1. Square root of Binary search

Algorithm:

1. First set the number for square root.
2. The number pass the method. First condition n<0 it is invalid calculate for square root.
3. (n==0 ||n==1) this condition return n.
4. Find the mid value of given number. The square the mid value.
5. Mid<n return low is a mid value. Mid>n return high is a mid value.
6. Last step return mid value.

```
package in;
        public class Binary {
           public static double square(double n, double c) {
              if (n < 0) {
             System.out.println("Cannot calculate square root for negative
number.");
                 return -1;
              }

              if (n == 0 || n == 1) {
                 return n;
              }

              double low = 0;
              double high = n;

              while (high - low > c) {
                 double mid = (high + low) / 2;
                 double midSquare = mid * mid;

                 if (midSquare < n) {
                    low = mid;
                 } else {
                    high = mid;
                 }
              }

              return (high +low) / 2;
           }

           public static void main(String[] args) {
```

```
            double n = 25;
            double c = 0.00001;

            double result = square(n,c );

            if (result != -1) {
               System.out.printf("The square root of %.5f is approximately
%.5f\n", n, result);
            }
         }
      }
```

Output:

The square root of 25.00000 is approximately 5.00000

2. BINARY SEARCH

Algorithm:

1. Initialize: Set left to 0 and right to the length of the array nums minus 1.
2. Binary Search:
3. While left is less than or equal to right, do the following:
4. Calculate the mid index: mid = left + (right - left) / 2.
5. If the element at mid is equal to the target (nums[mid] == target), return mid (target found).
6. If the target is greater than the element at mid (nums[mid] < target), update left = mid + 1 (search in the right half).
7. If the target is less than the element at mid (nums[mid] > target), update right = mid - 1 (search in the left half).
8. If the target is not found, return -1 to indicate that the target is not present in the array.

```java
package in;

public class Index {
   public static int search(int[] nums, int target) {
            int left = 0;
            int right = nums.length - 1;

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

               if (nums[mid] == target) {
                  return mid;
               } else if (nums[mid] < target) {
                  left = mid + 1;
               } else {
                  right = mid - 1;
               }
            }

            return -1;
         }

         public static void main(String[] args) {
            int[] nums = {-1, 0, 3, 5, 9, 12};
            int target = 9;

            int result = search(nums, target);

            if (result != -1) {
               System.out.println("Target " + target + " found at index:" +
result);
            } else {
               System.out.println("Target " + target + " not found in the array.");
            }
         }
      }
```

Output:

Target 9 found at index:4

Explanation:

- Initialize left to 0 and right to the length of the array nums minus 1.
- Repeat the following steps while left is less than or equal to right: a. Calculate the mid index: mid = left + (right - left) / 2. b. If the element at mid is equal to the target (nums[mid] == target), return mid (which is the index of the target). c. If the target is greater than the element at mid (nums[mid] < target), set left = mid + 1 (search in the right half). d. If the target is less than the element at mid (nums[mid] > target), set right = mid - 1 (search in the left half).
- If the target is not found in the array, return -1 to indicate that the target is not present.
- Following this approach, we'll find that for the input nums = [-1,0,3,5,9,12] and target = 9, the target 9 is found at index 4, so the output is 4, as indicated in the explanation.
- Top of Form