CODE_RENEGADES

```
/*==============================C++ cheat sheet ================================
  * 01- string
  * 02- vector
  * 03- pair
  * 04- set
  * 05- map
  * 10- general
  *  SHIHAB
  * ======================================*/

/*--------------------------01- string---------------------------*/
// --- transform char using #include <string.h>
toupper(s[i], locale());
tolower(s[i], locale());

// --- initialize a sequence using #include <algorithm>
fill_n(nums, sizeof(nums), true);

// --- transform string using #include <algorithm>
transform(word.begin(), word.end(), word.begin(), ::tolower);
transform(word.begin(), word.end(), word.begin(), ::toupper);

// --- use ignore after using cin
cin.ignore();   // Extracts chars from the input sequence and discards them.
getline(cin, line);

// --- use #include <sstream>
getline(cin, line);
stringstream stream(line);
stream >> word;


// --- find using #include <string>
// 2nd param: Position of the 1st char in the string to be searched.
// 3rd param: Length of sequence of chars to match.

string haystack = "There are two needles in this haystack with needles.";
string needle = "needle";

size_t found = haystack.find(needle);
if (found != ::npos) cout << "first needle at: " << found << '\n';

found = haystack.find("needles are small", found+1, 6);
if (found != ::npos) cout << "second needle at: " << found << '\n';


// --- replace using #include <string>
string str1 = "donkey ate the milk";
string str2 = "cat";
str1.replace(str1.begin(), str1.begin()+6, str2); // (from, to, replace)

string str1 = "donkey ate the milk";
string str2 = "012cat";
str1.replace(str1.begin(), str1.begin()+6, str2.begin()+3, str2.end()); // (from, to,
from, to)

str.replace(9,5,"hello");       // (from, len, replace) - string
str.replace(9,6,"hello",7,6);   // (from, len, replace, subFrom, subLen) - substring
str.replace(22,1,3,'!');        // (from, len, n, char c)   - fill
```

```
// replace the first needle:
str.replace(str.find(needle), needle.length(), "replace");

/*---------------------------02- vector----------------------------*/
#include <vector>;
vector<int> v;              // declaration
vector<int>::iterator it;   // pointer to the vector<int>

v.push_back(4);             // adding to the vector
v.push_back(5);
v.size();

for (it=v.begin(); it!=v.end(); it++) {
    cout << *it << endl;    // print the vector elements
}

cout << v.front() << endl;  // access the first element
cout << v.back() << endl;   // access the last element

v.resize(5);                // 5 is the new container size
v.resize(5, 0);             // 0 copied to the added elements if 5 > the current
container size

v.erase(v.begin());         // erase the first element
v.erase(v.begin() + 3);     // erase the 4th element
v.erase(v.begin(), v.begin() + 3);      // erase the 1, 2, 3 elements
v.pop_back();               // erase the last element
v.clear();                    // remember to clear the vector each loop

// use #include <algorithm>
reverse(v.begin(),v.end());
is_sorted(v.begin(),v.end());

int sum = 0;
while(!v.empty()) {         // return true if the container size = 0
    sum += v.back();
    v.pop_back();
}

// --- sort vector of nodes
bool vsort(const Node &left, const Node &right) {
    if (left.length == right.length) {
        return left.order < right.order;
    }
    return left.length > right.length;
}
struct Node {
    string word;
    int length;
    int order;
};
while (cin >> word) {
    Node *temp = new Node;
    temp->length = word.length();
    temp->order = count;
    temp->word = word;
    v.push_back(*temp);
}
sort(v.begin(), v.end(), vsort);
```

```
/*---------------------------03- pair---------------------------*/
pair<int, int> p0;          // declaration
p0.first = 2;
p0.second = 5;
pair<int, int> p1 = make_pair(2, 5);
pair<int, int> p2(2, 5);
pair<int, int> p3(p2);

pair<int, int> arr[10];     // array of pairs
arr[0].first = 2;
arr[0].second = 5;
arr[1] = make_pair(2, 5);

bool pSort(const pair<int,int> &left, const pair<int,int> &right) {
    if (left.second == right.second) {
        return left.first < right.second;
    }
    return left.second < right.second;
}
sort(arr, arr+10, pSort);   // sort the array of pairs according to pSort/\

vector< pair<int, int> > v0;    // vector of pairs
vector< pair<int, int> >::iterator it0;
v0.push_back( make_pair(2, 4) );
for (it0=v0.begin(); it0!=v0.end(); it0++) {
    cout << (*it0).first << " | " << it0->second << endl;    // print the vector elements
}


/*---------------------------04- set---------------------------*/
#include <set>
set<int> s; // declaration - used to store unique elements
set<int>::iterator it1;

s.insert(4);
for (it1=s.begin(); it1!=s.end(); it1++) {
    cout << *it << endl;  // print the set elements
}
cout << s.size() << endl;
cout << s.count(5); // return 1 if set contains 5 | otherwise return 0
it0 = s.find(5);    // if found return an iterator to the element, otherwise return
set.end()
s.erase(s.end());   // erase the last element
s.clear();
s.empty();  // true || false


/*---------------------------05- map---------------------------*/
#include <map>
map<char, int> m;
map<char, int>::iterator it2;

m['c'] = 3;
for(it2=m.begin(); it2!=m.end; it2++) {
    cout << it2->first << " => " << it2->second << endl;
}

it2 = m.find('c');
cout << it2->second << endl; // output => 3
```

```
cout << m.size() << endl;

m.erase('c');
m.erase(it2, m.end());  // erase from it2 to the end
m.clear();

m.empty();  // true || false

/*---------------------------10- general---------------------------*/
// --- generating a prime list in the first 1000 number
bool nums[1001];
fill_n(nums, sizeof(nums), true);
nums[0] = nums[1] = false;

for (int i=2; i<100; i++) {
    if (nums[i] == false) continue;
    int tmp;
    for (int j=2; tmp=i*j, tmp <=sizeof(nums); j++) {
        nums[tmp] = false;
    }
}

// --- get the greatest common devisor
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}


/*---------------------------SHIHAB---------------------------*/
```

```
//Binary Search

int binarySearch(int arr[],int low,int high,int n)

{

   while(low<=high){

      int mid=(low+high-1)/2;

      if(arr[mid]==n)return mid+1;


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

      else high=mid-1;

   }

   return -1;

}
```

```
void primeFactors(long long n)

{

   while(n%2==0){

      cout<<2<< " ";

      n/=2;

   }

   for(int i=3;i*i<=n;i++){

      while(n%i==0){

         cout<<i<< " ";

         n/=i;

      }

   }

   if(n>2)cout<<n;

}
```

//Normal sieve

```
#include<iostream>
#include<vector>
using namespace std;
#define endl '\n'                    //Sieve algorith to find Prime numbers upto 100000000 under 1 second
#define ll long long int

vector<int>primes;
bool a[100000000];
int n=100000000;

void findPrimes()
{
   a[0]=a[1]=true;
   primes.push_back(2);
   for(int i=3;i<n;i+=2){
      if(a[i]==false){
         primes.push_back(i);
         if(i*(ll)i<(ll)n)
         for(int j=i*i;j<=n;j+=i*2)
            a[j]=true;
      }
   }
}

int main()
{
   ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);

   findPrimes();
   int Size=primes.size();
   int i=0;
   while(i<Size){
      cout<<primes[i]<<endl;
      i++;
   }
   return 0;
}
```

```
//Mathematical Function
int GCD(int a, int b)
{
   while (b)
   {
      a %= b;
      swap(a, b);
   }
   return a;
}

ll mod_add(int a, int b, int m=MOD)
 {
   a = a % m;
   b = b % m;
   return (((a + b) % m) + m) % m;
}

ll mod_mul(int a, int b, int m=MOD)
 {
   a = a % m;
    b = b % m;
    return (((a * b) % m) + m) % m;
 }

ll mod_sub(int a, int b, int m=MOD)
 {
   a = a % m;
   b = b % m;
    return (((a - b) % m) + m) % m;
}
```

```
 // Calculating Power
ll modpow(int x, int n, int m = MOD)
{
   if (x == 0 && n == 0)
      return 0; // undefined case
   ll res = 1;
   while (n > 0)
   {
      if (n % 2)
         res = (res * x) % m;
      x = (x * x) % m;
      n /= 2;
   }
   return res;
}
// Modulo Inverse
int modinv(int x, int m = MOD)
{
   return modpow(x, m - 2, m);
}
int binpow(int a, int b) {
   if (b == 0)
      return 1;
   int res = binpow(a, b / 2);
   if (b % 2)
      return res * res * a;
   else
      return res * res;
}
```

$$\log_a xy = \log_a x + \log_a y$$

$$\log_a \frac{x}{y} = \log_a x - \log_a y$$

$$\log_a x^n = n \log_a x$$

$$\log_a b = \frac{\log_c b}{\log_c a}$$

$$\log_a b = \frac{1}{\log_b a}$$

The following can be derived f

$$\log_a 1 = 0$$

$$\log_a a = 1$$

$$\log_a a^r = r$$

$$\log_a \frac{1}{b} = -\log_a b$$

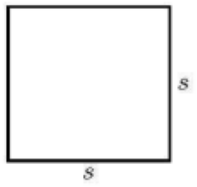
$$\log_{\frac{1}{a}} b = -\log_a b$$

$$\log_a b \log_b c = \log_a c$$

$$\log_{a^m} a^n = \frac{n}{m}, m \neq 0$$

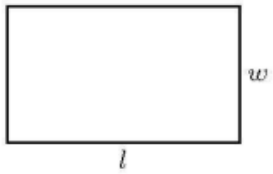
SQUARE

s = side
Area: $A = s^2$
Perimeter: $P = 4s$



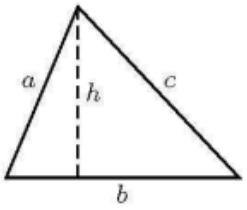
RECTANGLE

l = length, w = width
Area: $A = lw$
Perimeter: $P = 2l + 2w$



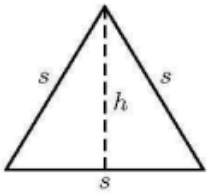
TRIANGLE

b = base, h = height
Area: $A = \frac{1}{2}bh$
Perimeter: $P = a + b + c$



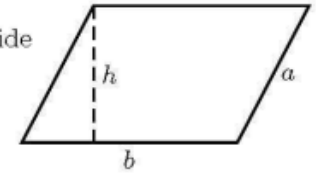
EQUILATERAL TRIANGLE

s = side
Height: $h = \frac{\sqrt{3}}{2}s$
Area: $A = \frac{\sqrt{3}}{4}s^2$



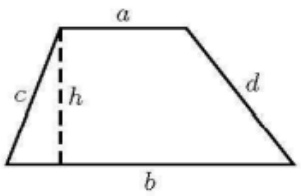
PARALLELOGRAM

b = base, h = height, a = side
Area: $A = bh$
Perimeter: $P = 2a + 2b$



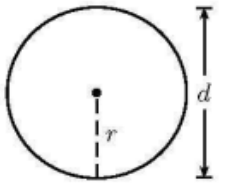
TRAPEZOID

a, b = bases; h = height;
c, d = sides
Area: $A = \frac{1}{2}(a + b)h$
Perimeter:
$P = a + b + c + d$



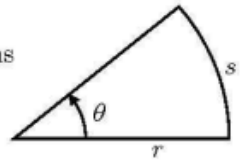
CIRCLE

r = radius, d = diameter
Diameter: $d = 2r$
Area: $A = \pi r^2$
Circumference: $C = 2\pi r = \pi d$



SECTOR OF CIRCLE

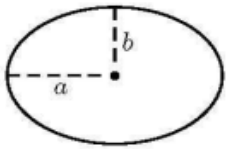
r = radius, θ = angle in radians
Area: $A = \frac{1}{2}\theta r^2$
Arc Length: $s = \theta r$



ELLIPSE

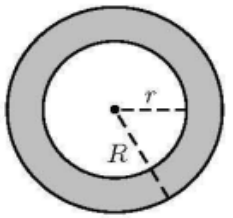
a = semimajor axis
b = semiminor axis
Area: $A = \pi ab$

Circumference:
$C \approx \pi \left(3(a + b) - \sqrt{(a + 3b)(b + 3a)}\right)$



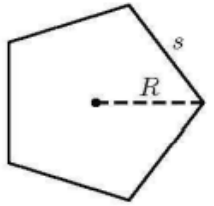
ANNULUS

r = inner radius,
R = outer radius
Average Radius: $\rho = \frac{1}{2}(r + R)$
Width: $w = R - r$
Area: $A = \pi(R^2 - r^2)$
or $A = 2\pi\rho w$



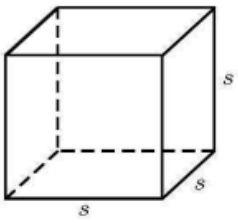
REGULAR POLYGON

s = side length,
n = number of sides
Circumradius: $R = \frac{1}{2}s \csc(\frac{\pi}{n})$
Area: $A = \frac{1}{4}ns^2 \cot(\frac{\pi}{n})$
or $A = \frac{1}{2}nR^2 \sin(\frac{2\pi}{n})$



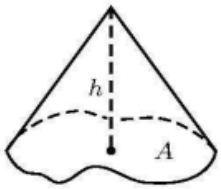
CUBE

s = side
Volume: $V = s^3$
Surface Area: $S = 6s^2$



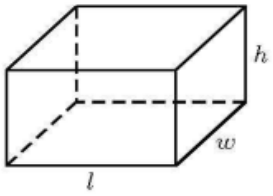
GENERAL CONE OR PYRAMID

A = area of base, h = height
Volume: $V = \frac{1}{3}Ah$



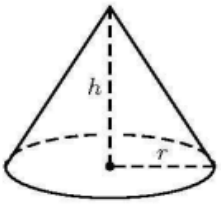
RECTANGULAR SOLID

l = length, w = width,
h = height
Volume: $V = lwh$
Surface Area:
$S = 2lw + 2lh + 2wh$



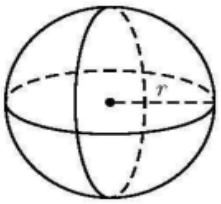
RIGHT CIRCULAR CONE

r = radius, h = height
Volume: $V = \frac{1}{3}\pi r^2 h$
Surface Area:
$S = \pi r\sqrt{r^2 + h^2} + \pi r^2$



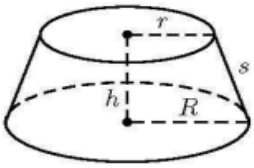
SPHERE

r = radius
Volume: $V = \frac{4}{3}\pi r^3$
Surface Area: $S = 4\pi r^2$



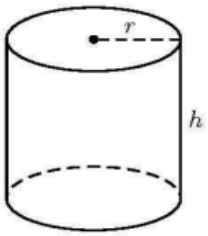
FRUSTUM OF A CONE

r = top radius, R = base radius,
h = height, s = slant height
Volume: $V = \frac{\pi}{3}(r^2 + rR + R^2)h$
Surface Area:
$S = \pi s(R + r) + \pi r^2 + \pi R^2$



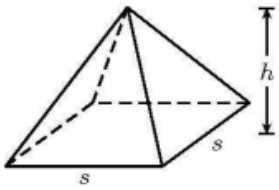
RIGHT CIRCULAR CYLINDER

r = radius, h = height
Volume: $V = \pi r^2 h$
Surface Area: $S = 2\pi rh + 2\pi r^2$



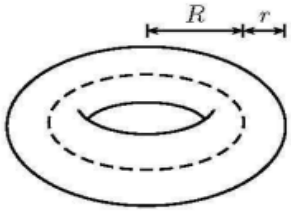
SQUARE PYRAMID

s = side, h = height
Volume: $V = \frac{1}{3}s^2 h$
Surface Area:
$S = s(s + \sqrt{s^2 + 4h^2})$



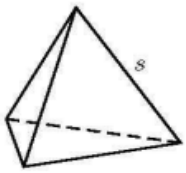
TORUS

r = tube radius,
R = torus radius
Volume: $V = 2\pi^2 r^2 R$
Surface Area: $S = 4\pi^2 rR$



REGULAR TETRAHEDRON

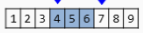
s = side
Volume: $V = \frac{1}{12}\sqrt{2}s^3$
Surface Area: $S = \sqrt{3}s^2$



C++ Standard Library Algorithms

Iterator Ranges

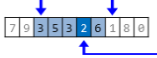
`distance(@begin, @end) → element_count`



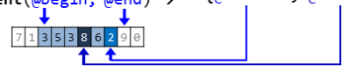
<iterator>

Non-Modifying Sequence Operations

`min_element(@begin, @end) → @minimum`



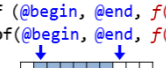
`minmax_element(@begin, @end) → {@minimum, @maximum}`



C++11

`any_of (@begin, @end, f(○)↦bool) →`
`all_of (@begin, @end, f(○)↦bool) →`
`none_of(@begin, @end, f(○)↦bool) →`

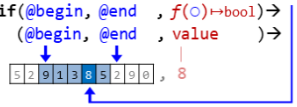
- true, if *f* yields true for *any*, *all* or *none* elements in the input range
- false otherwise



C++11

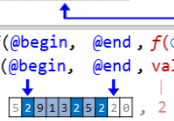
`find_if(@begin, @end , f(○)↦bool)→`
`find (@begin, @end , value )→`

- @1st match
- @end if no match



`count_if(@begin, @end , f(○)↦bool) →`
`count (@begin, @end , value ) →`

number of occurrences

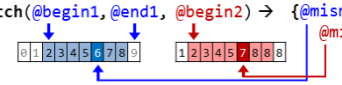


3

`equal(@begin1, @end1, @begin2) → true`

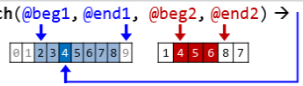
if all elements in both ranges are equal

`mismatch(@begin1,@end1, @begin2) → {@mismatch_in1, @mismatch_in2}`



`search(@beg1,@end1, @beg2,@end2) →`

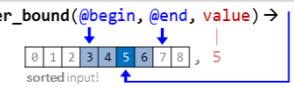
- @1stoccurrence of sequence 2 inside sequence 1
- @end1 otherwise



Binary Search On *Sorted* Sequences ⇒ $\mathcal{O}(\log n)$

`lower_bound(@begin, @end, value)→`

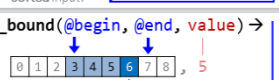
- @1st element **not** < value
- @end if no such element exists



sorted input!

`upper_bound(@begin, @end, value)→`

- @1st element > value
- @end if no such element exists



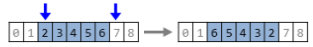
sorted input!

`equal_range(@begin, @end, value)→ {`

- @1st item **not** < value or @end if none such found),
- @1st item > value or @end if none such found)}

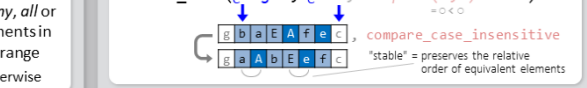
Reordering Elements

`reverse(@begin, @end)`

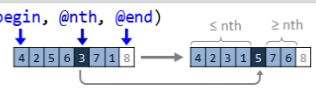


`sort(@begin, @end, f(○,○)↦bool)`
`sort(@begin, @end, std::less )`

`stable_sort(@begin, @end, compare(○,○)↦bool)`

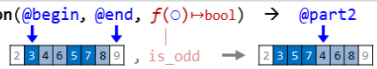


`nth_element(@begin, @nth, @end)`

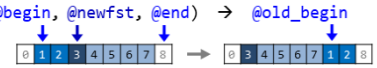


element at nth position → element that would be in that position in a sorted sequence

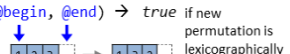
`partition(@begin, @end, f(○)↦bool) → @part2`



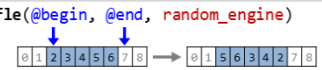
`rotate(@begin, @newfst, @end) → @old_begin`



`next_permutation(@begin, @end) → true` if new permutation is lexicographically greater



`shuffle(@begin, @end, random_engine)`



Manipulate *Sorted* Sequences ⇒ $\mathcal{O}(n)$

`merge(@1beg , @1end, @2beg,@2end, @out)`

`set_union(@1beg ,@1end, @2beg, @2end, @out)`

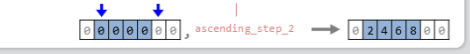
h/cpp hackingcpp.com

Changing Values

`copy(@begin, @end, @out)`

`transform(@begin, @end, @out, f(○)↦■)`

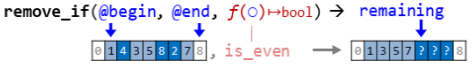
`generate(@begin, @end, f()↦■)`



`replace(@begin, @end, old, new)`

`replace_if(@begin,@end, f(○)↦bool, new)`

`remove (@begin, @end, value ) → @end_of_`
`remove_if(@begin, @end, f(○)↦bool) → remaining`



`unique(@begin, @end) → @end_of_remaining`

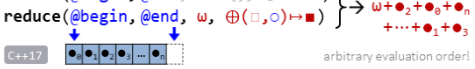
`erase( container , value) → erased_count`

C++20

Numeric Algorithms

<numeric>

`reduce(@begin, @end, ω = ●{}, ⊕ = ○+○ )`
`reduce(@begin, @end, ω, ⊕(□,○)↦■ )`



C++17

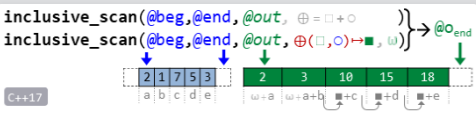
arbitrary evaluation order!

`transform_reduce(@●,@●, ω,⊕ = □+○ ,⊗ = ○×◊)`
`transform_reduce(@●,@●, @♦, ω,⊕(□,Δ)↦■,⊗(○,◊)↦▲)`
`transform_reduce(@●,@●, ω,⊕(□,Δ)↦■,f(○)↦▲) → R_f`

arbitrary evaluation order!

C++17 $R_f = \omega + f(\bullet_0) + f(\bullet_1) + \cdots$ $R_\otimes = \omega + (\bullet_0 \times \blacklozenge_0) + (\bullet_1 \times \blacklozenge_1) + \cdots$

`inclusive_scan(@beg,@end, @out, ⊕ = □+○ )`
`inclusive_scan(@beg,@end, @out, ⊕(□,○)↦■, ω)`



C++17

Sequence Queries

`all_of` (C++11)
`any_of` (C++11)
`none_of` (C++11)
`count` `count_if`
`find` `find_if`
`find_if_not` (C++11)
`find_end`
`find_first_of`
`adjacent_find`
`for_each`
`for_each_n` (C++17)
`sample` (C++20)
`equal`
`mismatch`
`search`
`search_n`
`lexicographical_compare`
`lexicographical_compare_three_way` (C++20)

Reordering Elements

`reverse` `reverse_copy`
`rotate` `rotate_copy`
`shift_left` (C++20) `shift_right` (C++20)
`shuffle` (C++11)
`swap`
`swap_ranges`
`iter_swap`

Partitioning

`is_partitioned`
`partition`
`stable_partition`
`partition_copy`
`partition_point`

Permutations

`is_permutation` (C++11)
`next_permutation` `prev_permutation`

Sorting

`sort`
`stable_sort`
`partial_sort`
`partial_sort_copy`
`is_sorted`
`is_sorted_until`
`nth_element`

Changing Elements

`copy`
`copy_backward`
`copy_if` (C++11)
`copy_n` (C++11)
`move` (C++11)
`move_backward` (C++11)
`fill`
`fill_n`
`generate`
`generate_n`
`transform`
`replace` `replace_if`
`replace_copy` `replace_copy_if`
`remove` `remove_if`
`remove_copy` `remove_copy_if`
`unique`
`unique_copy`

Binary Search on Sorted Ranges

`binary_search`
`lower_bound` `upper_bound`
`equal_range`
`includes`

Merging of Sorted Ranges

`merge`
`inplace_merge`
`set_union`
`set_intersection`
`set_difference`
`set_symmetric_difference`

Heaps

`make_heap`
`sort_heap`
`push_heap`
`pop_heap`
`is_heap` (C++11)
`is_heap_until` (C++11)

Minimum/Maximum

`min` `max`
`min_element` `max_element`
`minmax` (C++11)
`minmax_element` (C++11)
`clamp` (C++17)

Numeric

#include <numeric>

`accumulate`
`adjacent_difference`
`inner_product`
`partial_sum`
`iota` (C++11)
`reduce` (C++17)
`inclusive_scan` (C++17)
`exclusive_scan` (C++17)
`transform_reduce` (C++17)
`transform_inclusive_scan` (C++17)
`transform_exclusive_scan` (C++17)

v3.1

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

#define PI        acos(-1.0)
#define ll        long long
#define ull       unsigned long long
#define sp        ' '
#define vi        vector<int>
#define vll       vector<long long>
#define all(x)    (x).begin(), (x).end()
#define pb        push_back
#define fora(cn)  for(auto &x : (cn))
#define ff(i,n)   for(int i=0;i<n;i++)
#define ff1(i,n)  for(int i=1;i<=n;i++)
#define tc        \
    int t;        \
    cin >> t; \
    while (t--)
#define CY cout<<"YES\n";
#define CN cout<<"NO\n";

int main()
{
    #ifndef ONLINE_JUDGE
        freopen("input.txt","r",stdin);
        freopen("output.txt","w",stdout);
    #endif


    return 0;
}
```

```cpp
sort(all(v),greater<int>()); //boro theke choto
```

string substr(int pos, int n)	Creates a new string object of n characters
int size()	Return the length of the string in terms of bytes
void resize(int n)	Resizes the length of the string up to n characters
string& replace(int pos, int len, string& str)	Replaces the portion of the string beginning at character position pos and spans len characters
string& append(const string& str)	Adds a new character at the end of another string object
char& at(int pos)	Accesses an individual character at specified position pos
int find(string& str, int pos, int n)	Finds a string specified in the parameter
int find_first_of(string& str, int pos, int n)	Find the first occurrence of the specified sequence
int find_first_not_of(string& str, int pos, int n)	Searches for the string for the first character that does not match with any of the characters specified in the string
int find_last_of(string& str, int pos, int n)	Searches for the string for the last character of a specified sequence
int find_last_not_of(string& str, int pos)	Searches for the last character that does not match with the specified sequence
string& insert()	Inserts a new character before the character indicated by the position pos
int max_size()	Finds the maximum length of the string
void push_back(char ch)	Adds a new character ch at the end of the string
void pop_back()	Removes the last character of the string
string& assign()	Assigns new value to the string
int copy(string& str)	Copies the contents of string into another
void clear()	Removes all the elements from the string
const_reverse_iterator crbegin()	Points to the last character of the string
const_char* data()	Copies the characters of string into an array
bool empty()	Checks whether the string is empty or not

2.5 Converting to an array

If you have a vector of items, but you need to call a function that takes an array of items, what do you do?

```
void f(int *a);
vector<int> v;
// want to call f(v);
```

Take advantage of the fact that vectors are guaranteed to be implemented as arrays and pass a pointer to the beginning of the underlying array to the function:

```
void f(int *a);
vector<int> v;
f(&v[0]);
```

Get digits with log

```
int getDigits (ll i)
{
   return i > 0 ? (int) log10 ((double) i) + 1 : 1;
}
```

Compare two double values. (as there

```
// return 0 for a==b, 1 for a>b, -1 for a<b

int comp_double(double a, double b)
{
   if(fabs(a-b) <= 1e-10)
      return 0;
   return a < b ? -1 : 1;
}
```

Function to return LCM of two numbers

```
long long lcm(int a, int b)
{
   return (a / gcd(a, b)) * b;
}
```

Input

3

1 2 3

Output for this :

1

1 2

1 2 3

2

2 3

3

All subsequence of an array :

```
#include<stdio.h>
int main()
{
    int n,k,arr[1010],cnt=0;
    scanf("%d",&n);

    for(int i=0;i<n;i++)
        scanf("%d",&arr[i]);

    for(int i=0;i<n;i++)
    {
        for(int j=i;j<n;j++)
        {
            for(int k=i;k<=j;k++)
            {
                printf("%d ",arr[k]);
            }
            printf("\n");
        }
        //printf("\n");
    }
    return 0;
}
```