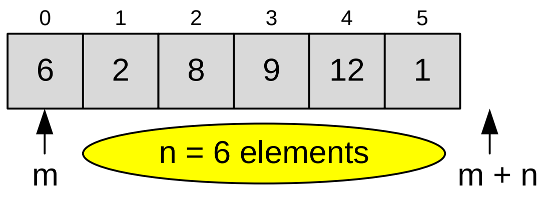
SORTING

Let m_0, m_1, …, m_{n-1} are n integers in array m (`int m[1001]`) to be sorted. You can use STL function ***sort***:

```
sort(m, m + n);
```

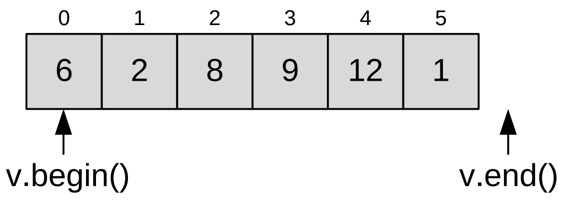
Here

- m = &m[0] is a pointer to the first element;
- $m + n$ = &m[n] is a pointer to the elment that is next to the last element;



If you have a vector (`vector<int> v`), use the next format:

```
sort(v.begin(), v.end());
```



E-OLYMP 2321. Sort Sort array of integers in nondecreasing order.

► Read the data into integer array. Sort the data using STL function ***sort***. Print the data.

Store the array of integers in vector v.

```
vector<int> v;
```

Read the numbers into the array.

```
scanf("%d",&n);
v.resize(n);
for(i = 0; i < n; i++)
  scanf("%d",&v[i]);
```

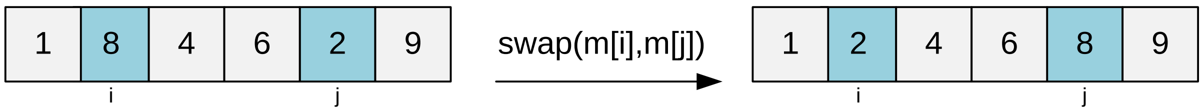
Sort it with ***sort*** function.

```
sort(v.begin(),v.end());
```

Print the elements in nondecreasing order.

```
for(i = 0; i < n; i++)
  printf("%d ",v[i]);
printf("\n");
```

In this problem you can use ***swap sort*** with complexity O(n^2). Since $n \le 1000$, this algorithm will pass the time limit. Iterate over all pairs (i, j), $0 \le i < j < n$, and if m[i] > m[j], swap them.



```
void sort(void)
{
  for (int i = 0; i < n; i++)
  for (int j = i + 1; j < n; j++)
    if (m[i] > m[j])
    {
      int temp = m[i]; m[i] = m[j]; m[j] = temp;
    }
}
```

To sort the array in increasing order in Java, you can use Arrays.sort() method.

To sort the data in increasing or decreasing order you can use **comparators**:

- `sort(m, m + n, less<int>())` – sort in increasing order;
- `sort(m, m + n, greater<int>())` – sort in decreasing order;

To sort the vector v, use

- `sort(v.begin(), v.end(), less<int>())` – sort in increasing order;
- `sort(v.begin(), v.end(), greater<int>())` – sort in decreasing order;

To sort the data in increasing order, you can omit the comparator. Default order is increasing.

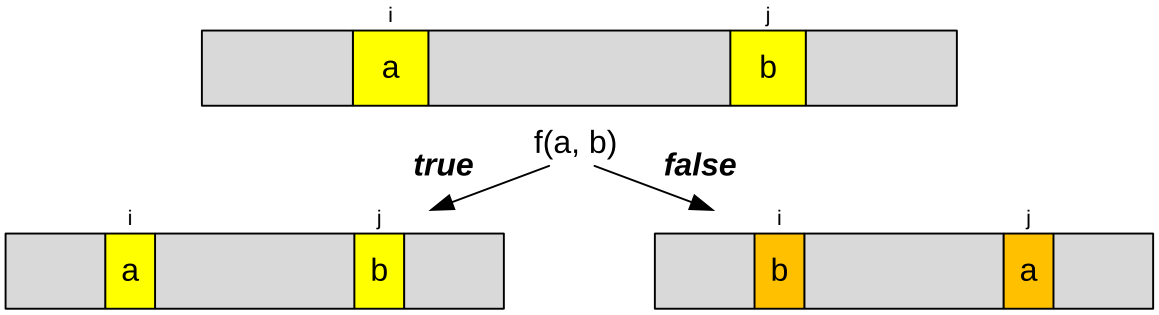
E-OLYMP 4738. Sorting Sort array of integers in non-increasing order.

► Read the data into integer array. Sort the data using STL function ***sort*** and `greater<int>()` comparator. Print the data.

You can write your own comparator. Comparator is a function of the form

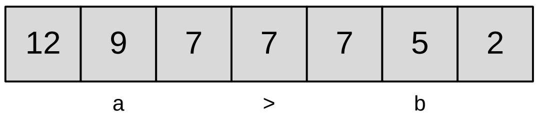
```
int f(type a, type b)
{
  . . .
}
```

that returns ***true*** (1), if elements *a* and *b* should **not** be swapped and ***false*** (0) otherwise. Consider array m and two elements: *a* = m[*i*], b = m[*j*], where *i* < *j*. If m[*i*] and m[*j*] must be swapped to preserve the sorting order, then comparator must return ***false***.



For ***sorted array*** the value of the function f(*a*, *b*) must be ***true*** for any elements *a* and *b* such that position of *a* is less than the position of *b*. The next comparator sorts array of integers in decreasing order:

```
int f(int a, int b)
{
  return a > b;
}
```



Comparator *a* < *b* is equivalent to `less<int>()`.
Comparator *a* > *b* is equivalent to `greater<int>()`.

`int f(int a, int b)`
`{`
`  return a < b;`
`}`
	`less<int>()`

`int f(int a, int b)`
`{`
`  return a > b;`
`}`
	`greater<int>()`

Let’s solve this problem using our own comparator.

```
#include <cstdio>
#include <algorithm>
#define MAX 1001
using namespace std;

int m[MAX];
int i, n;

int f(int a, int b)
{
  return a > b;
}

int main(void)
{
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d", &m[i]);

  sort(m, m + n, f);

  for (i = 0; i < n; i++)
    printf("%d ", m[i]);
  printf("\n");
  return 0;
}
```

E-OLYMP **4848. Quick sort** Sort the given sequence in non-decreasing order.

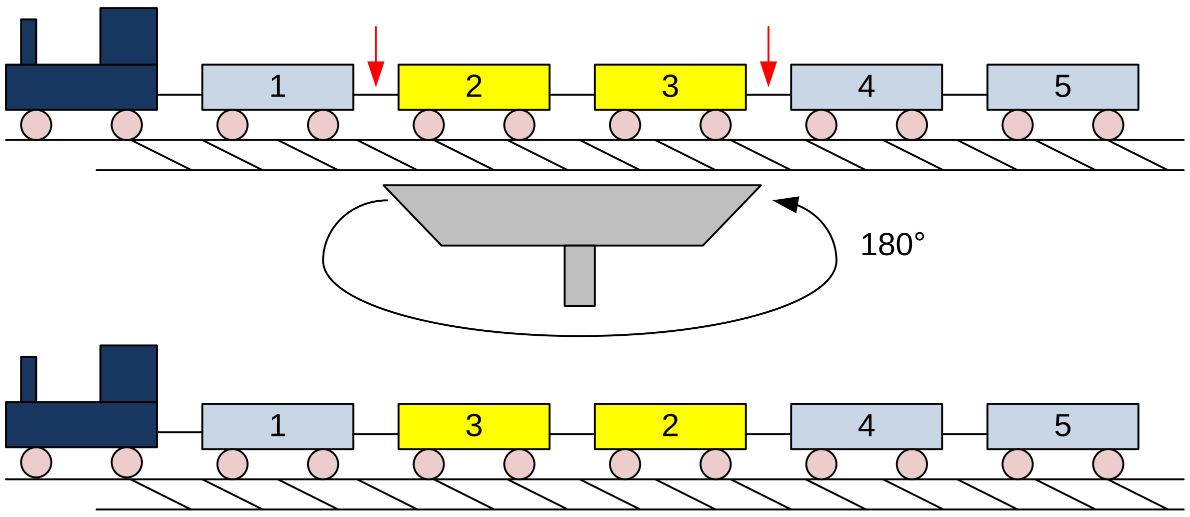
► Read the sequence of numbers into array till the *end of file* and use any sorting algorithm.

E-OLYMP **8735. Train swapping** At an old railway station, you may still encounter one of the last remaining "train swappers". A train swapper is an employee of the railroad, whose sole job it is to rearrange the carriages of trains. Once the carriages are arranged in the optimal order, all the train driver has to do, is drop the carriages off, one by one, at the stations for which the load is meant.

The title "train swapper" stems from the first person who performed this task, at a station close to a railway bridge. Instead of opening up vertically, the bridge rotated around a pillar in the center of the river. After rotating the bridge 90 degrees, boats could pass left or right.

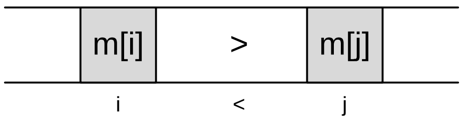
The first train swapper had discovered that the bridge could be operated with at most two carriages on it. By rotating the bridge 180 degrees, the carriages switched place, allowing him to rearrange the carriages (as a side effect, the carriages then faced the opposite direction, but train carriages can move either way, so who cares).

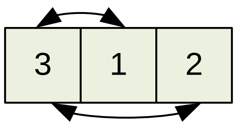
Now that almost all train swappers have died out, the railway company would like to automate their operation. Part of the program to be developed, is a routine which decides for a given train the least number of swaps of two adjacent carriages necessary to order the train. Your assignment is to create that routine.



► Let the array m contains the input permutation – the current order of the carriages. The required minimum number of permutations equals to the number of inversions in the permutation.

An ***inversion*** is a pair of numbers (m_i, m_j) such that $i < j$ but $m_i > m_j$. That is, a pair of numbers forms an inverse if they are not in the correct order.





For example, the array m = {3, 1, 2} has two inversions: (3, 1) and (3, 2). The pair (1, 2) does not form an inversion, since the numbers 1 and 2 stand in the correct order relative to each other.

The number of inversions in the array of length n can be calculated using a double loop: we iterate over all possible pairs (i, j) for which $1 \leq i < j \leq n$, and if $m_i > m_j$, then we have an inversion.

Consider an example of counting inversions in a permutation. Under each number we write down the number of inversions that it forms with the elements to the right of it. Let inv[i] contains the number of j such that $i < j$ and m[i] > m[j].

m[i]	4	8	2	6	5	1	7	3	
inv[i]	3	6	1	3	2	0	1	0	16

The total number of inversions is 16.

Simulate the solution for the next sample.

m[i]	7	2	5	3	6	1	8	4	
inv[i]									

Declare the array m.

```
int m[100010];
```

Sequentially, process the test for the problem.

```
scanf("%d", &tests);

while (tests--)
{
```

Read the input order of the carriages into the array m.

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

The minimum number of permutations for putting the train in order is calculated in the variable *res*.

```
  res = 0;
  for (i = 0; i < n - 1; i++)
  for (j = i + 1; j < n; j++)
    if (m[i] > m[j]) res++;
```

Print the answer.

```
  printf("Optimal train swapping takes %lld swaps.\n", res);
}
```

Sort the letters / strings

E-OLYMP 8316. Sort the letters A string consisting of lowercase Latin letters is given. Sort its letters in ascending and then in descending lexicographical order.

► Sort the strings using **sort** function from STL. Use the comparator `less<int>()` for sorting in increasing order and comparator `greater<int>()` for sorting in decreasing order.

Declare the character array.

```
#define MAX 101
char s[MAX];
```

Read the string.

```
gets(s);
```

Sort the letters in increasing order. Print the resulting string.

```
sort(s,s+strlen(s),less<int>());
puts(s);
```

Sort the letters in decreasing order. Print the resulting string.

```
sort(s,s+strlen(s),greater<int>());
puts(s);
```

Implementation using strings.

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

string s;

int main(void)
{
  cin >> s;
  sort(s.begin(),s.end(),less<int>());
  cout << s << "\n";

  sort(s.begin(),s.end(),greater<int>());
  cout << s << "\n";
  return 0;
}
```

E-OLYMP 2166. Anagrams The word is an anagram of another word, if it can be obtained by rearrangement of its letters.

► Sort the letters in each word in lexicographic order. If the obtained words are the same, then they consist of the same letters and thus are anagrams.

E-OLYMP 2323. Numbers from digits Given nonnegative integer *n*. Create from all its digits the biggest and then the smallest number. Print the sum of obtained numbers.

For example, for *n* = 56002 the biggest will be 65200 and the smallest will be 256 (the leading zeros in number 00256 do not count). The resulting sum is 65200 + 256 = 65456.

► Read the number into character array. Sort the digits in descending order, get the largest number. Sort the numbers in ascending order, get the smallest number. Find and print their sum.

For example, to get the biggest number you can the next way.

```
gets(s); // read the number to har array
sort(s,s+strlen(s),greater<char>()); // sort in decreasing order
sscanf(s,"%d",&a); // get a number from char array
```

E-OLYMP 2325. Two numbers Two integers are given. Print the maximum number that can be obtained from their digits.

For example, from digits of numbers 345 and 6090737 we can get the maximum number 9776543300.

► Read both numbers into a character array, thus gluing them together. Sort the numbers in decreasing order, and get the largest number.

Numbers can also be read in two strings and then concatenated. Then sort the numbers in decreasing order.

E-OLYMP 8317. Square of difference Given positive integer. Find and print the square of difference between the maximum and minimum numbers, composed from the digits of the given number.

For example, if given number is 30605, the maximum number, composed from its digits, is 65300, and minimum number is 356 (the minimum is 00356, but leading zeros are not counted). The required square of difference is (65300 – 356) * (65300 – 356) = 4217723136.

► Read the number into character array. Sort the digits in decreasing order and get the largest number *a*. Sort the digits in increasing order and get the smallest number *b*. Compute their squared difference.