Must Do **Searching and sorting** problems for **Placement**

Searching & Sorting	**Find first and last positions of an element in a sorted array**
Searching & Sorting	Find a Fixed Point (Value equal to index) in a given array
Searching & Sorting	Search in a rotated sorted array
Searching & Sorting	square root of an integer
Searching & Sorting	Maximum and minimum of an array using minimum number of comparisons
Searching & Sorting	Optimum location of point to minimize total distance
Searching & Sorting	Find the repeating and the missing
Searching & Sorting	find majority element
Searching & Sorting	Searching in an array where adjacent differ by at most k
Searching & Sorting	find a pair with a given difference
Searching & Sorting	find four elements that sum to a given value
Searching & Sorting	maximum sum such that no 2 elements are adjacent
Searching & Sorting	Count triplet with sum smaller than a given value
Searching & Sorting	merge 2 sorted arrays
Searching & Sorting	print all subarrays with 0 sum
Searching & Sorting	Product array Puzzle
Searching & Sorting	Sort array according to count of set bits
Searching & Sorting	minimum no. of swaps required to sort the array
Searching & Sorting	Bishu and Soldiers

Searching & Sorting	Rasta and Kheshtak
Searching & Sorting	Kth smallest number again
Searching & Sorting	Find pivot element in a sorted array
Searching & Sorting	K-th Element of Two Sorted Arrays
Searching & Sorting	Aggressive cows
Searching & Sorting	Book Allocation Problem
Searching & Sorting	EKOSPOJ:
Searching & Sorting	Job Scheduling Algo
Searching & Sorting	Missing Number in AP
Searching & Sorting	Smallest number with atleastn trailing zeroes infactorial
Searching & Sorting	Painters Partition Problem:
Searching & Sorting	ROTI-Prata SPOJ
Searching & Sorting	DoubleHelix SPOJ
Searching & Sorting	Subset Sums
Searching & Sorting	Findthe inversion count
Searching & Sorting	Implement Merge-sort in-place
Searching & Sorting	Partitioning and Sorting Arrays with Many Repeated Entries

Searching And Sorting

Ques → First and last occourance of x

Given a sorted array arr, containing n element with duplicate elements, task is to find out indexs of first and last occourance of element x.

Example : {1, 3, 5, 5, 5, 5, 8, 9, 10, 16} x = 5

O/P → 2 5

Intution :-

→ We use binary search in given array & arr+n for x.

→ If present return ind1 ind2 else {-1, -1};

→ 2 variable for lower & upper bound return them both

Code :-

bool isPresent = binary_search(arr, arr+n, x);

(1) if (!isPresent) return {-1, -1}

int ind1 = lower_bond(arr, arr+n, x) - arr;
ind2 = upper_bound(arr, arr+n, x) - arr - 1;

return {ind1, ind2};

(3)

Date: / / Page no:

Ques → Value equal to index value

Given arr of +ve int, find element whose value is equal to that of its index value

Example :

I/P = { 15, 2, 45, 12, 6 } O/P → 2.

Intuition :

if inside a loop,

if (i == arr[i]) return arr[i].

else

Code :-

```
vector<int> valueEqualIndex(int arr[], int n)
{
    vector<int> ans;
    for(i=1; i<=n)
    {
        if(i == arr[i-1])
            ans.push_back(i);
    }

    return ans;
}
```

Ques→ Search in a rotated sorted array

array[] sorted, prior to being passed to your func^n, nums is possibly rotating at unknown pivot index, k (1 <= k < length) such that resulting array [num[k], num[k+1] ---- num[n-1], num[0], num[1] ---, num[k-1]]

Example → [0,1,2,3,4,5,6] → rotate → [4,5,6,7,0,1,2]

I/P - [4, 5, 6, 7, 0, 1, 2] (0 1 2 3 4 5 6) target = 0

O/P → 4 (if not present return -1)

Intution :- (Binary Search)

Need to modify binary search, plan is to create a recursive func^n that takes l and r as range in input and the key.

Algo :-

1) find middle point mid = (l+h)/2
2) If key is present at middle point, return mid.
3) else if arr[l...mid] is sorted
   - If key to be searched lies in range from arr[l] to arr[mid], recurr for [l...mid]
   - else recurr for arr[mid+1...h]
4) Else (arr[mid+1...h] must be sorted)
   - if key to be searched lies in range from arr[mid+1] to arr[h], recurr for arr[mid+1...h]
   - else recurr for arr[l...mid]

Date: 14/2/22 Page no:

Ques → Middle of Three

Ques →

Given 3 no., A, B, C. find the middle no.

Example: A = 978 B = 518 C = 300

O/P - 518

since 518 > 300 (C) & 518 < 978 (A)

(518 = B)

Intution :-

Sum(A+B+C) - [max(A,B,C) - min(A,B,C)].

∵, Overall sum = Circular Range.

middle → (max - min) from sum.

Code :-

```
int middle (int A, int B, int C)
{
return (A+B+C) - max({A,B,C}) - min({A,B,C});
}
```

Ques → Find missing and repeating

unsorted array arr of size N of +ve integers. 1 number 'A' from set {1,2,3,...N} is missing and 1 no. 'B' occurs twice in array.
Find these 2 no.

Example :- arr[] = {2,2,3}
O/P → 2 1 → 2 is repeating no. & 1 is missing no.

* Intution :- (using count array)
Create temp array temp[], initial value is 0.
Traverse input array arr[], and do following for each arr[i].
- if (temp[arr[i]] == 0) temp[arr[i]] = 1;
- if (temp[arr[i]] == 1) output "arr[i]" repeat two this.

Traverse temp[] and output the array element having value as 0 (it is the missing element)
(TC - O(N) SC - O(N))

Ques- Find majority element

Given array A, find majority element in array. (size)

majority: any element appr more than N/2 times

Example: N=3, {1, 2, 3}, O/P = -1

N = 5, {3, 1, 3, 3, 2} => O/P = 3 (3 times)

Intution: (using moore Voting Technique).

→ 1st step is to check majority elem. if it is, then definately return majority element else return candidate for maj. ele.

→ check obtained elem. is maj. or not.

Algo.:- (TC-O(N))

1) loop trough each element & maintain count of majority element, and majority index => maj_index
2) If next ele. is same then, Inc count if next ele. is not same, then dec count.
3) if count reach 0 then, changes maj_index to curr. element & chang count to 1.
4) Again traverse through array & find count of majority element found.
5) If count > than half N/2, print element
6) else print -1.

Date: / / Page no:

Snippet :-

Step (1)

```
int findCandidate (int a[], int size)
{
    int maj_index = 0, count = 1;
    for(i=1 ; i<size) {
        if (a[maj_index == a[i])
            count++;
        else
            count--;
        if (count == 0) {
            maj_index = i;
            count = 1;
        }
    }
    return a[maj_index]
```

Step (2)

```
isMajority (int a[], int size, int cand)
{
    int count = 0;
    for (i=0; i<size(); i++) {
        if (a[i] == cand) count++;
    }
    if (count > size/2)
        return 1;
    else
        return 0;
}
```

Date 15/2/22 Page no.

Ques → Search in an array where adjacent differ by at most 'k'.

Step array is an array of int where each element has difference of atmost k.
Given a key 'x' find index value of 'x' if xpt ele. exist to return the 1st occourance of key.

* Example

arr[] = {4, 5, 6, 7, 6} (0 1 2 3 4) k = 1 x = 6

O/P = 2 // 1st index of 6 is 2.

* arr[] = {20, 40, 50, 70, 70, 60} (0 1 2 3 4 5) k = 20 x = 60.

O/P - 5 // 1st index of 60 is 5.

Intution :-

The diff. b/w all adj. ele is atmost K. Start compare leftmost ele. find the difference b/w curr. ele & 'x'. Let differce be 'diff'.

We know that x atleast 'diff/k' away, so instead of searching 1 by 1, we jump 'diff/k'.

* Code :-

```
int search (int arr[], int n, int x, int k)
{
    int i = 0;
    while (i < n)
    {
        if (arr[i] == x)
            return i;

        i = i + max (1, abs(arr[i] - x)/k);
    }
    cout << "not present";
    return -1;
}
```

Date: / / Page no:

Ques → Find a pair with given difference

Ques

Given unsorted array & no. 'n', find if their exist pair of element in array whose difference is n.

Example: arr[] = {5, 20, 3, 2, 50, 80} n = 78

O/P → (2, 80) ⇒ 80 - 2 = 78 = n.

Intuition :-

→ Hashing can be used. Create empty hash table as HT. Traverse in array. use array element as hash key & enter in HT.

→ Traverse array again look for value n+arr[i] in HT.

* Implementation :- bool findPair(arr[], size, n)

{

unordered_map<int, int> mpp;

for(int i=0; i<size)

mpp[arr[i]]++;

for(i=0; i<size) {

if (mpp.find(n+arr[i]) != mpp.end())

cout<<" found "<<arr[i]<<", "<<n+arr[i];

return true

}

}

cout<<" Not found ";

return false;

}

Date: 16/12/22 Page no.:

Ques- Find all four sum numbers

Given array int and another number. find all unique quadruple from given array that sums up to given number.

Example :- N = 5 K = 3 A[] = {0, 0, 2, 1, 1}.

O/P - 0 0 1 2 $ sum of 0, 0, 1, 2 is 3 = K.

Intution :- (Hashing Based solution ($O(n^2)$).

1) Store sums of all pair in hash table.
2) Traverse through all pairs again & search for x (curr pair sum) in hash table.
3) If pair is found with required sum, then make sure that all elements are distinct array elements and an element is not considered more than 1.

Pseudo code :- findElement(arr[], int n, int x).

```
{   unordered_map < int, pair < int, int > > mp;
    for ( i=0; i<n-1 )
{   for (int j = i+1; j<n)  mp[arr[i]+arr[j]] = {i,j};
    for (int i=0; i<n-1; i++) {
        for (j = i+1; j<n; j++)  for (j = i+1; j<n; j++)
sum =  if  arr[i] + arr[j];
        if (mp.find (x - sum) != mp.end()) {
pair <int, int> p = mp[x-sum];
    if ( p.first != i && p.first != j && p.second != i && p.sec != j)
{   cout << " arr[i] << "," << arr[j] << "," << arr[p.first]
    << "," << arr[p.second];
    return;
}
```

Date: / / Page no:

Ques → Stickler Thief

Given an array of +ve no., find max. sum of sub sequence with the constraints that no 2 num. in seq. should be adjacent in array.
So, 3 2 7 10 should return 13.
(3 + 10) OR 3 2 5 7 10 (3, 5, & 7) return 13. Ans
Answer in most efficient way.
Example : arr[] = { 5, 5, 10, 100, 10, 5 }. O/P - 110.

Intuition :-
Loop for all elem. in arr[], maintain 2 sums, incl & excl where, incl = max-sum including the previous elem. & excl = Max_Sum excluding the previous element.
Max_Sum excluding the current element will be max (incl, excl) and max-sum including current element will be excl + curr-ele.
At the end loop return max of incl and excl.

Code :-
```
find MaxSum ( Vector<int> arr, int n )
    int incl = arr[], excl = 0, excl_new, i;
For ( i = i ; i < n ; i++)
    excl_new = (incl > excl) ? incl : excl;
    incl = excl + arr[i];
    excl = excl_new ;
return ( (incl > excl) ? incl : excl);
```

Date 17/2/2022

Ques → Count triplet with sum smaller than x.

Given array, size, & sum, find the count of triplet (i, j, K) having (i<j<K) with sum of (arr[i] + arr[j] + arr[K]) smaller than given value sum.

Example - N = 5, sum = 12. arr[] = {5, 1, 3, 4, 7}.

O/P - 4. : (1,3,4), (1,3,5), (1,3,7), (1,4,5).

Intution :-

1) Sort in rising order, & initialize res = 0.

2) Loop i=0 to n-2, to find all triplet with arr[i] as first element.

◎ Initialize other two element as corner element of subarray arr[i+1, ..., n-1], i.e., j = i+1 & K = n-1.

◎ Move j & K toward each other until they meet, i.e., while (j<K) : (i) if arr[i] + arr[j] + arr[K] >= sum then K--.

else for current i & j, there can (K-j) possible 3rd element that satisfy the constraint

(ii) else do ans += (K-j) followed by j++.

```
Code :-  sort(arr, arr+n), int ans = 0;
for( i=0; i<n-2; i++) {
    int j = i+1, K = n-1;
    while (j<K) {
        if (arr[i] + arr[j] + arr[K] >= sum)
            K--;
        else {
            ans += (K-j); j++;
        }
    }
    return ans;
}
```

Date: / / Page no:

Ques- Merge 2 sorted array, without extra space

Given sorted arrays arr1[] & arr2[], of N, M each, merge 2 array into 1 sorted array without using extra space.

Example: N = 4 , M = 5

arr1[] = {1, 3, 5, 7}, arr2[] = {0, 2, 6, 8, 9}

o/p -> 0, 1, 2, 3, 5, 6, 7, 8, 9

Intution :-

1) Create swap of 2 pointer *a & *b.

2) write if while (start <= end and j < m).

then chick for arr1[start] > arr2[j]

- swap(&arr1[end], &arr2[j]);

end-- & j++.

- else start++;

Now sort both the array till its size

Code :-

```
void swap(int* a, int* b)
{ int swap(a,b); }
void merge(arr1, arr2, n, m)
{
start=0; end=n-1; j=0;
while (start <= end and j<m)
{
if (arr1[start] > arr2[j])
{ swap(&arr1[end], &arr2[j]);
end--; j++; }
else
start++;
}
sort(arr1, arr1+n);
sort(arr2, arr2+m);
}
```

Ques-

18/2/22

Ques- Zero sum subarray

Given array [] of N size, find total count of sub array having their sum equal 0.

Example :- n = 6 arr[] = {0, 0, 5, 5, 0, 0}

O/P → 6 → [0], [0], [0], [0], [0, 0] & [0, 0].

| 4 | 2 | -3 | 1 | 6 | k = 4

Intution :-

BF - check for every sub-arr if sum = 0, return

Optimal - (i) find prefix sum [4, 6, 3, 4, 10] 4 repeat 2 time. if we are getting any k value 2 or more time then it is sum subarray

using Hash map - maintain sum of element encountered so far in variable say sum.

if sum = 0, found it. Starting from 0 index till curr.

Check if current sum exist in hash table or not / if curr sum already exist in hash table, then this sum is sub array.

Subtract array ele. arr[0] to arr[i]. Same sum obtain.

Insert current sum into hash table.

Dry Run :-

| 6 | 3 | -1 | -3 | 4 | -2 | 2 |

sum = 0, map (empty) vector (empty)

	0	1	2	3	4	5	6
step 1 - arr	6	3	-1	-3	4	-2	2

sum = 6

map: 6 → 0

Step - 2

0	1	2	3	4	5	6
6	3	-1	-3	4	-2	2

Sum = 9

map

6 → 0
9 → 1

Step - 3

0	1	2	3	4	5	6
6	3	-1	-3	4	-2	2

Sum = 8

map

6 → 0
9 → 1
8 → 2

Step - 4

0	1	2	3	4	5	6
6	3	-1	-3	4	-2	2

Sum = 5

map

6 → 0
9 → 1
8 → 2
5 → 3

Step - 5

0	1	2	3	4	5	6
6	3	-1	-3	4	-2	2

Sum = 9

vector {(2, 4)}

map

6 → 0
9 → 1, 4
8 → 2
5 → 3

Step - 6

arr

0	1	2	3	4	5	6
6	3	-1	-3	4	-2	2

Sum = 7

vector {(2, 4)}

map

6 → 0
9 → 1, 4,
8 → 2
5, → 3
7 → 5

Step - 7

0	1	2	3	4	5	6
6	3	-1	-3	4	-2	2

Sum = 9

vector {(2, 4), (2, 6), (5, 6)}

map

6 → 0
9 → 1, 4, 6
8 → 2
5 → 3
7 → 5

Ques→ Product Array Puzzle

given array nums[], construct product array 'P', size is. : P[i] = product of all element of num except num[i].

Example : n=5

num[] = {10, 3, 5, 6, 2} o/p- 180, 600, 360, 300, 900

ex → 3*5*6*2 = 180 ; 10*5*6*2 = 600 , and so on...

Intution :-

2 int i & j for loop. i will track position of our cursor and j will do product calc. as:

if (i != j) product * num[j];

code :- vector <long long int> v;

for(i=0; i<n)

(1) long long int product = 1;

for(int j = 0; j < n; j++)

[2]

if (i != j)

product* = num[j];

[3]

v.push_back(product);

(3)

return V;

Ques → Sort by set bit count

Ques-

given array, sort the array (in descending order) acc. to count of set bits in binary representing of array

Example :

arr[] = {5, 2, 3, 9, 4, 6, 7, 15, 32}

15 - 1111, 7 - 0111, 5 - 0101, 3 - 0011, 9 - 1001, 6 - 0110, 2 - 0010, 4 - 0100, 32 - 100000.

O/P - {15}, {7}, {5, 3, 9, 6}, {2, 4, 32}.

Intution :-

create an array and store set-bit count of all int in aux-array.

Simultaneously sort both array acc to non-increasing order of auxiliary array.

Code :-

```
SortBySetBit (int arr[], int n)
{
    int aux[n];
    for (int i=0; i<n; i++)
    { aux[i] = countBits(arr[i]);
    insertionSort(arr, aux, n);
}
```

in insertion sort()

```
for (i=1 → i<n) { int k1 = aux[i];
k2 = arr[i]; int j = i-1;
while (j>=0 && aux[j] < k1) {
aux[j+1] = aux[j]; arr[j+1] = arr[j];
    j = j-1; }
aux[j+1] = key1; arr[j+1] = k2; } }
```

19/2/2022

Ques- Minimum swap to sort

Given array, find mini. no. swaps to sort in using order.

Example:
num = {2, 8, 5, 4} → 8↔4 → 2, 4, 5, 8

Intuition:
1) Take an empty vector & sort it, say b[].
2) Take a map to track the count, and enter array values in map.
3) Iterate a loop in which if (nums[i] != b[i])
4) swap pos = m[b[i]] with num[i], and use cnt++ till dO for all elements & return count → cnt in end

code:- minSwap (vector<int> & num) {
n = size, vector<int> b = nums;
sort (b.begin(), b.end())
map<int, int> m; for(int i=0; i<n; i++) m[num[i]]=
int cnt=0;
for(i=0 ; i<n) {
if (num[i] != b[i]) {
int pos = m[b[i]];
num[pos] = num[i];
m[num[i]] = pos;
num[i] = b[i];
m[num[i]] = i;
cnt ++;
}
}
}

Ques → Max. no. of people that can be killed with strength P

Infinit no. of people standing in row, from 1 to ∞
i person of order i has strength i^2. You've strength P, & you've to tell what is the max. no. of you can kill with strength P.
You can kill a person with strength x if $P \geq x$ & after kill strength lee by x.
Example :-
P = 14, 1st person has str. $1^2 = 1$ ⇒ $1 > P$
⇒ 14 - 1 = 13 (P) ; 2nd person ⇒ 13 - 4 = 9
; 3rd person ⇒ 9 - 9 ≤ kill = 3

Ans → 3

* Intuition :- TC (O(log N)) - SC - O(1)
We can use binary search.
Start binary search by considering low as 0 and high 10^{15}. We will calculate mid value, we will & using mid value we will change position of low and high.

as- mid = low + (high - low)/2.
then, (n * (n+1) * (2 * n + 1)) / 6 ;

Code :-

low = 0; high = 1000000L; ans = 0L;

while (low <= high)

(2) // to calculate mid value

long mid = low + (high - low) / 2;

long value = (n * (n+1) * (2 * n+1)) / 6;

if (value <= n) // compare value with n

{ [2] ans = mid;

low = mid + 1;

[3]

else

high = mid - 1;

(3)

return ans;

Date 20/2/22

Ques → Find K^{th} Smallest element in given n range

Given n & q, n = no. of range & q = no. of queries, find K^{th} smallest element for each query (let $K > 1$)
print K^{th} smallest ele. if exist else -1.

Example :-

I/P : arr[] = {{1,4}, {6,8}} ; queries = {2, 6, 8}

O/P - 2 7 -1

The no. becomes → 1 2 3 4 6 7 8 (0 1 2 3 4 5 6) → 2^{nd} element is 2.
6^{th} element is 7, 10 not exist so -1.

★ Intution :- (Merge Overlapping Intervals) TC - O(n log(n))

1) First we'll merge overlapping intervals & keep all intervals sorted in ascending order of start time.
2) after merges in merged[], we use linear search for K^{th} smallest element.

★ Implementation :- kthSmall (vector< interval > merge, int K)

```
{
    n = merge.size();
for (j=0 → j<n) {
  if (k <= abs(merge[j].e - merge[j].s + 1))
      return (merge[j].s + k - 1)

    k = k - abs(merge[j].e - merge[j].s + 1);
}
if (k)
    return -1.
}
```

Ques → Find minimum element in sorted and rotated array

Sorted array is rotated at unknown point, find min. element in it.

Example :- { 5, 6, 1, 2, 3, 4 } O/P - 1.

{ 1, 2, 3, 4 } O/P - 1. (∵ it is minimum)

* Intution :- (Binary Search - TC - O(log(n))

→ Mini. element is the element whose prev. ele. is greater than it. if no prev. ; then no rotation, (then 1st element is minimum).

→ Check this condt" for middle element by compairing with (mid-1)th term and (mid+1)th term.

→ If minimum ele. not at middle :- {

Then it lies either left half or right half

(1) if the middle element is smaller than last element, the mini. lies in LH.

(2) else, lies in Right half (RH).

* Implementation :- Find Min (int arr, low, high)

if (high < low) return arr[0];

if (high == low) return arr[low];

int mid = low + (high - low) / 2;

// to find mid.

// now we check if ele. (mid+1) is mini. elem.

if (mid < high && arr[mid+1] < arr[mid])
return arr[mid+1];

// check if mid itself is mini ele.
if (mid > low && arr[mid] < arr[mid-1])
return arr[mid];

// Decide whether we need to go LH or RH
if (arr[high] > arr[mid])
return findMin(arr, low, mid-1);
return findMin(arr, mid+1, high);

Ques:- K^th element of 2 sorted array

Given 2 sorted array and it's size of n1 & n2 respectively. Find element that would be at k^th position of final sorted array.

Example:

arr1[] = {2, 3, 6, 7, 9} arr2[] = {1, 4, 8, 10} k = 5

k = 5 ⇒ final sorted array = 1 2 3 4 6 7 8 9 10
0 1 2 3 4 5 6 7 8

O/P- 6

Intution:- (use upper bound STL)

Example - A1 - 2 3 6 7 9 , A2 - 1 4 8 10

k = 5 ans - 6.

→ 5^th ele. is 6, & 1^st elem. is 1. Notice here that upper_bound(6) gives 5, upper_bound(4) gives 4, it is the no. less than or equal to no. we are given as input to upper_bound().

→ using upper_bound to check what is position of element in sorted array.

upper_bound func^n returns pointer to ele. which is ≥ than the element we searched.

→ To find k^th element, need to just find upper_bound().

→ If we have given 2 array, we just need to sum of upper_bound for 2 array.

* I/P- 1- 2 3 6 7 9 2- 1 4 8 10 k = 5

value of upper_bound for value (6) in arr1 = 3

and array2 is 2.

Total = 5 which is ans.

Algo :-

1) We take a mid between [L, R] using mid = (L+R)/2
2) Check if mid can be K'th element using upper_bound() func^n.
3) Find the sum upper_bond() for both the array and if sum is >= K, its possible value of K'th ele.
4) if sum is >= K, then R = mid - 1.
5) else if sum < K, then curr. mid is too small & we assign L = mid + 1.
6) Repeat from top.
7) Return smallest value found

Date 21/2/22 Page No.

Ques- Aggressive Cow (spoj)

Given array of N length, which denote pos of stalls. Now N = no. of stalls & K = no. of aggressive cow. To prevent cow from hurting, you need to assign cow to stall, such that, minimum dist b/w 2 of them is as large as possible. Find largest minimum dist.

Example :-

arr : 1, 2, 4, 8, 9 & K = 3. Ans - O/P - 3.

1st cow @ stall 1, 2nd cow at stall 4, 3rd cow @ stall 8

* Intution :-

We need to define isPossible() that check if dist. x is possible between each of the cow. We use greedy approch, by placing cows at leftmost possible stall such that, atleast x dist away from last placed cow.

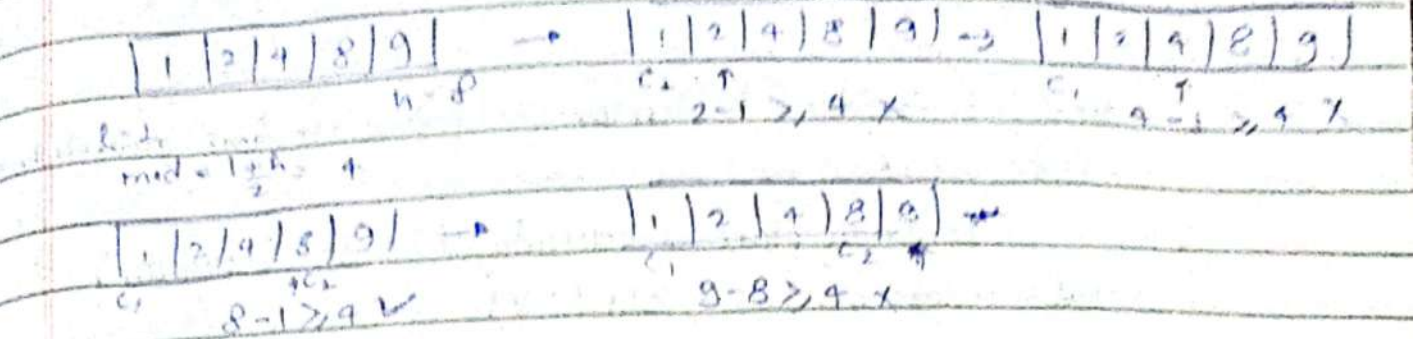
We need to sort given array / list, so once we have our sorted array to do greedy task, then we apply Binary search, on sorted input, & use isPossible() to find largest distance

Binary Search :-

Firstly, consider low & high starting & end pos low = 0 & high = end pos^.

Let mid be minimum dist., check in sorted array, how many position available of that mini. dist. b/w them is mid.

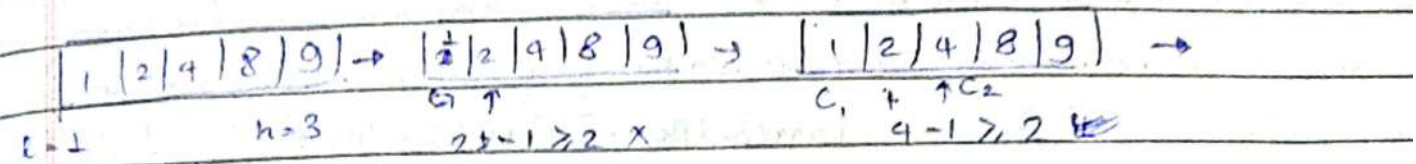
Dry Run arr = 1, 2, 4, 8, 9 low: l, high: h, mid.

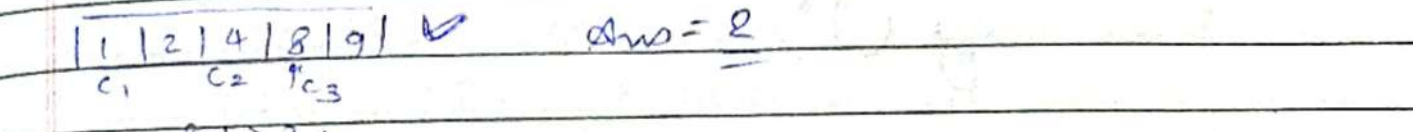


l: 1
mid = (l+h)/2 = 4

Now we could only place 2 cows not 3, so return false as '4' dist. btw cows is not possible, it has to be less than 4 so high = mid - 1 = 4 - 1 = (3)

Now, low = 1, high = 3, so mid = (l+h)/2 = 2.

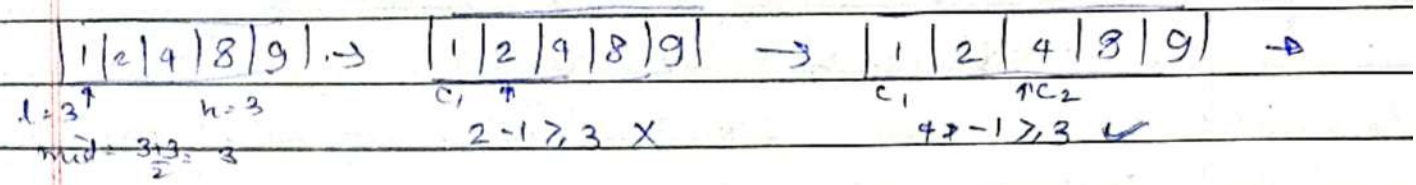


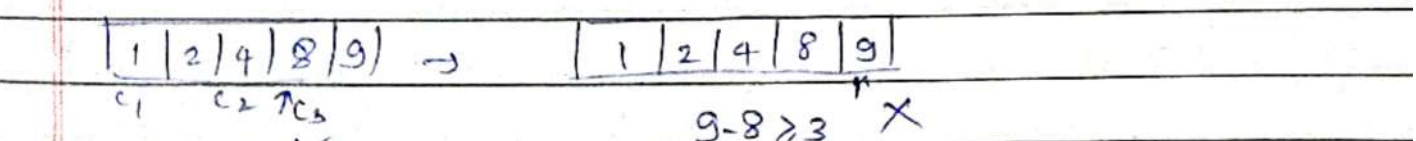


Now, we know, one possible ans is 2. ∴, we need to maximise mini. ans. so try to find for dist > 2.

so, low = mid + 1, 2 + 1 = 3.

low = 3, high = 3, mid = 3.





Ans = 1, 4, 8 ⇒ (3)

* isPossible() function

```
bool isPossible(vector<int>& stall, int minDist, k)
    int cow = 1;
    // already place 1st available slot (greedy)
    int lastCowPos = stall[0];

    for (int i=0; i<stall.size(); i++)
        if (stall[i] - lastCowPos >= minDist)
            cow++;
            lastCowPos = stall[i];
            if (cow >= k) return TRUE;

    return false;
```

Ques:- Allocate minimum number of pages

Given no. of pages, in m diff. books and n student. Book arranged in ascending order of no. of pages. Every student assigned to read some consecutive book. Task is to assign books in such a way, max no. of pages assigned to student is minimum.

Example:- Pages[] = {12, 34, 67, 90} m = 2

Ans - O/P - 113.

2 no. of student, books can be distributed as:

(1) [12] and [34, 67, 90], max pages allocate to Studn 2 with 34+67+90 = 191 page

(2) [12, 34] and [67, 90], max page allocate to studn 2 is, 67 + 90 = 157 pages

(3) [12, 34, 67] and [90] max page allocat to studn 2, 12 + 34 + 90 = (113 pages) ↓ minimum no. of pages

Intution :- (Binary search)

→ We fix value for no. of pages as mid of current minimum & maximum.

→ Initialize, minimum = 0 & maximum = sum of all pages.

→ If curr mid is sol^n, then search LH, else High Half (HH)

How to check mid value is feasible or not

* Basically, we need to check, if we can assign pages to all student in a way max. no. doesn't exceed curr. value. To do this,

* We sequentially assign pages to every student until curr. no. of assigned pages doesn't exceed value.

* If no. of studen > m, sol^n not possible/feasible. else possible/feasible.

Date 22/2/22

Ques- Job Scheduling Algo.

Given N jobs, every job is represented by following three element it.

1) Start Time 2) Finish Time 3) Profit or value associated

Find max. profit subset of jobs such that no two jobs in subset overlap.

(*) Example :- n = 4

Job details { start time, finish time, profit }

Job 1 : {1, 2, 50} Job 2 : {3, 5, 20}

Job 3 : {6, 19, 100} Job 4 : {2, 100, 200}

Ans → max. profit is 250.

We can get max. profit by scheduling job 1 & 4.

⇒ 50 + 200 ⇒ 250 profit

Intution :- (Recursion)

⇒ Sort jobs according to finish time.

Now, apply following recursion process.

ⓒ Here arr[] is array of n jobs.

findMaximumProfit (arr[], n)

{ (a) if (n == 1) return arr[1];

b) Return max. of following 2 profit.

i) max. profit by excluding curr. job, i.e. findMaxProfit (arr, n-1)

ii) Maximum profit by including curr job.

}

The idea is to find most latest job before curr. job in sorted array, that doesn't conflict with curr job, 'arr[n-1]'.
Once we find such a job, we recur for rest jobs till that job add profit of curr. job

∴ we are using binary search, and dp concept.
TC O(nlogn)

Ques → Arithmetic Number

Ques →

Given 3 integers 'A' denote first term of arithmetic sequence, 'C' denote common difference of arithmetic sequence and an integer 'B'.

Tell me whether 'B' exist in the arithmetic seq. or not.

Example :-

A = 1, B = 3, C = 2.

Output → 1.

3 is 2nd term of seq. starting with 1 & having common diff: 2.

Intution :-

Check if C = 0, if true then B = A, true than 1 else 0.

B-A remainder of (B-A)%C not equal 0, or B-A < 0 & C > 0 or B-A > 0 & C < 0).

then 0, otherwise 1 (true)

Ques → Smallest factorial number

Given a no. N, find smallest number whose factorial contain atleast n trailing zeroes.

Example :-

n = 1, output - 5

explanation - 5! = 120 which has atleast 1 trailing 0.

(*) Intuition :-

Trailing 0 in x! = count of 5s in prime factor of x!.

= floor(x/5) + floor (x/5^2) + floor (x/125) + ...

We can notice that max. value whose factorial contain n trailing zeroes is 5 * n

So to find max. value fact. contain n trailing zero, use binary search ranging 0 to 5 * n.

And, find smallest no. whose factorial contain n trailing zeroes.

Ques- Subset Sum

given list arr of N, print sums of all subset in it

Example :-

I/P - N=2 arr[] = {2,3}

O/P - 0, 2, 3, 5

when no element taken sum = 0,

when 2 taken sum = 2

when 3 taken sum = 3

when 2 & 3 taken sum = 2 + 3 = 5.

Intution :-

→ we'll solve this question using recursion,

we'll take a vector base[]; and if n==1,

→ then push 0 and push arr[0]. and return base

→ take 2 variable s, first and last.

and 2 vector ans & mkn.

⊙ ans contain (first, last);

⊙ mkn contain => subset (ans, s-1) ;

→ Iterate in loop, and, enter

⊙ a = mkn[i] + (*it) ; and push a into mkn vector.

→ and in end return mkn;

Date 24/2/22

Ques → Implement Merge Sort in place

Implement merge sort in standard implementation keeping sorting algo. as in-place.
In-place means it doesn't occupy extra memory for merge operation as in the standard case.

Example :
arr[] = {2,3,4,1} : o/p → 1, 2, 3, 4
arr[] = {56,2,45,3} : o/p → 2 45 56

(*) Intution :-
→ Let we've a no. A, & want to convert into no. B. & also a constraint that we can also recover no. A any time without using variable,
→ To achieve this we choose no. N, which is greater than both numbers and add B*N in A.
(#) To get B out from (A+B*N)
we divide (A+B*N) by N as (A+B*N)%N = A
(By taking modulus, we get old no. back, taking divide we get new no.)

Algorithms

mergeSort() :
→ Calculate mid two split array into two halves (left-sub-arr & right-sub-array)

→ Recursively call merge sort on left-subarray and right-sub-array to sort them.

→ Call merge func^n to merge left-sub and right sub.

merge() :

⊙ firstly find max. ele. in both sub array & increment by 1, to avoid collision by 0. & max. element during mod.

⊙ Idea is to traverse both sub-array from start symultenously.

⊙ One start from 'l' till 'm', another start from m+1, till 'l'. So, we will initialize 3 pointer say i, j, K.

⊙ i move from l till m; j will move from m+1 till r; K will move from l till r.

⊙ Now, update value a[K] by adding min(a[i], a[j]) * maximum_element.

⊙ Then also update those element which are left in both sub-array.

⊙ After updating all the elements divide all the elements by maximum_element. so we get updated array back.

Ques → How to sort a big array with many repeatations

Consider big array where elements are from small set and in any range. i.e. there are many repeatation. How efficiently sort the array?

Example : arr[] = {100, 12, 100, 1, 1, 12, 100, 1, 12, 100, 1, 1}
arr[] = {1, 1, 1, 1, 1, 12, 12, 12, 100, 100, 100}.

Intution :- (Binary Heap).

We can use Binary Heap to solve in O(n log m) time, also use Hashing to solve O(n + m log m).

(1) Create empty hash table. Input array values are stores as key and their count are sorted as value in hash table.

(2) For every element 'x' of arr[], do this :
  (a) if x is present in hashTab, Inc its value.
  (b) else insert x with value equal to 1.

(3) Consider all keys of hash table & sort them.

(4) Traverse all sorted keys and print every key its value times.

Implementation

[1] void sort (int arr[], int n)

```
    map<int, int> count;
for (int i=0; i<n; i++)
    count [arr[i]]++;

map<int, int> :: iteration it;
int index = 0;

for ( it = count.begin(); it != count.end(); ++it)
```

(2)

```
    while (it -> second--)
    arr[index++] = it -> first;
```

(3)

[3]

T.C- O(n + mlog m)