A sorting algorithm that runs O(NlogN) in best case is MergeSort, QuickSort, and HeapSort. Best Case for both is mergeSort. Out of these three sorts MergeSort is the one that is stable.

```
    public static void merge(int[] a, int[] left, int[]right, int leftSide, int rightSide){
        int i = 0;
        int j = 0;
        int k = 0;
        while(i < leftSide && j < rightSide){
            if(left[i] < right[j]){
                a[k++] = left[i++];
            }
            else{
                a[k++] = right[j++];
            }
        }
        while(i < leftSide){
            a[k++] = left[i++];
        }
        while(j < rightSide){
            a[k++] = right[j++];
        }
    }
    public static void mergeSort(int[]a, int size){
        if(size < 2) return;
        int mid = size/2;
        int[] left = new int[mid];
        int[] right = new int[size - mid];

        for(int i = 0; i<mid; i++){
            left[i] = a[i];
        }
        for(int i = 0; i<mid; i++){
            right[i] = a[i+mid];
        }
        mergeSort(left,mid);
        mergeSort(right,size-mid);
        merge(a,left,right,mid,size-mid);
    }
}
```

As we can see here MergeSort is stable because it does not create auxiliary arrays like quicksort, it splits it into left and right halves sorts them then connects it back.

The best case time complexity is O(NlogN) and worst case is also O(NlogN) because here the conditions are the while loops where it checks the left and right hand side, and the left and right hand side both iterate and sort.