Binary Search

Principles of Binary Search

1. We must guarantee that the **search space decreases** over time (after each iteration)
2. We must guarantee that the **target (if exists) cannot be ruled out** accidentally, when we change the value of Left or Right. **(It is critical to define the rule about how to move the range for search)**

Question 1: Classical Binary Search

--- to find an element/number in an array, → sorted array.

Example: a[7] = 1 2 4 5 7 8 9 whether target == **4** is in this array or not.

L=M+1R =M-1

index	0	1	2	3	4	5	6
A[7]	1	2	**4**	5	7	8	9

Iteration 1: L = 0, R = 6, M = 3 A[M] == A[3] == 5 > target == 4, so R = M-1 = 2;
Iteration 2: L = 0, R = 2, M = 1 A[M] == A[1] == 2 < target == 4, so L = M+1 = 2;
Iteration 3: L = 2, R = 2, M = 2 A[M] == A[2] == 4 == target , so Done!!!

Question 2: Classical Binary Search in 2D Space

2D matrix, sorted on each row, first element of next row is larger(or equal) to the last element of previous row, now giving a target number, returning the position that the target locates within the matrix

```
public boolean ifFind(int[][] matrix, int target) {
      if (matrix.length == 0 || matrix[0].length == 0)
            return false;

      int row = matrix.length;
      int col = matrix[0].length;
      int i = 0;                  // left
      int j = row * col - 1;      // right
      while (i <= j) {
            int mid = i + (j - i)/2;
            int r = mid / col; // helper function to map n-dimensional
                              // coordinate to 1D coordinate (vice versa)
            int c = mid % col;
            if (matrix[r][c] == target)
                  return true;
            else if (matrix[r][c] > target)
                  j = mid - 1;
            else
                  i = mid + 1;
      }
      return false;
}

Time = O(log m x n)
```

Question 3: Closest Element to Target

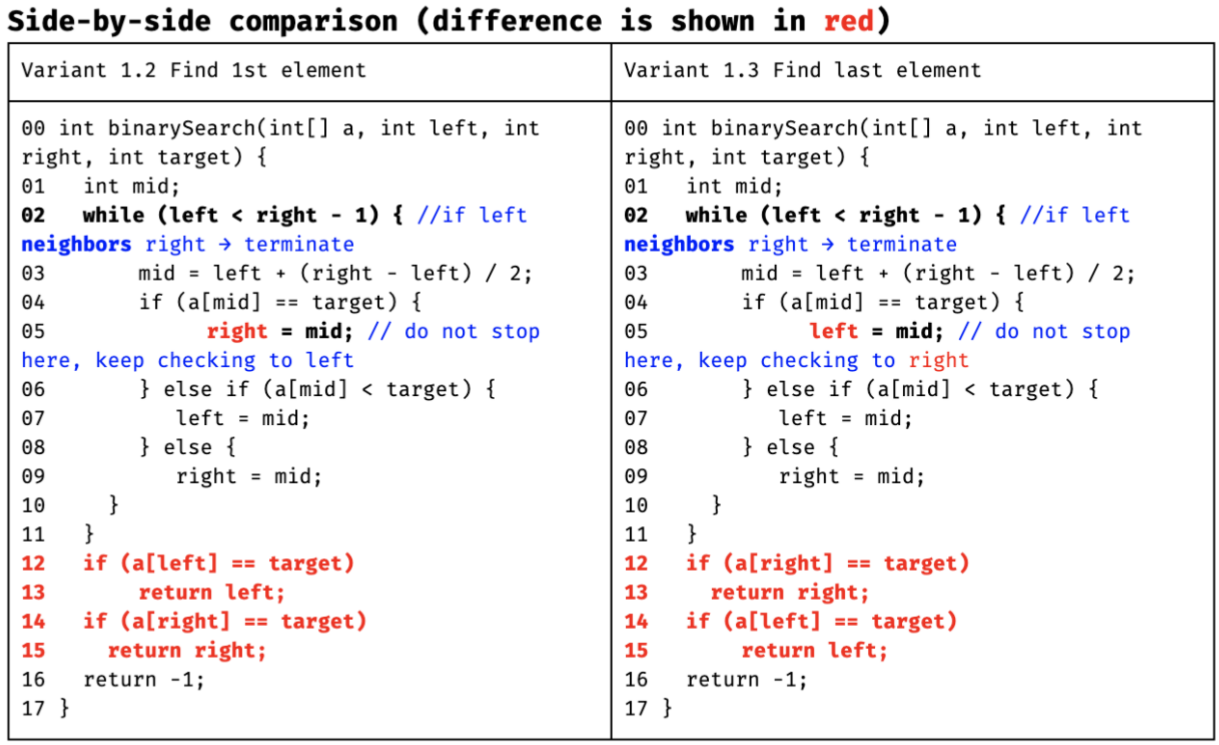
How to find an element in the array that is **closest** to the target number?

Target == 4;   **L=2 M=3   R=4**
                        index   0   1   2   3   4
// e.g. int a[5] = {1, 2, **3**,4 8, 9};
                                                L   R
                                    xxxxxxxxxxxxxxxx

When you only have two elements left in the valid range, then you only need to check whether the left choice or the right choice is the answer (it exists).

```
int binarySearch(int[] a, int left, int right, int target) {
  int mid;
  while (left < right - 1) { // if left neighbors right → terminate
    mid = left + (right - left) / 2;
    if (a[mid] == target) {
       return mid;
    } else if (a[mid] < target) {
       left = mid;            // left = mid + 1 (Wrong???)
    } else {
       right = mid;           // right = mid - 1 (Wrong)
    }
  }
  // Post-processing
  if (Math.abs(a[left] - target) <= Math.abs(a[right] - target)) // check a[left] against target first
    return left;
  else
    return right;
}
```

Question 4/4: Find First/Last element



Side-by-side comparison (difference is shown in red)

Variant 1.2 Find 1st element

```
int binarySearch(int[] a, int left, int right, int target) {
  int mid;
  while (left < right - 1) { //if left neighbors right → terminate
      mid = left + (right - left) / 2;
      if (a[mid] == target) {
           right = mid; // do not stop here, keep checking to left
      } else if (a[mid] < target) {
         left = mid;
      } else {
         right = mid;
     }
  }
  if (a[left] == target)
      return left;
  if (a[right] == target)
    return right;
  return -1;
}
```

Variant 1.3 Find last element

```
int binarySearch(int[] a, int left, int right, int target) {
  int mid;
  while (left < right - 1) { //if left neighbors right → terminate
      mid = left + (right - left) / 2;
      if (a[mid] == target) {
           left = mid; // do not stop here, keep checking to right
      } else if (a[mid] < target) {
         left = mid;
      } else {
         right = mid;
     }
  }
  if (a[right] == target)
    return right;
  if (a[left] == target)
      return left;
  return -1;
}
```