Name: Gaurav Yadav

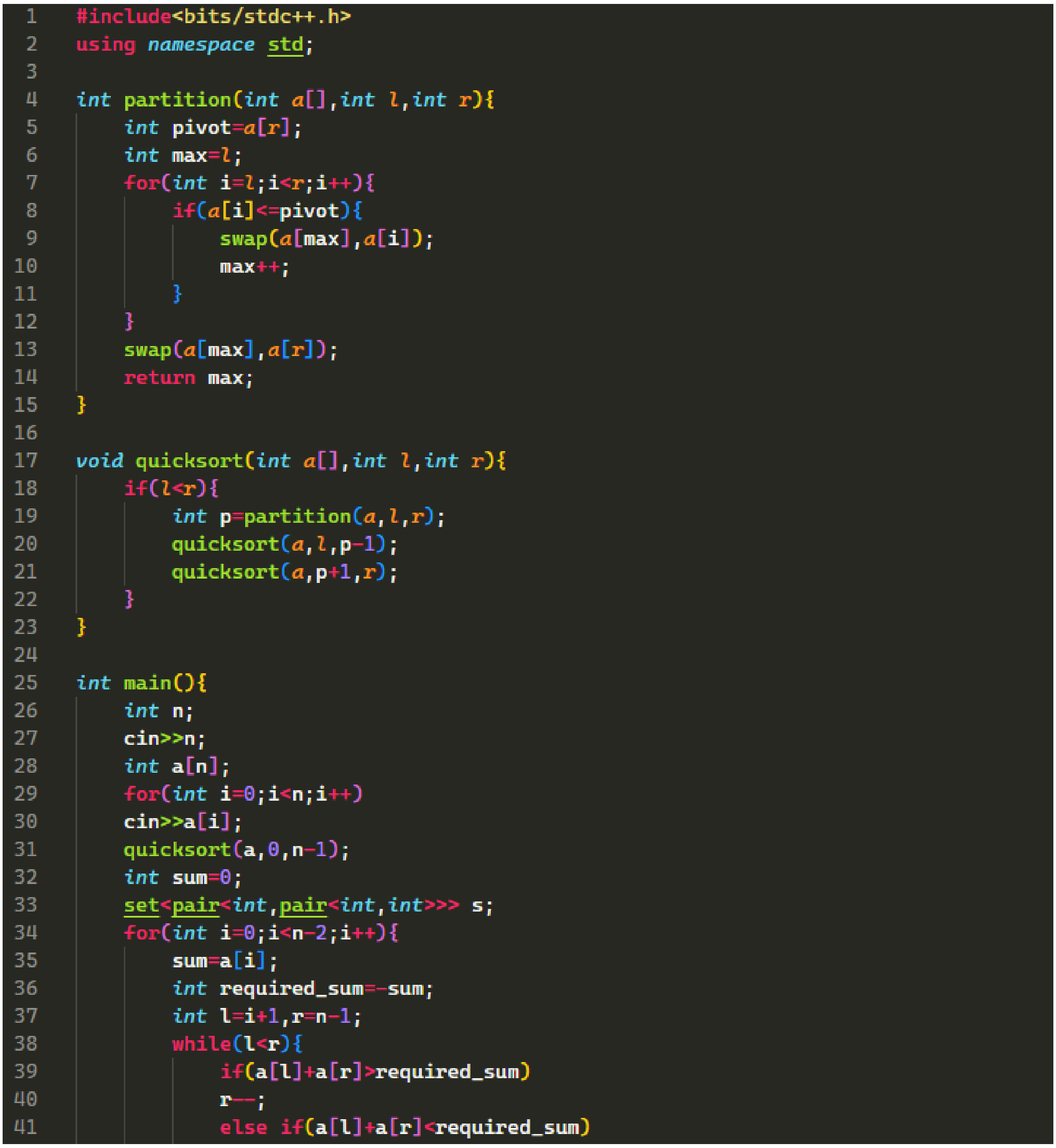
Roll number: 21075034

Dept: CSE(B.tech)

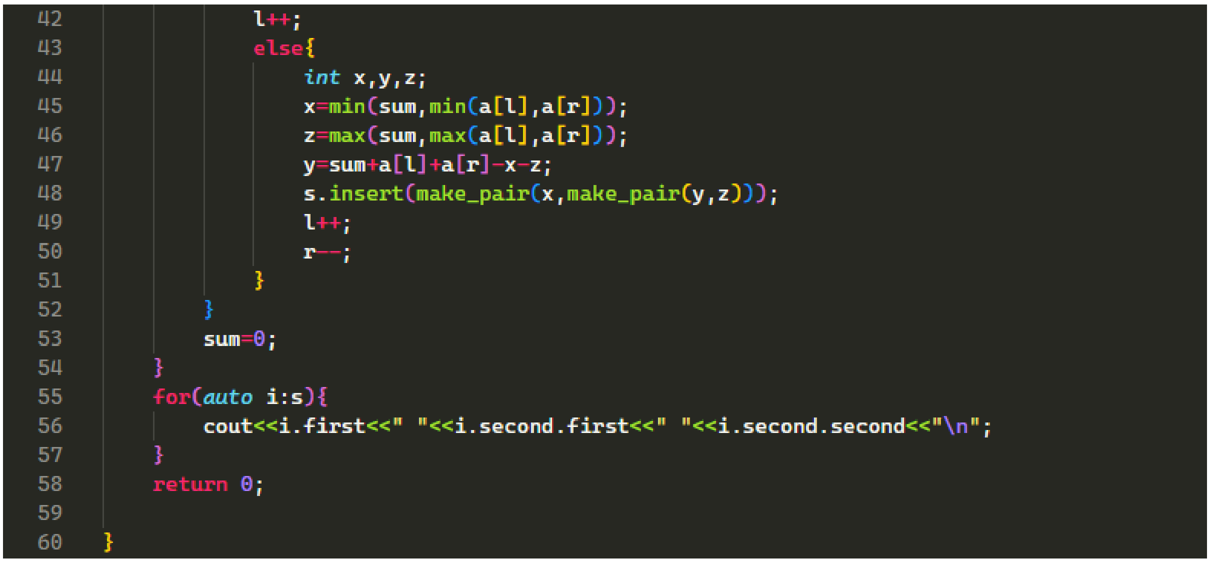
E-mail: gaurav.yadav.cse21@itbhu.ac.in

LAB-5

Q1) You have been given an integer array arr[], form triplets [arr[x], arr[y], arr[z]] such that x!=y, y!=z, z!=x, and arr[x] + arr[y] + arr[z] == 0. Return all such triplets.
Your solution must not contain duplicate triplets.

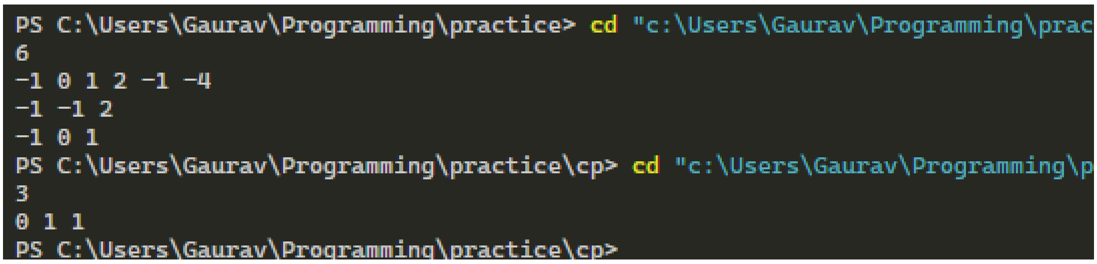


```
#include<bits/stdc++.h>
using namespace std;

int partition(int a[],int l,int r){
    int pivot=a[r];
    int max=l;
    for(int i=l;i<r;i++){
        if(a[i]<=pivot){
            swap(a[max],a[i]);
            max++;
        }
    }
    swap(a[max],a[r]);
    return max;
}

void quicksort(int a[],int l,int r){
    if(l<r){
        int p=partition(a,l,r);
        quicksort(a,l,p-1);
        quicksort(a,p+1,r);
    }
}

int main(){
    int n;
    cin>>n;
    int a[n];
    for(int i=0;i<n;i++)
    cin>>a[i];
    quicksort(a,0,n-1);
    int sum=0;
    set<pair<int,pair<int,int>>> s;
    for(int i=0;i<n-2;i++){
        sum=a[i];
        int required_sum=-sum;
        int l=i+1,r=n-1;
        while(l<r){
            if(a[l]+a[r]>required_sum)
            r--;
            else if(a[l]+a[r]<required_sum)
```



```
            l++;
            else{
                int x,y,z;
                x=min(sum,min(a[l],a[r]));
                z=max(sum,max(a[l],a[r]));
                y=sum+a[l]+a[r]-x-z;
                s.insert(make_pair(x,make_pair(y,z)));
                l++;
                r--;
            }
        }
        sum=0;
    }
    for(auto i:s){
        cout<<i.first<<" "<<i.second.first<<" "<<i.second.second<<"\n";
    }
    return 0;

}
```

Output:



```
PS C:\Users\Gaurav\Programming\practice> cd "c:\Users\Gaurav\Programming\prac
6
-1 0 1 2 -1 -4
-1 -1 2
-1 0 1
PS C:\Users\Gaurav\Programming\practice\cp> cd "c:\Users\Gaurav\Programming\p
3
0 1 1
PS C:\Users\Gaurav\Programming\practice\cp>
```

Recurrence relation for quicksort:

T(n)=T(i)+T(n-i-1)+cn for n>1

T(n)=c for n=1

Worst Case:

i=0 or n-1

T(n)=T(n-1)+cn

T(n-1)=T(n-2)+c(n-1)

T(n-2)=T(n-3)+c(n-2)

. . .

. . .

. . .

T(2) = T(1) + c(2)

T(n)=T(1)+c(n+n-1+n-2+n-3+……………………+2)

T(n)=c(1+2+3+4+…………………………………..+n)

21075034 Gaurav Yadav

T(n)=c(n*(n+1))/2

=> Time complexity in worst case is O(n^2)+O(n^2)

=>O(n^2)

Best case:

i=n/2;

T(n)=2T(n/2)+cn

Which is same as merge sort recurrence relation

So T(n)=O(nlogn)

=>Time complexity in best case is O(nlogn)+O(n^2)

=>O(n^2)

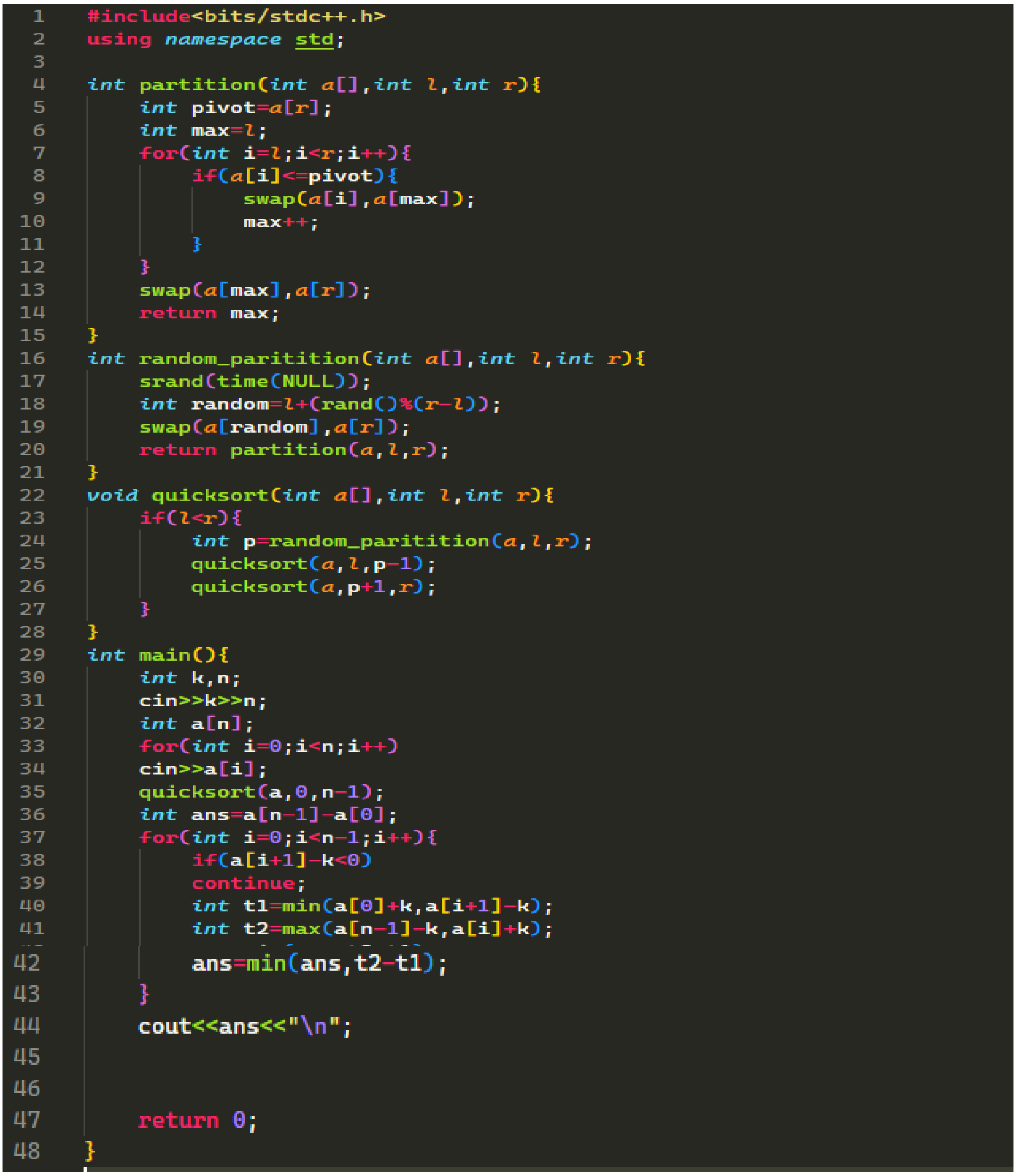
21075034 Gaurav Yadav

Q2) You have been given an integer array arr[] denoting the heights of N towers and a positive integer K.

You **must** carry out **one** of the following operations **exactly once** for each tower while making sure that its height remains **non-negative**

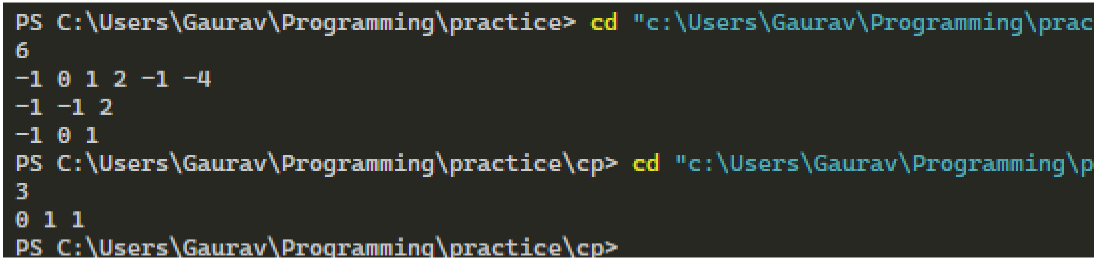
- ➔ **Increase** the height of the tower by **K**
- ➔ **Decrease** the height of the tower by **K**

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



```
#include<bits/stdc++.h>
using namespace std;

int partition(int a[],int l,int r){
    int pivot=a[r];
    int max=l;
    for(int i=l;i<r;i++){
        if(a[i]<=pivot){
            swap(a[i],a[max]);
            max++;
        }
    }
    swap(a[max],a[r]);
    return max;
}
int random_paritition(int a[],int l,int r){
    srand(time(NULL));
    int random=l+(rand()%(r-l));
    swap(a[random],a[r]);
    return partition(a,l,r);
}
void quicksort(int a[],int l,int r){
    if(l<r){
        int p=random_paritition(a,l,r);
        quicksort(a,l,p-1);
        quicksort(a,p+1,r);
    }
}
int main(){
    int k,n;
    cin>>k>>n;
    int a[n];
    for(int i=0;i<n;i++)
    cin>>a[i];
    quicksort(a,0,n-1);
    int ans=a[n-1]-a[0];
    for(int i=0;i<n-1;i++){
        if(a[i+1]-k<0)
        continue;
        int t1=min(a[0]+k,a[i+1]-k);
        int t2=max(a[n-1]-k,a[i]+k);
        ans=min(ans,t2-t1);
    }
    cout<<ans<<"\n";


    return 0;
}
```

Output:



```
PS C:\Users\Gaurav\Programming\practice> cd "c:\Users\Gaurav\Programming\prac
6
-1 0 1 2 -1 -4
-1 -1 2
-1 0 1
PS C:\Users\Gaurav\Programming\practice\cp> cd "c:\Users\Gaurav\Programming\p
3
0 1 1
PS C:\Users\Gaurav\Programming\practice\cp>
```

Time complexity analysis:

Average Time complexity of the randomized quicksort is O(nlogn)

Time complexity for the for loop is O(n)

=>Time complexity=O(nlogn)