**Sliding window all sub arrays**

**import java.util.\*;**

**public class trailer**

**{**

**public static void main(String[] args) {**

**int a[]={1,2,3,4,5,6,7};**

**int k=2;**

**ArrayList<Integer>a1=new ArrayList<>();**

**while(k<a.length)**

**{**

**System.out.println("THE SUB ARRAY OF LENGTH :"+k);**

**for(int i=0;i<k;i++)**

**a1.add(a[i]);**

**System.out.println(a1);**

**for(int i=k;i<a.length;i++)**

**{**

**a1.remove(0);**

**a1.add(a[i]);**

**System.out.println(a1);**

**}**

**a1.clear();**

**k+=1;**

**}**

**}**

**}**

**//printing all sub arrays**

**void printsub(int a[],int n)**

**{**

**for(int i=0;i<n;i++)**

**{**

**for(int j=i;j<n;j++)**

**{**

**cout<<"[";**

**for(int k=i;k<=j;k++)**

**cout<<a[k]<<" ";**

**cout<<"]"<<endl;**

**}**

**}**

**}**

**?????????printnig all subsets??????????????????**

**Void printsubset(int arr[]){**

**For(int i=0;i<(1<<n);i++){**

**For(int j=0;j<n;j++)**

**{**

**If(i&(1<<j)){**

**Cout<<A[j]<<” “;**

**}**

**}**

**}**

**///max sum of sub array of size k**

**long a[]=new long[N];**

**a[0]=Arr.get(0);**

**for(int i=1;i<N;i++)**

**a[i]=a[i-1]+Arr.get(i);**

**long max=a[K-1];**

**for(int i=K;i<N;i++)**

**if(max<(a[i]-a[i-K]))**

**max=a[i]-a[i-K];**

**return max;**

HashMap<Integer,Integer> h=new HashMap<>();

ArrayList<Integer> a=new ArrayList<>();

int max1=arr[0];

for(int i=0;i<k;i++)

{

if(h.containsKey(arr[i]))

{

h.put(arr[i],h.get(arr[i])+1);

}

else

{

h.put(arr[i],1);

}

if(max1<arr[i])

{

max1=arr[i];

}

}

a.add(max1);

// System.out.println(h);

for(int i=k;i<n;i++)

{

h.put(arr[i-k],(int)h.get(arr[i-k])-1);

if(h.get(arr[i-k])==0)

{

h.remove(arr[i-k]);

}

if(h.containsKey(arr[i]))

{

h.put(arr[i],(int)h.get(arr[i])+1);

}

else

{

h.put(arr[i],1);

}

if(h.containsKey(max1))

{

max1=Math.max(max1,arr[i]);

}

else

{

max1=arr[i];

for(int z:h.keySet())

if(z>max1)

max1=z;

}

a.add(max1);

}

**BIT MANIPULATIONS**

**Bitwise operator’s**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **a** | **b** | **&** | **|** | **^** |
| **1** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **1** | **1** |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **1** | **1** |

**To convert decimal number to binary**

**while(n>0)**

**{**

**cout<<((n>>1)&1);**

**n=n>>1;**

**}**

**To know of set bits**

**Int c=0;**

**While(n){**

**If(n&1)**

**C+=1;**

**N=n>>1;**

**}**

**Return c;**

**0 0 0 1 0 1 1 0 1**

**| |**

**Msb lsb**

**| least significant bit**

**Most significant bit**

**Set bit in given position**

**Int a=12;**

**Int p=4;**

**Cout<<(12|(1<<p))<<endl;**

**Unset given position**

**Int i=3;**

**Cout<<(12&(~(1<<i)))<<endl;**

**Toggle bit in given position**

**Cout<<(12^(1<<i))<<endl;**

**setBit(n,p)**

**{**

**Int k=1<<P;**

**Return n|k;**

**}**

**unsetBit(n,p)**

**{**

**Int k=(~(1<<p));**

**Return n&k;**

**}**

**Flipgivenbit(n,p)**

**{**

**int k=(1<<p);**

**cout<<(n^k);**

**}**

**To check even or odd**

**If(n&1==0)**

**Return true;**

**Return false;**

**}**

**To convert uppercase to lower case**

**Uppertolower(char n)**

**Return (n|’ ‘);**

**Lowertoupper(char n)**

**Return (n&’A’)**

**Subarray generation bbrute force**

**void printsub(int a[],int n)**

**{**

**for(int i=0;i<n;i++)**

**{**

**for(int j=i;j<n;j++)**

**{**

**for(int k=i;k<=j;k++)**

**cout<<a[k]<<" ";**

**cout<<endl;**

**}**

**}**

**}**

**//GAME OF XOR**

**#include<stdio.h>**

**#include<string.h>**

**char\* encryp(char a[],int k);**

**char\* decryp(char a[],int k);**

**int main()**

**{**

**printf("enter the string:::");**

**char a[200];**

**scanf("%s",a);**

**char\* b=encryp(a,3);**

**// // strcpy(b,encryp(a,3));**

**// char c[200];**

**// strcpy(c,decryp(b,3));**

**printf("%s\n",b);**

**// printf("%s\n",c);**

**}**

**char\* encryp(char a[],int k)**

**{**

**for(int i=0;i<strlen(a);i++)**

**if(((int)a[i]>=65) && ((int) a[i]<=90))**

**a[i]=(((int)a[i]+k-65)%26+65);**

**else if(((int)a[i]>=97&&(int)a[i]<=122))**

**a[i]=(((int)a[i]+k-97)%26+97);**

**return a;**

**}**

**// char\* decryp(char a[],int k)**

**// {**

**// for(int i=0;i<strlen(a);i++)**

**// if(a[i]>='A' && a[i]<='Z')**

**// a[i]=(((int)a[i]-k-65)%26+65);**

**// else if(a[i]>='a'&&a[i]<='z')**

**// a[i]=(((int)a[i]-k-97)%26+97);**

**// return a;**

**// }**

**GCD----ECULIDEAN ALOGRITHM**

**GCD(A,B)=GCD(A-B,B)=GCD(A,A-B)**

**EX::: GCD(8,2)=GCD(6,2)=GCD(8,6)**

**SUBSET GENERATION USING RECURSION:::::::::**

 vector<vector<int>> v;

    vector<vector<int>> subsets(vector<int>& nums) {

       subsets(0,nums,nums.size(),{});

       return v;

    }

    void subsets(int ind,vector<int> a,int n,vector<int> k)

    {

        if(ind==n)

        {

            v.push\_back(k);

            return;

        }

        k.push\_back(a[ind]);

        subsets(ind+1,a,n,k);

        k.pop\_back();

        subsets(ind+1,a,n,k);

    }

**To get only unique values in vector**

**Sort(Mainans.begin(),MAinans.end())**

**Mainans.erase(unique(mainans.begin(),mainans.end()),mainans.end());**

**To get faster input and output**

**ios\_base::sync\_with\_stdio(false);**

**cin.tie(NULL);**

**cout.tie(NULL);**

**Sorting algorithms:**

**Bubble sort:**

bubbleSort(array)

n = length(array)

repeat

  swapped = **false**

**for** i = 1 to n - 1

**if** array[i - 1] > array[i], then

         swap(array[i - 1], array[i])

         swapped = **true**

         end **if**

   end **for**

   n = n - 1

 until not swapped

end bubbleSort

**Time complexity O(n),O(n2)**

**Bucket Sort(A[])**

Let B[0....n-1] be a new array

n=length[A]

for i=0 to n-1

make B[i] an empty list

for i=1 to n

do insert A[i] into list B[n a[i]]

for i=0 to n-1

do sort list B[i] with insertion-sort

Concatenate lists B[0], B[1],........, B[n-1] together in order

End

**Time complexity O(n+k),O(n2)**

**implementation**

#include <stdio.h>

int getMax(int a[], int n) // function to get maximum element from the given array

{

int max = a[0];

for (int i = 1; i < n; i++)

if (a[i] > max)

max = a[i];

return max;

}

void bucket(int a[], int n) // function to implement bucket sort

{

int max = getMax(a, n); //max is the maximum element of array

int bucket[max], i;

for (int i = 0; i <= max; i++)

{

bucket[i] = 0;

}

for (int i = 0; i < n; i++)

{

bucket[a[i]]++;

}

for (int i = 0, j = 0; i <= max; i++)

{

while (bucket[i] > 0)

{

a[j++] = i;

bucket[i]--;

}

}

}

**Heap sort**

void heapify(int arr[], int N, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < N && arr[left] > arr[largest])

largest = left;

if (right < N && arr[right] > arr[largest])

largest = right;

if (largest != i) {

swap(arr[i], arr[largest]);

heapify(arr, N, largest);

}}

void heapSort(int arr[], int N)

{

for (int i = N / 2 - 1; i >= 0; i--)

heapify (arr, N, i);

for (int i = N - 1; i > 0; i--) {

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

int N = sizeof(arr) / sizeof(arr[0]);

heapSort(arr, N);

}

**Time complexity O(n log n)**

**Insertion sort:**

void insert(int a[], int n)

{

int i, j, temp;

for (i = 1; i < n; i++) {

temp = a[i];

j = i - 1;

while(j>=0 && temp <= a[j ])

{

a[j+1] = a[j];

j = j-1;

}

a[j+1] = temp;

} }

**Time complexity O(n)O(n2)**

**MergeSort**

MERGE\_SORT(arr, beg, end)

if beg < end

set mid = (beg + end)/2

MERGE\_SORT(arr, beg, mid)

MERGE\_SORT(arr, mid + 1, end)

MERGE (arr, beg, mid, end)

end of if

void merge(int a[], int beg, int mid, int end)

{

int i, j, k;

int n1 = mid - beg + 1;

int n2 = end - mid;

int LeftArray[n1], RightArray[n2]; //temporary arrays

/\* copy data to temp arrays \*/

for (int i = 0; i < n1; i++)

LeftArray[i] = a[beg + i];

for (int j = 0; j < n2; j++)

RightArray[j] = a[mid + 1 + j];

i = 0

j = 0;

k = beg;

while (i < n1 && j < n2)

{

if(LeftArray[i] <= RightArray[j])

{

a[k] = LeftArray[i];

i++;

}

else

{

a[k] = RightArray[j];

j++;

}

k++;

}

while (i<n1)

{

a[k] = LeftArray[i];

i++;

k++;

}

while (j<n2)

{

a[k] = RightArray[j];

j++;

k++;

}

}

**Time complexity O(nlog n)**

**Qucik sort**

int partition (int a[], int start, int end)

{

int pivot = a[end]; // pivot element

int i = (start - 1);

for (int j = start; j <= end - 1; j++)

{

if (a[j] < pivot)

{

i++; // increment index of smaller element

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}

int t = a[i+1];

a[i+1] = a[end];

a[end] = t;

return (i + 1);

}

void quick(int a[], int start, int end)

{

if (start < end)

{

int p = partition(a, start, end); //p is the partitioning index

quick(a, start, p - 1);

quick(a, p + 1, end);

}

}

**Time complexity O(nlog n),O(n2)**

SQL QUERIES:

* Sql is case in-sensitive SELECT and select are same.
* The SELECT statement is used to select data from a database.

Ex:SELECT column1, column2, ...FROM table\_name;

* SELECT DISTINCT statement is used to return only distinct (different) values.  
  ex:SELECT DISTINCT column1, column2, ...FROM table\_name;

The WHERE clause is used to filter records.  
Ex:SELECT column1, column2, ...FROM table\_nameWHERE condition;

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

The AND operator displays a record if all the conditions separated by AND are TRUE.

The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.  
ex: SELECT column1, column2, ...  
FROM table\_name  
WHERE not (condition1 AND condition2 OR condition3 ...);

* The ORDER BY keyword is used to sort the result-set in ascending or descending order.  
  ex: SELECT column1, column2, ...FROM table\_name ORDER BY column1, column2, ... ASC|DESC;
* It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table\_name* (*column1*,*column2*,*column3*, ...)  
VALUES (*value1*,*value2*,*value3*, ...);

2. INSERT INTO *table\_name* VALUES (*value1*,*value2*,*value3*, .

* A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!  
  to check NULL value we use IS NULL and IS NOT NULL
* The UPDATE statement is used to modify the existing records in a table.  
  UPDATE table\_name SET column1 = value1, column2 = value2, ...  
  WHERE condition;
* The DELETE statement is used to delete existing records in a table.  
  DELETE FROM table\_name WHERE condition;  
  TO DELETE ALL ROWS IN TABLE  
  🡪delete from table\_name;
* database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.  
  SELECT column\_name(s) FROM table\_name WHERE condition LIMIT number;

**Agregate functions**

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

* j

To create procedures

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;