Binary Search

14. Classical Binary Search

public class Solution {

 public int binarySearch(int[] array, int target) {

   // Write your solution here

   int left = 0;

   int right = array.length-1;

   while(left<=right){  // use sigle element to test

     int mid = left+(right-left)/2;  // avoid overflow

     if(array[mid]==target){

       return mid;

     }

     if(array[mid]>target){

       right = mid-1;

     }else{

       left = mid+1;

     }

   }

   return -1;

 }

}

267. Search In Sorted Matrix I

public class Solution {

 public int[] search(int[][] matrix, int target) {

   // Write your solution here

   int row = matrix.length;

   int col = matrix[0].length;

   int[] result = new int[2];

   int left = 0;

   int right = row\*col-1;  // be careful about the index

   while(left<=right){

     int mid = left+(right-left)/2;

     int x = mid/col;

     int y = mid%col;

     if(matrix[x][y]==target){

       result[0] = x;

       result[1] = y;

       return result;

     }

     if(matrix[x][y]>target){

       right = mid-1;

     }else{

       left = mid+1;

     }

   }

   result[0] = -1;

   result[1] = -1;

   return result;

 }

}

C++:

class Solution {

public:

 vector<int> search(vector<vector<int>> matrix, int target) {

   // write your solution here

   int left = 0;

   int right = matrix.size() \* matrix[0].size() - 1;

   vector<int> result;

   while (left <= right) {

     int mid = left + (right - left) / 2;

     int x = mid / matrix[0].size();

     int y = mid % matrix[0].size();

     if (matrix[x][y] == target) {

       result.push\_back(x);

       result.push\_back(y);

       return result;

     }

     if (matrix[x][y] > target) {

       right = mid - 1;

     } else {

       left = mid + 1;

     }

   }

   result.push\_back(-1);

   result.push\_back(-1);

   return result;

 }

};

17. Closest In Sorted Array

public int closest(int[] array, int target) {

   // Write your solution here

   // Need to find the closest TWO elements using binary search and then determine which one is the closest

   if(array.length==0)

     return -1;

   int left = 0;

   int right = array.length-1;  // remember -1

   while(left<right-1){  // because this ensure two elements after the loop

     int mid = left+(right-left)/2;

     if(array[mid]==target)

       return mid;

     // 不能使用left=mid+1和right=mid-1! [...3,4,...],当target=3.5的时候mid=left+1就会跳过target

     if(array[mid]<target){

       left = mid;

     }else if(array[mid]>target){

       right = mid;

     }

   }

   // judge which one is the closest between the two elements

   if(target-array[left]>array[right]-target)

     return right;

   else

     return left;

 }

C++:

class Solution {

public:

 int closest(vector<int> array, int target) {

   // write your solution here

   if (array.size() == 0) {

     return -1;

   }

   // 1. need to find the last two elements that are wrapping around the target value

   int left = 0;

   int right = array.size() - 1;

   while (left < right - 1) {

     int mid = left + (right - left) / 2;

     if (array[mid] == target) {

       return mid;

     }

     if (array[mid] < target) {

       left = mid;

     } else if (array[mid] > target) {

       right = mid;

     }

   }

   // 2. check which one is the closest one

   if (abs(target - array[left]) < abs(array[right] - target)) {

     return left;

   } else {

     return right;

   }

 }

};

15. First Occurrence

public int firstOccur(int[] array, int target) {

   // Write your solution here

   if(array.length==0)

     return -1;

   // consider this as: aaaaaaaaaaaBbbbbbbbbb, then find B

   // define left and right

   int left = 0;

   int right = array.length-1;

   while(left<right-1){  // in the last two elements array[left]==array[right]==target, if using while(left<=right), then give wrong result

     int mid = left+(right-left)/2;

     if(array[mid]==target){

       right = mid;  // do not return, keep moving toward middle to find the leftmost target

       continue;     // NEED TO CONTINUE otherwise it will keep doing the rest in the loop

     }

     if(array[mid]<target)

       left = mid+1;  // did not skip target

     else

       right = mid-1;  //did not skip target

   }

   if(array[left]==target)

     return left;

   else if(array[right]==target)

     return right;

   else

     return -1;

 }

C++:

class Solution {

public:

 int firstOccur(vector<int> array, int target) {

   // write your solution here

   // Time: O(logn)

   // Space: stack O(n), heap 0

   if (array.size() == 0) {

     return -1;

   }

   int left = 0;

   int right = array.size() - 1;

   while (left < right - 1) {

     int mid = left + (right - left) / 2;

     if (array[mid] >= target) {

       right = mid;  // don't want to skip

     } else {

       left = mid;

     }

   }

   // Now have 2 element. So check left first

   if (array[left] == target) {

     return left;

   } else if (array[right] == target) {

     return right;

   } else {

     return -1;

   }

 }

};

16. Last Occurrence

public class Solution {

 public int lastOccur(int[] array, int target) {

   // Write your solution here

   if(array.length==0)

     return -1;

   int left = 0;

   int right = array.length-1;

   while(left<right-1){  // 注意退出的condition

     int mid = left+(right-left)/2;

     if(array[mid]==target){

       left = mid;

     }

     if(array[mid]>target){

       right = mid-1;

     }else if(array[mid]<target){

       left = mid+1;

     }

   }

   if(array[right]==target)

     return right;

   else if(array[left]==target)

     return left;

   return -1;

 }

}

19. K Closest In Sorted Array

public class Solution {

 public int[] kClosest(int[] array, int target, int k) {

      // Write your solution here

      int left = 0;

      int right = array.length-1;

      while(left<right-1){

        int mid = left+(right-left)/2;

        if(array[mid]>=target){

          right = mid;

        }else if(array[mid]<target){

          left = mid;

        }

      }

      int[] result = new int[k];

      int i;

      for(i=0;i<k;i++){

        if(left<0 || right>=array.length)

          break;

        if((target-array[left])>(array[right]-target)){

          result[i] = array[right];

          right++;

        }else{

          result[i] = array[left];

          left--;

        }

      }

      if(i==k){

        return result;

      }else if(left<0){

        while(i<k){

          result[i] = array[right];

          right++;

          i++;

        }

      }else if(right>=array.length){

        while(i<k){

          result[i] = array[left];

          left--;

          i++;

        }

      }

      return result;

    }

}

C++:

class Solution {

public:

 vector<int> kClosest(vector<int> array, int target, int k) {

   // write your solution here

   int left = 0;

   int right = array.size() - 1;

   while (left < right - 1) {

     int mid = left + (right - left) / 2;

     if (array[mid] >= target) {

       right = mid;

     } else {

       left = mid;

     }

   }

   vector<int> result;

   int i = 0;

   while (left >=0 && right < array.size() && i < k) {

     if (abs(array[left] - target) <= abs(array[right] - target)) {

       result.push\_back(array[left]);

       left--;

     } else {

       result.push\_back(array[right]);

       right++;

     }

     i++;

   }

   while (i < k && left >= 0) {

     result.push\_back(array[left]);

     left--;

     i++;

   }

   while (i < k && right < array.size()) {

     result.push\_back(array[right]);

     right++;

     i++;

   }

   return result;

 }

};

636. Smallest Element Larger than Target

public class Solution {

 public int smallestElementLargerThanTarget(int[] array, int target) {

   // Write your solution here

   if(array.length==0)

     return -1;

   int left = 0;

   int right = array.length-1;

   while(left<right-1){

     int mid = left+(right-left)/2;

     if(array[mid]>target){

       right = mid; // cannot use mid-1;

     }else if(array[mid]<=target){

       left = mid;

     }

   }

   if(array[left]>target)

     return left;

   else if(array[right]>target)

     return right;

   return -1;

 }

}

C++:

class Solution {

public:

 int smallestElementLargerThanTarget(vector<int> array, int target) {

   // write your solution here

   // Similar to first occurance

   if (array.size() == 0) {

     return -1;

   }

   int left = 0;

   int right = array.size() - 1;

   while (left < right - 1) {

     int mid = left + (right - left) / 2;

     if (array[mid] > target) {

       right = mid;

     } else {

       left = mid;

     }

   }

   if (array[left] > target) {

     return left;

   } else if (array[right] > target) {

     return right;

   } else {

     return -1;

   }

 }

};

20. Search In Unknown Sized Sorted Array

/\*

\*  interface Dictionary {

\*    public Integer get(int index);

\*  }

\*/

// You do not need to implement the Dictionary interface.

// You can use it directly, the implementation is provided when testing your solution.

public class Solution {

 public int search(Dictionary dict, int target) {

   // Write your solution here

   // 1. Find the size of the dict by jumping by \*2

   int left = 0;

   int right = 1;  // 特别注意必须是1

   while(dict.get(right)!=null && dict.get(right)<target){

     left = right;

     right \*= 2;

   }

   // 2. Start binary search for numbers between left and right

   while(left<=right){

     int mid = left+(right-left)/2;

     if(dict.get(mid)==null){

       right = mid-1;

       continue;

     }

     if(dict.get(mid)==target){

       return mid;

     }

     if(dict.get(mid)<target){

       left = mid+1;

     }else if(dict.get(mid)>target){

       right = mid-1;

     }

   }

   return -1;

 }

}

C++:

// class UnknownSizeVector {

//  public:

//   int get(int index) {

//     // Return INT\_MIN if out of bound,

//     // otherwise return the element value.

//   }

// }

class Solution {

public:

 int solve(UnknownSizeVector input, int target) {

   // 1. increase the searching range by 2 fold

   int left = 0;

   int right = 1;

   while (input.get(right) != INT\_MIN && input.get(right) < target) {

     left = right;

     right \*= 2;

   }

   // 2. use binary search to find the target

   while (left <= right) {

     int mid = left + (right - left) / 2;

     if(input.get(mid) == INT\_MIN || input.get(mid) > target) {

       right = mid - 1;

     } else if (input.get(mid) < target) {

       left = mid + 1;

     } else {

       return mid;

     }

   }

   return -1;

 }

};

21. Search In Shifted Sorted Array I

public class Solution {

 public int search(int[] array, int target) {

   // Write your solution here

     if(array.length==0)

        return -1;

      int left = 0;

      int right = array.length-1;

      // 分三种情况讨论, ==, >, < target

      while(left<=right){

        int mid = left+(right-left)/2;

        if(array[mid]==target){

          return mid;

        }

        if(array[mid]<array[right]) {  // 如果用 array[mid]>=array[left]会把[2,1]中target=1 rule out

          if(target>array[mid] && target<=array[right]) {  // 保证右边是ascending的

        left = mid+1;

          }else {

        right = mid-1;

          }

        }else {  // array[mid]>=array[right]

          if(target>=array[left] && target<array[mid]) {

        right = mid-1;

          }else {

        left = mid+1;

          }

        }

      }

      return -1;

 }

}