Homework 1
Divide and Conquer

Multiplication

- Long multiplication takes $\Theta(n^2)$
  - $20\times20000^2\approx8$ billion
  - Need ≥4 billion operations per second
  - It is impossible to use this method to solve the large input.
- Karatsuba multiplication $\Theta(n^{1.59})$
  - $20\times20000^{1.59}\approx138$ million
  - Relatively hard to implement and the constant is very large.

≈1.1 seconds on Macbook Air 2013

```java
import java.lang.*;
import java.io.*;
import java.math.*;
import java.util.*;

public class SolutionAlt{
    public static void main(String args[]) throws Exception
    {
        int nCases = 0;
        Scanner sc = new Scanner(System.in);
        nCases = sc.nextInt();
        while(nCases-->0){
            BigInteger x, y, product;
            int head, tail;
            x=new BigInteger(sc.next());
            y=new BigInteger(sc.next());
            head=sc.nextInt();
            tail=sc.nextInt();
            product=x.multiply(y);
            System.out.println("DONE");
            //System.out.println(product.toString().substring(head-1,tail));
        }
        return;
    }
}
```

≈5.6 seconds on Macbook Air 2013

```java
import java.lang.*;
import java.io.*;
import java.math.*;
import java.util.*;

public class SolutionAlt{
   public static void main(String args[]) throws Exception
   {
      int nCases = 0;
      Scanner sc = new Scanner(System.in);
      nCases = sc.nextInt();
      while(nCases-->0){
         BigInteger x, y, product;
         int head, tail;
         x=new BigInteger(sc.next());
         y=new BigInteger(sc.next());
         head=sc.nextInt();
         tail=sc.nextInt();
         product=x.multiply(y);
         //System.out.println("DONE");
         System.out.println(product.toString().substring(head-1,tail));
      }
      return;
   }
}
```

BigDecimal.toString() is slow!

Surprise!

- Macbook Air 2013: Intel Core i5 1.3Ghz
  - Turbo Boost: 2.6Ghz
- Note: JAVA use the long multiplication method!
- Unreasonable?!
- Actually, n is not as large as 20000!
  - Base-10 versus base-2^k
  - Use base-2^{32}: $n \approx 2077$ and $n^2 \approx 4.3$M

Surprise?!

- BigInteger.toString() is much slower than BigInteger.multiply().
  - Why???
- Base-2^k to base-2 can be fast.
  - Hint: BigInteger.testBit()
- Inspiration: Base-10^9 to base-10 is fast
- $n \le 2223$
- $20 \times 2223^2 \approx 100$M ... 80 times faster!

$\approx$0.6 seconds on Macbook Air 2013

Surprise!!!

CPython: ≈0.85 seconds on Macbook Air 2013

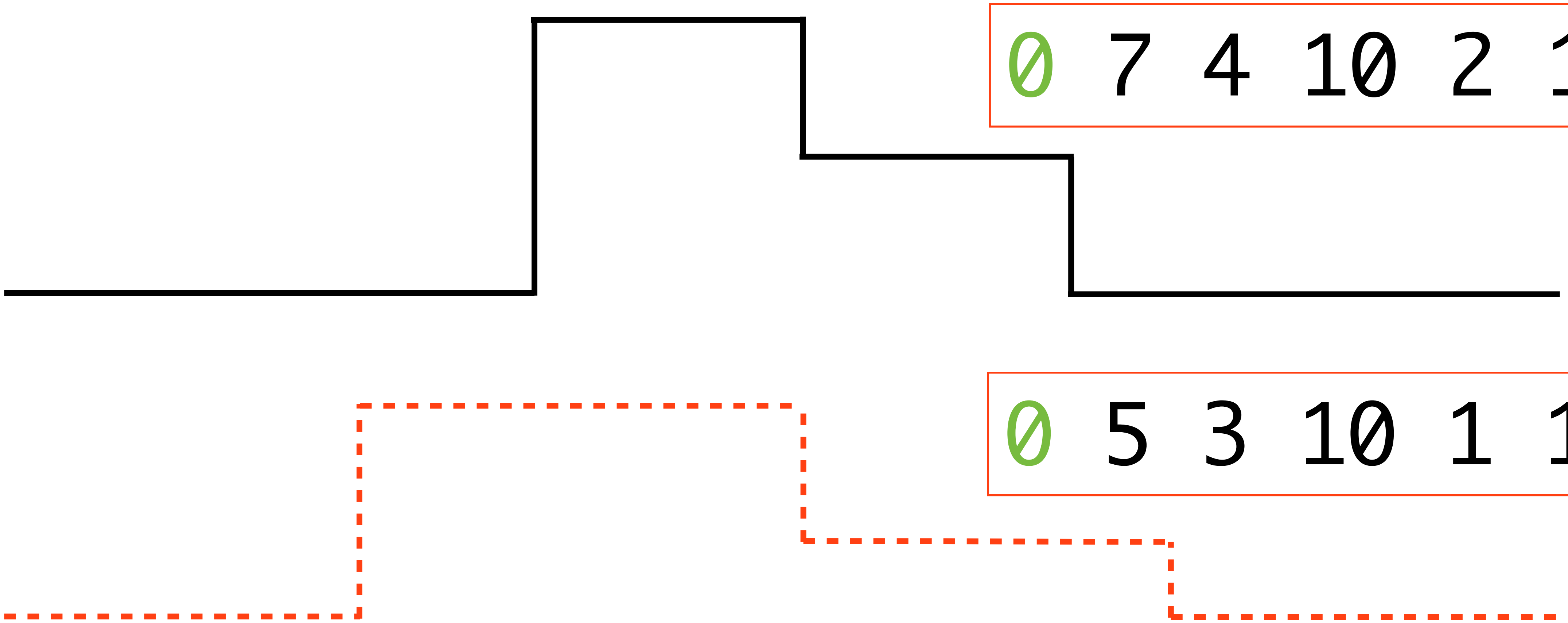
```
import sys

cas = input()
for i in range(cas):
    instr = raw_input()
    strlist = instr.split(' ')
    a = int(strlist[0])
    b = int(strlist[1])
    start = int(strlist[2])-1
    end = int(strlist[3])
    ans = str(a*b)
    print ans[start:end]
```

Modified 0110737_hw1–1_v2.py

Skyline Problem

- Merge two skylines into one



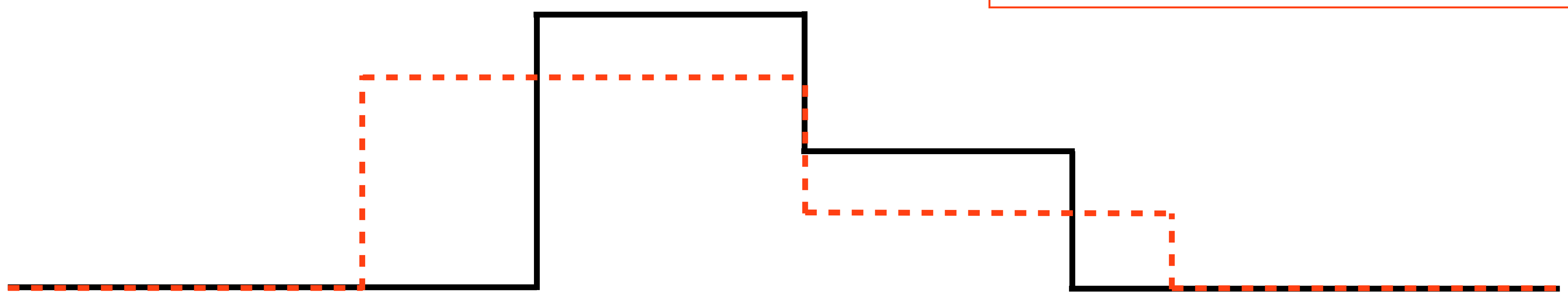
```
0 7 4 10 2 13 0
```

```
0 5 3 10 1 14 0
```

Skyline Problem

- Merge two skylines into one

0	7	4	10	2	13	0
0	5	3	10	1	14	0

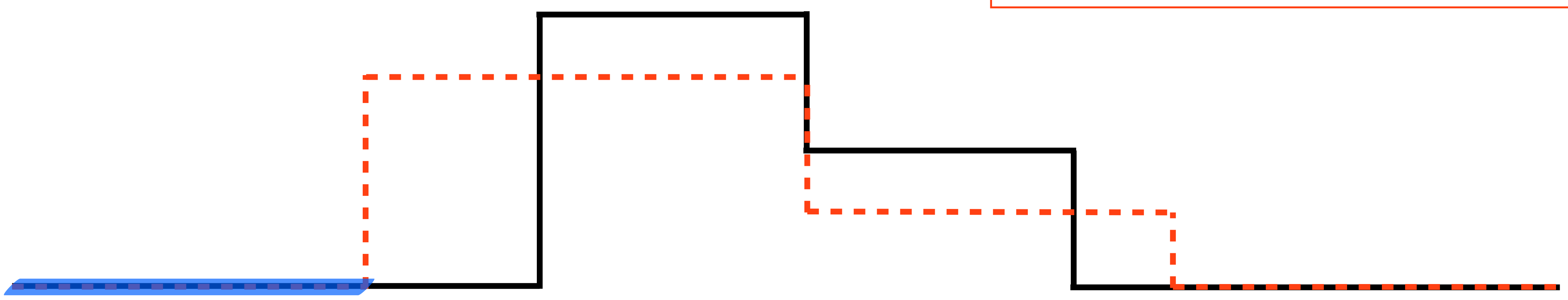


Skyline Problem

- Merge two skylines into one

0 7 4 10 2 13 0

0 5 3 10 1 14 0

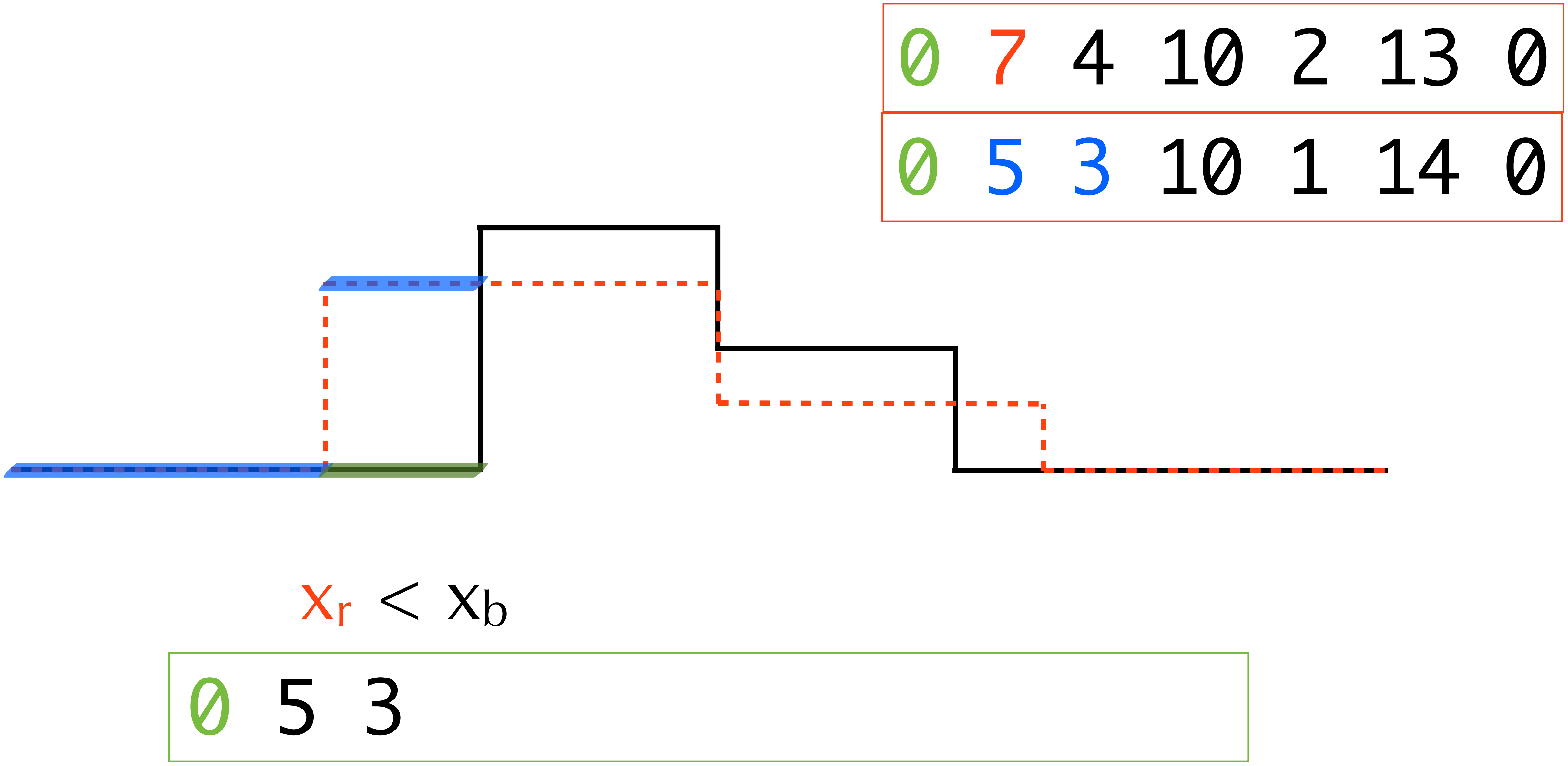


$x_r < x_b$

0 5

Skyline Problem

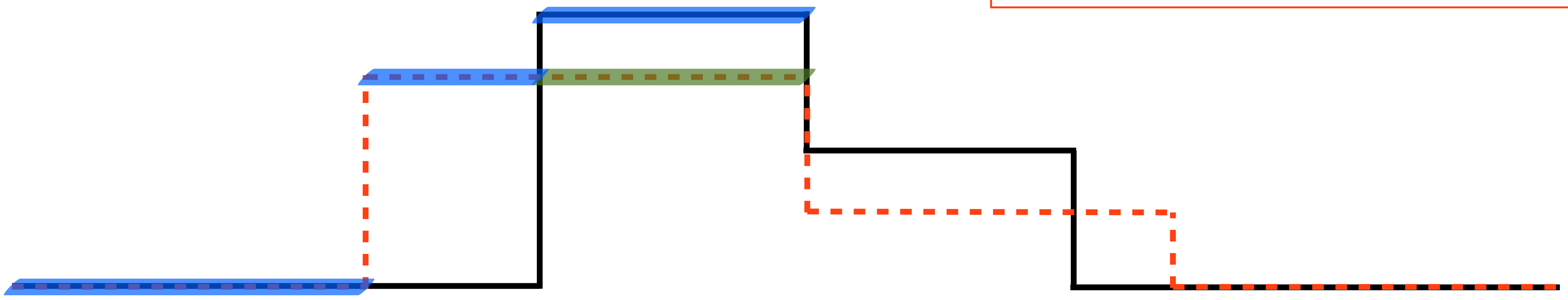
- Merge two skylines into one



Skyline Problem

- Merge two skylines into one

0	7	4	10	2	13	0
0	5	3	10	1	14	0



$x_b < x_r$

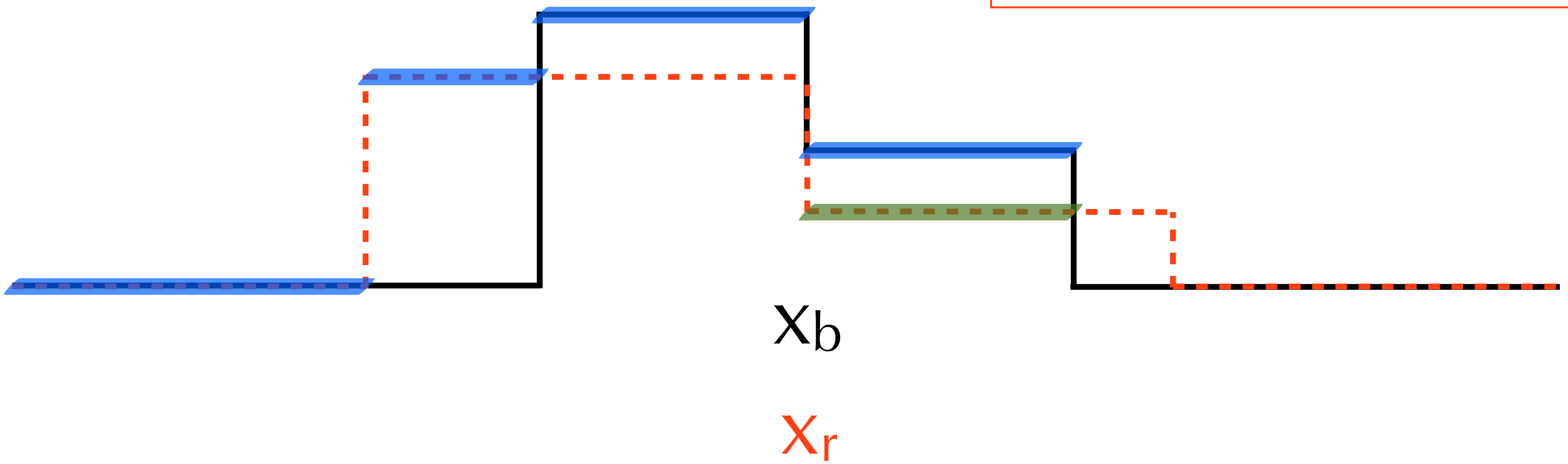
```
0 5 3 7 4
```

Skyline Problem

- Merge two skylines into one

0 7 4 10 2 13 0

0 5 3 10 1 14 0



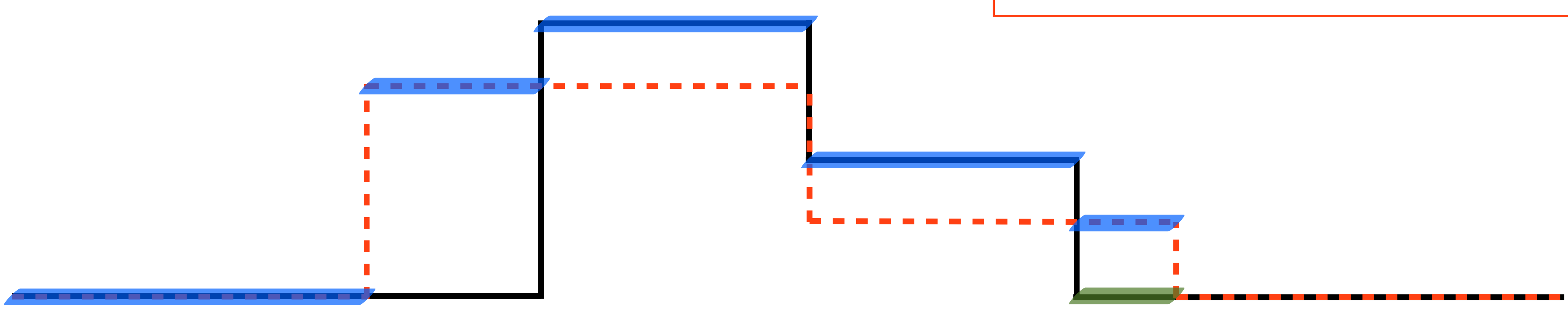
0 5 3 7 4 10 2

Skyline Problem

- Merge two skylines into one

0 7 4 10 2 13 0

0 5 3 10 1 14 0



$x_b < x_r$

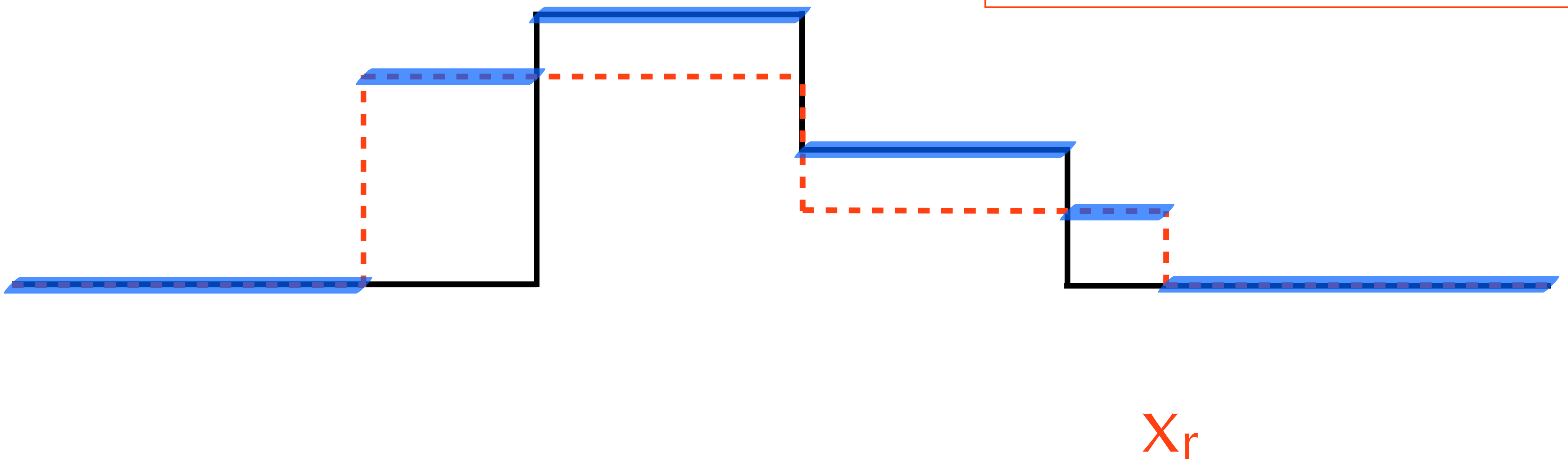
0 5 3 7 4 10 2 13 1

Skyline Problem

- Merge two skylines into one

0 7 4 10 2 13 0

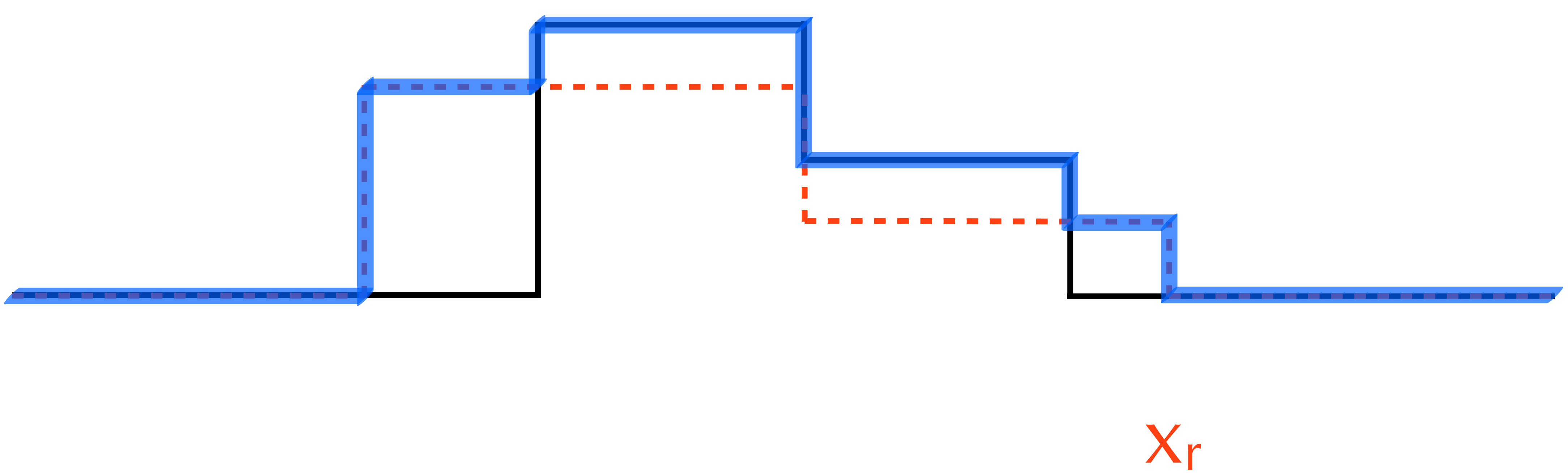
0 5 3 10 1 14 0



0 5 3 7 4 10 2 13 1 14 0

Skyline Problem

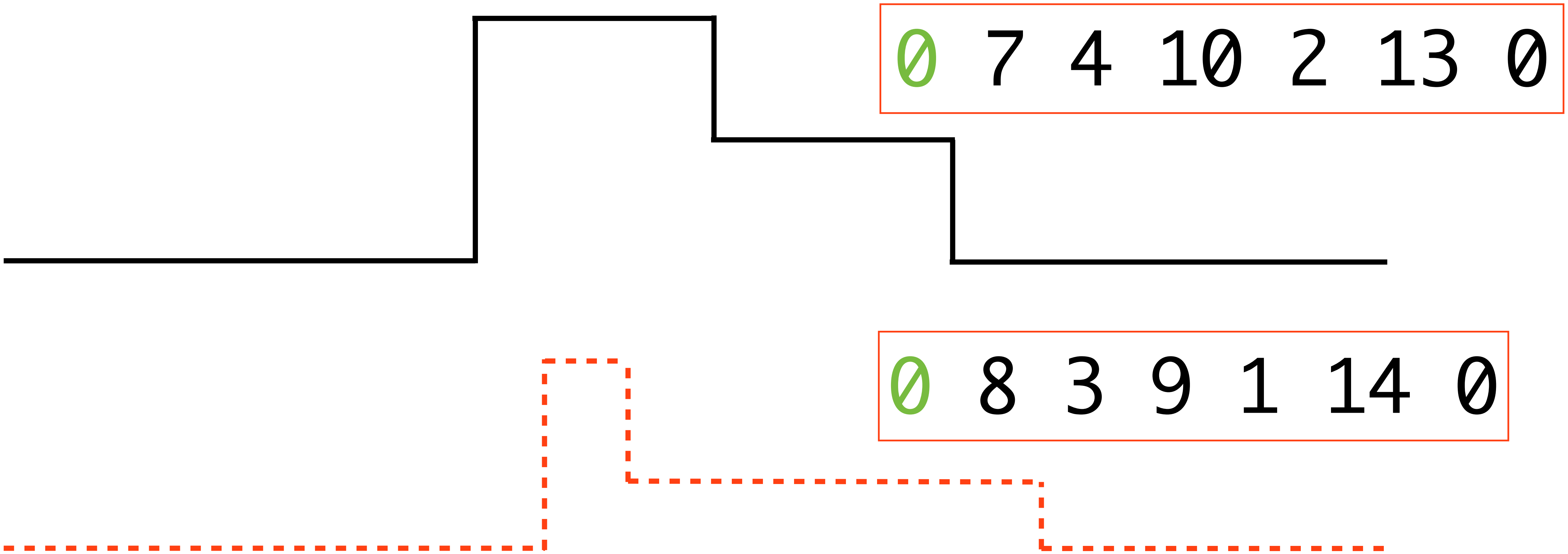
- Merge two skylines into one




```
0 5 3 7 4 10 2 13 1 14 0
```

Skyline Problem

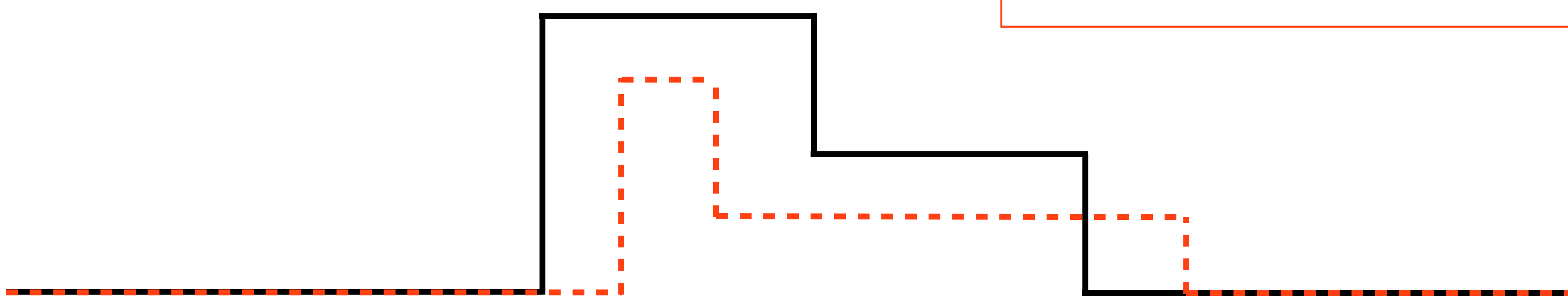
- Merge two skylines into one



Skyline Problem

- Merge two skylines into one

0	7	4	10	2	13	0
0	8	3	9	1	14	0

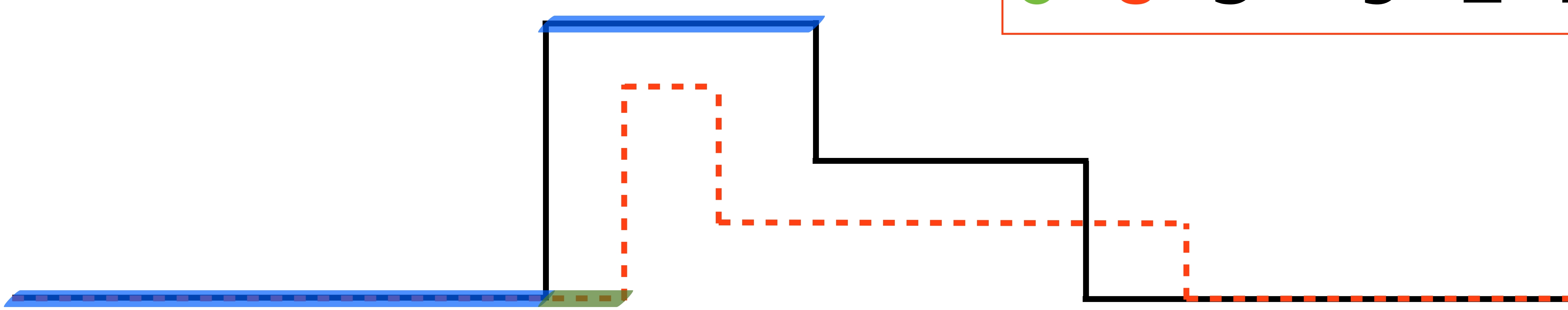


Skyline Problem

- Merge two skylines into one

0 7 4 10 2 13 0

0 8 3 9 1 14 0



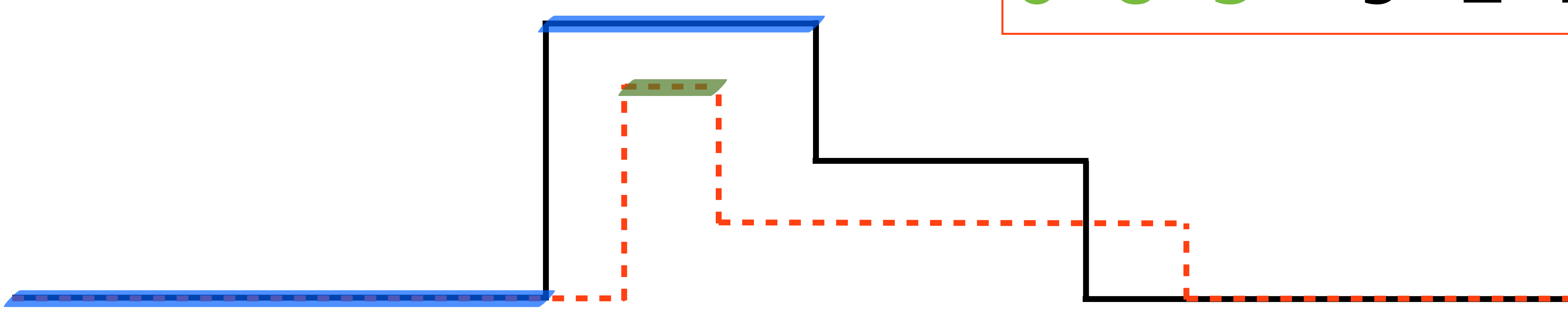
0 7 4

Skyline Problem

- Merge two skylines into one

0 7 4 10 2 13 0

0 8 3 9 1 14 0

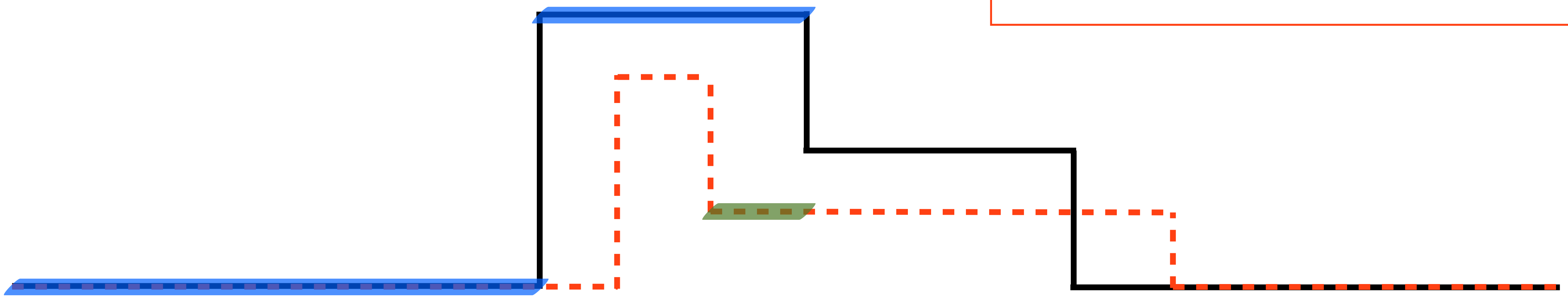


0 7 4

Skyline Problem

- Merge two skylines into one

0	7	4	10	2	13	0
0	8	3	9	1	14	0



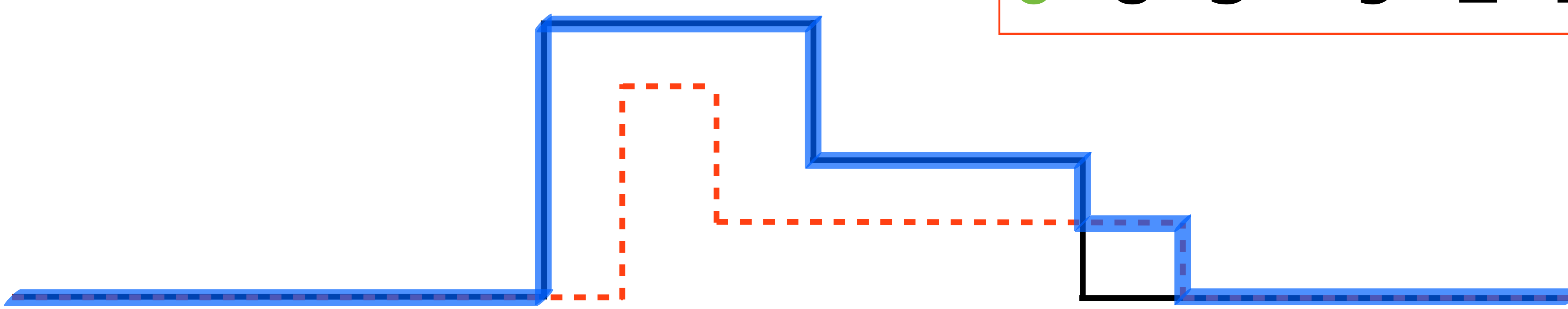
0 7 4

Skyline Problem

- Merge two skylines into one

0 7 4 10 2 13 0

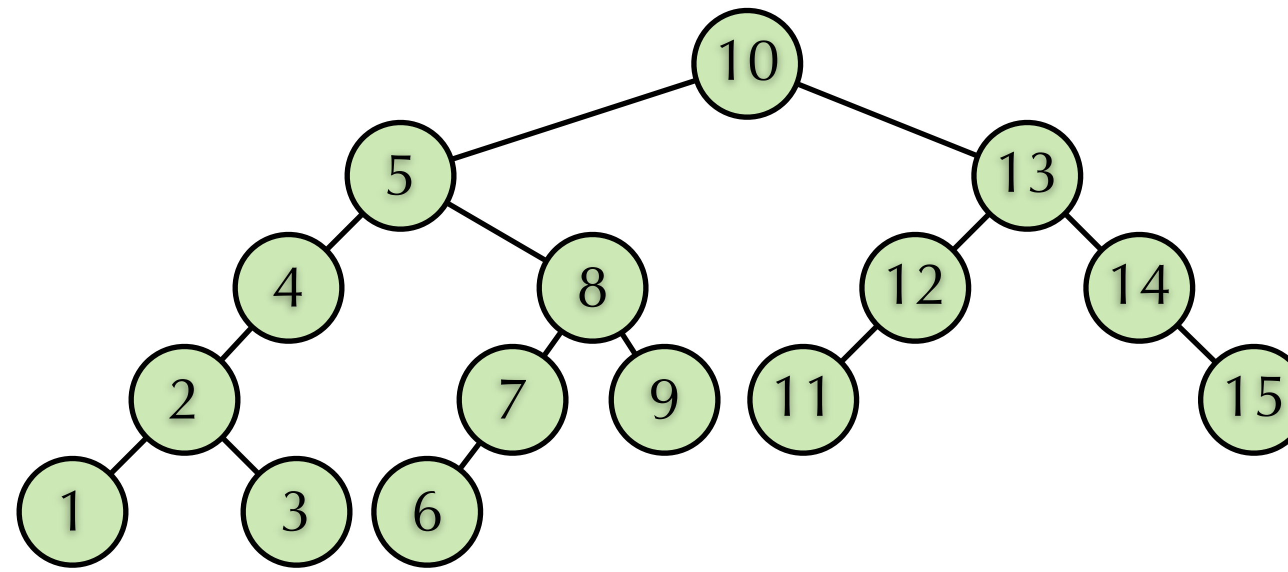
0 8 3 9 1 14 0



0 7 4 10 2 13 1 14 0

Height

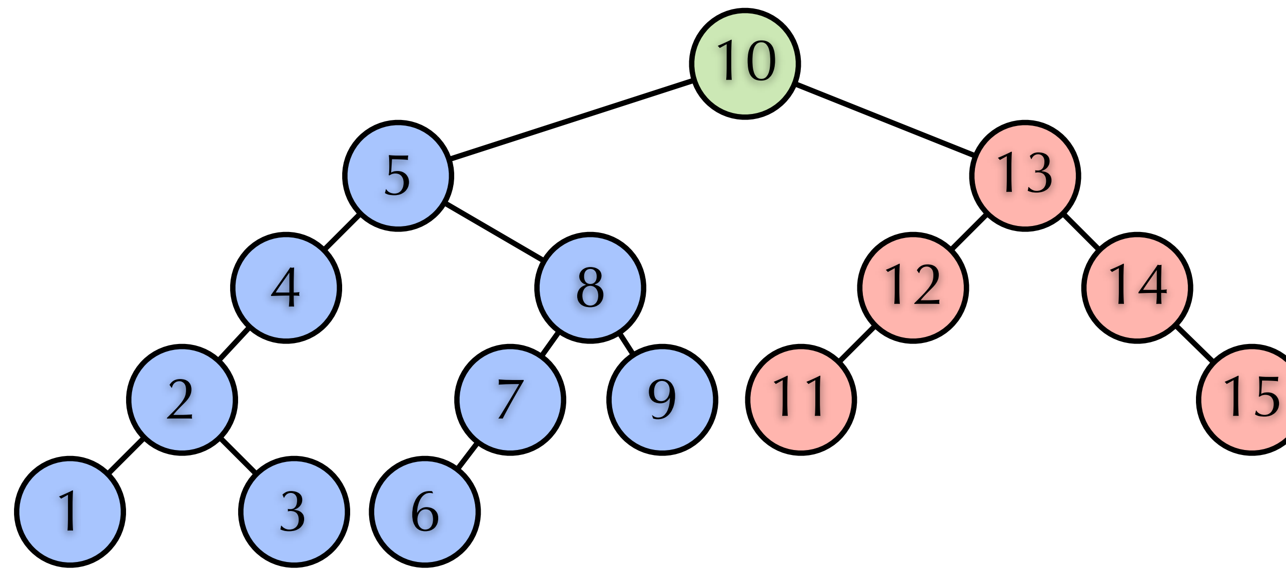
- $h_T = \max(h_R, h_L) + 1$
- Preorder traversal
- BST



10	5	4	2	1	3	8	7	6	9	13	12	11	14	15

Partition: Binary Search

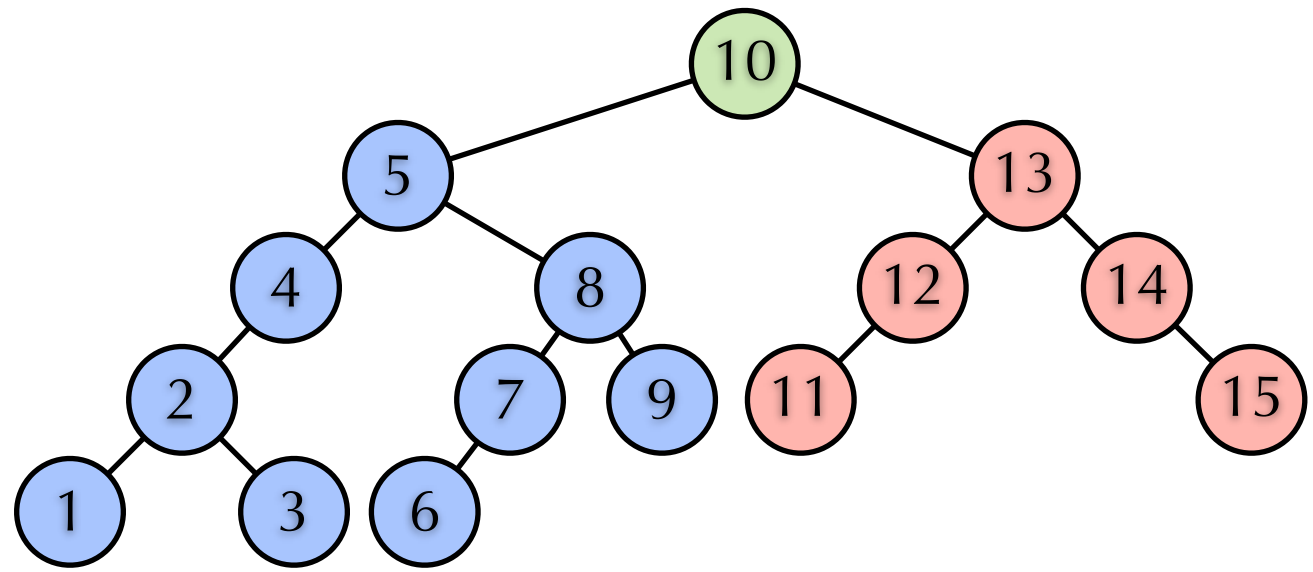
- $h_T = \max(h_R, h_L) + 1$
- Preorder traversal
- BST



10	5	4	2	1	3	8	7	6	9	13	12	11	14	15

Partition: Binary Search

- $h_T = \max(h_R, h_L) + 1$
- Preorder traversal
- BST

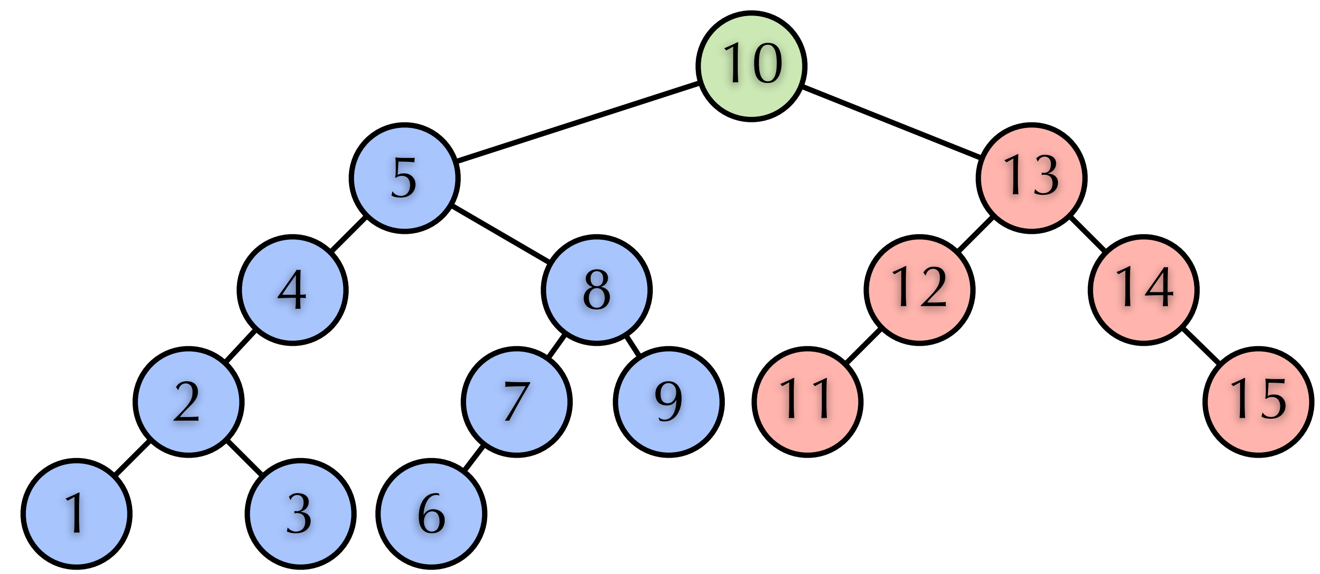


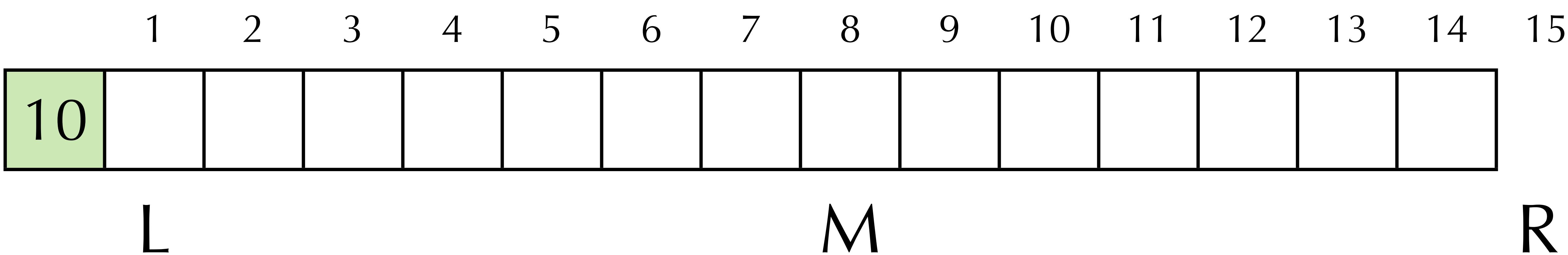
10	5	4	2	1	3	8	7	6	9	13	12	11	14	15

10	<10	>10

Partition: Binary Search

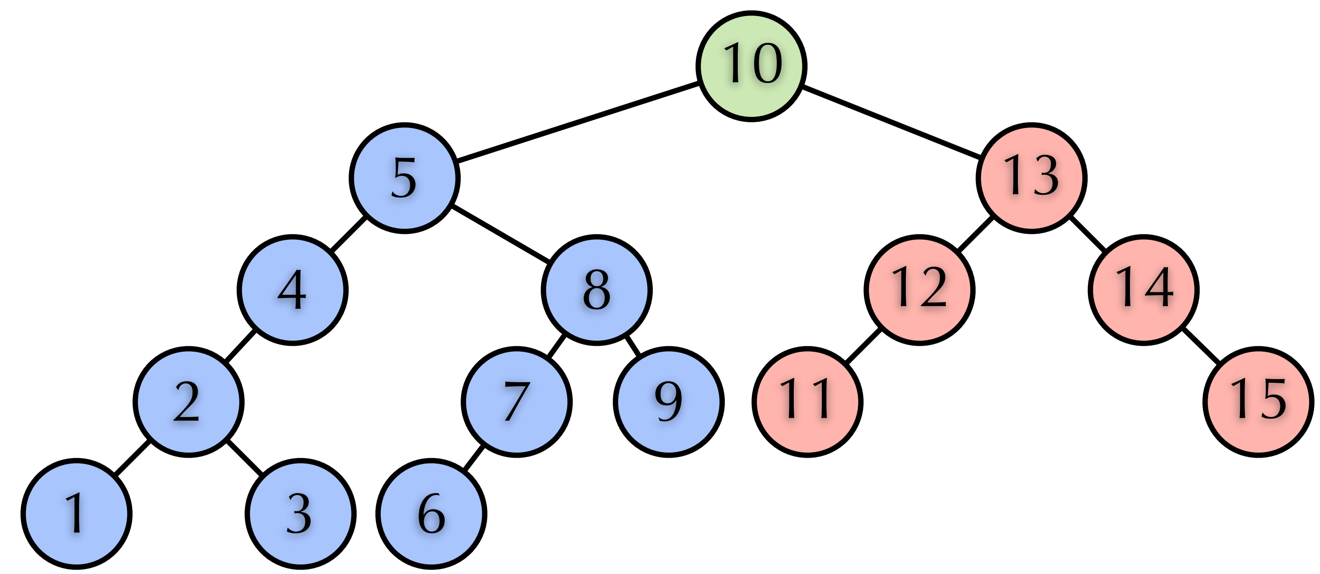
- $h_T = \max(h_R, h_L) + 1$
- Preorder traversal
- BST

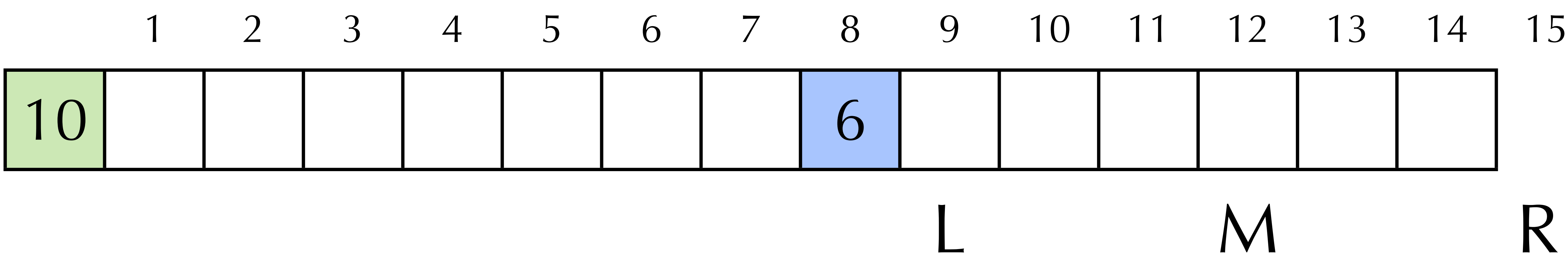




Partition: Binary Search

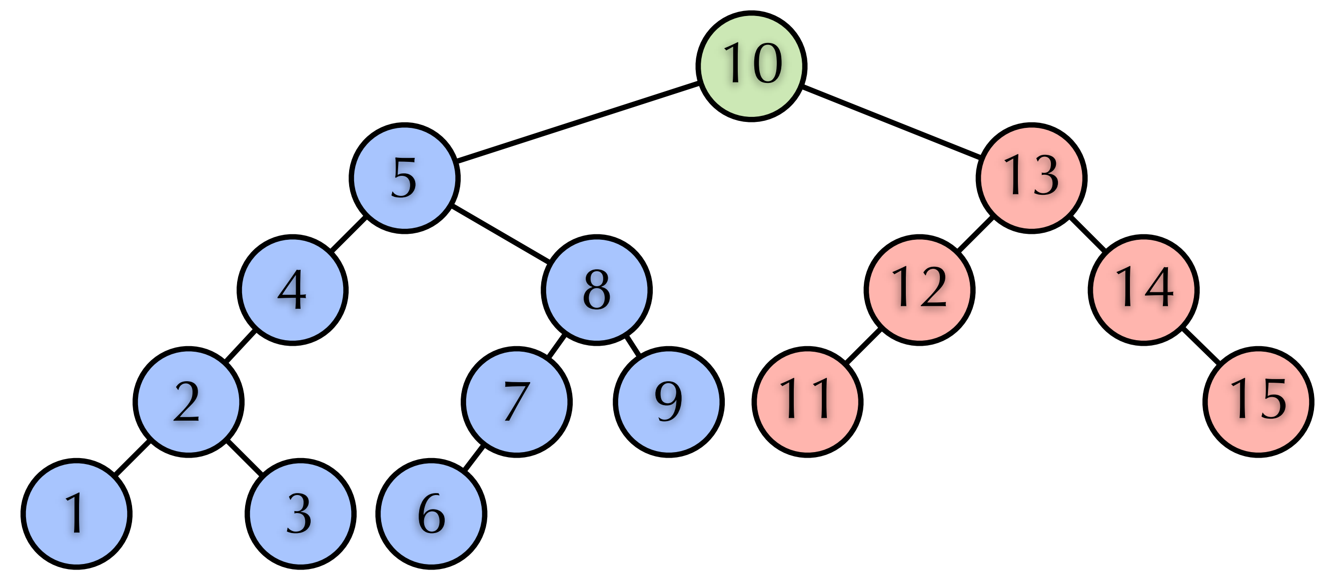
- $h_T = \max(h_R, h_L) + 1$
- Preorder traversal
- BST

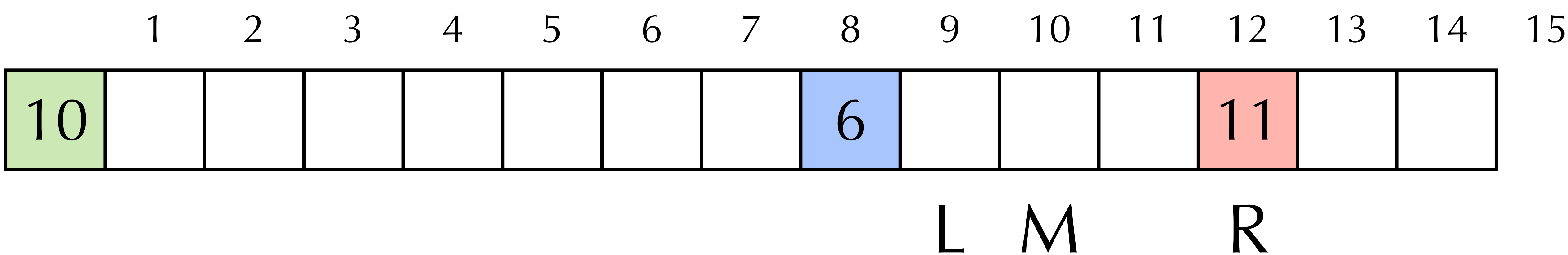




Partition: Binary Search

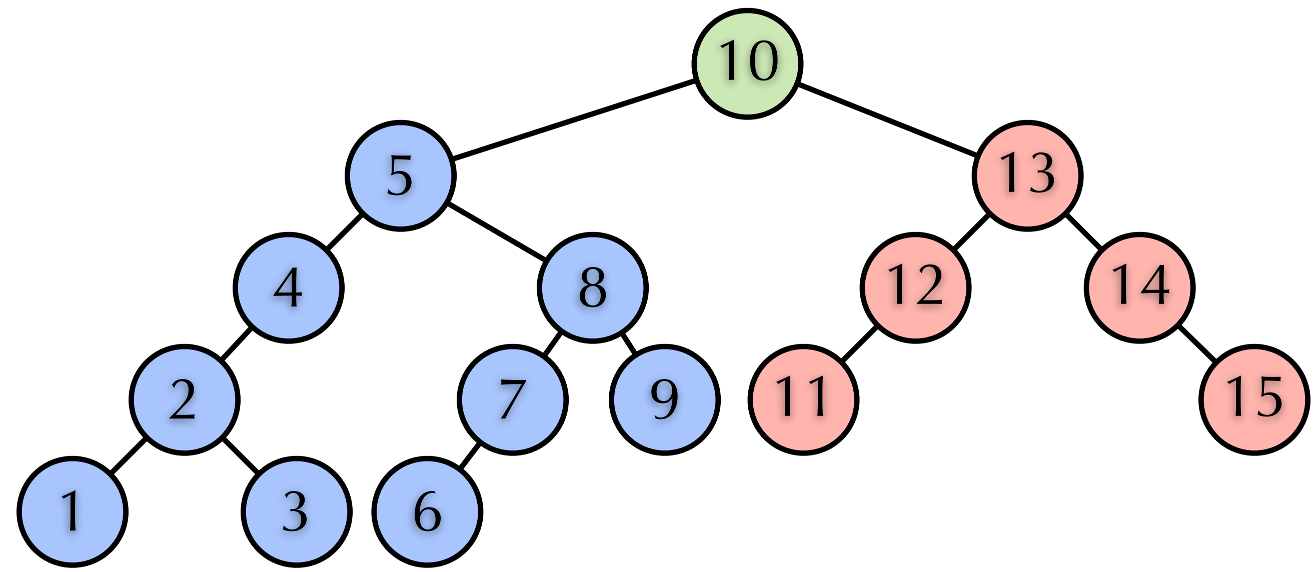
- $h_T = \max(h_R, h_L) + 1$
- Preorder traversal
- BST





Partition: Binary Search

- $h_T = \max(h_R, h_L) + 1$
- Preorder traversal
- BST



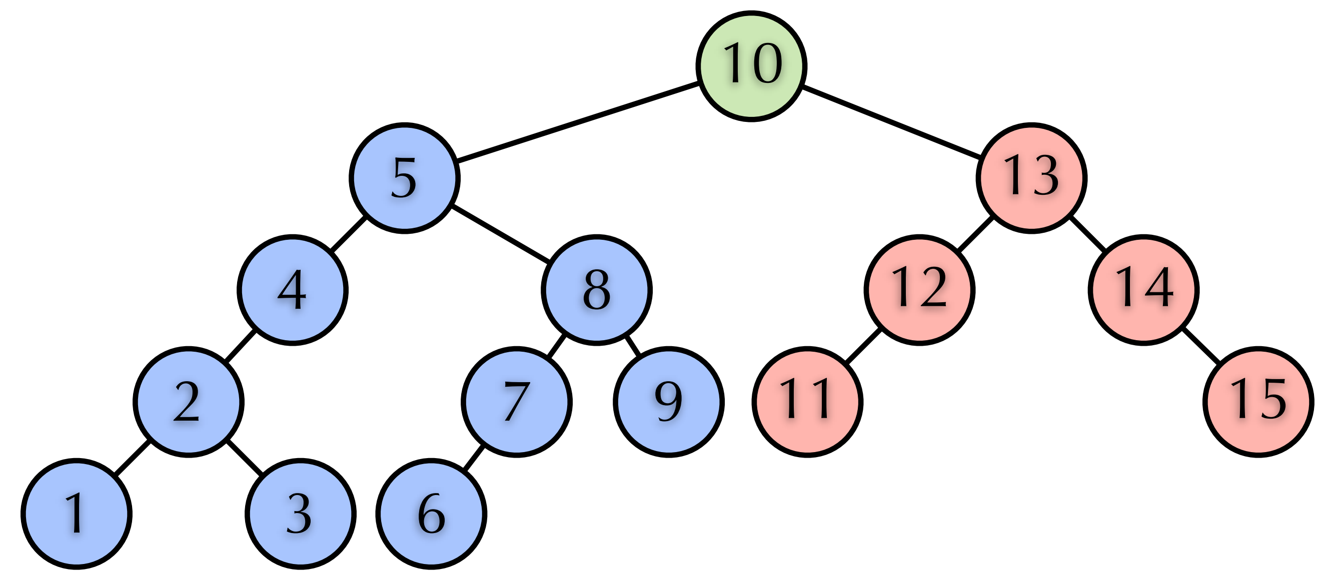
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10								6		13		11			

L R

M

Partition: Binary Search

- $h_T = \max(h_R, h_L) + 1$
- Preorder traversal
- BST



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10								6	9	13		11			

R

L