Подредување на елементите кај еднодимензионални низи (selection sort, bucket sort, merge sort, bubble sort)

1. Selection Sort

Прво се бара најмалиот елемент и тој го заменува своето место со првиот елемент во низата. Постапката се повторува онолку пати колку што е долга низата без разлика дали низата веќе е сортирана. Овој алгоритам е временски спор. Подредува без разлика дали почетната низа е веќе подредена или не е. Вкупниот број е:

$$n(n — 1)/2 + 3(n — 1) = O(n^2).$$

Пр.1. Да се напише програма со која се подредуваат елементите од низа а со н елементи во растечки редослед и да се отпечати новодобиената низа.

```
#include <iostream>
using namespace std;
int main()
{
   int n,i,j,a[100],p;
   cout<<"Vnesi go brojot na elementi vo nizata"<<endl;
   cin>>n;
   for(i=0;i<n;i++)
   {
      cin>>a[i];
   }

   for(i=0;i<n-1;i++)
   {
      for(j=i+1;j<n;j++)
          {
      if(a[i]>a[j])
      {
         p=a[i];
         a[i]=a[j];
         a[j]=p;
      }
          }
    }
    cout<<"Podredena nizata e "<<endl;

    for(i=0;i<n;i++)
   {
      cout<<a[i]<<"  ";
   }
   return 0;
}
```

Пр. Да се напише програма со која се подредуваат елементите од низа а со н елементи во опаѓачки редослед и да се отпечати новодобиената низа.

```
#include <iostream>
using namespace std;
int main()
```

```
{
   int n,i,j,a[100],p;
   cout<<"Vnesi go brojot na elementi vo nizata"<<endl;
   cin>>n;
   for(i=0;i<n;i++)
   {
      cin>>a[i];
   }

   for(i=0;i<n-1;i++)
   {
      for(j=i+1;j<n;j++)
          {
      if(a[i]<a[j])
      {
          p=a[i];
          a[i]=a[j];
          a[j]=p;
      }
          }
    }
    cout<<"Podredena nizata e "<<endl;

    for(i=0;i<n;i++)
   {
      cout<<a[i]<<"  ";
   }
   return 0;
}
```

Пр. Да се подреди низата [ai]n во растечки редослед, со истовремено наоѓање и на најмалиот и на најголемиот елемент.
Објаснување: Се употребува следната постапка: Најмалиот елемент од a_1 до a_n го доведуваме на прво место, а најголемиот на n-то место, најмалиот елемент од a_2 до a_{n-1} го доведуваме на второ место, а најголемиот на (n-1)-во место итн.

```
#include<iostream>
using namespace std;
main()
{
int a[100];
int n,i,j,pom;
cout << "Vnesi go brojot na elementi "<<endl;
cin>>n;
cout << "Vnesi gi elementite na nizata " << endl;
for(i=0;i<n;i++)
{
     cin >> a[i];
}
for(i=0;i<n/2;i++)
{
  for(j=i;j<=n-i-1;j++)
```

```
  {
    if(a[j]<a[i])
    {
      pom=a[j];
      a[j]=a[i];
      a[i]=pom;
    }
    if(a[j]>a[n-i-1])
    {
      pom=a[j];
      a[j]=a[n-i-1];
      a[n-i-1]=pom;
    }
  }
}
cout<<"Podredenata niza e:"<<endl;
for(i=0;i<n;i++)
  cout<<a[i]<<" ";
cout<<endl;
return 0;
}
```

2. bubble sort:

Се споредуваат два соседни елементи од низата и ако се погрешно распоредени ги заменуваат местата. По споредување на секои два соседни парови најмалиот елемент е на крајот на низата. Постапката се повторува се до подредувањето на низата.

```
#include <iostream>
using namespace std;
int main()
{
int n,pom,i,j;
int a[100];
cout<<"Vnesi go n"<<endl;
cin>>n;
cout<<"Vnesi gi elementite na nizata"<<endl;
for(i=0; i<n; i++)
{
   cin>>a[i];
}
for(i=n-1; i>0; i--)
{
for(j=0; j<i; j++)
{
if(a[j]<a[j+1])
{
pom=a[j];
a[j]=a[j+1];
a[j+1]=pom;
}
}
}
```

```
 cout<<"Podredenata niza e"<<endl;
 for(i=0; i<n; i++)
 cout<<a[i]<<" ";
 return 0;
 }
```

3. Сортирање со спојување (Merge Sort)

Алгоритмот работи со рекурзија. Низата се дели на две помали поднизи со иста должина над кои се повикува рекурзивната постапка за Merge sort. Потоа двете поднизи се спојуваат во една.

Овој алгоритам е еден од најбрзите за подредување. Предноста е тоа што може да се подреди голема количина на податоци, а недостаток е големото количество на меморија што се користи бидејќи се креира дополнителна низа за спојување при подредувањето.

```
#include<iostream>
using namespace std;
void zamena(int &a, int &b) {     //swap the content of a and b
   int temp;
   temp = a;
   a = b;
   b = temp;
}
void pecati(int *a, int n)
 {
   for(int i = 0; i<n; i++)
      cout << a[i] << " ";
   cout << endl;
}
void mergesort(int *a, int l, int m, int r)
{
   int i, j, k, nl, nr;
   //size of left and right sub-arrays
   nl = m-l+1; nr = r-m;
   int b[nl], c[nr];
   //fill left and right sub-arrays
   for(i = 0; i<nl; i++)
      b[i] = a[l+i];
   for(j = 0; j<nr; j++)
      c[j] = a[m+1+j];
   i = 0; j = 0; k = l;
   //marge temp arrays to real array
   while(i < nl && j<nr) {
      if(b[i] <= c[j]) {
         a[k] = b[i];
         i++;
      }else{
         a[k] = c[j];
         j++;
      }
      k++;
   }
   while(i<nl) {       //extra element in left array
      a[k] = b[i];
      i++; k++;
   }
   while(j<nr) {     //extra element in right array
      a[k] = c[j];
      j++; k++;
   }
}
```

```
void mergeSortt(int *a, int l, int r) {
  int m;
  if(l < r) {
    int m = l+(r-l)/2;
    // Sort first and second arrays
    mergeSortt(a, l, m);
    mergeSortt(a, m+1, r);
    mergesort(a, l, m, r);
  }
}
int main() {
  int n;
  cout << "vnesi broj na elementi: ";
  cin >> n;
  int a[n];    //create an array with given number of elements
  cout << "vnesi elementi:" << endl;
  for(int i = 0; i<n; i++) {
    cin >> a[i];
  }
  cout << "nesortirano pole: ";
  pecati(a, n);
  mergeSortt(a, 0, n-1);    //(n-1) for last index
  cout << "sortirano pole: ";
  pecati(a, n);
}
```

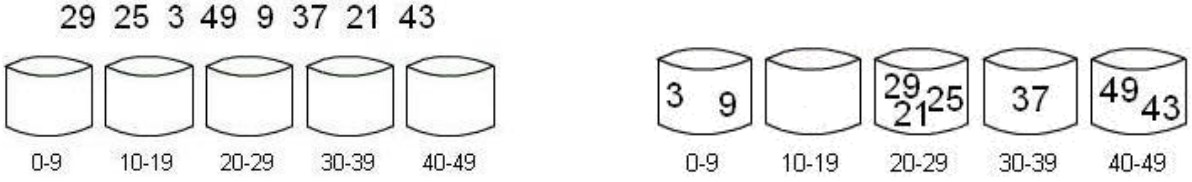
4. Bucket sort (кофа сортирање)- O(број на кофи * временска сложеност на алгоритмот за сортирање)- најчесто со Insertion sort кој има O(n) .

Insertion sort – код

```
for(i=1;i<=n;i++)
 {
    j=i;
    pom=A[j];
    while((j>1) && (A[j-1]>pom))
    {
      A[j]=A[j-1];
      j--;
    }
   A[j]=pom;
 }
```

Алгоритам за Bucket sort:

1)Се распределуваат елементите во поднизи (кофи), со задоволување на одреден критериум.



2) Секоја подниза (кофа) се сортира посебно со некој метод за сортирање

3) Се спојуваат сортираните поднизи (кофи) во оригинална низа.