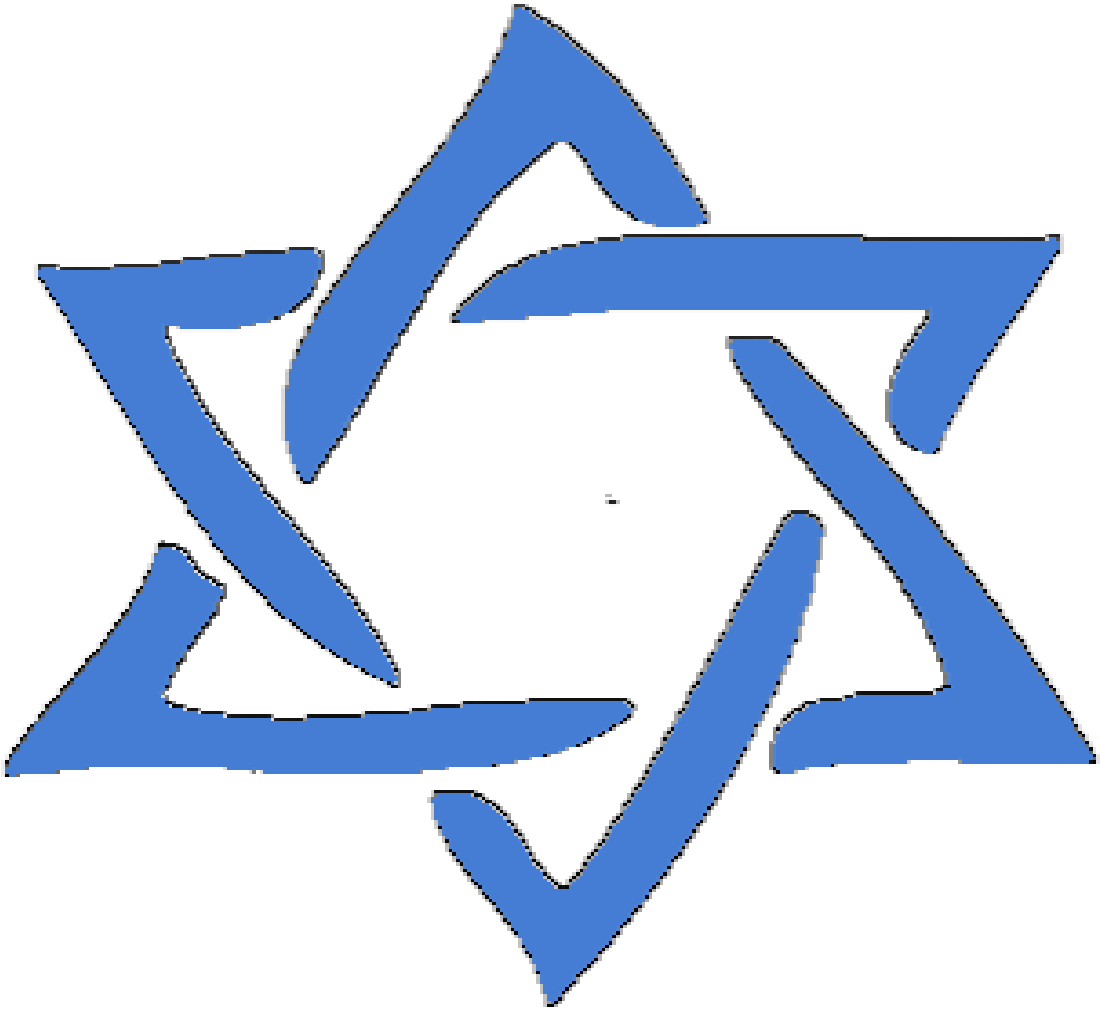
Binary search
Lesson 2



FIND THE HIGHEST NUMBER

Given an array in such a way that the elements stored in array are in increasing order initially and then after reaching to a peak element, elements stored are in decreasing order. Find the highest element.

Note: A[i] != A[i+1]

Input:

11

1 2 3 4 5 6 5 4 3 2 1

Output:

6

Explanation: Highest element is 6.

Input:
5
1 2 3 4 5
Output:
5
Explanation: Highest element is 5.

ALGORITHM:
NAIVE APPROACH,by simply traversing the array we can find the highest number,in O(N) time complexity, where N is the number of elements in the array.

Using BINARY SEARCH,we can solve this is O(logN) complexity.

First find the middle element, if the middle element is greater than the previous and next element in the array,then the middle element in the highest number in the array.
Else If the middle element is smaller than the previous element ,then further check in the left half of the array.
Else check in the right half of the array.

CODE:

```
1.  int findPeakElement(vector<int>& a)
2.  {
3.     int low = 0;
4.     int high = a.size() - 1;
5.
6.     while(low<=high)
7.     {
8.        int mid = low + (high - low)/2;
9.        if((a[mid-1] < a[mid]) && (a[mid+1] < a[mid]))
10.          return a[mid];
11.
12.       else if(a[mid-1] > a[mid])
13.          high = mid - 1;
14.       else
15.          low = mid + 1;
16.    }
17. }
```

QUES LINK : [find the highest number]

FIND PAIR GIVEN DIFFERENCE

Given an array Arr[] of size L and a number N, you need to write a program to find if there exists a pair of elements in the array whose difference is N.

Input:

L = 6, N = 78

arr[] = {5, 20, 3, 2, 5, 80}

Output: 1

Explanation:

(2, 80) have a difference of 78.

Input:

L = 5, N = 45

arr[] = {90, 70, 20, 80, 50}

Output: -1

Explanation:

There is no pair with a difference of 45.

ALGORITHM:

NAIVE ALGORITHM, IN O(n^2) time complexity ,we can do this,by selecting every element of the array one by one then find its pair element in the array.

BINARY SEARCH ALGO: firstly ,sort the array . By selecting every element of the array one by one and then find its pair element using binary search in the array. TIME COMPLEXITY: O(NlogN)

CODE:

```
1.  bool binarysearch(int arr[], int n, int ele)
2. {
3.     int low = 0;
4.     int high = n-1;
5.
6.     while(low <= high)
7.     {
8.         int mid = low + (high - low)/2;
9.         if(arr[mid] == ele)
10.            return true;
11.        else if(arr[mid] < ele)
12.            low =  mid + 1;
13.        else
14.            high = mid - 1;
15.    }
16.    return false;
17.}
18.
19.bool findPair(int arr[], int size, int n){
20.        sort(arr, arr + size);
21.
```

```
22.      for(int i = 0; i < size; i++)
23.      {
24.          if(binarysearch(arr, size, abs(n - arr[i])))
25.              return true;
26.      }
27.      return false;
28.  }
```

QUES LINK: find pair given difference

MINIMUM NUMBER IN A SORTED ROTATED ARRAY

Given an array of distinct elements which was initially sorted. This array is rotated at some unknown point. The task is to find the minimum element in the given sorted and rotated array.

Example 1:

Input:
N = 10
arr[] = {2,3,4,5,6,7,8,9,10,1}
Output: 1

Given an array of distinct elements which was initially sorted. This array is rotated at some unknown point. The task is to find the minimum element in the given sorted and rotated array.

Input:
N = 10
arr[] = {2,3,4,5,6,7,8,9,10,1}

Output: 1

Explanation: The array is rotated once anti-clockwise. So minimum element is at last index (n-1) which is 1.

Input:
N = 6
arr[] = { 50, 60, 10, 20, 30, 40 }

Output: 10

Explanation: The array is rotated twice clockwise. So minimum element is at 1st index.

ALGORITHM:

A simple solution is to traverse the complete array and find a minimum. This solution requires O(n) time.

We can do it in O(Logn) using Binary Search. If we take a closer look at the above examples, we can easily figure out the following pattern:

- The minimum element is the only element whose previous and next both are greater than it. If there is no previous element, then there is no rotation (the first element is minimum). If there is no next element, then there is single anticlockwise rotation(the last element is minimum). We check this condition for the middle element by comparing it with (mid-1)th and (mid+1)th elements.
- If the minimum element is not at the middle, then the minimum element lies in either the left half or right half.
    1. If the middle element is smaller than the last element, then the minimum element lies in the left half
    2. Else minimum element lies in the right half.

CODE:

```
1. int minNumber(int a[], int low, int high)
```

```
2. {
3.    while(low<=high)
4.    {
5.        if(high == low)
6.            return a[high];
7.        int mid = low + (high - low)/2;
8.
9.        if((mid == low || a[mid-1] > a[mid]) && (mid == high || a[mid+1] >
    a[mid]))
10.           return a[mid];
11.       else if(a[mid] > a[high])
12.           low = mid + 1;
13.       else if(a[mid] < a[high])
14.           high = mid - 1;
15.   }
16.}
```

QUES LINK : minimum no in a sorted rotated array

K SORTED ARRAY

Given an array of n distinct elements. Check whether the given array is a k sorted array or not. A k sorted array is an array where each element is at most k distance away from its target position in the sorted array.

Input:

N = 6

arr[] = {3, 2, 1, 5, 6, 4}

K = 2

Output: Yes

Explanation:

Every element is at most 2 distance away from its target position in the sorted array.

Input:

N = 7

arr[] = {13, 8, 10, 7, 15, 14, 12}

K = 1

Output: No

Explanation:

7, 12, 13, 15 are more than 1 distance away from their target position.

ALGORITHM:

NAIVE ALGO.: Copy elements of original array arr[] to an arr2[].

Sort arr2[]. Now, for each element at index i in arr[], find its index j in arr2[] . If for any element k < abs(i-j), then arr[] is not a k sorted array. Else it is a k sorted array. Here abs is the absolute value.

Time complexity:O(N^2), SPACE complexity:O(N)

USING BINARY SEARCH:

Copy elements of original array arr[] to an arr2[].

Sort arr2[]. Now, for each element at index i in arr[], find its index j in arr2[] using binary search . If for any element k < abs(i-j), then arr[] is not a k sorted array. Else it is a k sorted array. Here abs is the absolute value.

Time complexity:O(NlogN), SPACE complexity:O(N)

CODE:

```
1.  int binarysearch(int a[], int low, int high, int ele)
2.  {
3.      if(low>high)
4.          return -1;
5.
6.      int mid = low + (high - low)/2;
7.      if(a[mid] == ele)
8.          return mid;
9.      else if(a[mid] > ele)
10.         return binarysearch(a, low, mid-1, ele);
11.     else
12.         return binarysearch(a, mid+1, high, ele);
```

```
13.}
14.
15.string isKSortedArray(int arr[], int n, int k)
16.{
17.   int arr2[n];
18.   for(int i = 0; i<n; i++)
19.   {
20.         arr2[i] = arr[i];
21.   }
22.      sort(arr2, arr2 + n);
23.
24.      for(int i = 0; i<n; i++)
25.      {
26.         int pos = binarysearch(arr2, 0, n-1, arr[i]);
27.         if(abs(pos - i) > k)
28.            return "No";
29.      }
30.      return "Yes";
31.}
```

QUES LINK : [k sorted array]

ALL NUMBERS WITH SPECIFIC DIFFERENCE

Given a positive number N and a number D. Find the count of positive numbers smaller or equal to N such that the difference between the number and sum of its digits is greater than or equal to given specific value D.

Input:

N = 13 , D = 2

Output:

4

Explanation:

There are 4 numbers satisfying the conditions. These are 10,11,12 and 13.

10 – sumofdigit(10) = 9 >= 2

11 – sumofdigit(11) = 9 >= 2

12 – sumofdigit(12) = 9 >= 2

13 – sumofdigit(13) = 9 >= 2

ALGORITHM:

We can solve this problem by observing a fact that for a number k less than N,

if k – sumofdigit(k) >= diff then

above equation will be true for (k+1)

also because we know that sumofdigit(k+1)

is not greater than sumofdigit(k) + 1

so, k + 1 - sumofdigit(k + 1) >= k - sumofdigit(k)

but we know that right side of above inequality is greater than diff, so left side will also be greater than diff.

So, finally we can say that if a number k satisfies the difference condition then (k + 1) will also satisfy same equation so our job is to find the smallest number which satisfies the difference condition then all numbers greater than this and up to N will satisfy the condition so our answer will be N – smallest number we found.

We can find the smallest number satisfying this condition using binary search so total time complexity of solution will be O(log N)

CODE:

```
1.  long long utility(long long n)
2. {
3.    long long diff = 0, sod = 0, num = n;
4.    while(num>0)
5.    {
6.        sod += num % 10;
7.        num = num/10;
8.    }
9.    diff = n - sod;
10.   return diff;
11.}
12.
13.long long getCount(long long N , long long D)
```

```
14.{
15.   if(utility(N)<D)
16.       return 0;
17.   long long low = 1;
18.   long long high = N;
19.   long long ans = 1;
20.    while(low <= high)
21.   {
22.          long long mid = low + (high - low)/2;
23.          if(utility(mid) >= D)
24.              high = mid-1;
25.          else
26.              low = mid + 1;
27.       }
28.       return N - high;
29.    }
```

QUES LINK : [all the numbers with specific difference number and the digit sum]

COUNTING ELEMENTS IN TWO ARRAYS

Given two unsorted arrays arr1[] and arr2[]. They may contain duplicates. For each element in arr1[] count elements less than or equal to it in array arr2[].

Input:
m = 6, n = 6
arr1[] = {1,2,3,4,7,9}
arr2[] = {0,1,2,1,1,4}
Output:
4 5 5 6 6 6
Explanation:
Number of elements less than or equal to 1, 2, 3, 4, 7, and 9 in the second array are respectively 4,5,5,6,6,6.

ALGORITHM:

1. Sort the second array.
2. Traverse through the elements of the first array from start to end.
3. For every element in the first array.
4. Do a binary search on the second array and find the index of the largest element smaller than or equal to element of first array.
5. The index of the largest element will give the count of elements. Print the count for every index.

TIME COMPLEXITY: O(NlogN)

CODE:

```
1. int binary_search(int arr[], int l, int h, int x)
2. {
```

```
3.    while (l <= h) {
4.        int mid = (l + h) / 2;
5.        if (arr[mid] <= x)
6.            l = mid + 1;
7.        else
8.            h = mid - 1;
9.    }
10.   return h;
11.}
12.
13.void countEleLessThanOrEqual(
14.   int arr1[], int arr2[],
15.   int m, int n)
16.{
17.   sort(arr2, arr2 + n);
18.   for (int i = 0; i < m; i++)
19.   {
20.          int index = binary_search(
21.          arr2, 0, n - 1, arr1[i]);
22.          cout << (index + 1) << " ";
23.      }
24.   }
```

QUES LINK : [counting elements in two arrays]

SMALLEST FACTORIAL NUMBER

Given a number **n**. The task is to find the smallest number whose factorial contains at least n trailing zeroes.

Input:
n = 1
Output: 5

Explanation : 5! = 120 which has at least 1 trailing 0.

Input:

n = 6

Output: 25

Explanation : 25! has at least

6 trailing 0.

ALGORITHM :

The number of zeroes is equal to number of 5's in prime factors of x!.

Trailing 0s in x! = Count of 5s in prime factors of x!

= floor(x/5) + floor(x/25) + floor(x/125) + ...

Let us take few examples to observe pattern

5! has 1 trailing zeroes

[All numbers from 6 to 9

have 1 trailing zero]

10! has 2 trailing zeroes

[All numbers from 11 to 14

have 2 trailing zeroes]

15! to 19! have 3 trailing zeroes

20! to 24! have 4 trailing zeroes

25! to 29! have 6 trailing zeroes

We can notice that, the minimum value whose factorial contain n trailing zeroes is 5*n.

So, to find minimum value whose factorial contains n trailing zeroes, use binary search on range from 0 to 5*n. And, find the smallest number whose factorial contains n trailing zeroes.

CODE:

```
1.  int trailingzeroes(int n)
2. {
3.     int count=0;
4.     while(n>0)
5.     {
6.         n/=5;
7.         count+=n;
8.     }
9.     return count;
10.}
```

```
class Solution
{
  public:
      int findNum(int n)
      {
      int low=0,high=1000000;
      while(low<high)
      {
          int mid=(low+high)/2;

              int count=trailingzeroes(mid);
                  if(count<n)
                    low=mid+1;
                  else
                    high=mid;
          }
          return low;
          }
   };
```

Time Complexity: O($\log_2 N * \log_5 N$).

Auxiliary Space: O(1).

QUES LINK : [smallest factorial number]

PRACTICE QUESTIONS :

array subset of another array

balance with respect to an array

binary search in forest

geeks and his marks 3

geeks and his marks 2

find the minimum time