Arrays

An array is a collection of items of the same data type stored at contiguous memory locations. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

Defining an Array: Array definition is similar to defining any other variable. There are two things that are needed to be kept in mind, **the data type of the array elements** and the **size** of the array. The size of the array is fixed and the memory for an array needs to be allocated before use, the size of an array cannot be increased or decreased dynamically.

Advantages of using arrays:

- Arrays allow random access of elements. This makes accessing elements by their position faster.
- Arrays have better cache locality that can make a pretty big difference in performance.

Time Complexity of this search operation will be O(N) in the worst case as we are checking every element of the array from 1st to last, so the number of operations is N.

Time Complexity of this insert operation is constant, i.e. O(1) as we are directly inserting the element in a single operation.

Time Complexity in worst case of this insertion operation can be linear i.e. O(N) as we might have to shift all of the elements by one place to the left.

Arrays can be classified into two categories:

1. Fixed-sized arrays
2. Dynamic-sized arrays

Operation on array

Insert:

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

int insert(int arr[], int n, int x, int cap, int pos)
{
       if(n == cap)
              return n;

       int idx = pos - 1;

       for(int i = n - 1; i >= idx; i--)
       {
              arr[i + 1] = arr[i];
       }

       arr[idx] = x;

       return n + 1;
}



int main() {

       int arr[5], cap = 5, n = 3;

       arr[0] = 5; arr[1] = 10; arr[2] = 20;

       cout<<"Before Insertion"<<endl;

       for(int i=0; i < n; i++)
       {
              cout<<arr[i]<<" ";
       }

       cout<<endl;

       int x = 7, pos = 2;

       n = insert(arr, n, x, cap, pos);

       cout<<"After Insertion"<<endl;
       for(int i=0; i < n; i++)
       {
                     cout<<arr[i]<<" ";
       }

}
```

OUTPUT: Before Insertion

5 10 20

After Insertion

5 7 10 20

Search in Unsorted array:

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

int search(int arr[],int n, int x)
{
    for(int i=0;i<n;i++)
    {
        if(arr[i]==x)
            return i;
    }
    return -1;
}

int main()
{
    int arr[] ={20,5,7,25}, x=25;
    cout<<search(arr,4,x);
}
```

OUTPUT:

3

Operation on array

Delete:

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

int deleteEle(int arr[],int n,int x)
{   int i;
    for(i=0;i<n;i++)
        if(arr[i]==x)
            break;

    if(i==n)
        return n;

    for(int j=i;j<n-1;j++)
        arr[j]=arr[j+1];

    return n-1;
}
int main()
{
    int arr[]={3,8,12,5,6},x=12,n=5;

    cout<<"Before Deletion "<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

    cout<<endl;

    n=deleteEle(arr,n,x);
    cout<<"After Deletion :"<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

}
```

OUTPUT:

```
Before Deletion

3 8 12 5 6

After Deletion :

3 8 5 6
```

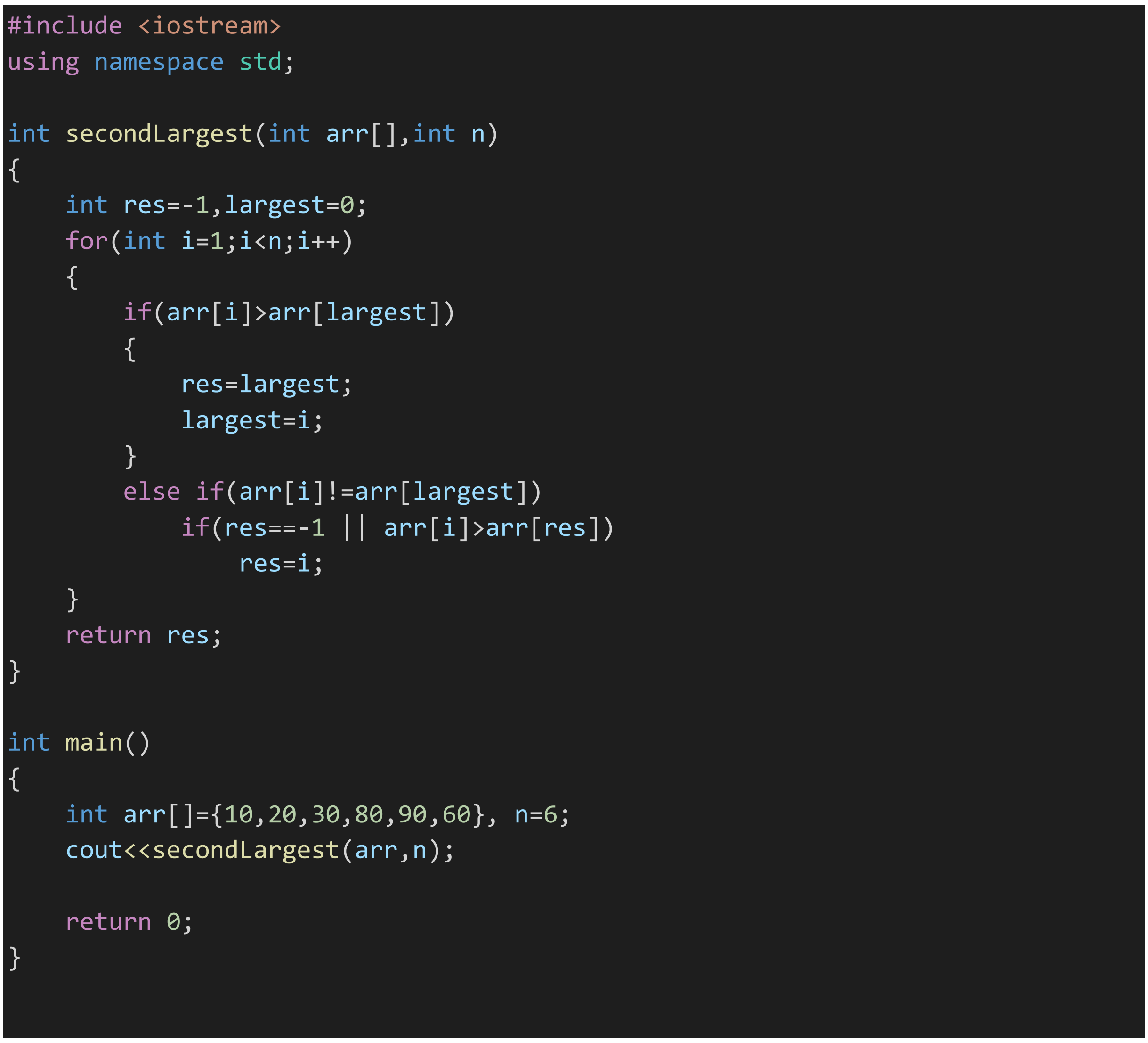
Largest Element in an Array Index

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

int largetInx(int arr[],int n)
{
    int res=0;
    for(int i=1;i<n;i++)
        if(arr[i]>arr[res])
            res=i;

    return res;
}

int main()
{
    int arr[]={10,20,30,80,50,60}, n=6;
    cout<<largetInx(arr,n);

    return 0;
}
```

OUTPUT:

```
3
```

Second Largest Element in an Array Index



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

int secondLargest(int arr[],int n)
{
    int res=-1,largest=0;
    for(int i=1;i<n;i++)
    {
        if(arr[i]>arr[largest])
        {
            res=largest;
            largest=i;
        }
        else if(arr[i]!=arr[largest])
            if(res==-1 || arr[i]>arr[res])
                res=i;
    }
    return res;
}

int main()
{
    int arr[]={10,20,30,80,90,60}, n=6;
    cout<<secondLargest(arr,n);

    return 0;
}
```

OUTPUT:

```
3
```

Check if an Array is Sorted

Naïve:

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

bool isSorted(int arr[], int n)
{
    for(int i = 0; i < n; i++)
    {
        for(int j = i + 1; j < n; j++)
        {
            if(arr[j] < arr[i])
                return false;
        }
    }

    return true;
}


int main() {

      int arr[] = {7, 2, 30, 10}, n = 4;

      printf("%s", isSorted(arr, n)? "true": "false");

}
```

OUTPUT:

```
false
```

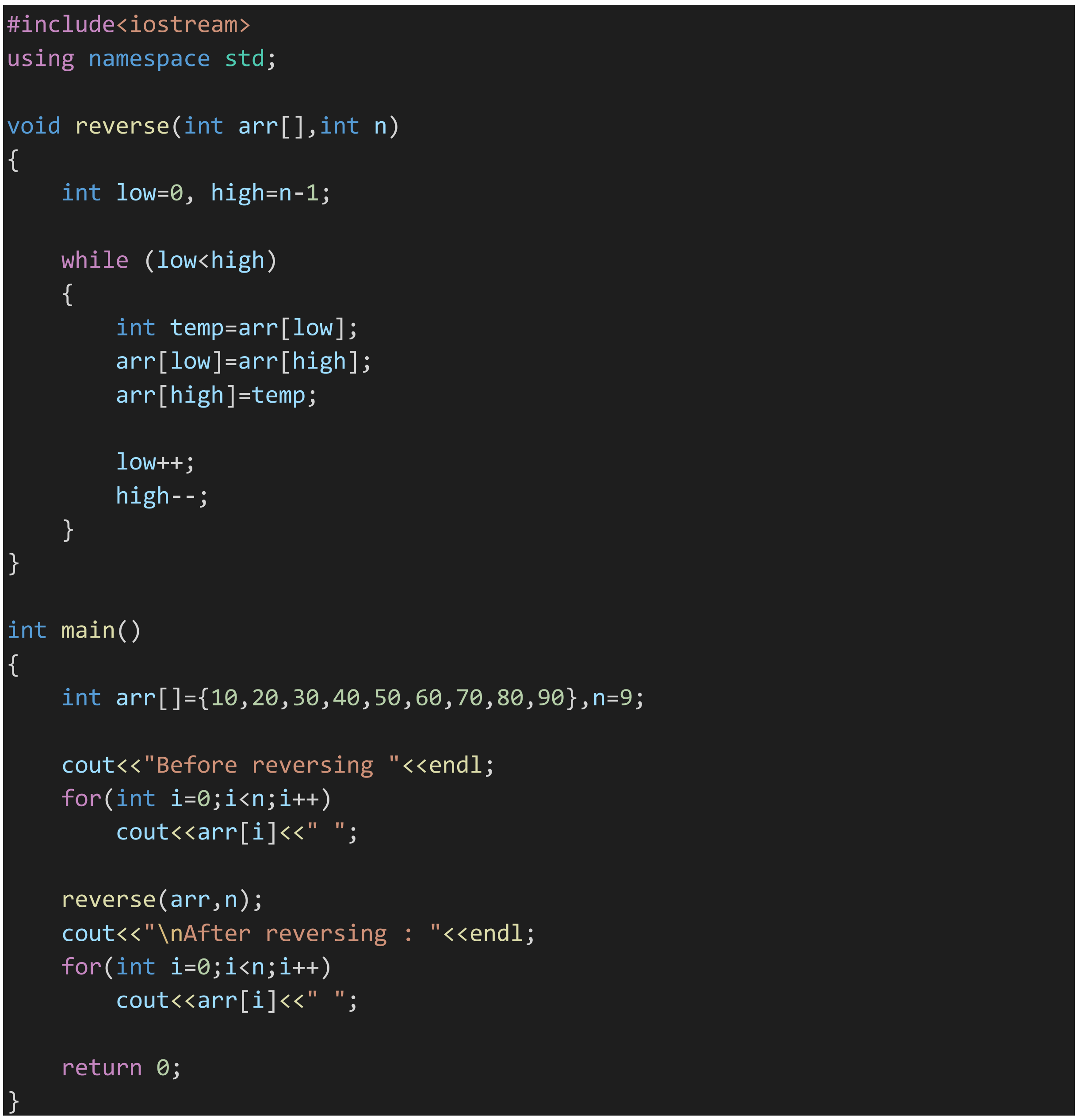
Efficient for check if an array is sorted:

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

//2 for loop pn use karu shakto pn te naive soution hoil
// taypeksha 1 for loop use karun simulatenouly check karaych adjacent barobar

bool isSorted(int arr[],int n)
{
    for(int i=1;i<n;i++)
        if(arr[i]<arr[i-1])
            return false;

    return true;
}

int main() {

      int arr[] = {5, 12, 30, 33, 35}, n = 5;

      cout<<(isSorted(arr, n)? "true": "false");

}
```

OUTPUT:

```
true
```

Reverse an array using swap



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

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

    while (low<high)
    {
        int temp=arr[low];
        arr[low]=arr[high];
        arr[high]=temp;

        low++;
        high--;
    }
}

int main()
{
    int arr[]={10,20,30,40,50,60,70,80,90},n=9;

    cout<<"Before reversing "<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

    reverse(arr,n);
    cout<<"\nAfter reversing : "<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

    return 0;
}
```

OUTPUT:

```
Before reversing

10 20 30 40 50 60 70 80 90

After reversing :

90 80 70 60 50 40 30 20 10
```

Remove duplicate From sorted array

Naïve:

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

int remDups(int arr[],int n)
{
    int temp[n];
    temp[0]=arr[0];

    int res=1;
    for(int i=1;i<n;i++)
    {
        if(arr[i-1]!=arr[i])
        {
            temp[res]=arr[i];
            res++;
        }
    }

    for(int i=0;i<res;i++)
        arr[i]=temp[i];

    return res;
}

int main()
{
    int arr[]={10,20,20,30,30,30},n=6;

    cout<<"Before Removal "<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

    cout<<endl;
    n=remDups(arr,n);

    cout<<"After removal "<<endl;
    for (int i = 0; i < n; i++)
        cout<<arr[i]<<" ";
}
```

OUTPUT

```
Before Removal    10 20 20 30 30 30

After removal   10 20 30
```

Efficient for remove duplicate from sorted array

```
#include <iostream>
using namespace std;
int remDups(int arr[],int n)
{
    int res=1;
    for(int i=1;i<n;i++)
    {
        if (arr[res-1]!=arr[i])
        {
            arr[res]=arr[i];
            res++;
        }
    }
    return res;
}

int main() {

      int arr[] = {10, 20, 20, 30, 30, 30}, n = 6;

      cout<<"Before Removal"<<endl;

       for(int i = 0; i < n; i++)
       {
            cout<<arr[i]<<" ";
       }

       cout<<endl;

       n = remDups(arr, n);

       cout<<"After Removal"<<endl;

       for(int i = 0; i < n; i++)
       {
            cout<<arr[i]<<" ";
       }

}
```

OUTPUT:

Before Removal   10 20 20 30 30 30

After Removal   10 20 30

Move Zero to End

Naïve:

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

void moveToEnd(int arr[],int n)
{
    for (int i = 0; i < n ; i++)
        if(arr[i]==0)
            for (int j = i+1; j < n; j++)
                if(arr[j]!=0)
                {
                    swap(arr[i],arr[j]);
                    break;
                }

}

int main()
{
    int arr[] = {10, 20, 0, 30, 50, 60}, n = 6;

    cout<<"Before Moving"<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

    moveToEnd(arr,n);

    cout<<"\nAfter Moving"<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

    return 0;

}
```

OUTPUT: Before Moving

10 20 0 30 50 60

After Moving

10 20 30 50 60 0

Efficient for Move Zero to End

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

void moveToEnd(int arr[],int n)
{
    int count=0;
    for (int i = 0; i < n; i++)
    {
        if(arr[i]!=0)
        {
            swap(arr[i],arr[count]);
            count++;
        }
    }
}

int main()
{
    int arr[] = {10, 20, 0, 30, 50, 60}, n = 6;

    cout<<"Before Moving"<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

    moveToEnd(arr,n);

    cout<<"\nAfter Moving"<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";

    return 0;

}
```

OUTPUT:

```
Before Moving

10 20 0 30 50 60

After Moving

10 20 30 50 60 0
```

Left Rotate an Array By one

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

 void lRotateOne(int arr[], int n)
{
      int temp = arr[0];
      for(int i = 1; i < n; i++)
            arr[i - 1] = arr[i];
      arr[n - 1] = temp;
}

int main() {

      int arr[] = {1, 2, 3, 4, 5}, n = 5;

      cout<<"Before Rotation"<<endl;

       for(int i = 0; i < n; i++)
                  cout<<arr[i]<<" ";

       cout<<endl;

       lRotateOne(arr, n);

       cout<<"After Rotation"<<endl;

       for(int i = 0; i < n; i++)
                  cout<<arr[i]<<" ";

}
```

Output:

```
Before Rotation

1 2 3 4 5

After Rotation

2 3 4 5 1
```

Left Rotate an Array By D places

Naïve:

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

void lRotateOne(int arr[],int n)
{
    int temp=arr[0];
    for(int i=1;i<n;i++)
        arr[i-1]=arr[i];
    arr[n-1]=temp;
}

void leftRotate(int arr[],int d, int n)
{
    for(int i=0;i<d;i++)
        lRotateOne(arr,n);
}

int main() {

      int arr[] = {1, 2, 3, 4, 5}, n = 5, d = 2;

      cout<<"Before Rotation"<<endl;
       for(int i = 0; i < n; i++)
            cout<<arr[i]<<" ";

       cout<<endl;

       leftRotate(arr, d, n);

       cout<<"After Rotation"<<endl;

       for(int i = 0; i < n; i++)
            cout<<arr[i]<<" ";

}
```

OUTPUT:

```
Before Rotation  1 2 3 4 5

After Rotation  3 4 5 1 2
```

Efficient for left rotate an array By D places

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

void leftRotate(int arr[],int d,int n)
{
    int temp[d];
    for(int i=0;i<d;i++)
        temp[i]=arr[i];

    for(int i=d;i<n;i++)
        arr[i-d]=arr[i];

    for(int i=0;i<d;i++)
        arr[n-d+i]=temp[i];
}

int main() {

      int arr[] = {1, 2, 3, 4, 5}, n = 5, d = 2;

      cout<<"Before Rotation"<<endl;
       for(int i = 0; i < n; i++)
            cout<<arr[i]<<" ";

       cout<<endl;

       leftRotate(arr, d, n);

       cout<<"After Rotation"<<endl;

       for(int i = 0; i < n; i++)
            cout<<arr[i]<<" ";

}
```

OUTPUT:

```
Before Rotation

1 2 3 4 5

After Rotation

3 4 5 1 2
```

Reversal Method for left rotate an array By D places

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

void reverse(int arr[],int low,int high)
{
    while (low<high)
    {
        swap(arr[low],arr[high]);
        low++;
        high--;
    }
}

void leftRotate(int arr[],int d, int n)
{
    reverse(arr,0,d-1);
    reverse(arr,d,n-1);
    reverse(arr,0,n-1);
}

int main() {

      int arr[] = {1, 2, 3, 4, 5}, n = 5, d = 2;

      cout<<"Before Rotation"<<endl;

       for(int i = 0; i < n; i++)
       {
            cout<<arr[i]<<" ";
       }

       cout<<endl;

       leftRotate(arr, d, n);

       cout<<"After Rotation"<<endl;

       for(int i = 0; i < n; i++)
       {
            cout<<arr[i]<<" ";
       }

}
```

OUTPUT: Before Rotation   1 2 3 4 5

After Rotation   3 4 5 1 2

Leaders in an array problem

An element is called the leader of an array if there is no element greater than it on the right side.

Naïve:

```
#include<iostream>
using namespace std;
// An element is called the leader of an array if there is no element greater
than it on the right side
void leaders(int arr[],int n)
{
    for(int i=0;i<n;i++)
    {
        bool flag=false;
        for(int j=i+1;j<n;j++)
        {
            if(arr[i]<=arr[j])
            {
                flag=true;
                break;
            }
        }
        if(!flag)
            cout<<arr[i]<<" ";

    }
}

int main() {

      int arr[] = {7, 10, 4, 10, 6, 5, 2}, n = 7;

      leaders(arr, n);

}
```

OUTPUT:

```
10 6 5 2
```

Efficient for leaders in an array

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

// An element is called the leader of an array if there is no element greater
than it on the right side

void leaders(int arr[],int n)
{
    int curr_ldr=arr[n-1];
    cout<<curr_ldr<<" ";

    for(int i=n-2;i>=0;i--)
    {
        if(curr_ldr<arr[i])
        {
            curr_ldr=arr[i];
            cout<<curr_ldr<<" ";
        }
    }
}

int main() {

      int arr[] = {7, 10, 4, 10, 6, 5, 2}, n = 7;

      leaders(arr, n);

}
```

OUTPUT:

```
2 5 6 10
```

Maximum Difference problem with order

Maximum Difference problem is to find the maximum of arr[j] - arr[i] where j>i.

Naïve 1:

```cpp
//Maximum Difference problem is to find the maximum of arr[j] - arr[i] where j>i.

#include<iostream>
using namespace std;

int maxDiff(int arr[],int n)
{
    int res=arr[1]-arr[0];
    for(int i=0;i<n-1;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(arr[j]-arr[i]>res)
                res=arr[j]-arr[i];
        }
    }
    return res;
}

int main() {

      int arr[] = {2, 3, 10, 6, 4, 8, 1}, n = 7;

      cout<<maxDiff(arr, n);

}
```

OUTPUT:

```
8
```

Maximum Difference problem with order

Naïve 2 :

```
//Maximum Difference problem is to find the maximum of arr[j] - arr[i] where j

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


int maxDiff(int arr[], int n)
{
    int res = arr[1] - arr[0];

    for(int i = 0; i < n - 1; i++)
    {
        for(int j = i + 1; j < n; j++)
        {
            res = max(res, arr[j] - arr[i]);
        }
    }

    return res;
}



int main() {

      int arr[] = {2, 3, 10, 6, 4, 8, 1}, n = 7;

      cout<<maxDiff(arr, n);

}
```

OUTPUT:

```
8
```

Efficient 1 for maximum difference problem with order:

```
//Maximum Difference problem is to find the maximum of arr[j] - arr[i] where j
>i.
//here we only need to keep track of small and large element
//i.e mainting hte small and large

#include<iostream>
using namespace std;

int maxDiff(int arr[],int n)
{
    int res=arr[1]-arr[0], minVal=arr[0];

    for(int i=1;i<n;i++)
    {
        if(res<(arr[i]-minVal))
            res=arr[i]-minVal;

        if(minVal>arr[i])
            minVal=arr[i];

    }
    return res;
}

int main() {

      int arr[] = {2, 3, 10, 6, 4, 8, 1}, n = 7;

      cout<<maxDiff(arr, n);

}
```

OUTPUT:

```
8
```

Efficient 2 for maximum difference problem with order:

```cpp
//Maximum Difference problem is to find the maximum of arr[j] - arr[i] where j
>i.
//here we only need to keep track of small and large element
//i.e mainting hte small and large

#include<iostream>
using namespace std;

int maxDiff(int arr[],int n)
{
    int res=arr[1]-arr[0], minVal=arr[0];

    for(int i=1;i<n;i++)
    {
        res=max(res,arr[i]-minVal);
        minVal=min(minVal,arr[i]);
    }
    return res;
}

int main() {

      int arr[] = {2, 3, 10, 6, 4, 8, 1}, n = 7;

      cout<<maxDiff(arr, n);

}
```

OUTPUT:

```
8
```

Frequencies in a sorted array

```
//frequieces is nothing but number of occurances

#include<iostream>
using namespace std;

void printFreq(int arr[],int n)
{
    int freq=1,i=1;
    while (i<n)
    {
        while(i<n && arr[i]==arr[i-1])
        {
            freq++;
            i++;
        }

        cout<<arr[i-1]<<" "<<freq<<endl;

        i++;
        freq=1;
    }

}

int main() {

      int arr[] = {10, 10, 20, 30, 30, 30}, n = 6;

      printFreq(arr, n);

}
```

OUTPUT:

```
10 2

20 1

30 3
```

Stock BUY and SELL problem part 1

Naïve:

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

int maxProfit(int price[],int start ,int end)
{
    if(end<=start)
        return 0;

    int profit=0;

    for(int i=start; i<end ; i++)
    {
        for(int j=i+1; j<=end; j++)
        {
            if(price[j]>price[i])
            {
                int curr_profit=price[j]-price[i]
                                +maxProfit(price,start,i-1)
                                +maxProfit(price,j+1,end);

                profit=max(profit,curr_profit);
            }
        }
    }

    return profit;
}

int main() {

      int arr[] = {1, 5, 3, 8, 12}, n = 5;

      cout<<maxProfit(arr, 0, n - 1);

}
```

OUTPUT:

```
13
```

Stock BUY and SELL problem part 2

Efficient

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

int maxProfit(int price[],int n)
{
    int profit=0;
    for (int i = 1; i < n; i++)
    {
        if(price[i]>price[i-1])
            profit+=price[i]-price[i-1];
    }

    return profit;
}

int main() {

      int arr[] = {1, 5, 3, 8, 12}, n = 5;

      cout<<maxProfit(arr, n);

}
```

OUTPUT:

```
13
```

Trapping rain water

Naïve:

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

int getWater(int arr[],int n)
{
    int res=0;
    for (int i = 1; i < n-1; i++)
    {
        int lMax=arr[i];

        for(int j=0;j<i;j++)
            lMax=max(lMax,arr[j]);

        int rMax=arr[i];

        for(int j=i+1;j<n;j++)
            rMax=max(rMax,arr[j]);

        res=res+(min(rMax,lMax))-arr[i];
    }

    return res;

}

int main() {

     int arr[] = {3, 0, 1, 2, 5}, n = 5;

     cout<<getWater(arr, n);

}
```

OUTPUT:

```
6
```

Efficient for trapping rain water :

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


int getWater(int arr[], int n)
{

        int res = 0;

        int lMax[n];
        int rMax[n];

        lMax[0] = arr[0];
        for(int i = 1; i < n; i++)
            lMax[i] = max(arr[i], lMax[i - 1]);


        rMax[n - 1] = arr[n - 1];
        for(int i = n - 2; i >= 0; i--)
            rMax[i] = max(arr[i], rMax[i + 1]);

        for(int i = 1; i < n - 1; i++)
            res = res + (min(lMax[i], rMax[i]) - arr[i]);

        return res;

}


int main() {

     int arr[] = {5, 0, 6, 2, 3}, n = 5;

     cout<<getWater(arr, n);

}
```

OUTPUT:

```
6
```

Maximum consecutive 1s

Find count of maximum consecutive 1s in a binary array. Two approaches are discussed, one is O(n^2) and other is O(n). Both of these approaches require O(1) auxiliary space.

Naïve:

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

int maxConsecutiveOnes(int arr[],int n)
{
    int res=0;
    for(int i=0;i<n;i++)
    {
        int count=0;
        for(int j=i;j<n;j++)
        {
            if(arr[j]==1) count++;
            else break;
        }

        res=max(res, count);
    }
    return res;
}

int main() {

     int arr[] = {0, 1, 1, 1, 0, 1, 1}, n = 7;

     cout<<maxConsecutiveOnes(arr, n);

}
```

OUTPUT:

```
3
```

Efficient for maximum consecutive 1s:

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

int maxConsecutiveOnes(int arr[], int n)
{
    int res=0,count=0;
    for(int i=0;i<n;i++)
    {
        if(arr[i]==0)
            count=0;
        else
            count++;
            res=max(res,count);
    }

    return res;
}

int main() {

     int arr[] = {0, 1, 1, 0, 1, 1, 1}, n = 7;

     cout<<maxConsecutiveOnes(arr, n);

}
```

OUTPUT:

```
3
```

Maximum Subarray Sum:

Naïve:

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

int maxSum(int arr[],int n)
{
    int res=arr[0];
    for(int i=0;i<n;i++)
    {
        int curr=0;
        for(int j=i;j<n;j++)
        {
            curr=curr+arr[j];
            res=max(res,curr);
        }

    }
    return res;
}

int main() {

     int arr[] = {1, -2, -3, -100, -2}, n = 5;

     cout<<maxSum(arr, n);

}
```

OUTPUT:

```
1
```

Kadane's algorithm.

Efficient for maximum subarray sum:

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

//Kadane's algorithm.
int maxSum(int arr[], int n)
{
    int res = arr[0];

    int maxEnding = arr[0];

    for(int i = 1; i < n; i++)
    {
        maxEnding = max(maxEnding + arr[i], arr[i]);

        res = max(maxEnding, res);
    }

    return res;
}



int main() {

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

     cout<<maxSum(arr, n);

}
```

OUTPUT:

```
4
```

Longest EVEN ODD subarray

Naïve:

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

int maxEvenOdd(int arr[],int n)
{
    int res=1;
    for(int i=1;i<n;i++)
    {
        int curr=1;
        for(int j=i+1;j<n;j++)
        {
            if((arr[j]%2==0 && arr[j-1]%2!=0)
                || (arr[j]%2!=0 && arr[j-1]%2==0))
                curr++;
            else
                break;
        }
        res=max(res,curr);
    }

    return res;
}

int main() {

     int arr[] = {5, 10, 20, 6, 3, 8}, n = 6;

     cout<<maxEvenOdd(arr, n);

}
```

OUTPUT:

```
3
```

Efficient for Longest EVEN ODD subarray:

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

int maxEvenOdd(int arr[],int n)
{
    int res=1;
    int curr=1;

    for(int i=1;i<n;i++)
    {
        if((arr[i]%2==0 && arr[i-1]%2!=0)
            || arr[i]%2!=0 && arr[i-1]%2==0)
            {
                curr++;
                res=max(res,curr);
            }

        else
            curr=1;

    }
    return res;
}

int main() {

     int arr[] = {5, 10, 20, 6, 3, 8}, n = 6;

     cout<<maxEvenOdd(arr, n);

}
```

OUTPUT:

```
3
```

Maximum Circular sum subarray

Two approaches are discussed, one is O(n^2) and other is O(n).

Naïve:

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

int maxCircularSum(int arr[], int n)
{
    int res=arr[0];
    for(int i=0;i<n;i++)
    {
        int curr_max=arr[i];
        int curr_sum=arr[i];

        for(int j=1;j<n;j++)
        {
            int index=(i+j)%n;
            curr_sum+=arr[index];
            curr_max=max(curr_max,curr_sum);
        }
        res=max(res,curr_max);
    }
    return res;
}

int main() {

     int arr[] = {5, -2, 3, 4}, n = 4;

     cout<<maxCircularSum(arr, n);

}
```

OUTPUT:

```
12
```

Efficient for maximum circular sum subarray:

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

int normalMaxSum(int arr[], int n )
{
    int res=0;
    int maxEnding=arr[0];
    for(int i=1;i<n;i++)
    {
        maxEnding=max(maxEnding+arr[i],arr[i]);

        res=max(res,maxEnding);
    }

    return res;
}

int overallMaxSum(int arr[], int n)
{
    int max_normal=normalMaxSum(arr,n);

    if(max_normal<0)
        return max_normal;

    int arr_sum=0;
    for(int i=0;i<n;i++)
    {
        arr_sum+=arr[i];
        arr[i]=-arr[i];
    }

    int max_circular=arr_sum+normalMaxSum(arr,n);

    return max(max_circular, max_normal);
}

int main() {

     int arr[] = {8, -4, 3, -5, 4}, n = 5;

     cout<<overallMaxSum(arr, n);

}
```

OUTPUT: 12

Majority Element

Majority element is an element that appears more than n/2 times in an array of size n

Naïve:

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

// majority element means element which occure more than n/2 times

int findMajority(int arr[], int n)
{
    for(int i=0;i<n;i++)
    {
        int count=1;
        for(int j=i+1;j<n;j++)
        {
            if(arr[i]==arr[j])
                count++;
        }
        if(count>n/2)
            return i;
    }

    return -1;
}

int main() {

     int arr[] = {8, 7, 6, 8, 6, 6, 6, 6}, n = 8;

     cout<<findMajority(arr, n);

}
```

OUTPUT:

```
2
```

Efficient for majority element :

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

//Using Moore’s Voting Algorithm
int findMajority(int arr[], int n)
{
    int res=0,count=1;
    for(int i=1;i<n;i++)
    {
        if(arr[res]==arr[i])
            count++;
        else
            count--;
        if(count==0)
        {
            res=i;
            count=1;
        }
    }

    count=0;
    for(int i=0;i<n;i++)
        if(arr[res]==arr[i])
            count++;
    if(count<=n/2)
        res=-1;

    return res;
}

int main() {

     int arr[] = {8, 8, 6, 6, 6, 4, 6}, n = 7;

     cout<<findMajority(arr, n);

}
```

OUTPUT:

```
3
```

Minimum Consecutive Flips

Given a binary array, we need to find the minimum of number of group flips to make all array elements same. In a group flip, we can flip any set of consecutive 1s or 0s.

Efficient:

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

void printGroups(int arr[], int n)
{
    for(int i=1;i<n;i++)
    {
        if(arr[i]!=arr[i-1])
        {
            if(arr[i]!=arr[0])
                cout<<"From "<<i<<" to ";
            else
                cout<<(i-1)<<endl;
        }

    }
    if(arr[n-1]!=arr[0])
        cout<<n-1<<endl;
}

int main() {

     int arr[] = {0, 0, 1, 1, 0, 0, 1, 1, 0}, n = 9;

     printGroups(arr, n);

}
```

OUTPUT:

```
From 2 to 3

From 6 to 7
```

Sliding Window Technique

This technique shows how a nested for loop in few problems can be converted to single for loop and hence reducing the time complexity.

N- bonacci numbers:

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

void bonacciseries(long n,int m)
{
    int a[m]={0};
    a[n-1]=1;
    a[n]=1;

    int sum=1;
    for(int i=n+1;i<m;i++)
    {
        sum+=a[i-1]-a[i-1-n];
        a[i]=sum;
    }

    for(int i=0;i<m;i++)
        cout<<a[i]<<" ";
}

int main()
{
    int N = 5, M = 15;
    bonacciseries(N, M);
    return 0;
}
```

Output:

```
0 0 0 0 1 1 2 4 8 16 31 61 120 236 464
```

Sliding Window Technique

Find subarray with given sum :

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

int subArraySum(int arr[], int n , int sum)
{
    int curr_sum=arr[0], start=0, i;

    for(int i=1;i<n;i++)
    {
        while (curr_sum>sum && start<i-1)
        {
            curr_sum-=arr[start];
            start++;
        }

        if(curr_sum==sum)
        {
            cout<<"Sum found between indexes "<<start<<" and "<<i-1;
            return 1;
        }

        if(i<n)
            curr_sum+=arr[i];

    }
    cout<<"No Subarray found ";
    return 0;
}

int main()
{
    int arr[] = {15, 2, 4, 8, 9, 5, 10, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = 23;
    subArraySum(arr, n, sum);
    return 0;
}
```

OUTPUT:

Sum found between indexes 1 and 4

Sliding Window Technique

Maximum Sum of K Consecutive elements(Naïve) :

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

int maxSum(int arr[], int n, int k)
{
    int max_sum=INT16_MIN;
    for(int i=0;i+k-1<n;i++)
    {
        int sum=0;
        for(int j=0;j<k;j++)
            sum+=arr[i+j];

        max_sum=max(max_sum,sum);
    }
    return max_sum;
}

int main() {

     int arr[] = {1, 8, 30, -5, 20, 7}, n = 6, k = 3;

     cout<<maxSum(arr, n, k);

}
```

OUTPUT:

45

Sliding Window Technique

Efficient for Maximum sum of K consecutive elements (efficient):

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

//window sliding technique

int maxSum(int arr[], int n, int k)
{
    int curr_sum=0;
    for(int i=0;i<k;i++)
        curr_sum+=arr[i];

    int max_sum=curr_sum;

    for(int i=k;i<n;i++)
    {
        curr_sum+=(arr[i]-arr[i-k]);
        max_sum=max(max_sum, curr_sum);
    }

    return max_sum;
}

int main() {

     int arr[] = {1, 8, 30, -5, 20, 7}, n = 6, k = 3;

     cout<<maxSum(arr, n, k);

}
```

OUTPUT:

45

Prefix sum Technique part 1

Prefix sum array :

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

int prefix_sum[10000];

void preSum(int arr[], int n)
{
    prefix_sum[0]=arr[0];

    for(int i=1;i<n;i++)
        prefix_sum[i]=prefix_sum[i-1]+arr[i];
}

int getSum(int arr[], int l, int r)
{
    if(l!=0)
        return prefix_sum[r]-prefix_sum[l-1];
    else
        return prefix_sum[r];
}

int main() {

      int arr[] = {2, 8, 3, 9, 6, 5, 4}, n = 7;

      preSum(arr, n);


     cout<<getSum(prefix_sum, 1, 3)<<endl;

     cout<<getSum(prefix_sum, 0, 2)<<endl;

}
```

OUTPUT:

```
20

13
```

Prefix sum Technique part 1

Naïve for equilibrium point :

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

bool checkEquilibrium(int arr[], int n)
{
    for( int i=0;i<n;i++)
    {
        int l_sum=0, r_sum=0;

        for(int j=0;j<i;j++)
            l_sum+=arr[j];

        for(int k=i+1;k<n;k++)
            r_sum+=arr[k];

        if(l_sum==r_sum)
            return true;
    }
    return false;
}

int main() {

      int arr[] = {2,3,2,-2}, n = 4;

      cout<<(checkEquilibrium(arr, n)? "true" : "false");


}
```

OUTPUT:

```
False
```

Prefix sum Technique part 1

Efficient for Equilibrium point :

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

bool checkEquilibrium(int arr[], int  n)
{
    int sum=0;
    for(int i=0;i<n;i++)
        sum=+arr[i];

    int l_sum=0;
    for(int i=0;i<n;i++)
    {
        if(l_sum==sum-arr[i])
            return true;

        l_sum+=arr[i];

        sum-=arr[i];
    }

    return false;
}

int main() {

      int arr[] = {3, 4, 8, -9, 20, 6}, n = 6;

      cout<<(checkEquilibrium(arr, n)? "true" : "false");


}
```

OUTPUT:

False

Prefix sum Technique part 2

Maximum Occuring Elements

```
#include <iostream>
#include <cmath>
#include <bits/stdc++.h>
#include <climits>
using namespace std;
int maxOcc(int L[], int R[], int n)
{
        int arr[1000];

        memset(arr, 0, sizeof(arr));

        for(int i = 0; i < n; i++)
        {
            arr[L[i]]++;

            arr[R[i] + 1]--;
        }

        int maxm = arr[0], res = 0;

        for(int i = 1; i < 1000; i++)
        {
            arr[i] += arr[i - 1];

            if(maxm < arr[i])
            {
                maxm = arr[i];

                res = i;
            }
        }

        return res;
}

int main() {

      int L[] = {1, 2, 3}, R[] = {3, 5, 7}, n = 3;

      cout<<maxOcc(L, R, n);


}
OUTPUT : 3
```