Nitin Chaudhary
9921103163
F7
APS Lab Week 3

Q.1.Cubic integer root x of n is largest number x such that x3<=n. Find the value of x given n using divide and conquer approach. Also analyse the complexity.

```
#include <iostream>
using namespace std;
// int tellCuberoot(int n,int end){
//     int mid=end/2;
//     cout<<mid<<endl;
//     if(mid*mid*mid<=n&&(mid+1)*(mid+1)*(mid+1)>n)return mid;
//     else if((mid+1)*(mid+1)*(mid+1)==n)return (mid+1);//check for n=27
//     else if((mid-1)*(mid-1)*(mid-1)==n)return (mid-1);
//     else if(mid*mid*mid>n)return tellCuberoot(n,mid-1);
//     else {
//         return tellCuberoot(n,mid+1);
//     }
// }

int tellCuberoot(int n, int start,int end){
    int mid=(start+end)/2;
    if(mid*mid*mid<=n&&(mid+1)*(mid+1)*(mid+1)>n)return mid;
    else if(mid*mid*mid>n)return tellCuberoot(n,start,mid-1);
    else return tellCuberoot(n,mid+1,end);
}
int main()
{
    cout<<"Enter n to get the cube root of that number\n";
    int n;
    cin>>n;
    if(n==1||n==0)cout<<"The cube root is:"<<n;
    else{
```

```
        int cuberoot=tellCuberoot(n,0,n-1);
        cout<<"The cube root x of the given number such that x*x*x<=n is:"<<cuberoot;
    }
    return 0;
}
```

```
Enter n to get the cube root of that number
27
The cube root x of the given number such that x*x*x<=n is:3

...Program finished with exit code 0
Press ENTER to exit console.
```

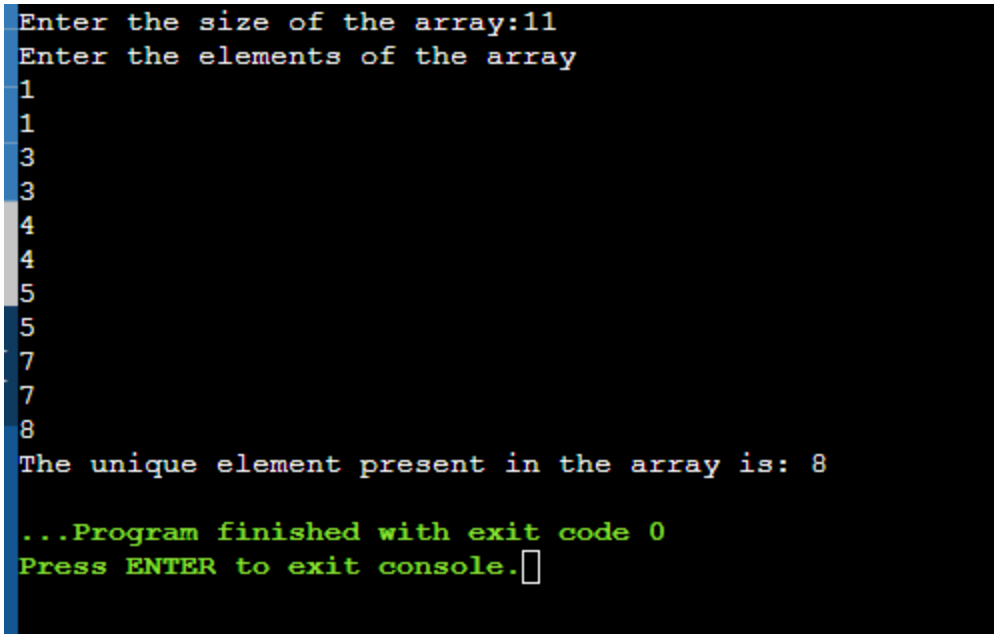
```
Enter n to get the cube root of that number
0
The cube root is:0

...Program finished with exit code 0
Press ENTER to exit console.
```

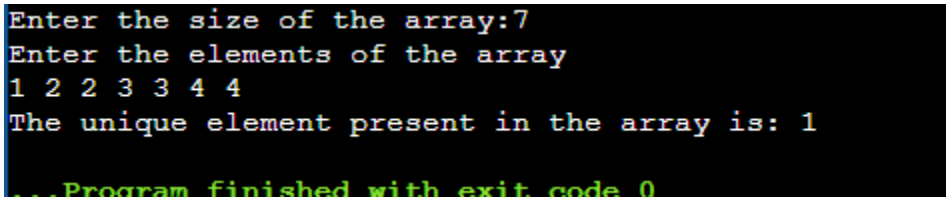
```
Enter n to get the cube root of that number
34
The cube root x of the given number such that x*x*x<=n is:3

...Program finished with exit code 0
Press ENTER to exit console.
```

As I am using binary Search here, hence the time complexity will be O(log n)

Q2. Given a sorted array in which all elements appear twice (one after one) and one element appears only once. Find that element in O(log n) complexity. Example: Input: arr[] = {1, 1, 3, 3, 4, 5, 5, 7, 7, 8, 8} Output: 4 Input: arr[] = {1, 1, 3, 3, 4, 4, 5, 5, 7, 7, 8} Output: 8

```
#include <iostream>
using namespace std;
int findUnique(int arr[],int start,int end){
    if(start<=end){
        int mid=(start+end)/2;
        if(arr[mid]!=arr[mid-1]&&arr[mid]!=arr[mid+1])return mid;
        else if(mid%2!=0){
            if(arr[mid]==arr[mid+1])return findUnique(arr,start,mid-1);
            else return findUnique(arr,mid+1,end);
        }
        else{
            if(arr[mid]==arr[mid-1])return findUnique(arr,start,mid-1);
            else return findUnique(arr,mid+1,end);
        }
    }
}
int main()
{
   cout<<"Enter the size of the array:";
   int size;
   cin>>size;
   cout<<"Enter the elements of the array\n";
   int arr[size];
   for(int i=0;i<size;i++)cin>>arr[i];
   int index=findUnique(arr,0,size);
   cout<<"The unique element present in the array is: "<<arr[index];
    return 0;
}
```





Q3. List of points have been given on 2D Plane. Calculate K closest points to the origin (0,0) (Consider euclidean distance to find the distance between two points). Write a code to return the answer in any order. The solution is guaranteed to be unique.

Example 1:

```
Input: points = [[1,3],[-2,2]], K = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest K = 1 points from the origin, so the answer is just
[[-2,2]].
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], K = 2
Output: [[3,3],[-2,4]]
(The answer [[-2,4],[3,3]] would also be accepted.)
```

Note:

1. 1 <= K <= points.length <= 10000
2. -10000 < points[i][0] < 10000
3. -10000 < points[i][1] < 10000

Code:

Firstly I am writing a programme to find the closest distance between two points in a plane then I will implement the approach asked in question-

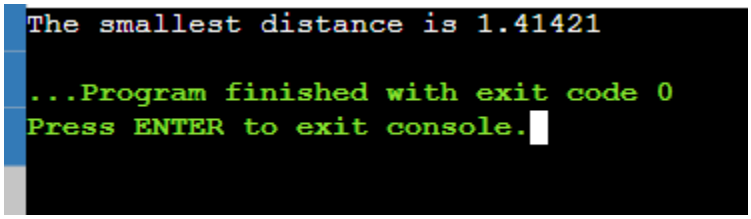
```
#include <bits/stdc++.h>
using namespace std;
class Point
{
    public:
        int x, y;
};
int compareX(const void* a, const void* b)
```

```
{
	Point *p1 = (Point *)a, *p2 = (Point *)b;
	return (p1->x != p2->x) ? (p1->x - p2->x) : (p1->y - p2->y);
}
int compareY(const void* a, const void* b)
{
	Point *p1 = (Point *)a, *p2 = (Point *)b;
	return (p1->y != p2->y) ? (p1->y - p2->y) : (p1->x - p2->x);
}

float dist(Point p1, Point p2)
{
	return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
				(p1.y - p2.y)*(p1.y - p2.y)
			);
}
float bruteForce(Point P[], int n)
{
	float min = FLT_MAX;
	for (int i = 0; i < n; ++i)
		for (int j = i+1; j < n; ++j)
			if (dist(P[i], P[j]) < min)
				min = dist(P[i], P[j]);
	return min;
}

float min(float x, float y)
{
	return (x < y)? x : y;
}
float stripClosest(Point strip[], int size, float d)
{
	float min = d;
	for (int i = 0; i < size; ++i)
		for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
			if (dist(strip[i],strip[j]) < min)
				min = dist(strip[i], strip[j]);

	return min;
}
```

```
float closestUtil(Point Px[], Point Py[], int n)
{
	if (n <= 3)
		return bruteForce(Px, n);
	int mid = n/2;
	Point midPoint = Px[mid];
	Point Pyl[mid];
	Point Pyr[n-mid];
	int li = 0, ri = 0;
	for (int i = 0; i < n; i++)
	{
	if ((Py[i].x < midPoint.x || (Py[i].x == midPoint.x && Py[i].y < midPoint.y)) &&
li<mid)
		Pyl[li++] = Py[i];
	else
		Pyr[ri++] = Py[i];
	}
	float dl = closestUtil(Px, Pyl, mid);
	float dr = closestUtil(Px + mid, Pyr, n-mid);
	float d = min(dl, dr);
	Point strip[n];
	int j = 0;
	for (int i = 0; i < n; i++)
		if (abs(Py[i].x - midPoint.x) < d)
			strip[j] = Py[i], j++;
	return stripClosest(strip, j, d);
}
float closest(Point P[], int n)
{
	Point Px[n];
	Point Py[n];
	for (int i = 0; i < n; i++)
	{
		Px[i] = P[i];
		Py[i] = P[i];
	}

	qsort(Px, n, sizeof(Point), compareX);
	qsort(Py, n, sizeof(Point), compareY);
```

```
        return closestUtil(Px, Py, n);
}
int main()
{
        Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
        int n = sizeof(P) / sizeof(P[0]);
        cout << "The smallest distance is " << closest(P, n);
        return 0;
}
```

Output-

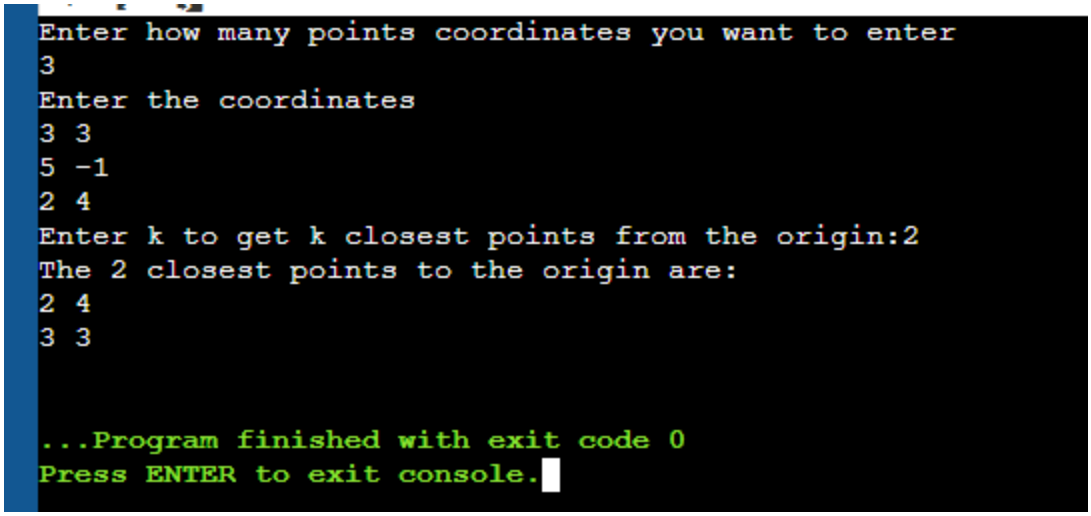


Now the below programme is as per question requirement-

```
#include<bits/stdc++.h>
using namespace std;
//This is a very good approach to find k points closest to origin.
//We have used the property of max heap which is used to implement the
priority_queue.
//Firstly I store all distances along with the coordinates of the points and then I has
removed all above points except k points.
//The k poits left in the priority_queue will be our k closest points as per the property of
the priority_queue.
void kClosest(vector<vector<int>>& points, int k) {
        priority_queue<pair<int,pair<int,int>>> pq;
        for(int i=0;i<points.size();i++){
            int sq = points[i][0]*points[i][0]  + points[i][1]*points[i][1];
            pq.push({sq,{points[i][0],points[i][1]}});
        }
        int n = points.size()-k;
        while(n--){
            pq.pop();
```

```
        }
        while(!pq.empty()){
            pair<int,pair<int,int>> p = pq.top();
            pq.pop();
            cout<<p.second.first<<" "<<p.second.second<<endl;
        }
}
int main(){
    vector<vector<int>> v;
    cout<<"Enter how many points coordinates you want to enter\n";
    int row;
    cin>>row;
    cout<<"Enter the coordinates\n";
    while(row--){
        vector<int>temp;
        int x,y;
        cin>>x>>y;
        temp.push_back(x);
        temp.push_back(y);
        v.push_back(temp);
    }
    cout<<"Enter k to get k closest points from the origin:";
    int k;
    cin>>k;
    cout<<"The "<<k<<" closest points to the origin are:\n";
    kClosest(v,k);
    return 0;
}
```

```
Enter how many points coordinates you want to enter
2
Enter the coordinates
1 3
-2 2
Enter k to get k closest points from the origin:1
The 1 closest points to the origin are:
-2 2
```



Q4. Let there be an array of N random elements. We need to sort this array in ascending order. If n is very large (i.e. N= 1,00,000) then Quicksort may be considered as the fastest algorithm to sort this array. However, we can further optimize its performance by hybridizing it with insertion sort. Therefore, if n is small (i.e. N<= 10) then we apply insertion sort to the array otherwise Quick Sort is applied. Implement the above discussed hybridized Quick Sort and compare the running time of normal Quick sort and hybridized quick sort. Run each type of sorting 10 times on a random set of inputs and compare the average time returned by these algorithms.

//Simple hybridized code is given below:

```
#include <iostream>
using namespace std;
void insertionSort(int arr[],int low,int high)
{
    int i, key, j;
    for (i = low+1; i < high; i++)
    {
        key = arr[i];
```

```
        j = i - 1;
        while (j >= low && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
 int partition (int arr[], int low, int high)
    {
        //remember the indexing of i and j
        //this code is understood by me from abdul bari youtube
        int i=low-1;
        int j=high+1;
        int pivot=arr[low];
        while(i<j){
        do{
           i++;
        }while(arr[i]<=pivot);
        do{
            j--;
        }while(arr[j]>pivot);
        if(i<j)swap(arr[i],arr[j]);
        }
        swap(arr[j],arr[low]);
        return j;
    }
void hybridQuickSort(int arr[], int low, int high)
    {
        if(low<high){
            if(high-low<=10)insertionSort(arr,low,high);
            else{
            int pi=partition(arr,low,high);
            hybridQuickSort(arr,low,pi-1);
            hybridQuickSort(arr,pi+1,high);
            }
        }
    }
```

```
int main()
{
   cout<<"This is the programme to sort the give array by using hybridized quick sort";
   cout<<"\nEnter the size of the array";
   int size;
   cin>>size;
   cout<<"Enter the elements of the array\n";
   int arr[size];
   for(int i=0;i<size;i++){
      cin>>arr[i];
   }
   hybridQuickSort(arr,0,size);
   cout<<"The array after sorting using hybridized quick sort is:\n";
   for(int i=0;i<size;i++)cout<<arr[i]<<" ";
   return 0;}
```

```
This is the programme to sort the give array by using hybridized quick sort
Enter the size of the array15
Enter the elements of the array
2  3 1 5 4 6 7 8 10 9 12 11 14 13 15
The array after sorting using hybridized quick sort is:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
This is the programme to sort the give array by using hybridized quick sort
Enter the size of the array20
Enter the elements of the array
1 2 3 4 9 8 7 6 5 10 11 12 13 20 19 18 17 16 15 14
The array after sorting using hybridized quick sort is:
0 1 2 3 4 6 7 8 5 9 11 12 13 14 15 16 17 18 19 20

...Program finished with exit code 0
Press ENTER to exit console.
```

//For Average time complexity analysis-

```
#include<bits/stdc++.h>
#include <sys/time.h>
using namespace std;
void insertionSort(int arr[],int low,int high)
{
    int i, key, j;
    for (i = low+1; i < high; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= low && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
 int partition (int arr[], int low, int high)
    {
        //remember the indexing of i and j
        //this code is understood by me from abdul bari youtube
        int i=low-1;
        int j=high+1;
        int pivot=arr[low];
        while(i<j){
        do{
           i++;
        }while(arr[i]<=pivot);
        do{
            j--;
        }while(arr[j]>pivot);
        if(i<j)swap(arr[i],arr[j]);
        }
        swap(arr[j],arr[low]);
        return j;
    }
    void quickSort(int arr[], int low, int high)
    {
        if(low<high){
```

```
            if(high-low<=10)insertionSort(arr,low,high);
            else{
            int pi=partition(arr,low,high);
            quickSort(arr,low,pi-1);
            quickSort(arr,pi+1,high);
            }
        }
    }

void hybridQuickSort(int arr[], int low, int high)
    {
        if(low<high){
            if(high-low<=10)insertionSort(arr,low,high);
            else{
            int pi=partition(arr,low,high);
            hybridQuickSort(arr,low,pi-1);
            hybridQuickSort(arr,pi+1,high);
            }
        }
    }



int main()
{
    cout<<"This is the programme to compare the average time complexity by sorting the
give array by using hybridized quick sort, insertion sort and simple quick sort ";
    double timeHybrid=0;
    double timeQuick=0;
    double timeInsertion=0;

    for(int k=0;k<10;k++){
    int arr1[100];
    int arr2[100];
    int arr3[100];
    for(int i=0;i<100;i++){
        int num=rand()%500;
        arr1[i]=num;
        arr2[i]=num;
        arr3[i]=num;
```

```
    }
    cout<<"The randomly generated array is:\n";

    for(int i=0;i<100;i++)cout<<arr1[i]<<" ";
struct timeval stop, start;
gettimeofday(&start, NULL);
hybridQuickSort(arr1,0,100);
gettimeofday(&stop, NULL);
cout<<"\nTime taken by hybrid quick sort :"<<k<<"th time :"<<(stop.tv_sec - start.tv_sec)
* 1000000 + stop.tv_usec - start.tv_usec<<" us";
timeHybrid+=((stop.tv_sec - start.tv_sec) * 1000000 + stop.tv_usec - start.tv_usec);
gettimeofday(&start, NULL);
insertionSort(arr2,0,99);
gettimeofday(&stop, NULL);
cout<<"\nTime taken by insertion sort"<<k<<"th time :"<<(stop.tv_sec - start.tv_sec) *
1000000 + stop.tv_usec - start.tv_usec<<" us";
timeInsertion+=((stop.tv_sec - start.tv_sec) * 1000000 + stop.tv_usec - start.tv_usec);

gettimeofday(&start, NULL);
quickSort(arr3,0,100);
gettimeofday(&stop, NULL);
cout<<"\nTime taken by quick sort"<<k<<"th time :"<<(stop.tv_sec - start.tv_sec) *
1000000 + stop.tv_usec - start.tv_usec<<" us";
timeQuick+=((stop.tv_sec - start.tv_sec) * 1000000 + stop.tv_usec - start.tv_usec);
cout<<"The array after sorting:\n";
for(int i=0;i<100;i++)cout<<arr1[i]<<" ";
}
cout<<"The average running time complexity of hybridQuickSort by running it 10 times
is:"<<timeHybrid/10.0<<"us\n";
cout<<"The average running time complexity of insertion sort by running it 10 times
is:"<<timeInsertion/10.0<<"us\n";
cout<<"The average running time complexity of quick sort by running it 10 times
is:"<<timeQuick/10.0<<"us\n";
    return 0;
}
```

Output-

```
This is the programme to compare the average time complexity by sorting the give array by using hybridized quick sort, insertion sort and s
imple quick sort The randomly generated array is:
383 386 277 415 293 335 386 492 149 421 362 27 190 59 263 426 40 426 172 236 211 368 67 429 282 30 362 123 67 135 429 302 22 58 69 167 393
456 11 42 229 373 421 419 284 37 198 324 315 370 413 26 91 480 456 373 362 170 496 281 305 425 84 327 336 5 346 229 313 357 124 395 82 45 3
14 367 434 364 43 250 87 308 276 178 288 84 403 151 254 399 432 60 176 368 239 12 226 86 94 39
Time taken by hybrid quick sort :0th time :7 us
Time taken by insertion sort0th time :11 us
Time taken by quick sort0th time :7 usThe array after sorting:
5 11 22 26 27 30 37 39 40 42 12 43 45 59 58 60 67 67 69 82 84 84 86 87 91 123 124 135 149 151 167 94 170 176 178 172 190 198 211 226 229 22
9 236 239 250 254 263 277 281 282 284 288 276 293 302 305 308 313 314 315 324 327 335 346 336 357 362 362 362 364 367 368 368 370 373 373 3
83 383 386 386 393 395 399 403 413 415 419 421 421 425 426 426 429 429 432 434 456 456 480 492 The randomly generated array is:
295 70 434 378 467 101 97 402 317 492 152 256 301 280 286 441 365 189 444 119 440 229 31 117 97 271 481 175 209 427 67 356 497 353 86 465 3
06 183 219 124 28 371 232 329 3 19 270 368 208 215 340 149 296 223 118 245 346 451 421 55 379 488 264 228 341 350 193 0 34 264 124 414 487
356 243 491 227 365 359 436 432 51 437 228 275 407 474 121 358 395 29 237 235 293 318 428 143 11 428 29
Time taken by hybrid quick sort :1th time :7 us
Time taken by insertion sort1th time :11 us
Time taken by quick sort1th time :6 usThe array after sorting:
0 11 19 28 29 29 3 31 34 51 55 67 70 86 97 97 101 117 118 119 121 124 124 149 152 175 143 183 189 193 208 209 215 219 227 228 223 228 229 2
32 235 237 245 243 256 264 270 271 275 280 286 293 295 264 295 296 301 306 317 318 329 340 341 346 350 353 356 358 359 365 365 356 368 378
379 395 371 402 407 414 421 427 428 428 432 434 437 440 441 444 451 436 465 467 474 481 487 488 492 497 The randomly generated array is:
276 404 443 263 113 38 106 340 404 318 128 188 369 417 417 496 324 243 470 183 490 499 272 225 144 90 5 139 454 286 169 82 42 464 197 7 355
 304 348 111 122 328 299 343 246 68 340 422 311 310 105 301 161 230 378 305 320 236 444 126 22 465 208 416 282 258 424 137 62 124 100 36 45
2 399 379 50 468 71 473 131 381 430 433 394 160 163 199 481 399 496 459 273 313 168 190 95 426 466 84 340
Time taken by hybrid quick sort :2th time :6 us
Time taken by insertion sort2th time :9 us
Time taken by quick sort2th time :6 usThe array after sorting:
5 7 22 36 38 50 62 68 42 71 82 90 95 100 84 105 106 111 113 122 126 128 124 131 139 137 144 160 161 163 168 169 183 188 190 197 199 208 230
 236 243 246 258 263 272 225 273 276 276 282 286 301 304 305 299 310 311 318 313 320 324 328 340 340 343 340 348 355 369 378 379 381 394 39
9 399 404 404 416 417 417 422 424 426 430 433 443 452 444 454 459 464 465 466 468 470 473 481 490 496 496 The randomly generated array is:
90 184 376 42 436 107 445 256 179 418 387 412 348 172 159 9 336 210 342 87 206 301 213 372 321 255 319 99 221 404 439 311 440 167 205 228 1
27 150 484 158 420 224 422 269 396 81 130 84 292 472 172 350 125 385 222 299 140 42 398 213 298 190 24 90 209 81 319 336 232 155 494 4 379
269 273 276 350 255 360 142 79 384 493 205 121 67 4 113 461 254 326 259 444 202 202 6 284 21 342 368
Time taken by hybrid quick sort :3th time :7 us
Time taken by insertion sort3th time :12 us
Time taken by quick sort3th time :6 usThe array after sorting:
4 4 6 9 21 42 24 42 79 81 81 67 84 87 90 90 90 99 107 113 121 125 127 130 140 150 142 155 158 159 167 172 179 184 190 202 202 205 205 206 1
72 209 210 213 213 221 222 224 228 254 255 255 256 259 269 232 269 273 276 284 292 298 301 311 319 299 319 321 326 336 336 342 342 348 350
350 360 368 372 376 379 384 385 396 398 404 412 418 420 422 436 439 387 440 444 461 472 445 484 493 The randomly generated array is:
28 189 372 408 458 498 36 308 253 248 303 333 133 148 390 254 67 246 368 29 0 46 288 297 249 490 303 33 363 497 253 392 186 125 152 496 475
 188 157 229 436 460 414 421 460 304 28 27 50 248 56 402 294 197 199 43 39 2 428 403 0 181 147 38 159 151 35 134 339 192 215 127 4 129 49 4
64 285 429 343 335 177 400 238 471 449 289 367 488 292 295 243 144 329 390 182 340 41 69 326 232
Time taken by hybrid quick sort :4th time :6 us
Time taken by insertion sort4th time :11 us
Time taken by quick sort4th time :7 usThe array after sorting:
0 0 2 4 27 28 28 28 29 35 36 38 33 39 41 43 49 50 56 67 69 46 125 127 129 134 144 133 147 148 151 152 157 159 177 181 182 186 188 192 189 1
97 215 229 232 238 243 246 248 248 249 253 199 253 254 285 288 292 289 294 295 297 303 303 304 308 326 329 333 335 339 340 343 363 367 368
372 390 392 400 402 403 390 408 414 421 428 429 436 449 458 460 464 471 475 488 490 496 497 498 The randomly generated array is:
261 42 360 117 23 261 81 309 190 425 496 367 177 234 190 126 24 57 114 168 205 358 312 386 100 346 226 494 416 52 78 29 446 290 147 470 51
80 131 93 357 127 312 386 214 355 12 90 412 479 110 469 189 274 355 141 120 433 487 388 338 66 270 284 356 417 106 260 349 237 205 59 217 1
8 445 283 373 458 373 137 289 483 107 478 257 314 471 229 100 459 118 438 25 388 74 233 157 181 493 358
Time taken by hybrid quick sort :5th time :11 us
Time taken by insertion sort5th time :10 us
Time taken by quick sort5th time :7 usThe array after sorting:
```

```
Time taken by quick sort5th time :7 usThe array after sorting:
12 18 24 25 23 29 42 51 52 57 59 66 74 78 80 81 90 93 100 106 107 110 114 100 117 118 120 127 131 137 126 141 147 157 168 177 181 190 190 2
05 189 205 214 217 226 233 234 237 229 257 261 261 260 261 270 274 283 284 290 309 312 289 312 314 338 346 349 355 355 356 357 358 358 360
367 373 373 386 386 388 388 412 416 417 425 433 445 438 446 458 459 469 470 471 478 483 487 493 494 496 The randomly generated array is:
270 199 417 339 69 363 122 294 173 347 431 462 182 390 292 291 57 115 21 157 74 491 447 451 231 21 37 240 54 30 98 325 81 16 16 2 231 139 2
96 404 338 80 218 21 470 362 312 379 477 185 36 404 176 483 207 259 357 244 499 411 127 450 236 60 318 105 63 49 244 211 305 434 291 375 45
5 114 89 268 493 418 305 382 322 482 217 30 93 74 126 93 486 253 43 74 314 213 179 377 262 275
Time taken by hybrid quick sort :6th time :6 us
Time taken by insertion sort6th time :11 us
Time taken by quick sort6th time :6 usThe array after sorting:
2 16 16 21 21 21 30 30 36 43 49 54 57 60 63 69 74 74 37 74 80 89 93 93 98 105 114 115 122 81 126 127 139 157 173 179 182 185 199 207 211 17
6 213 217 218 231 231 236 244 240 244 253 262 268 270 259 270 275 291 292 294 291 296 305 305 312 314 318 322 325 338 347 339 357 362 363 3
75 377 382 390 404 404 411 417 418 379 431 447 450 451 434 455 462 470 477 482 483 486 491 499 The randomly generated array is:
88 419 210 232 294 17 346 235 137 191 153 443 73 328 425 291 210 18 217 336 463 55 90 358 130 404 71 161 133 185 289 73 104 351 305 250 368
 3 485 6 195 139 449 120 467 226 263 177 96 481 365 60 36 455 270 18 211 342 32 196 379 321 270 484 172 427 234 40 283 72 398 330 63 347 45
0 30 73 214 59 22 47 424 82 435 232 204 454 443 398 486 140 278 159 262 262 183 41 348 223 324
Time taken by hybrid quick sort :7th time :6 us
Time taken by insertion sort7th time :10 us
Time taken by quick sort7th time :6 usThe array after sorting:
3 6 17 18 18 22 32 30 36 40 41 47 59 60 63 55 71 72 73 73 82 88 73 88 90 96 120 104 130 133 137 139 140 153 159 161 172 183 177 185 191 195
 196 210 204 210 211 214 217 223 232 232 226 234 235 250 262 262 263 270 270 278 283 289 294 305 291 321 324 328 330 336 342 347 348 351 34
6 358 365 368 379 398 398 404 419 424 427 435 443 443 425 449 450 454 455 463 481 484 485 486 The randomly generated array is:
272 122 154 335 321 457 365 247 171 276 269 218 201 203 153 433 407 459 228 306 297 220 84 308 334 198 491 376 398 215 52 171 189 59 6 10 1
6 224 109 39 0 378 109 53 81 114 338 489 426 67 147 223 287 231 32 122 281 375 350 179 90 254 350 131 313 357 494 181 81 103 220 433 482 18
1 487 415 296 325 404 222 392 51 297 32 134 181 6 415 57 208 95 499 462 297 483 276 154 477 309 87
Time taken by hybrid quick sort :8th time :6 us
Time taken by insertion sort8th time :9 us
Time taken by quick sort8th time :7 usThe array after sorting:
6 0 6 10 16 32 32 39 51 53 52 57 59 67 81 81 84 90 95 103 109 109 114 87 122 122 131 134 147 153 154 171 171 154 179 181 181 181 189 198 20
1 208 215 218 220 203 220 222 223 224 228 231 247 254 269 272 272 276 276 281 287 296 297 297 297 308 309 313 321 325 306 334 335 338 350 3
```

```
Time taken by quick sort8th time :7 usThe array after sorting:
6 0 6 10 16 32 32 39 51 53 52 57 59 67 81 81 84 90 95 103 109 109 114 87 122 122 131 134 147 153 154 171 171 154 179 181 181 181 189 198 20
1 208 215 218 220 203 220 222 223 224 228 231 247 254 269 272 272 276 276 281 287 296 297 297 297 308 309 313 321 325 306 334 335 338 350 3
57 365 350 375 376 378 392 398 404 407 415 415 426 433 433 459 462 477 457 482 483 487 489 494 499 The randomly generated array is:
432 382 21 266 63 360 182 211 185 86 285 430 490 83 314 476 116 320 392 25 28 339 25 490 136 360 118 143 337 428 82 121 310 455 388 225 315
 70 437 353 8 222 283 350 157 97 327 126 269 71 151 149 410 28 139 398 388 110 393 77 390 476 199 52 431 87 277 99 157 66 452 17 141 235 36
8 298 184 195 276 305 266 428 454 28 308 93 278 197 55 172 274 445 0 325 497 283 412 127 382 421
Time taken by hybrid quick sort :9th time :7 us
Time taken by insertion sort9th time :10 us
Time taken by quick sort9th time :6 usThe array after sorting:
0 8 17 21 25 28 28 25 28 52 55 63 66 71 77 82 70 83 86 93 97 99 110 116 118 121 126 87 127 136 141 143 139 149 157 151 157 182 172 184 185
195 197 199 211 222 235 266 266 269 225 274 276 277 278 283 285 298 305 308 310 283 314 315 320 325 327 337 339 350 353 360 360 368 382 382
 388 390 392 393 398 388 410 421 428 428 430 431 432 412 432 437 445 452 454 455 476 476 490 497 The average running time complexity of hyb
ridQuickSort by running it 10 times is:6.9us
The average running time complexity of insertion sort by running it 10 times is:10.4us
The average running time complexity of quick sort by running it 10 times is:6.4us


...Program finished with exit code 0
Press ENTER to exit console.
```

Q5. Consider a sorted array A of n elements. The array A may have repetitive/duplicate elements. For a given target element T, design and implement an efficient algorithm to find T's first and last occurrence in the array A. Also print the message if an element was not present in the array. For Example, Input: arr = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9] target = 5 Output: The first occurrence of element 5 is located at index 1 The last occurrence of element 5 is located at index 3 Input: arr = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9] target = 4 Output: Element not found in the array

```
#include <iostream>
```

```
using namespace std;
int firstBinarySearch(int arr[],int start,int end,int target){
    if(start<=end){
        int mid=(start+end)/2;
        if(mid!=0&&arr[mid]==target&&arr[mid-1]!=target)return mid;
        else if(mid==0&&arr[mid]==target)return mid;
        else if(arr[mid]>=target)return firstBinarySearch(arr,start,mid-1,target);
        else return firstBinarySearch(arr,mid+1,end,target);
    }
    else return -1;
    //return 0;
}
int lastBinarySearch(int arr[],int start,int end,int target){
    if(start<=end){
        int mid=(start+end)/2;
        if(mid!=end&&arr[mid]==target&&arr[mid+1]!=target)return mid;
        else if(mid==end&&arr[mid]==target)return mid;
        else if(arr[mid]<=target)return lastBinarySearch(arr,mid+1,end,target);
        else return lastBinarySearch(arr,start,mid-1,target);
    }
    else return -1;
    //return 0;
}
int main()
{
    cout<<"This is the programme to find the first and last occurrence of a number in a
sorted array";
    cout<<"\nEnter the size of the sorted array";
    int size;
    cin>>size;
    cout<<"Enter the elements of the array in sorted array";
    int arr[size];
    for(int i=0;i<size;i++){
        cin>>arr[i];
    }
    int target;
    cout<<"Enter the target value";
    cin>>target;
    int first=firstBinarySearch(arr,0,size-1,target);
```

```
    int last=lastBinarySearch(arr,0,size-1,target);
    if(first==-1||last==-1){
        cout<<"Element not found in the give array";
        return 0;
    }
    cout<<"\nThe first occurrence of the "<<target<<" is "<<first+1;
    cout<<"\nThe last occurrence of the "<<target<<" is "<<last+1;

    return 0;
}
```