其中，统计某个区间的数据之和，与统计某个区间的数据个数的处理方式类似，我们只需要将节点中的统计变量由 count 换成 sum。而对于统计某个区间的第 K 大值，也只需要在统计某个区间的数据个数的基础之上，稍作改动就能实现。其中，构建线段树、插入数据和删除数据的代码完全不变，查询区间第 K 大值的代码如下所示。

```
public int getKth(int left, int right, int k) {
  return getKthInternal(left, right, 1, k);
}

private int getKthInternal(int left, int right, int i, int k) {
  if (segments[i].left == left
      && segments[i].right == right) {
    if (k != -1) {
      return -1; //表示第k大值存在
    } else {
      return segments[i].left;
    }
  }
  int mid = (left + right) / 2;
  int rightSegmentCount = countInternal(mid+1, right, i*2+1);
  if (rightSegmentCount >= k) {
    return getKthInternal(mid+1, right, i*2+1, k);
  } else {
    return getKthInternal(left, mid, i*2, k-rightSegmentCount);
  }
}
```

求区间最大值和最小值的处理思路完全一致，我们用求区间最大值来举例讲解。对于这个区间统计问题，线段树中每个节点记录本区间的最大值。构建空线段树、插入数据、区间查询与其他的区间统计问题类似，唯一比较特殊的是删除操作，我们需要在递归删除数据的同时，更新节点所属区间的最大值，具体代码如下所示。

```
public void delete(int data) {
  deleteInternal(data, 1);
}

private void deleteInternal(int data, int i) {
  if (segments[i].left == segments[i].right) {
    segments[i].max = Integer.MIN_VALUE;
    return;
  }
  int mid = (segments[i].left + segments[i].right)/2;
  if (data <= mid) {
    deleteInternal(data, i*2);
  } else {
    deleteInternal(data, i*2+1);
  }
  segments[i].max = Math.max(segments[i*2].max, segments[i*2+1].max);
}
```

上述删除操作的代码实现针对的是集合中没有重复数据的情况。如果集合中存在重复数据，那么我们需要再用另外一个数据结构记录重复数据的个数。假设要删除的数据是 a，如果 a 在集合中存在多个，对于删除操作，我们只需要将记录的数据个数减 1，不需要更新线段树。如果 a 在集合中只存在一个，对于删除操作，我们需要按照上面的代码处理方式更新线段树。