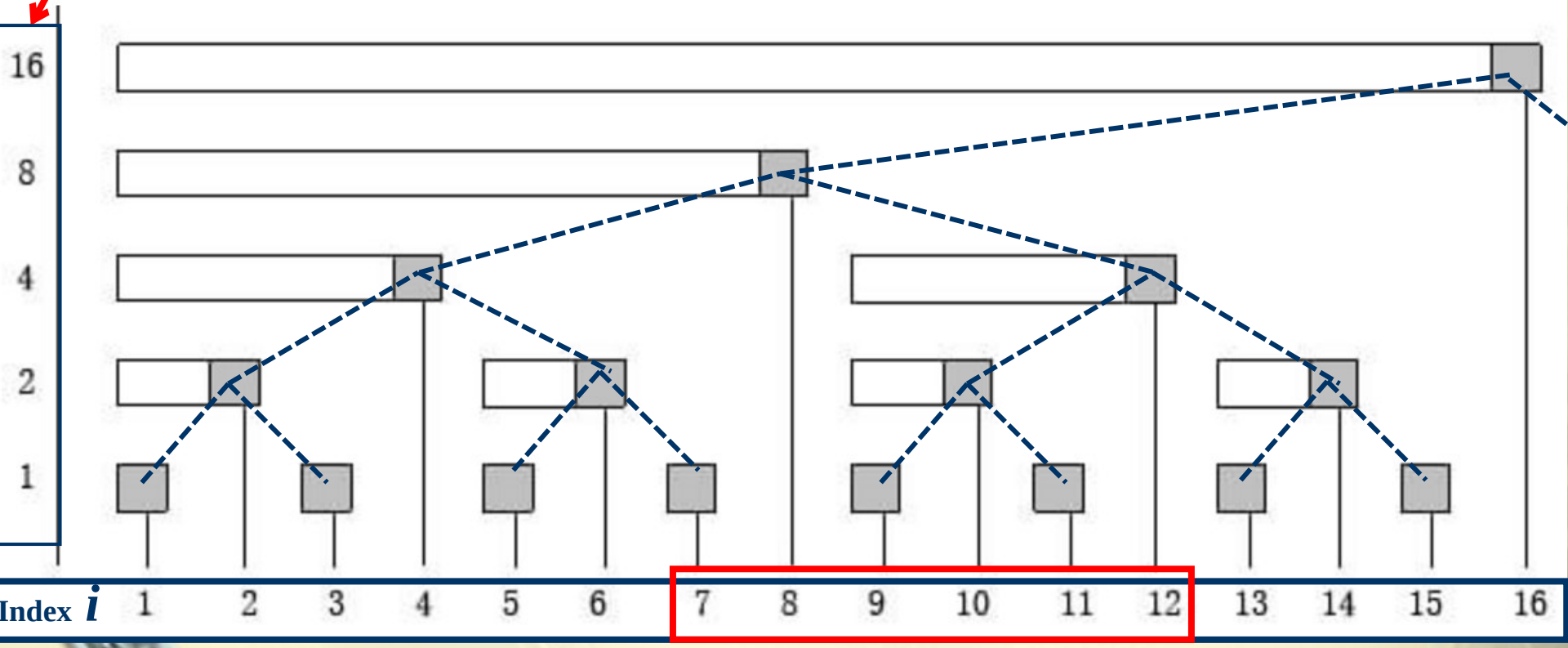


BIT, sum (12)


k-=lowbit(k)
16
8
4
2
1
1000
Index i
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
1100

BIT, rsum (7,12)

rsum(7,12) = sum(12)-sum(6)



rsum(7,12) = sum(12)-sum(6)

Bit Index Tree (BIT)

- Given an array with n integers, and n queries and n modifications?

- Time Complexity:
  - Build Bit Index Tree: O(nlogn)
  - O(logn) for each query.
  - Therefore, O(nlogn) for n queries.
  - O(logn) for each modification.
  - Therefore, O(nlogn) for n modifications.
  - Overall: O(nlogn)

Solution

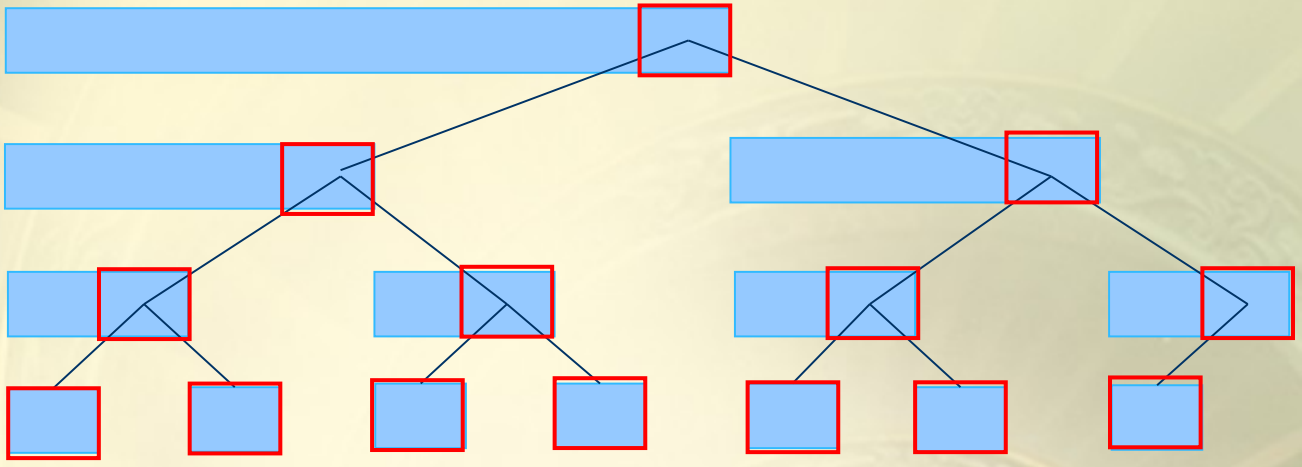
1	2	3	4	5	6	7	pos
7	6	5	4	3	2	1	

top

7	1
6	2
5	3
4	4
3	5
2	6
1	7

pos

8	5
7	1
6	2
5	3
4	4
3	6
2	6
1	7



7 6

5 2 1 5 6 3

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	1	4	1	2	1	7	0	0	0	0	0	0

4

7	6	5	4	3	2	1
1	1	1	1	1	1	1

Solution

1	2	3	4	5	6	7	pos
7	6	5	4	3	2	1	

top

7	1
6	2
5	3
4	4
3	5
2	6
1	7

pos

8	5
7	1
6	2
5	3
4	4
3	5
2	6
1	7

7 6

5 2 1 5 6 3

4

bit

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	1	4	1	2	1	7	0	0	0	0	0	0

7-bit(3)=7-(bit(3)+bit(2))=7-(1+2)=4

add(3,-1)

bit

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	0	3	1	2	1	6	0	0	0	0	0	0

1	2	3	4	5	6	7	pos
7	6	5	4	8	2	1	

add(8,1)

bit

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	0	3	1	2	1	7	0	0	0	0	0	0

Solution

top

8	5
7	1
6	2
5	3
4	4
3	5
2	6
1	7

pos

9	2
8	5
7	1
6	2
5	3
4	4
3	5
2	6
1	7

7 6

5 2 1 5 6 3

4 5

1	2	3	4	5	6	7	pos
7	6	5	4	8	2	1	

bit

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	0	3	1	2	1	7	0	0	0	0	0	0

7-bit(6)=7-(bit(6)+bit(4))=7-(2+3)=2

bit

add(6,-1)

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	0	3	1	1	1	6	0	0	0	0	0	0

1	2	3	4	5	6	7	pos
7	9	5	4	8	2	1	

add(9,1)

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	0	3	1	1	1	6	1	1	0	1	0	0

Sample code

```cpp
#include <cstdio>
#include <cstring>

#define lowbit(x) ((x)&(-x))

const int maxn = 1e5 + 5;

int bit[2 * maxn], pos[maxn];
int n, m;

void add (int p, int v)
{
    while (p < 2 * maxn)
    {
        bit[p]+= v;
        p += lowbit(p);
    }
}

int sum (int p)
{
    int ret = 0;
    while (p > 0)
    {
        ret += bit[p];
        p -= lowbit(p);
    }
    return ret;
}
```

Sample code

```
31  int main ()
32  {
33      int T;
34      scanf ("%d", &T);
35
36      while (T--)
37      {
38          memset (bit, 0, sizeof (bit));
39          scanf ("%d%d", &n, &m);
40
41          for (int i = 1; i <= n; i++)
42          {
43              pos[i] = n - i + 1;
44              add(pos[i], 1);
45          }
46
47          int req_num;
48          int N = n;
49
50          for (int i = 1; i <= m; i++)
51          {
52              scanf ("%d",&req_num);
53              printf ("%d%c", n - sum(pos[req_num]), i == m ? '\n':' ');
54              add(pos[req_num], -1);
55              pos[req_num] = ++N;
56              add(pos[req_num], 1);
57          }
58      }
59      return 0;
60  }
```

build bit index tree

handle request