Sortari:

**Direct sort:**

For (int i=0;i<N;++i) {

for(int j=i+1;j<N;++j) {

If (v[i] > v[j]) {

swap(v[i],v[j])

}

}

}

1 **3** 3 2 -> 1 2 3 **3**

**Bubble sort:**

Void bubble\_sort(int a[], int len)

{  
for (int i = 0;i<len;i++)

    for(int j = 0;i<len-i-1;j++)

if(a[j+1] < a[j])

  swap(a[j+1],a[j]);

}

// stabil

Merge sort:

#include <iostream>

typedef cout\_std std::cout

int v[10000], w[10000];

void Interclasare(int v, int s, int d)

{

    int m = (s + d) / 2;

    int i1 = s;

    int i2 = m + 1;

    int k = 0;

    while (i1 <= m and i2 <= d)

    {

**if(v[i1]<=v[i2])**

        {

            w[k] = v[i1];

            k++;

            i1++;

        }

        else

        {

            w[k] = v[i2];

            k++;

            i2++;

        }

    }

    while (i1 <= m)

    {

        w[k] = v[i1];

        i1++;

    }

    while (i2 <= d)

    {

        w[k] = v[i2];

        i2++;

    }

    k++;

    for (int i = s; i <= d; i++)

        v[i] = w[i - s];

}

void MergeSort(v,s,d)

{

    if (d - s <= 2)

    {

        if(v[s]>v[d])

            swap(v[s],v[d]);

    }

    else

    {

        int m = (s + d) / 2;

        MergeSort(v, s, m);

        MergeSort(v, m + 1, d);

        Interclasare(v, s, d);

    }

}

**Quick sort:**

Void quick\_sort(int a[],int len)

{

   If len<=1

     Return;

Int pivot = a[0];

Int l[len],r[len],e[len];

Int i1 = 0, i2 = 0, i3 =0;

for(int i = 0; i<len;i++){

  if(a[i] <= pivot)

L[i1++] = a[i];

Else if(a[i] > pivot)

R[i2++] = a[i];

Else

E[i3++] = a[i];

}

quick\_sort(l,i1);

quick\_sort(r,i2);

Int k = 0;

for(int i = 0;i<i1;i++)

A[k++] =l[i];

for(int i = 0;i<i3;i++)

A[k++] =e[i];

for(int i = 0;i<i2;i++)

A[k++] =r[i];

}

// O(1) memorie in plus

quick\_sort(int v[], int st,int dr) {

Int piv = v[st];

Int idxf = dr;

for(int i = st; i<  idxf; i++)

if( v[i]> piv)

{

int aux = v[idxf];

v[idxf] = v[i];

v[i] = aux;

idxf --;

i--;

}

}

**1**,1,1,0 -> 0,1,1,**1**

1 **3** 3 2 -> 1 2 3 **3**

While (stP < drP) {  
 If (v[stP] > piv) {

-- drP //

} else {

++ stP //

}

}

1 …. N

2 …. N

3 …. N

4 …. N

…

N … N

N + (N-1) + … + 1 =

quick\_sort(A, 1000)

quick\_sort(A, 999)

quick\_sort(A, 998)

…

quick\_sort(A, 1)

3000 + 999 \* 3 + 998 \* 3 … 1 \* 3

3 \* (N \* (N+1))/2

**Radix Sort (varianta LSD):**

#include <iostream>

using namespace std;

int v[100], vnou[100];

void Radix(int v[], int ordin, int nr)

{

    int put = 1;

    for (int k = 0; k < ordin; k++)

    {

        int index = 0;

        for (int i = 0; i <= 9; i++)

        {

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

            {

                if ((v[j] / put) % 10 == i)

                    vnou[index++] = v[j];

            }

        }

        for (int i = 0; i < nr; i++)

            v[i] = vnou[i];

        put \*= 10;

    }

}

int main()

{

    int n, maxim = -999, ord = 0;

    cin >> n;

    for (int i = 0; i < n; i++)

    {

        cin >> v[i];

        if (maxim < v[i])

            maxim = v[i];

    }

    do {

        maxim /= 10;

        ord++;

    } while (maxim>0);

    Radix(v, ord, n);

    for (int i = 0; i < n; i++)

        cout << v[i]<<" ";

    return 0;

}

void RadixSort(int v[], int ordin, int s, int d)

{

    if (d - s +1 >= 2)

    {

        capat = d;

        for (int i = s; i < = capat; i++)

        {

**doi = 1 << ordin;**

            if (v[i] & doi >0)

            {

                swap(v[i], v[capat])

                    capat--;

                i--;

            }

        }

        RadixSort(v, ordin - 1, s, capat);

        RadixSort(v, ordin - 1, capat + 1, d);

    }

}

1010111

0101110

1010111

0111011

1111111

---

1 \* 2 ^ ordin

1, 10, 100, 1000

X << Y

X \* 2 ^ Y

X = 1010110

Y = 5

->

X = 101011000000

X >> Y

X / 2  ^ Y

X = 1010110

Y = 5

->

000000010

V[i] = 11110111011

Doi = 0000010000

V[i] & Doi = 0000010000

V[i] = 11110111011

Doi = 0000100000

V[i] & Doi = 0

**Tema** (deadline pe 14 Martie, sfarsitul zilei):

Implementați următorii 5 algoritmi de sortare (puteti alege altii cat timp sunt peste 5 si nivelul de complexitate e similar, nu scrieti doar algoritmi simplii):

○ BubbleSort

○ CountSort

○ RadixSort

* Prefix
* Suffix

○ MergeSort

○ QuickSort

● Comparati timpii de rulare pe mai multe teste cu numere naturale, între cei 5 algoritmi

dar și cu timpul de rulare al algoritmului de sortare nativ al limbajul de programare ales.

● Ideal este sa veniți cu o prezentare a acestor comparații (chiar cu niște slide-uri,

rapoarte.)

* Tipuri de teste:
  + Vectori aproape sortati
  + Vectori aproape sortati, descrescator
  + Non-integer values (double, string etc.)
  + Complet random
  + Vector constant

* T = numarul de teste
* N = lumgimea vectorului (10, 100, … 10^6)
* A = string algoritmul
* TT = test type (Aproape sortat etc.)

Codul sa il puneti pe github.

Codul:

Test\_vector = generaza(test);

startTime = time()

sort\_x(Test\_vector);

endTime = time();

print(endTime);