Microsoft Technical Interview Questions and Answers

### **Write a program that prints the k largest elements in an unordered array efficiently.**

Approach 1:

1. To implement the bubble sort Algorithm & sort the array in descending order.
2. To iterate the loop from 0 to k the print the first k element.

Code:

int main ()

{

int array [] = {1,5,3,6,8,4,2};

int size = 7;

int k=4;

//bubble sort in descending order

for (int i=0; i<size; i++)

{

for (int j=i+1; j<size; j++)

{

If (array[i]<array[j])

{

Swap (array[i], array[j]);

}

}

}

for (int i=0; i<k; i++)

{

cout<<array[i]<<” “;

}  
}

Output: 5 6 5 4

Time Complexity: O (size\*size + k)

Space Complexity: O (1)

Approach 2:

1. To sort array using the quick sort algorithm that is predefine sort function.
2. Againe the print the first k element from array.

Code:

int main ()

{

int array [] = {1,5,3,6,8,4,2};

int size = 7;

int k=4;

Sort (array, array + size, greater<int> ());

// greater<int> () its return either true of false while compare the element of array.

for (int i=0; i<k; i++)

{

cout<<array[i]<<” “;

}  
}

Time complexity: O (size \* log (size) + k)

Space Complexity: O (1);

Quicksort:

int part (int array [], int low, int high)

{

int i=low-1;

int pivot=array[high];

for (int j=low; j<=high; j++)

{

If (pivot<array[j])

{

i++;

swap (array[i], array[j]);

}

}

swap(array[i+1], array[high]);

return i+1;

}

void quicksort (int array [], int low, int high)

{

if(low<high) {

int partition=part (array, low, high);

quicksort (array, low, partition-1);

quicksort (array, partition+1, high);

}

}

void kLarge (int array [], int n, int k)

{

quicksort (ar,0,n-1);

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

cout << ar[j] << " ";

}

==================================================================================

### **2. Define objects and classes in C++?**

Answer:

A class is user define data type that contain the member variable and member function.

An object is instant of class. We can also say that object is variable of class due to class is user define data type.

Class A

{

Int member\_variable;

Void member\_function()

{}

};

Int main ()

{

A object =new A();

object.member\_variable;

object.member\_function();

}

==================================================================================

### **3. Write a function returning true if there is a triplet (a, b, c) that satisfies a2 + b2 = c2 given an array of numbers.**

Solution:

Approach 1:

1. Brute Force approach is to iterate the for loop three time then inside the third loop we check the condition that is a2 + b2 = c2. If it is equal, return true.
2. After iterate all the three loops, if not return function yet then return false. (Triplet is not found)

Time Complexity: O (n3);

Space Complexity: O (1);

Approach 2:

1. Using two pointer method.
2. At initial sort the array element.
3. 1st we can calculate the target element with the help of first iteration of the loop.
4. After that we can use the two pointers approach in which take the sum of first and last element. If its equal to target return true, else if target is greater than sum increase the lower index , else if target is less than sum decrease the high index.
5. Finally, if not return yet then return false.

Time complexity: O (N2);

Space Complexity: O (1);

Code:

bool triplet (vector<int> &array)

{

sort (array. begin (), array. end ());

int n=array.size();

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

{

Int target=array[i]\*array[i];

Int low=0;

Int high=n-1;

while (low<high)

{

If (target==(array[low]+array[high])&&low!=i&&high!=i)

{

return true;

}

else if (target>(array[low]+array[high]))

{

low++;

}

else

{

high--;

}

}

}

return false;

}

Time complexity: O (N2);

Space complexity: O (1);

Approach 3:

1. Third approach is to using hashing technique we can solve.
2. Initial we calculate the maximum value of the array.
3. We create a hash array of size maximum +1 and initial with zero.
4. Using for loop we assign the frequency of the element in hash array.
5. After that we can iterate the two loop and check the condition if hash array found the value zero then continue the loop and also check the element not present in the same index and sum of square root of the element is not outside the range that is maximum value of array.
6. Inside the second loop we can calculate the square of element if it is prefect square is not found continue the loop.
7. It is found return true.
8. End of function return false;

Code:

bool triplet (vector<int> &array)

{

int n=array.size();

int maxi=INT\_MIN;

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

{

Maxi=max (array[i],maxi);

}

int hsah\_array[maxi+1]={0};

for (int i=0; i<maxi+1; i++)

{

Hash\_array[i]++;

}

for (int i=0 ;i<maxi+1 ;i++)

{

If(hash\_array[i]==0)

{

continue;

}

for (int j=1 ;j<maxi+1 ;j++)

{

If(hash\_array[i]>=1&&i==j||hash\_array[j]==0)

{

Continue;

}

Int target=sqrt(i\*i+j\*j);

If(target\*target!=(i\*i+j\*j))

{

continue;

}

If(target>maxi+1)

{

continue;

}

If(hash\_array[target])

{

Return true;

}

}

}

return false;

}

Time complexity: O (N + O (max (array[]))2);

Space complexity: O (max (array[])));

=================================================================================

### **Q4: How does operator overloading work?**

Solution:

Define: Operator overloading is user define data type. Its critical component. The default meaning such as + - \* / and so on can change using operator overloading.

Example:

In the example we can demonstrate operator addition using class object.

you we can add the custom object directly but with the help of operator overloading we can do.

#inclide<iostream>

using namespace std;

class A

{

int mark;//private variable by default.

int extramarks; //private variable by default.

public:

A ()

{

mark=0;

extramarks=0;

}

A (int ma, int ex)

{

mark=ma;

extramarks=ex;

}

void display ()

{

cout<<mark<<” “<<extramarks<<endl;

}

A operator+ (A m)

{

A temp;

temp.mark=mark+m.mark;

temp.extramarks=extramarks+m.extramarks;

return temp;

}

}

int main ()

{

A m1 (10,20), m2 (20,13);

A m3= m1 + m2;

m3.display();

}

==================================================================================

### **5. In C++, how do you allocate and release memory?**

Solution:

In C++, new operator is used to allocate the memory and the delete operator is used to release the memory.

For Allocate: int \*ptr = new int[10];

For Release: delete ptr;//delete [] ptr;

==================================================================================

### **Determine if there are any two integers in the array whose sum equals the provided value, given an array of integers and a value.**

Example:

Array[] = { 5, 90, 45, 20, 13, 111 };

        int x = 25;

output: 5 20;

Approach 1:

Brute force Approach:

1. We the iterate two for loop and inside the second for loop check the sum of two value is equal to target. If true then print the value.

Time Complexity: O (N2);

Space Complexity: O (1);

Approach 2:

Two pointer Approach:

1. Initially, sort the array.
2. Using two pointers approach we can sum of last and first element and check the condition.
3. If target is found then print it.
4. Sum is less than target increment first index.
5. Sum is greater than target decrement last index.
6. End.

Code:

Int low=0;

Int high=n-1;

while (low<high)

{

If (target==(array[low]+array[high]))

{

cout<<array[low]<<” “<<array[high]<<endl;

}

else if (target>(array[low]+array[high]))

{

low++;

}

else

{

high--;

}

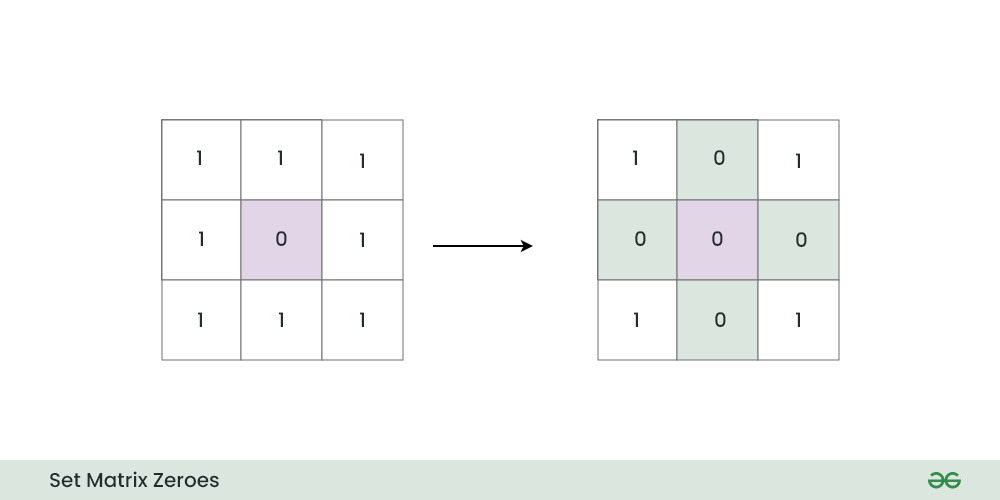
}

Time Complexity: O (N\*log (N));

Space Complexity: O (1);

=================================================================================

### **7. If any element in the given two-dimensional array is zero, make the entire row and column zero.**



Solution:

Approach 1:

1. In first traversal of the matrix marks all the zero elements with another value that is -1.
2. In second traversal make all the row and column 0 of mark element.
3. End

Time complexity: O (M\*N\*N+M);

Space complexity: O (1);

If all the element of matrix is zero and one then solution is:

Code:

    int n=matrix.size();

    int m=matrix[0].size();

        for(int i=0;i<n;i++)

        {

            for(int j=0;j<m;j++)

            {

                if(matrix[i][j]==0)

                {

                    matrix[i][j]=-1;

                }

            }

        }

        for(int i=0;i<n;i++)

        {

            for(int j=0;j<m;j++)

            {

                if(matrix[i][j]==-1)

                {

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

                     {

                         if(matrix[k][j]!=-1)

                         {

                             matrix[k][j]=0;

                         }

                     }

                     for(int k=0;k<m;k++)

                     {

                         if(matrix[i][k]!=-1)

                         {

                             matrix[i][k]=0;

                         }

                     }

                }

            }

        }

         for(int i=0;i<n;i++)

        {

            for(int j=0;j<m;j++)

            {

                if(matrix[i][j]==-1)

                {

                    matrix[i][j]=0;

                }

            }

        }

Approach 1:

* If all the elements are in the range -231 <= matrix[i][j] <= 231 - 1

1. With the help of extra space. we take another matrix and assign all element is zero.
2. The matrix will take update that which cell of the matrix visited.
3. And similarly, update the original matrix.

Code:

vector<vector<int>> ma(n,vector<int>(m,0));

         for(int i=0;i<n;i++)

         {

             for(int j=0;j<m;j++)

             {

                 if(matrix[i][j]==0&&ma[i][j]!=1)// important condition.

                 {

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

                     {

                         if(matrix[k][j]!=0)

                         {

                             matrix[k][j]=0;

                             ma[k][j]=1;

                         }

                     }

                     for(int k=0;k<m;k++)

                     {

                         if(matrix[i][k]!=0)

                         {

                             matrix[i][k]=0;

                             ma[i][k]=1;

                         }

                     }

                 }

             }

         }

Time complexity: O (M\*N\* (N+M));

Space complexity: O (N + M);

Approach 2:

Using array row and col.

* 1. We can store true in row and col all the cell if found zero.
  2. Traversal the matrix all element if found either of row and col is true update matrix is 0.
  3. End.

Code:

  void setZeroes(vector<vector<int>>& matrix) {

        int n=matrix.size();

        int m=matrix[0].size();

        int row[n]={0};

        int col[m]={0};

 for(int i=0;i<n;i++)

        {

            for(int j=0;j<m;j++)

            {

                if(matrix[i][j]==0)

                {

                    row[i]=1;

                    col[j]=1;

                }

            }

        }

        for(int i=0;i<n;i++)

        {

            for(int j=0;j<m;j++)

            {

                if(row[i]||col[j])

                {

                    matrix[i][j]=0;

                }

            }

        }

}

Time complexity: O (N\*M)

Space complexity: O (N) + O (M)

Approach 3:

1. In place optimisation.

Modify this.

Code:

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> zeroMatrix(vector<vector<int>> &matrix, int n, int m) {

// int row[n] = {0}; --> matrix[..][0]

// int col[m] = {0}; --> matrix[0][..]

int col0 = 1;

// step 1: Traverse the matrix and

// mark 1st row & col accordingly:

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

if (matrix[i][j] == 0) {

// mark i-th row:

matrix[i][0] = 0;

// mark j-th column:

if (j != 0)

matrix[0][j] = 0;

else

col0 = 0;

}

}

}

// Step 2: Mark with 0 from (1,1) to (n-1, m-1):

for (int i = 1; i < n; i++) {

for (int j = 1; j < m; j++) {

if (matrix[i][j] != 0) {

// check for col & row:

if (matrix[i][0] == 0 || matrix[0][j] == 0) {

matrix[i][j] = 0;

}

}

}

}

//step 3: Finally mark the 1st col & then 1st row:

if (matrix[0][0] == 0) {

for (int j = 0; j < m; j++) {

matrix[0][j] = 0;

}

}

if (col0 == 0) {

for (int i = 0; i < n; i++) {

matrix[i][0] = 0;

}

}

return matrix;

}

int main()

{

vector<vector<int>> matrix = {{1, 1, 1}, {1, 0, 1}, {1, 1, 1}};

int n = matrix.size();

int m = matrix[0].size();

vector<vector<int>> ans = zeroMatrix(matrix, n, m);

cout << "The Final matrix is: n";

for (auto it : ans) {

for (auto ele : it) {

cout << ele << " ";

}

cout << "n";

}

return 0;

}

==================================================================================

### **Given the head pointers of two linked lists, add them and return the new linked list. Each linked list represents an integer number (each node is a digit).**

Example:

The First List is 14 51 19 42 16

The Second List is 83 34

The Final List is 1 9 4 2 0 0

Solution:

Approach 1:

1. If both the lists are zero node return any of the node.
2. Reverse the both link lists.
3. Traversal the both link lists and compute the sum of each list node.
4. If any of the list node is not found sum with zero.
5. And finally, update into new link list and reverse the answer list.
6. End.

Code:

/\* Online C++ Compiler and Editor \*/

#include <iostream>

using namespace std;

class Node

{

public:

int val;

Node \*next;

Node(int val)

{

this->val=val;

this->next=NULL;

}

};

void display(Node \*l1)

{

while(l1!=NULL)

{

cout<<l1->val<<" ";

l1=l1->next;

}

cout<<endl;

}

Node\* reverse1(Node \*l1)

{

Node \*p,\*q,\*r;

p=l1;

q=NULL;

r=NULL;

while(p!=NULL)

{

r=q;

q=p;

p=p->next;

q->next=r;

}

return q;

}

Node\* sumoflist(Node \*l1,Node \*l2)

{

Node \*first,\*last;

int carray=0,p=0;

if(l1==NULL&&l1==NULL)

return l1;

while(l1!=NULL||l2!=NULL||carray)

{

int sum=0;

if(l1!=NULL)

{

sum+=l1->val;

l1=l1->next;

}

if(l2!=NULL)

{

sum+=l2->val;

l2=l2->next;

}

sum+=carray;

carray=(sum/10);

Node \*temp=new Node(sum%10);

if(p==0)

{

first=temp;

last=temp;

p++;

}

else

{

last->next=temp;

last=temp;

}

}

return first;

}

int main()

{

int arr1[]={14,51,19,42,16};

int arr2[]={83,34};

Node \*l1,\*l2,\*last;

// for l1 Link list;

int n=sizeof(arr1)/sizeof(arr1[0]);

if(n==0){

l1=NULL;

}

else{

Node \*temp= new Node(arr1[0]);

l1=temp;

last=temp;

}

for(int i=1;i<n;i++)

{

Node \*temp= new Node(arr1[i]);

last->next=temp;

last=temp;

}

display(l1);

// for l2 Link list

int m=sizeof(arr2)/sizeof(arr2[0]);

if(m==0){

l2=NULL;

}

else{

Node \*temp1= new Node(arr2[0]);

l2=temp1;

last=temp1;

}

for(int i=1;i<m;i++)

{

Node \*temp1= new Node(arr2[i]);

last->next=temp1;

last=temp1;

}

display(l2);

// reverse both the linklist

Node \*revl1=reverse1(l1);

Node \*revl2=reverse1(l2);

display(revl1);

display(revl2);

//sum of each node of both linklist

Node \*reslist=sumoflist(revl1,revl2);

display(reslist);

// reverse the answer

Node \*anslist=reverse1(reslist);

display(anslist);

return 0;

}

Output sequence:

14 51 19 42 16

83 34

16 42 19 51 14

34 83

0 0 2 4 9 1

1 9 4 2 0 0

Time complexity: O (N)

Space complexity: O (1)

==================================================================================

### **9. You're given a linked list with two pointers at each node. The regular 'next' pointer is the first. 'Arbitrary pointer' is the second pointer, and it can point to any node in the linked list. Your task is to write code that creates a deep copy of the linked list provided. Any operations on the original list (inserting, updating, and removing items) should have no effect on the cloned list.**

A linked list of length n is given such that each node contains an additional random pointer, which could point to any node in the list, or null.

Construct a [**deep copy**](https://en.wikipedia.org/wiki/Object_copying#Deep_copy) of the list. The deep copy should consist of exactly n **brand new** nodes, where each new node has its value set to the value of its corresponding original node. Both the next and random pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. **None of the pointers in the new list should point to nodes in the original list**.

For example, if there are two nodes X and Y in the original list, where X.random --> Y, then for the corresponding two nodes x and y in the copied list, x.random --> y.

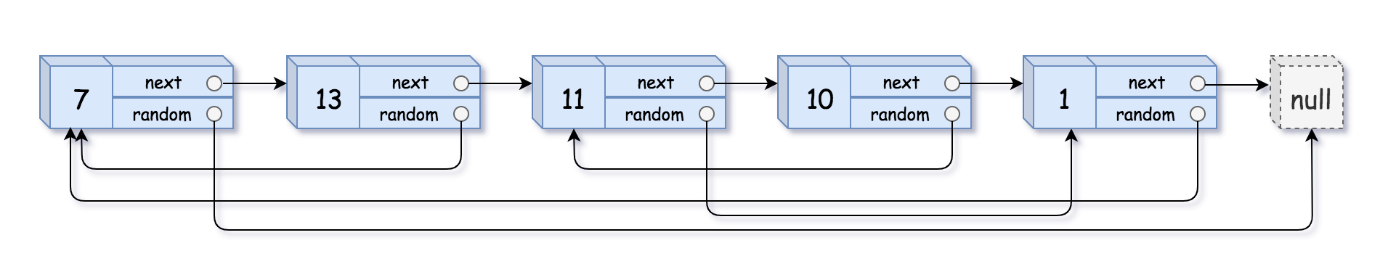
Return *the head of the copied linked list*.

The linked list is represented in the input/output as a list of n nodes. Each node is represented as a pair of [val, random\_index] where:

* val: an integer representing Node.val
* random\_index: the index of the node (range from 0 to n-1) that the random pointer points to, or null if it does not point to any node.

Your code will **only** be given the head of the original linked list.

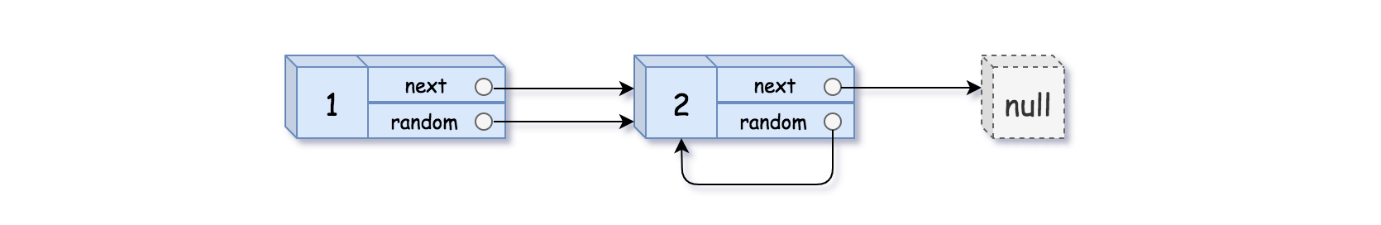
**Example 1:**



**Input:** head = [[7,null],[13,0],[11,4],[10,2],[1,0]]

**Output:** [[7,null],[13,0],[11,4],[10,2],[1,0]]

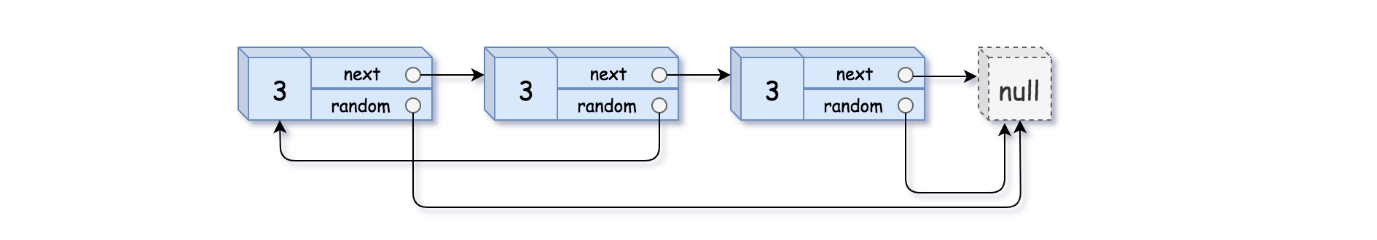
**Example 2:**



**Input:** head = [[1,1],[2,1]]

**Output:** [[1,1],[2,1]]

**Example 3:**

****

**Input:** head = [[3,null],[3,0],[3,null]]

**Output:** [[3,null],[3,0],[3,null]]

**Constraints:**

* 0 <= n <= 1000
* -104 <= Node.val <= 104
* Node.random is null or is pointing to some node in the linked list.

Approach 1: using hash map.

* 1. Create a link list and as well take care about the random and next pointer.
  2. Make a print function.
  3. Make a deep copy function with the help of map data structure.
  4. End.

Code:

/\* Online C++ Compiler and Editor \*/

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

class Node

{

public:

int val;

Node\* next;

Node\* random;

Node(int val)

{

this->val=val;

this->next=NULL;

this->random=NULL;

}

};

Node\* deepCopy(Node \*head)

{

unordered\_map<Node\*,Node\*> m;

if(head==NULL)

{

return NULL;

}

Node \*first, \*second=NULL;

Node \*temp=head;

int flag=0;

while(temp!=NULL)

{

Node \*node = new Node(temp->val);

if(flag==0)

{

first=node;

second=node;

flag=1;

}

else

{

second->next=node;

second=node;

}

m.insert({temp,second});

temp=temp->next;

}

temp=head;

//Importanat point

while(temp!=NULL)

{

Node \*node=m[temp];

node->random=temp->random;

temp=temp->next;

}

//this part is to show that create a node then assign the value into next random pointer.

t=head;

        temp=first;

        while(t!=NULL)

        {

             Node \*random=t->random;

             temp->random=m[random];

             t=t->next;

             temp=temp->next;

        }

return first;

}

void print(Node \*head)

{

while(head!=NULL) {

cout<<head->val<<" : ";

head->next!=NULL?cout<<head->next->val<<"NV ":cout<<"NULL"<<" ";

head->random!=NULL?cout<<head->random->val<<"RV ":cout<<"NULL"<<" ";

head=head->next;

}

cout<<endl;

}

int main()

{

Node \*head=NULL;

Node \*node1= new Node(1);

Node \*node2= new Node(2);

Node \*node3= new Node(3);

Node \*node4= new Node(4);

head=node1;

head->next=node2;

head->next->next=node3;

head->next->next->next=node4;

head->random=node2;

head->next->random=node4;

head->next->next->random=NULL;

head->next->next->next->random=node1;

print(head);

Node \*NewNode=deepCopy(head);

print(NewNode);

return 0;

}

Time Complexity: O (N)

Space Complexity: O (N)

### IMPORTANT : 1. Minimize the Heights II(gfg).

Given an array **arr[]** denoting heights of **N** towers and a positive integer **K.**

For **each**tower, you must perform **exactly one** of the following operations **exactly once**.

* **Increase**the height of the tower by **K**
* **Decrease**the height of the tower by **K**

Find out the **minimum**possible difference between the height of the shortest and tallest towers after you have modified each tower.

You can find a slight modification of the problem [here](https://practice.geeksforgeeks.org/problems/minimize-the-heights-i/1/).  
**Note:** It is **compulsory**to increase or decrease the height by K for each tower. **After** the operation, the resultant array should **not** contain any **negative integers**.

**Example 1:**

**Input:**

K = 2, N = 4

Arr[] = {1, 5, 8, 10}

**Output:**

5

**Explanation:**

The array can be modified as   
{1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.   
The difference between

the largest and the smallest is 8-3 = 5.

**Example 2:**

**Input:**

K = 3, N = 5

Arr[] = {3, 9, 12, 16, 20}

**Output:**

11

**Explanation:**

The array can be modified as  
{3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.   
The difference between

the largest and the smallest is 17-6 = 11.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **getMinDiff()** which takes the **arr[], n,** and **k**as input parameters and returns an integer denoting the minimum difference.

**Expected Time Complexity:** O(N\*logN)  
**Expected Auxiliary Space:** O(N)

**Constraints**  
1 ≤ K ≤ 109  
1 ≤ N ≤ 105  
1 ≤ Arr[i] ≤ 109

Approach:

1 5 8 10

I=0; min=1, max=10. Ans= max-min=9

I=1 min=min(1+2,5-2)=3,max=max(10-2, 1+2)=8 ans=min(9,8-3)=5;

I=2 min=min(1+2,8-2)=3,max=max(10-2, 5+2)=8 ans=min(5,8-3)=5;

I=3 min=min(1+2,10-2)=3,max=max(10-2, 8+2)=10 ans=min(5,10-3)=5;

Final answer is 5;

int getMinDiff(int arr[], int n, int k) {

// code here

sort(arr,arr+n);

int mini=arr[0],maxi=arr[n-1];

int ans=maxi-mini;

for(int i=1;i<n;i++)

{

if((arr[i]-k)<0)continue;

mini=min(arr[0]+k,arr[i]-k);

maxi=max(arr[n-1]-k,arr[i-1]+k);

ans=min(ans,maxi-mini);

}

return ans;}