SEARCHING

1. ITERATIVE BINARY SEARCH

```
#include <iostream>
using namespace std;


int bSearch(int arr[], int n, int x)
{
        int low = 0, high = n - 1;

        while(low <= high)
        {
                int mid = (low + high) / 2;

                if(arr[mid] == x)
                        return mid;

                else if(arr[mid] > x)
                        high = mid - 1;

                else
                        low = mid + 1;
        }

        return -1;
}

int main() {

    int arr[] = {10, 20, 30, 40, 50, 60}, n = 6;
```

```
        int x = 25;

        cout<<bSearch(arr, n, x);
        return 0;
}
```

2. RECURSIVE BINARY SEARCH

```
#include <iostream>
using namespace std;


int bSearch(int arr[], int low, int high, int x)
{
        if(low > high)
                return -1;

        int mid = (low + high) / 2;

        if(arr[mid] == x)
                return mid;

        else if(arr[mid] > x)
                return bSearch(arr, low, mid - 1, x);

        else
                return bSearch(arr, mid + 1, high, x);
}

int main() {

    int arr[] = {10, 20, 30, 40, 50, 60, 70}, n = 7;

        int x = 20;
```

```
        cout<<bSearch(arr, 0, n - 1, x);
        return 0;
}
```

3. INDEX OF FIRST OCCURRENCE IN SORTED

ITERATIVE

```
#include <iostream>
using namespace std;


int firstOcc(int arr[], int n, int x)
{
        int low = 0, high = n - 1;

        while(low <= high)
        {
                int mid = (low + high) / 2;

                if(x > arr[mid])
                        low = mid + 1;

                else if(x < arr[mid])
                        high = mid - 1;

                else
                {
                        if(mid == 0 || arr[mid - 1] != arr[mid])
                                return mid;

                        else
                                high = mid - 1;
```

```
            }

        }

        return -1;
}

int main() {

    int arr[] = {5, 10, 10, 10, 20}, n = 5;

        int x = 10;

        cout<<firstOcc(arr, n, x);
        return 0;
}
```

RECURSIVE

```
#include <iostream>
using namespace std;


int firstOcc(int arr[], int low, int high, int x)
{
        if(low > high)
                return -1;

        int mid = (low + high) / 2;

        if(x > arr[mid])
                return firstOcc(arr, mid + 1, high, x);

        else if(x < arr[mid])
                return firstOcc(arr, low, mid - 1, x);
```

```
        else
        {
                if(mid == 0 || arr[mid - 1] != arr[mid])
                        return mid;

                else
                        return firstOcc(arr, low, mid - 1, x);
        }
}

int main() {

    int arr[] = {5, 10, 10, 15, 20, 20, 20}, n = 7;

        int x = 20;

        cout<<firstOcc(arr, 0, n - 1, x);
        return 0;
}
```

4. INDEX OF LAST OCCURRENCE IN SORTED

ITERATIVE

```
#include <iostream>
using namespace std;


int lastOcc(int arr[], int n, int x)
{
        int low = 0, high = n - 1;

        while(low <= high)
```

```
        {
                int mid = (low + high) / 2;

                if(x > arr[mid])
                        low = mid + 1;

                else if(x < arr[mid])
                        high = mid - 1;

                else
                {
                        if(mid == n - 1 || arr[mid + 1] != arr[mid])
                                return mid;

                        else
                                low = mid + 1;
                }

        }

        return -1;
}


int main() {

   int arr[] = {5, 10, 10, 10, 10, 20, 20}, n = 7;

        int x = 10;

    cout << lastOcc(arr, n, x);
        return 0;
}
```

RECURSIVE

```
#include <iostream>
using namespace std;


int lastOcc(int arr[], int low, int high, int x, int n)
{
	if(low > high)
		return -1;

	int mid = (low + high) / 2;

	if(x > arr[mid])
		return lastOcc(arr, mid + 1, high, x, n);

	else if(x < arr[mid])
		return lastOcc(arr, low, mid - 1, x, n);

	else
	{
		if(mid == n - 1 || arr[mid + 1] != arr[mid])
			return mid;

		else
			return lastOcc(arr, mid + 1, high, x, n);
	}
}


int main() {

   int arr[] = {5, 10, 10, 10, 10, 20, 20}, n = 7;

	int x = 10;

    cout << lastOcc(arr, 0, n - 1, x, n);
```

```
        return 0;
}
```

5. COUNT OCCURRENCES IN SORTED

```
#include <iostream>
using namespace std;

int firstOcc(int arr[], int n, int x)
{
        int low = 0, high = n - 1;

        while(low <= high)
        {
                int mid = (low + high) / 2;

                if(x > arr[mid])
                        low = mid + 1;

                else if(x < arr[mid])
                        high = mid - 1;

                else
                {
                        if(mid == 0 || arr[mid - 1] != arr[mid])
                                return mid;

                        else
                                high = mid - 1;
                }

        }

        return -1;
```

```
}

int lastOcc(int arr[], int n, int x)
{
	int low = 0, high = n - 1;

	while(low <= high)
	{
		int mid = (low + high) / 2;

		if(x > arr[mid])
			low = mid + 1;

		else if(x < arr[mid])
			high = mid - 1;

		else
		{
			if(mid == n - 1 || arr[mid + 1] != arr[mid])
				return mid;

			else
				low = mid + 1;
		}

	}

	return -1;
}

int countOcc(int arr[], int n, int x)
{
	int first = firstOcc(arr, n, x);

	if(first == -1)
```

```
            return 0;
        else
            return lastOcc(arr, n, x) - first + 1;
}

int main() {

  int arr[] = {10, 20, 20, 20, 40, 40}, n = 6;

  int x = 20;

  cout << countOcc(arr, n, x);

        return 0;
}
```

6. COUNT 1s IN A BINARY SORTED ARRAY

```
#include <iostream>
using namespace std;

int countOnes(int arr[], int n)
{
        int low = 0, high = n - 1;

        while(low <= high)
        {
            int mid = (low + high) / 2;

            if(arr[mid] == 0)
                low = mid + 1;
            else
            {
                if(mid == 0 || arr[mid - 1] == 0)
```

```
                return (n - mid);
            else
                high = mid -1;
        }
    }

    return 0;
}

int main() {

  int arr[] = {0, 0, 1, 1, 1, 1}, n = 6;

  cout << countOnes(arr, n);

    return 0;
}
```

7. SQUARE ROOT

```
#include <iostream>
using namespace std;

int sqRootFloor(int x)
{
    int low = 1, high = x, ans = -1;

    while(low <= high)
    {
        int mid = (low + high) / 2;

        int mSq = mid * mid;

        if(mSq == x)
```

```
                return mid;
        else if(mSq > x)
                high = mid - 1;
        else
        {
                low = mid + 1;
                ans = mid;
        }
    }

    return ans;
}

int main() {


  cout << sqRootFloor(10);

    return 0;
}
```

8. SEARCH IN INFINITE SIZED ARRAY

```
#include <iostream>
using namespace std;

int bSearch(int arr[], int low, int high, int x)
{
    if(low > high)
        return -1;

    int mid = (low + high) / 2;

    if(arr[mid] == x)
```

```
            return mid;

        else if(arr[mid] > x)
            return bSearch(arr, low, mid - 1, x);

        else
            return bSearch(arr, mid + 1, high, x);
}


int search(int arr[], int x)
{
    if(arr[0] == x) return 0;

        int i = 1;

        while(arr[i] < x)
            i = i * 2;

        if(arr[i] == x) return i;


        return bSearch(arr, i / 2 + 1, i - 1, x);
}

int main() {


   int arr[] = {1, 2, 3, 5, 5};

        int x = 4;

        cout << search(arr, x);

        return 0;
```

```
}
```

9. SEARCH IN SORTED ROTATED ARRAY

```
#include <iostream>
using namespace std;



int search(int arr[], int n, int x)
{
	int low = 0, high = n - 1;

	while(low <= high)
	{
		int mid = (low + high) / 2;

		if(arr[mid] == x)
			return mid;
		if(arr[low] < arr[mid])
		{
			if(x >= arr[low] && x < arr[mid])
				high = mid - 1;
			else
				low = mid + 1;
		}
		else
		{
			if(x > arr[mid] && x <= arr[high])
				low = mid + 1;
			else
				high = mid - 1;
		}
	}
```

```
        return -1;
}

int main() {


  int arr[] = {10, 20, 40, 60, 5, 8}, n = 6;

  int x = 5;

        cout << search(arr, n, x);

        return 0;
}
```

10. FIND A PEAK ELEMENT

```
#include <iostream>
using namespace std;



int getPeak(int arr[], int n)
{
        int low = 0, high = n - 1;

                while(low <= high)
                {
                        int mid = (low + high) / 2;

                        if((mid == 0 || arr[mid - 1] <= arr[mid]) &&
                                (mid == n - 1 || arr[mid + 1] <= arr[mid]))
```

```
                return mid;
            if(mid > 0 && arr[mid - 1] >= arr[mid])
                high = mid -1;
            else
                low = mid + 1;
        }
    return -1;
}

int main() {


 int arr[] = {5, 20, 40, 30, 20, 50, 60}, n = 7;

 cout << getPeak(arr, n);

    return 0;
}
```

11. Find pair in unsorted array which gives sum X

```
// C++ program to check if given array
// has 2 elements whose sum is equal
// to the given value
#include <bits/stdc++.h>

using namespace std;

void printPairs(int arr[], int arr_size, int sum)
{
    unordered_set<int> s;
    for (int i = 0; i < arr_size; i++) {
        int temp = sum - arr[i];
```

```
                if (s.find(temp) != s.end())
                        cout << "Pair with given sum " << sum << " is (" << arr[i]
<< ", " << temp << ")" << endl;

                s.insert(arr[i]);
        }
}

/* Driver program to test above function */
int main()
{
        int A[] = { 1, 4, 45, 6, 10, 8 };
        int n = 16;
        int arr_size = sizeof(A) / sizeof(A[0]);

        // Function calling
        printPairs(A, arr_size, n);

        return 0;
}
```

12. Find pair in sorted array which gives sum X

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

bool isPresent(int arr[], int n, int sum)
{
   int l = 0, h = n-1;
   int mid;
   while(l <= h)
   {
      if(arr[l] + arr[h] == sum)
        return true;
```

```
        else if(arr[l] + arr[h] > sum)
            h--;
        else l++;
    }
    return false;
}

int main()
{
    int arr[] = {2, 3, 7, 8, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = 14;

    cout << isPresent(arr, n, sum);
}
```

13. Find triplet in an array which gives sum X

```
// C++ program to find a triplet
#include <bits/stdc++.h>
using namespace std;

// returns true if there is triplet with sum equal
// to 'sum' present in A[]. Also, prints the triplet
bool find3Numbers(int A[], int arr_size, int sum)
{
        int l, r;

        /* Sort the elements */
        sort(A, A + arr_size);

        /* Now fix the first element one by one and find the
        other two elements */
        for (int i = 0; i < arr_size - 2; i++) {
```

```
                // To find the other two elements, start two index
                // variables from two corners of the array and move
                // them toward each other
                l = i + 1; // index of the first element in the
                // remaining elements

                r = arr_size - 1; // index of the last element
                while (l < r) {
                        if (A[i] + A[l] + A[r] == sum) {
                                printf("Triplet is %d, %d, %d", A[i],
                                        A[l], A[r]);
                                return true;
                        }
                        else if (A[i] + A[l] + A[r] < sum)
                                l++;
                        else // A[i] + A[l] + A[r] > sum
                                r--;
                }
        }

        // If we reach here, then no triplet was found
        return false;
}

/* Driver program to test above function */
int main()
{
        int A[] = { 1, 4, 45, 6, 10, 8 };
        int sum = 22;
        int arr_size = sizeof(A) / sizeof(A[0]);

        find3Numbers(A, arr_size, sum);
```

```
        return 0;
}
```

14. MEDIAN OF TWO SORTED ARRAYS

```
#include <iostream>
#include <cmath>
#include <bits/stdc++.h>
#include <climits>
#include <deque>
using namespace std;



double getMed(int a1[], int a2[], int n1, int n2)
{
        int begin1 = 0, end1 = n1;

        while(begin1 <= end1)
        {
                int i1 = (begin1 + end1) / 2;
                int i2 = (n1 + n2 + 1) / 2 - i1;

                int min1 = (i1 == n1)?INT_MAX:a1[i1];
                int max1 = (i1 == 0)?INT_MIN:a1[i1 - 1];

                int min2 = (i2 == n2)?INT_MAX:a2[i2];
                int max2 = (i2 == 0)?INT_MIN:a2[i2 - 1];

                if(max1 <= min2 && max2 <= min1)
```

```
                {
                        if((n1 + n2) % 2 == 0)
                                return ((double)max(max1, max2) +
min(min1, min2)) / 2;
                        else
                                return (double)max(max1, max2);
                }
                else if(max1 > min2)
                        end1 = i1 - 1;
                else
                        begin1 = i1 + 1;
        }
}


int main() {


 int a1[] = {10, 20, 30, 40, 50}, n1 = 5, a2[] = {5, 15, 25, 35, 45}, n2 =
5;

cout << getMed(a1, a2, n1, n2);


        return 0;
}
```

15. REPEATING ELEMENTS

```
#include <iostream>
#include <cmath>
```

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



int repeat(int arr[], int n)
{
	bool visit[n];

	memset(visit, false, sizeof(visit));

	for(int i = 0; i < n; i++)
	{
		if(visit[arr[i]])
			return arr[i];
		visit[arr[i]] = true;
	}

	return -1;
}


int main() {


 int arr[] = {0, 2, 1, 3, 2, 2}, n= 6;

 cout << repeat(arr, n);

	return 0;
}
```

16. REPEATING ELEMENTS PART 2

```
#include <iostream>
#include <cmath>
#include <bits/stdc++.h>
#include <climits>
#include <deque>
using namespace std;



int repeat(int arr[], int n)
{
        int slow = arr[0], fast = arr[0];

        do{
                slow = arr[slow];
                fast = arr[arr[fast]];

        }while(slow != fast);

        slow = arr[0];

        while(slow != fast)
        {
                slow = arr[slow];
                fast = arr[fast];
        }
        return slow;
}

int main() {


 int arr[] = {1, 3, 2, 4, 6, 5, 7, 3}, n= 8;
```

```
 cout << repeat(arr, n);

        return 0;
}
```

17. ALLOCATE MINIMUM PAGES

NAIVE

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

int sum(int arr[],int b, int e){
    int s=0;
    for(int i=b;i<=e;i++)
        s+=arr[i];
    return s;
}

int minPages(int arr[],int n, int k){
    if(k==1)
        return sum(arr,0,n-1);
    if(n==1)
        return arr[0];
    int res=INT_MAX;
    for(int i=1;i<n;i++){
        res=min(res,max(minPages(arr,i,k-1),sum(arr,i,n-1)));
    }
    return res;
}

int main()
{
```

```
    int arr[]={10,20,10,30};
    int n=sizeof(arr)/sizeof(arr[0]);
    int k=2;

    cout<<minPages(arr,n,k);
}
```

BINARY SEARCH

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

bool isFeasible(int arr[],int n,int k, int ans){
    int req=1,sum=0;
    for(int i=0;i<n;i++){
        if(sum+arr[i]>ans){
            req++;
            sum=arr[i];
        }
        else{
            sum+=arr[i];
        }
    }
    return (req<=k);
}

int minPages(int arr[],int n, int k){
    int sum=0,mx=0;
    for(int i=0;i<n;i++){
        sum+=arr[i];
        mx=max(mx,arr[i]);
    }
    int low=mx,high=sum,res=0;

    while(low<=high){
```

```
        int mid=(low+high)/2;
        if(isFeasible(arr,n,k,mid)){
            res=mid;
            high=mid-1;
        }else{
            low=mid+1;
        }
    }
    return res;
}

int main()
{
    int arr[]={10,20,10,30};
    int n=sizeof(arr)/sizeof(arr[0]);
    int k=2;

    cout<<minPages(arr,n,k);
}
```