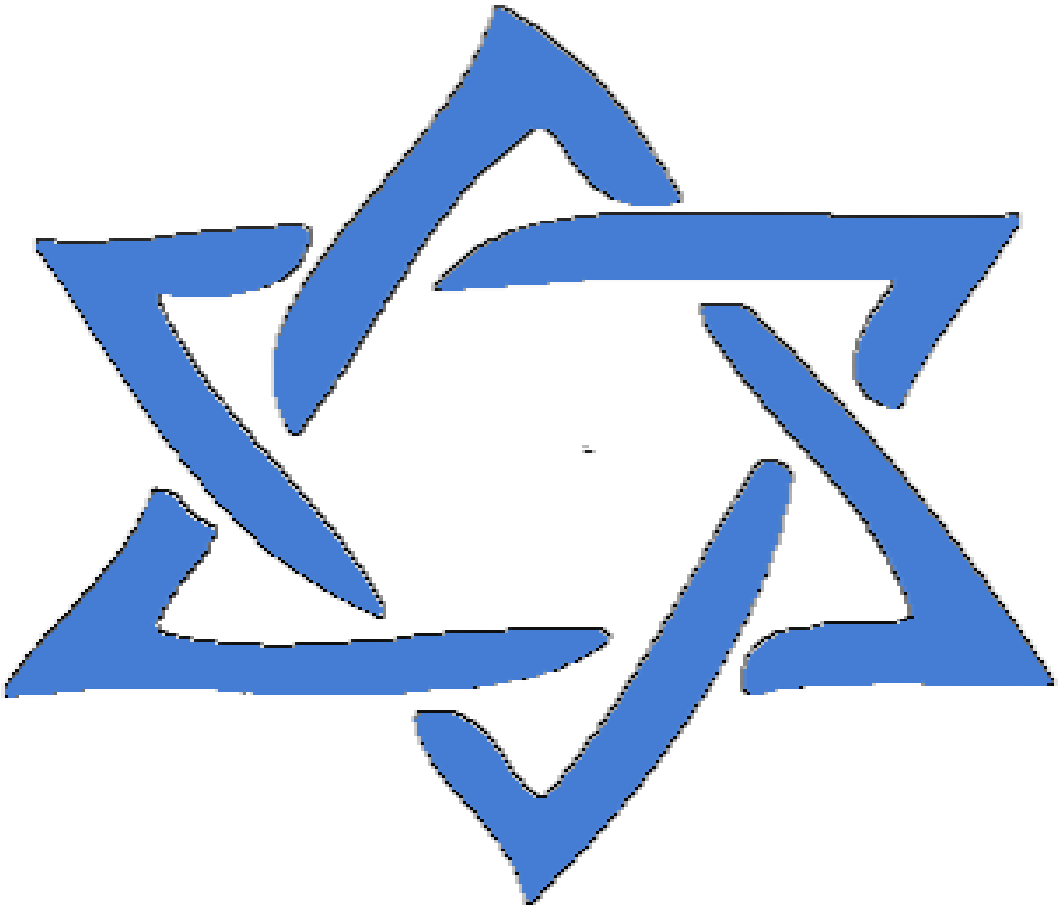
Binary search
Lesson 3



SQUARE ROOT OF A NUMBER

Given an integer x, find the square root of x. If x is not a perfect square, then return floor(√x).

Example 1:

Input:
x = 5
Output: 2
Explanation: Since, 5 is not a perfect square, floor of square_root of 5 is 2.

ALGORITHM:

Simple Approach: To find the floor of the square root, try with all-natural numbers starting from 1. Continue incrementing the number until the square of that number is greater than the given number.

- Algorithm:
    1. Create a variable (counter) *i* and take care of some base cases, i.e when the given number is 0 or 1.
    2. Run a loop until *i*i <= n* , where n is the given number. Increment i by 1.
    3. The floor of the square root of the number is *i - 1*

- *Complexity Analysis:*

    - *Time Complexity: O(√ n).*

      Only one traversal of the solution is needed, so the time complexity is O(√ n).
    - *Space Complexity: O(1).*

      Constant extra space is needed.

Better Approach: we can solve this by binary search.

1. Take care of some base cases, i.e when the given number is 0 or 1.
2. Create some variables, lowerbound *low = 0*, upperbound *high = x*, where x is the given number, *mid* for the mid values .
3. Run a loop until *low <= high* , the search space vanishes
4. Check if the square of mid (*mid = (low + high)/2*) is less than or equal to n, If yes then search for a larger value in second half of search space, i.e low = mid + 1.
5. Else if the square of mid is less than n then search for a smaller value in first half of search space, i.e high = mid - 1
6. Print the value of high.

Complexity Analysis:

- Time complexity: O(log n).

  The time complexity of binary search is O(log n).
- Space Complexity: O(1).

CODE:

```
1.  long long int floorSqrt(long long int x)
2.  {
3.      if(x==0)
4.          return 0;
5.      if(x==1)
6.          return 1;
7.     long long int low=0,high=x,mid;
8.     while(low<=high)
9.     {
10.            mid=(low+high)/2;
11.        if(mid*mid==x)
12.              return mid;
13.            else if(mid*mid<x)
14.              low=mid+1;
15.            else
16.              high=mid-1;
17.        }
18.        return high;
19.    }
```

Que link:[square root of a number]

KILLING SPREE

There are Infinite People Standing in a row, indexed from 1.A person having index 'i' has strength of (i*i).

You have Strength 'P'. You need to tell what is the maximum number of People You can Kill With your Strength P.You can only Kill a person with strength 'X' if P >= 'X' and after killing him, Your Strength decreases by 'X'.

Example 1:

Input:
N = 14
Output: 3
Explanation:
The strengths of people is 1, 4, 9, 16, ...
and so on. WE can kill the first 3 person ,
after which our Power becomes 0 and we cant
kill anyone else. So answer is 3

ALGORITHM:

Simple solution:

Take a variable sum intialized with 0 and i initialized with 1 and start a loop and till sum is less than or equal to the strength P. Every time sum is increased by (i*i) .

Time complexity:O(N^($\frac{1}{2}$))

BETTER SOLUTION:

Using binary search,

- Create some variables, lowerbound *low = 0*, upperbound *high = x*, where x is the given number, *mid* for the mid values .
- Run a loop until *low <= high* , the search space vanishes
- Check if the square of mid (*mid = (low + high)/2*) if sum of square of mid no.'s less than or equal to n, If yes then search for a larger value in second half oF search space, i.e low = mid + 1.
- Else if the square of mid is less than n then search for a smaller value in first half oF search space, i.e high = mid - 1
- Print the value of high.

Time complexity:O(logN)

CODE:

```
1. long long int killinSpree(long long int n)
2.   {
3.       long long int low=0,high=1e6,mid=0;
4.       while(low<=high)
5.       {
6.          mid=(low+high)/2;
7.          long long int x=(mid*(mid+1)*(2*mid+1))/6;
8.          if(x>n)
9.            high=mid-1;
```

```
10.          else
11.            low=mid+1;
12.      }
13.      return high;
14.  }
```

Que link:killing spree

MR. MODULO AND ARRAYS

Mr. Modulo lives up to his name and is always busy taking modulo of numbers and making new problems.
Now he comes up with another problem. He has an array of integers with n elements and wants to find a pair of elements {arr[i], arr[j]} such that arr[i] ≥ arr[j] and arr[i] % arr[j] is maximum among all possible pairs where 1 ≤ i, j ≤ n.

Mr. Modulo needs some help to solve this problem. Help him to find this maximum value arr[i] % arr[j].

Example 1:

Input:
n=3
arr[] = {3, 4, 7}
Output: 3
Explanation: There are 3 pairs which satisfies
arr[i] >= arr[j] are:-
4, 3 => 4 % 3 = 1
7, 3 => 7 % 3 = 1
7, 4 => 7 % 4 = 3
Hence maximum value among all is 3.

ALGORITHM:

A Naive approach is to run two nested for loops and select the maximum of every possible pairs after taking modulo of them. Time complexity of this approach will be O(n^2) which will not be sufficient for large value of n.

An efficient approach (when elements are from small range) is to use sorting and binary search method. Firstly we will sort the array so that we would able to apply binary search on it. Since we need to maximize the value of arr[i] mod arr[j] so we iterate through each x(such x divisible by arr[j]) in range from 2*arr[j] to M+arr[j], where M is Maximum value of sequence. For each value of x we need to find maximum value of arr[i] such that arr[i] < x. By doing this we would assure that we have chosen only those values of arr[i] that will give the maximum value of arr[i] mod arr[j]. After that we just need to repeat the above process for other values of arr[j] and update the answer by value a[i] mod arr[j].
Time complexity:O(nlogn+mlogm),where n is total no of elements and m is the maximum element.

CODE:

```
1.  int maxModValue(int arr[], int n)
2.  {
3.    int ans=0;
4.    sort(arr,arr+n);
5.    for(int j=n-2;j>=0;--j)
6.    {
7.       if(ans>=arr[j])
8.         break;
9.       for (int i = 2 * arr[j]; i <= arr[n - 1] + arr[j]; i += arr[j]) {
10.          // Fetch the index which is greater than or
11.          // equals to arr[i] by using binary search
12.
13.          int ind = lower_bound(arr, arr + n, i) - arr;
```

```
14.             ans = max(ans, arr[ind - 1] % arr[j]);
15.         }
16.     }
17.     return ans;
18. }
```

Que link: [Mr. Modulo and arrays]

MEDIAN IN ROW-WISE SORTED MATRIX

Given a row wise sorted matrix of size RxC where R and C are always odd, find the median of the matrix.

Input:
R = 3, C = 3
M = [[1, 3, 5],
[2, 6, 9],
[3, 6, 9]]
Output: 5
Explanation:
Sorting matrix elements gives us
{1,2,3,3,5,6,6,9,9}. Hence, 5 is median.

ALGORITHM:
Simple Method: The simplest method to solve this problem is to store all the elements of the given matrix in an array of size r*c. Then we can either sort the array and find the median element in O(r*clog(r*c)).

An **efficient approach** for this problem is to use a binary search algorithm. The idea is that for a number to be median there should be exactly (n/2) numbers which are less than this number. So, we try to find the count of numbers less than all the numbers. Below is the step by step algorithm for this approach:

Algorithm:

1. First, we find the minimum and maximum elements in the matrix.

   The minimum element can be easily found by comparing the first

element of each row, and similarly, the maximum element can be found by comparing the last element of each row.

2. Then we use binary search on our range of numbers from minimum to maximum, we find the mid of the min and max and get a count of numbers less than our mid. And accordingly change the min or max.
3. For a number to be median, there should be (r*c)/2 numbers smaller than that number. So for every number, we get the count of numbers less than that by using upper_bound() in each row of the matrix, if it is less than the required count, the median must be greater than the selected number, else the median must be less than or equal to the selected number.

Time Complexity: O(32 * r * log(c)). The upper bound function will take log(c) time and is performed for each row. And since the numbers will be max of 32 bit, so binary search of numbers from min to max will be performed in at most 32 (log2(2^32) = 32) operations.

CODE:

```
1.  int median(vector<vector<int>> &matrix, int r, int c){
2.      int min=INT_MIN,max=INT_MAX;
3.      for(int i=0;i<r;i++)
4.      {
5.          if(matrix[i][0]<min)
```

```
6.          {
7.              min=matrix[i][0];
8.          }
9.          if(matrix[i][c-1]>max)
10.         {
11.             max=matrix[i][c-1];
12.         }
13.     }
14.     int desired=(r*c+1)/2;
15.     while(min<max)
16.     {
17.         int mid=(min+max)/2;
18.         int place=0;
19.         for(int i=0;i<r;i++)
20.
    place+=(upper_bound(matrix[i].begin(),matrix[i].end(),mid)-matrix[i].be
    gin());
21.     if(place<desired)
22.          min=mid+1;
23.         else
24.          max=mid;
25.         }
26.         return min;
27.     }
```

Que link:[median in a row-wise sorted matrix]

NINE DIVISORS

Find the count of numbers less than equal to N having exactly 9 divisors.

Example 1:

Input:
N = 100
Output:
2
Explanation:
The two numbers which have
exactly 9 divisors are 36 and 100.

ALGORITHM:

A naive approach is to iterate for all numbers till N and count the numbers that have exactly 9 divisors. For counting number of divisors, one can easily iterate till N and check if N is divisible by i or not and keep a count.
TIME COMPLEXITY:O(N^2)

An **efficient approach** is to use the prime factors property to count the number of divisors of a number. The method can be found here. If any number(let x) can be expressed in terms of (p^2 * q^2) or (p^8), where p and q are prime factors of X, then X has a total of 9 divisors. The below steps can be followed to solve the above problem.

1. Use Sieve technique to mark the smallest prime factor of a number.

2. We just need to check for all the numbers in **range[1-sqrt(n)]** that can be expressed in terms of p*q since **(p^2*q^2)** has 9 factors, hence **(p*q)^2** will also have exactly 9 factors.
3. Iterate from 1 to sqrt(n) and check if i can be expressed as **p*q**, where p and q are prime numbers.
4. Also check if i is prime then **pow(i, 8)<=n** or not, in that case count that number also.
5. The summation of count of numbers that can be expressed in form p*q and p^8 is our answer.

TIME COMPLEXITY:O(sqrt(N))

CODE:

```
long long int nineDivisors(long long int N){
    long long int c=0,limit;
    limit=sqrt(N);
    long long int prime[limit+1];

   for(long long int i=1;i<=limit;i++)
   {
      prime[i]=i;
   }
   for(long long int i=2;i*i<=limit;i++)
   {
      if(prime[i]==i)
      {
         for(long long int j=i*i;j<=limit;j+=i)
         {
```

```
            if(prime[j]==j)
              prime[j]=i;
          }
      }
      }
  for(long long int i=2;i<=limit;i++)
      {
         long long int p=prime[i];
         long long int q=prime[i/prime[i]];
         if(p*q==i && q!=1 && p!=q)
            c+=1;
         else if(prime[i]==i)
         {
            if(pow(i,8)<=N)
             c+=1;
      }
      }
      return c;
      }
```

Que link: [nine divisors]

Empty The Tank

Given a tank with capacity C litres which is completely filled in starting. At the end of every day, tank is filled with L litres of water and in the case of overflow extra water is thrown out. Now on i-th day i litres of water is taken out for drinking. We need to find out the day at which tank will become empty the first time.

Input: C = 5, L = 2
Output: 4
Explanation:

- At the start of 1st day, water in tank = 5 and at the end of the 1st day = (5 - 1) = 4
- At the start of 2nd day, water in tank = 4 + 2 = 6 but tank capacity is 5 so water = 5 and at the end of the 2nd day = (5 - 2) = 3.
- At the start of 3rd day, water in tank = 3 + 2 = 5 and at the end of the 3rd day = (5 - 3) = 2.
- At the start of 4th day, water in tank = 2 + 2 = 4 and at the end of the 4th day = (4 - 4) = 0. So final answer will be 4.

Input: C = 6, L = 1
Output: 4
Explanation:

- At the start of 1st day,water in tank = 6 and at the end of the 1st day = (5 - 1) = 5.
- At the start of 2nd day, water in tank = 5 + 1 = 6 and at the end of the 2nd day = (5 - 2)= 3.
- At the start of 3rd day, water in tank = 3 + 1 = 4 and at the end of the 3rd day = (4 - 3) = 1.

- At the start of 4th day, water in tank = 1 + 1 = 2 and at the end of the 4th day = (2 - 4) = 0. So final answer will be 4.

ALGORITHM:

NAIVE APPROACH : Time complexity : O(C)

Run the loop for Capacity > 0;

- At the start of day L water is added so Capacity = Capacity + L ;
- Condition of overflow : if Capacity > C ,then Capacity = C;
- At the end of day i water is taken out so Capacity = Capacity - i

Lastly, return i

EFFICIENT APPROACH (using binary search): Time Complexity: O(log C)

We can see that tank will be full for starting (l + 1) days because water taken out is less than water being filled. After that, each day water in the tank will be decreased by 1 more liter and on (l + 1 + i)th day (C – (i)(i + 1) / 2) liter water will remain before taking drinking water.

Now we need to find a minimal day (l + 1 + K), in which even after filling the tank by l liters we have water less than l in tank i.e. on (l + 1 + K – 1)th day tank becomes empty so our goal is to find minimum K such that,
C – K(K + 1) / 2 <= l

We can solve above equation using binary search and then (l + K) will be our answer. Total time complexity of solution will be O(log C)

CODE:

```
1.  long long int minDaysToEmpty(long long int C, long long int l) {
2.      // code here
3.      if(C <= l)
4.          return C;
5.
6.      //for l days there is overflow condition so, tank is full
7.      long long low = 0;
8.      long long high = 1e5;
9.
10.     while(low <= high)
11.     {
12.         long long mid = low + (high- low)/2;
13.         if(C - l - (mid * (mid + 1))/2  <= 0)
14.             high = mid - 1;
15.         else
16.             low = mid + 1;
17.     }
18.     return l + low;
19.
20.     }
```

QUES LINK : [Empty The Tank]

Equalize the Towers

Given heights h[] of N towers, the task is to bring every tower to the same height by either adding or removing blocks in a tower. Every addition or removal operation costs, cost[] a particular value for the respective tower. Find out the Minimum cost to Equalize the Towers.

Input: N = 3, h[] = {1, 2, 3}
cost[] = {10, 100, 1000}
Output: 120
Explanation: The heights can be equalized by either
"Removing one block from 3 and adding one in 1"
or
"Adding two blocks in 1 and adding one in 2".
Since the cost of operation in tower 3 is 1000, the first process would yield 1010 while the second one yields 120. Since the second process yields the lowest cost of operation, it is the required output.

ALGORITHM:
NAIVE APPROACH : Time complexity : $O(N^2)$
Find the cost of each and every possible operation set, and then find the minimum cost of operation.

EFFICIENT APPROACH (binary search): Time Complexity: O(Nlog N)

1. Find the minimum and maximum height in the h[] and set them as the low and high value respectively.
2. Create a variable ans which will keep track of minimum cost during the iteration.

3. Run a loop if low <= high .
4. For every iteration, find the mid (mid = low + (high - low)/2).
5. Find the
    a. cost of operation to equalise all the towers to height mid and store in variable Cost
    b. cost of operation to equalise all the towers to height (mid - 1) and store in variable Lcost. (do this operation only if mid > 0)
    c. cost of operation to equalise all the towers to height (mid + 1) and store in variable Ucost.
6. Update ans variable to Cost, if ans > Cost
7. Check if the Ucost <= Cost; search in right half i.e. low = mid + 1
8. Check if the Lcost <= Cost; search in left half i.e. high = mid - 1
9. Else (Lcost > Cost and Ucost > Cost) return Cost
10. Return ans

CODE:

```
// to find cost of operation
long long utility(int h[], int cost[], int n, int x)
{
    long long ans = 0;
    for(int i = 0; i < n; i++)
        ans += ( abs(h[i] - x) * cost[i] ) ;
    return ans;
}

long long int Bsearch(int n, int h[], int cost[])
{
```

```
12.         if(n == 1)
13.             return 0;
14.
15.         int low = *min_element(h, h + n);
16.         int high = *max_element(h, h + n);
17.         long long ans = LLONG_MAX;
18.
19.         while(low <= high)
20.         {
21.             int mid = low + (high - low)/2;
22.             long long Cost = utility(h, cost, n, mid);
23.             long long Lcost = mid>0 ? utility(h, cost, n, mid - 1) :
    LLONG_MAX;
24.             long long Ucost = utility(h,cost,n,mid + 1);
25.
26.
27.                 //being done so that if it doesn't fall into
28.                 //else block we can return the minimum cost
29.             if(ans >= Cost)
30.                 ans = Cost;
31.
32.
33.             if( Cost >= Ucost)
34.                 low = mid + 1;
35.             else if( Cost >= Lcost )
```

```
36.             high = mid - 1;
37.         else  //if( Cost < Ucost && Cost < Lcost )
38.             return Cost;
39.     }
40.     return ans;
41.}
```

QUES LINK : [Equalize the Towers]

Find the closest number

Given an array of unsorted integers. The task is to find the closest value to the given number in array. Array may contain duplicate values.
Note: If the difference is same for two values print the value which is greater than the given number.

Example 1:

Input : Arr[] = {1, 3, 6, 7} and K = 4.
Output : 3
Explanation:
We have an array [1, 3, 6, 7] and target is 4. If we look at the absolute difference of target with every element of an array we will get
[|1-4|, |3-4|, |6-4|, |7-4|].
So, the closest number is 3.

ALGORITHM:
Naive Approach: O(n)
Traverse the complete array and find the absolute difference between the element of array and target (K) . Now return the minimum of those differences.
Using binary search,

- Sort the array
- Assign low = 0 and high = n-1
- Run a loop until low <= high .
- Find the mid element mid = low + (high - low)/2

- Check if the mid element == target, return target (this means the closest element to K is K itself because the difference is 0).
- If target > mid , search in right half i.e. low = mid + 1
- Else (target < mid) search in left half i.e. high = mid - 1
- Here, we observe at the end of loop, high will be pointing to the maximum element less than target and low will be pointing to the minimum element greater than target,
- Now, find which of the above two elements are closer, if both are at equal distance return the greater one.

Time complexity:O(logN)

CODE:

```
1. int findClosest(int arr[], int n, int target)
2. {
3.   sort(arr, arr + n);
4.   if(target <= arr[0])
5.       return arr[0];
6.   if(target >= arr[n-1])
7.       return arr[n-1];
8.
9.   int low = 0;
10.    int high = n-1;
11.    int mid;
12.
13.    while(low <= high)
14.    {
15.        mid = low + ( high - low)/2;
16.        if(arr[mid] == target)
```

```
17.             return arr[mid];
18.
19.         else if ( arr[mid] < target)
20.             low = mid + 1;
21.         else
22.              high = mid - 1;
23.     }
24.
25.     //high is number just less than target
26.     //and low is just greater than target
27.     return (abs(arr[low] - target) <= abs(arr[high] -
  target)) ? arr[low] : arr[high];
28.   }
```

PRACTICE:

[N trailing zeros in factorials]