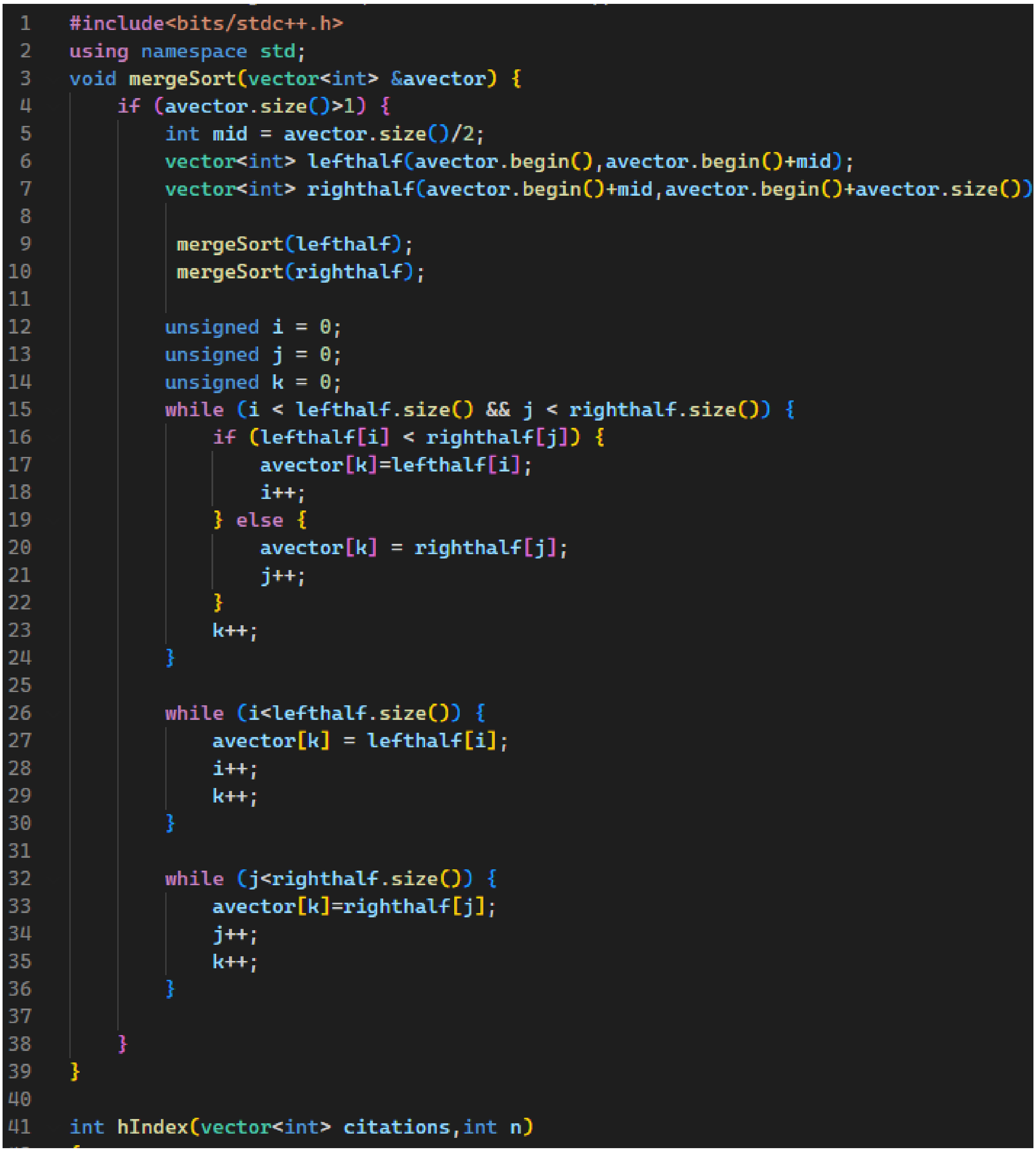
**NAME: GAURAV YADAV**

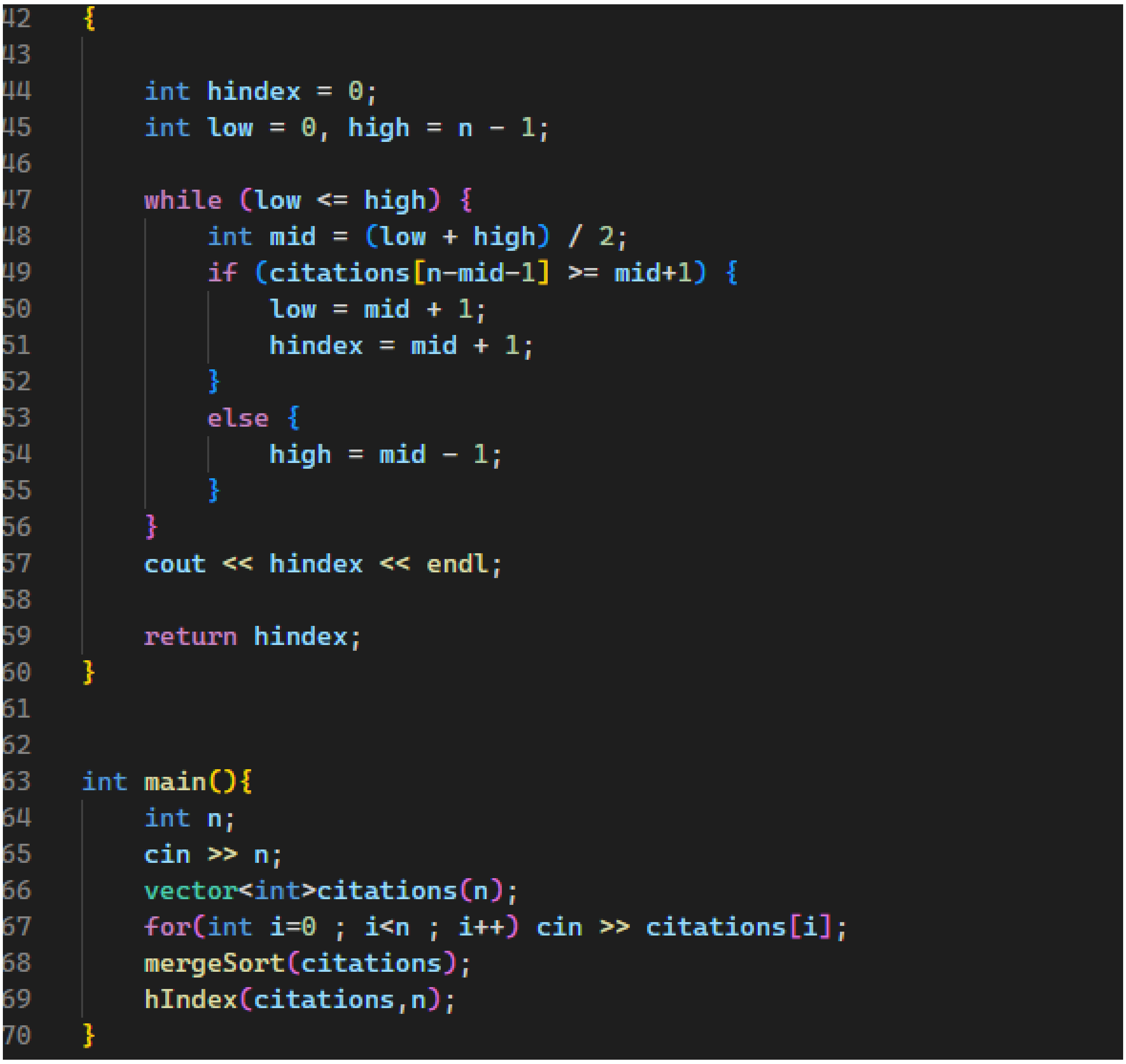
**ROLL NUMBER: 21075034**

**E-MAIL: gaurav.yadav.cse21@itbhu.ac.in**

**Q1)** Given an array of integers citations where citations[i] is the number of citations a researcher received for their *ith* paper, return the researcher's h-index. A scientist has an index *h* if *h* of their *n* papers have at least *h* citations each, and the other *n* − *h* papers have no more than *h* citations each. If there are several possible values for *h*, the maximum one is taken as the h-index. (Time complexity of your code should not exceed O(nlogn))

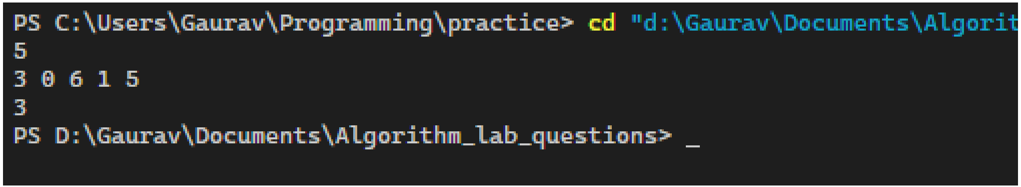


```
#include<bits/stdc++.h>
using namespace std;
void mergeSort(vector<int> &avector) {
    if (avector.size()>1) {
        int mid = avector.size()/2;
        vector<int> lefthalf(avector.begin(),avector.begin()+mid);
        vector<int> righthalf(avector.begin()+mid,avector.begin()+avector.size())

         mergeSort(lefthalf);
         mergeSort(righthalf);

        unsigned i = 0;
        unsigned j = 0;
        unsigned k = 0;
        while (i < lefthalf.size() && j < righthalf.size()) {
            if (lefthalf[i] < righthalf[j]) {
                avector[k]=lefthalf[i];
                i++;
            } else {
                avector[k] = righthalf[j];
                j++;
            }
            k++;
        }

        while (i<lefthalf.size()) {
            avector[k] = lefthalf[i];
            i++;
            k++;
        }

        while (j<righthalf.size()) {
            avector[k]=righthalf[j];
            j++;
            k++;
        }

    }
}

int hIndex(vector<int> citations,int n)
```

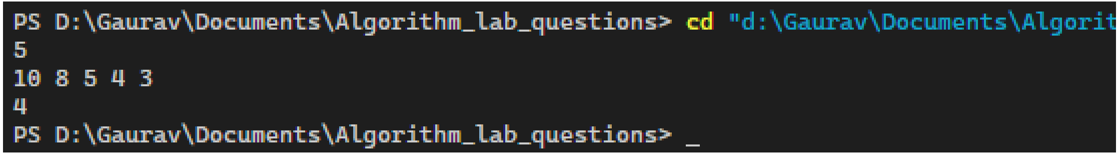


```
{

    int hindex = 0;
    int low = 0, high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        if (citations[n-mid-1] >= mid+1) {
            low = mid + 1;
            hindex = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    cout << hindex << endl;

    return hindex;
}


int main(){
    int n;
    cin >> n;
    vector<int>citations(n);
    for(int i=0 ; i<n ; i++) cin >> citations[i];
    mergeSort(citations);
    hIndex(citations,n);
}
```

**Output:**



```
PS C:\Users\Gaurav\Programming\practice> cd "d:\Gaurav\Documents\Algorit
5
3 0 6 1 5
3
PS D:\Gaurav\Documents\Algorithm_lab_questions> _
```



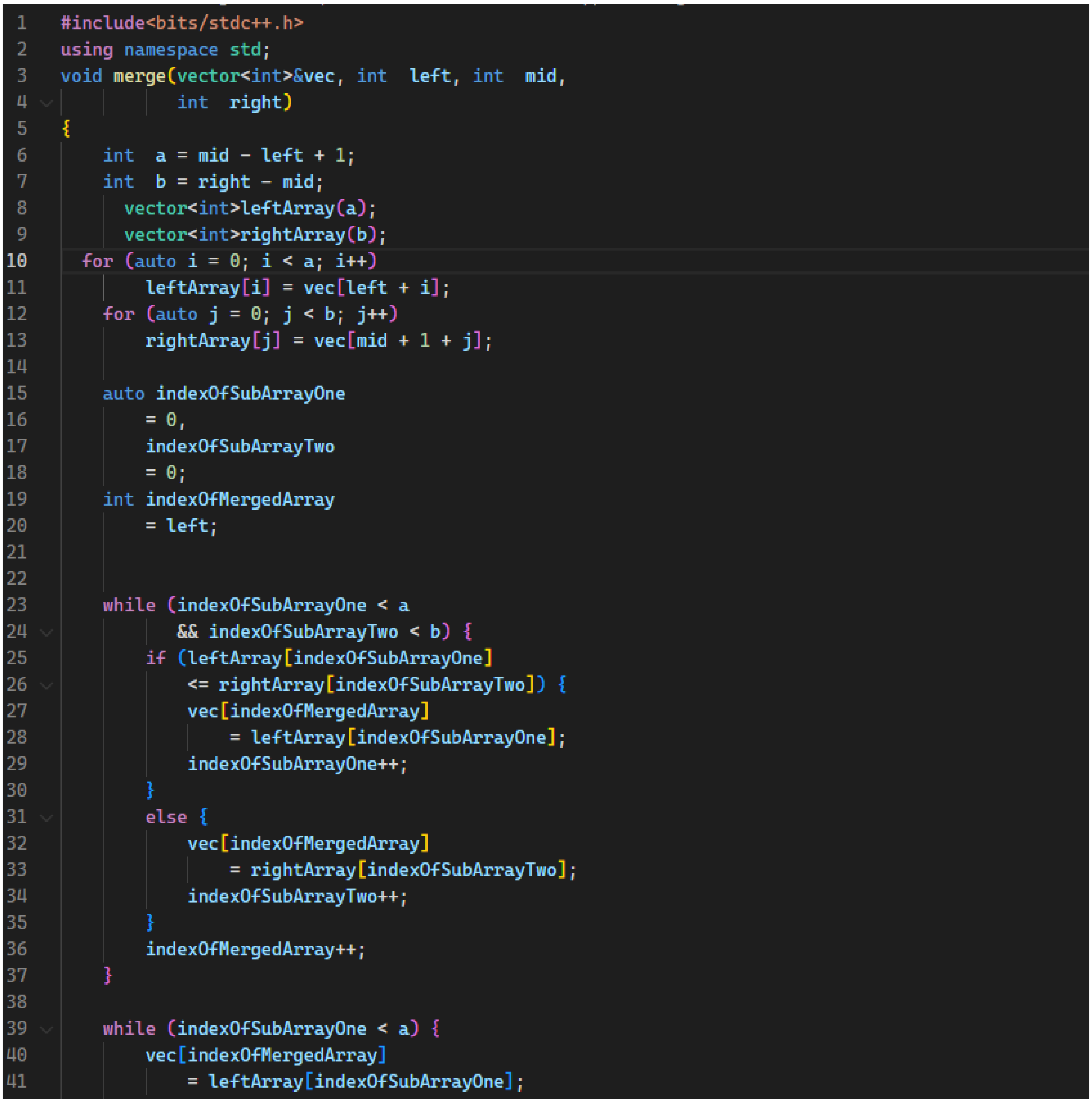
```
PS D:\Gaurav\Documents\Algorithm_lab_questions> cd "d:\Gaurav\Documents\Algorit
5
10 8 5 4 3
4
PS D:\Gaurav\Documents\Algorithm_lab_questions> _
```

**Time Complexity:** Since we are using the merge sort algorithm which has a time complexity of O(nlogn) and we also used binary search for finding the required value of h so binary search has time complexity of O(logn). So the overall time complexity will be that of merge sort which is O(nlogn).

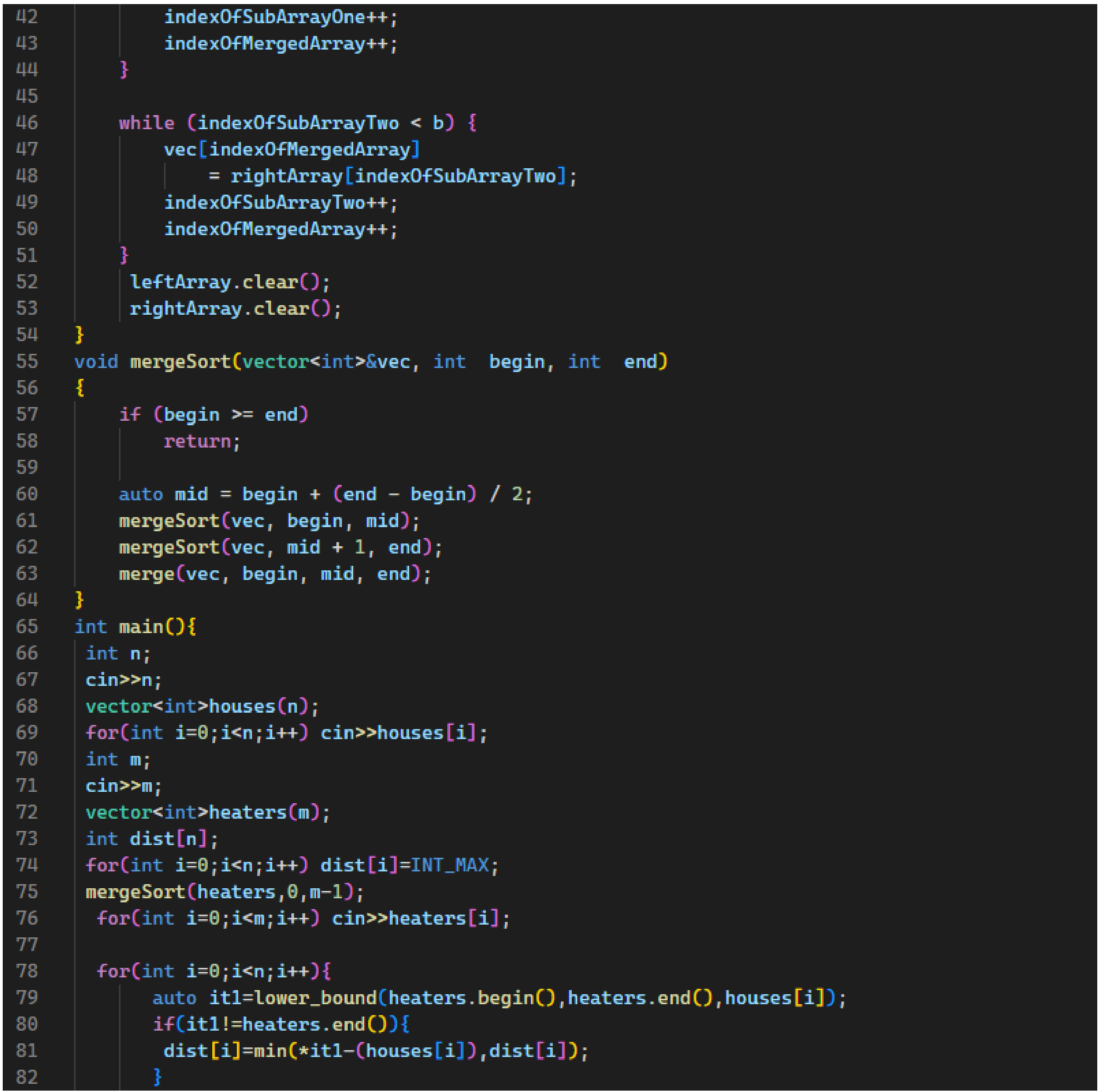
GAURAV YADAV

21075034

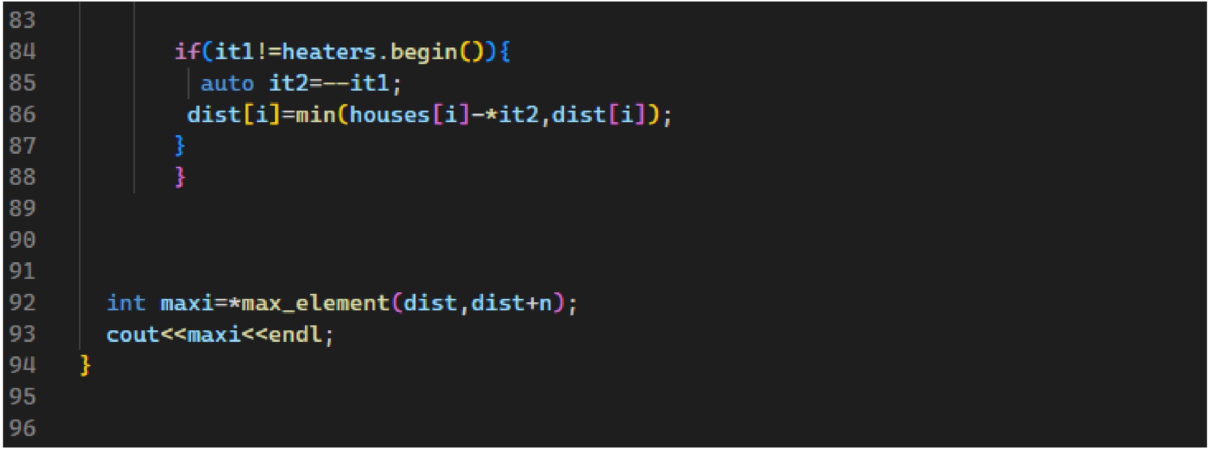
**Q2) Design a standard heater with a fixed warm radius to warm all the houses.**
**Every house can be warmed, as long as the house is within the heater's warm radius range. Given the positions of houses and heaters on a horizontal line, return *the minimum radius standard of heaters so that those heaters could cover all houses.* (Try to write the most optimized code)**



```
#include<bits/stdc++.h>
using namespace std;
void merge(vector<int>&vec, int  left, int  mid,
           int  right)
{
    int  a = mid - left + 1;
    int  b = right - mid;
      vector<int>leftArray(a);
      vector<int>rightArray(b);
  for (auto i = 0; i < a; i++)
        leftArray[i] = vec[left + i];
    for (auto j = 0; j < b; j++)
        rightArray[j] = vec[mid + 1 + j];

    auto indexOfSubArrayOne
        = 0,
        indexOfSubArrayTwo
        = 0;
    int indexOfMergedArray
        = left;


    while (indexOfSubArrayOne < a
           && indexOfSubArrayTwo < b) {
        if (leftArray[indexOfSubArrayOne]
            <= rightArray[indexOfSubArrayTwo]) {
            vec[indexOfMergedArray]
                = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            vec[indexOfMergedArray]
                = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    while (indexOfSubArrayOne < a) {
        vec[indexOfMergedArray]
            = leftArray[indexOfSubArrayOne];
```

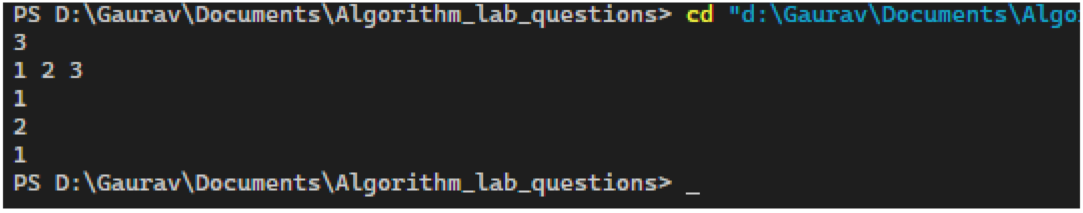


```
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }

    while (indexOfSubArrayTwo < b) {
        vec[indexOfMergedArray]
            = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
     leftArray.clear();
     rightArray.clear();
}
void mergeSort(vector<int>&vec, int  begin, int  end)
{
    if (begin >= end)
        return;

    auto mid = begin + (end - begin) / 2;
    mergeSort(vec, begin, mid);
    mergeSort(vec, mid + 1, end);
    merge(vec, begin, mid, end);
}
int main(){
 int n;
 cin>>n;
 vector<int>houses(n);
 for(int i=0;i<n;i++) cin>>houses[i];
 int m;
 cin>>m;
 vector<int>heaters(m);
 int dist[n];
 for(int i=0;i<n;i++) dist[i]=INT_MAX;
 mergeSort(heaters,0,m-1);
  for(int i=0;i<m;i++) cin>>heaters[i];

  for(int i=0;i<n;i++){
       auto it1=lower_bound(heaters.begin(),heaters.end(),houses[i]);
       if(it1!=heaters.end()){
        dist[i]=min(*it1-(houses[i]),dist[i]);
       }
```

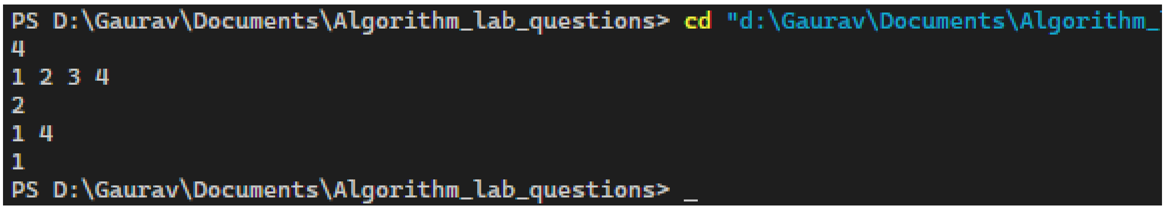


```

       if(it1!=heaters.begin()){
         auto it2=--it1;
        dist[i]=min(houses[i]-*it2,dist[i]);
       }
       }



  int maxi=*max_element(dist,dist+n);
  cout<<maxi<<endl;
}


```

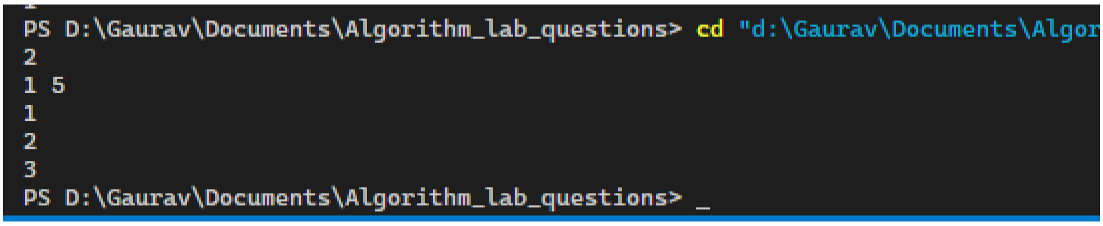
**Output:**



```
PS D:\Gaurav\Documents\Algorithm_lab_questions> cd "d:\Gaurav\Documents\Algo
3
1 2 3
1
2
1
PS D:\Gaurav\Documents\Algorithm_lab_questions> _
```



```
PS D:\Gaurav\Documents\Algorithm_lab_questions> cd "d:\Gaurav\Documents\Algorithm_
4
1 2 3 4
2
1 4
1
PS D:\Gaurav\Documents\Algorithm_lab_questions> _
```



```
PS D:\Gaurav\Documents\Algorithm_lab_questions> cd "d:\Gaurav\Documents\Algor
2
1 5
1
2
3
PS D:\Gaurav\Documents\Algorithm_lab_questions> _
```

**Time Complexity:**

The time complexity of the code is O(nlogm) where n is the number of houses and m is the number of heaters used to heat the houses.