**QUICK SUMMARY**

**Prasanje 7. Binarno prebaruvanje**

#include <iostream>

using namespace std;

int bsearch(int n, int a[10], int K)

{

int left = 0, right = n - 1;

while (left <= right)

{

int middle = (left + right) / 2;

if(a[middle] == K) return middle;

else

if(K < a[middle]) right = middle - 1;

else

left = middle + 1;

}

return -1;

}

**Kruzno pomestuvanje**

#include <iostream>

using namespace std;

int main()

{

int n; cout<<" vnesi broj na elementi vo niza: "; cin>>n;

int a[n];

for(int i=0; i<n; i++) { cout<<" vnesi element "<<i+1<<":"; cin>>a[i]; }

cout<<" vnesovte niza: ";

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

int tmp = a[n-1];

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

{

a[n-i]=a[n-i-1];

}

a[0]=tmp;

cout<<" obrabotenata niza e:";

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

return 0;

}

**Selection Sort -> O(n^2)**

#include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

void selectionSort(int arr[], int n)

{

    int i, j, min\_idx;

    for (i = 0; i < n-1; i++)

    {

        min\_idx = i;

        for (j = i+1; j < n; j++)

        if (arr[j] < arr[min\_idx])

            min\_idx = j;

        swap(&arr[min\_idx], &arr[i]);

    }

}

**Bubble Sort -> O(n^2)**

#include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

void bubbleSort(int arr[], int n)

{

    int i, j;

    for (i = 0; i < n-1; i++)

    for (j = 0; j < n-i-1; j++)

        if (arr[j] > arr[j+1])

            swap(&arr[j], &arr[j+1]);

}

**Quick Sort -> O(n log(n))**

#include <bits/stdc++.h>

using namespace std;

void swap(int\* a, int\* b)

{

    int t = \*a;

    \*a = \*b;

    \*b = t;

}

int partition (int arr[], int low, int high)

{

    int pivot = arr[high];

    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)

    {

        if (arr[j] < pivot)

        {

            i++;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[high]);

    return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

**Merge Sort -> O(n\*log(n))**

#include <iostream>

using namespace std;

void merge(int a[], int l, int s, int d)

{

int i, j, k;

int n1 = s - l + 1;

int n2 = d - s;

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

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

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

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

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

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{ a[k] = L[i];

i++;

}

else

{ a[k] = R[j];

j++;

}

k++;

}

**Bucket Sort -> O(n+k)**

void bucketSort(float arr[], int n)

{

    for (int i = 0; i < n; i++) {

        int bi = n \* arr[i]; // Index in bucket

        b[bi].push\_back(arr[i]);

    }

    for (int i = 0; i < n; i++) {

        sort(b[i].begin(), b[i].end());

    int index = 0;

    for (int i = 0; i < n; i++)

        for (int j = 0; j < b[i].size(); j++)

            arr[index++] = b[i][j];

}

**zamena na elementi na pozicii s, t:**

int main()

{

int n; cout<<" vnesi broj na elementi vo niza: "; cin>>n;

int A[n], temp;

for(int i=0; i<n; i++) { cout<<" vnesi element "<<i+1<<":"; cin>>A[i]; }

cout<<" vnesovte niza: ";

for(int i=0; i<n; i++) { cout<<A[i]<<" "; }

int s,t;

cout<<endl<<" vnesi pozicii s,t:"; cin>>s>>t;

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

{

if ( i == s ) {

temp = A[i];

A[i] = A[t];

A[t] = temp; }

}

cout<<" obrabotenata niza e:";

for(int i=0; i<n; i++) { cout<<A[i]<<" "; }

return 0;

**BFS & DFS search**

BFS: zapocnuvame so bilo koj node I gi proveruvame site negovi povrzani jazli, (pr ako sme pocnale od 1, a 1 e povrzan so 5 4 2, gi zapisuvame 5, 4, 2 vo bilo kakov redosled), proveruvame dali tie jazli imaat nivni vrski I tie vrski drugi se dodeka ne gi istrazime site jazli

DFS: depth first search -> (so primer na zadaca 8) pocnuvame od node A: A, C, B, D ili A, B, D, C

**SE OSTANATO OD DOKUMENTITE:**

Тема 2: Алгоритми кај повеќе-димензионални низи и графови

Потсетување за еднодимензионални низи.

(дефинирање, декларирање и иницијализација и поминување низ еднодимензионална низа)

Вовед

ОСНОВНИ ТИПОВИ ПОДАТОЦИ

Прво да се потсетиме за основните типови податоци.

Основните податочни типови во C/C++ се:

* bool (boolean – точно и неточно),
* char (скратено од character - знак),
* int (integer - цел број),
* float (скратено од floating-point number – број со подвижна децимална точка - децимален број т.е. реален број) и
* double (double precision floating point number - децимален број со двојна прецизност).

Дефиницијата на некои од овие типови може да се измени/дообјасни.

Тоа се прави со помош на зборовите: signed, unsigned, short и long. Сите расположливи основни податочни типови се дадени во следната табела:

|  |  |  |  |
| --- | --- | --- | --- |
| **име** | **опис** | **големина** | **опсег** |
| bool | две вредности - точно (true) и погрешно (false) | 1 бајт (8 бита, од кои 7 се игнорираат) | true и false |
| char | може да се користи за чување знаци или цели броеви | 1 бајт | Доколку се чуваат цели броеви, опсегот е од -128 до 127, но може да биде и 0-255 (кај некои архитектури). За правилно извршување, кога сакате да чувате цели броеви, секогаш користете unsigned char или signed char (види ги следните 2 податочни типови). |
| signed char | исто како char, но гарантирано може да чува и негативни броеви | 1 бајт | -128 до 127 |
| unsigned char | исто како char, но гарантирано може да чува само позитивни броеви | 1 бајт | 0 до 255 |
| short short int signed short signed short int (сите имаат ист ефект) | цели броеви (позитивни и негативни) | 2 бајти | -32768 до 32767 |
| unsigned short unsigned short int | позитивни цели броеви | 2 бајти | 0 до 65535 |
| long long int signed long signed long int | цели броеви (позитивни и негативни) | 4 бајти | -2147483648 до 2147483647 |
| unsigned long unsigned long int | позитивни цели броеви | 4 бајти | 0 до 4294967295 |
| long long long long int signed long long signed long long int | цели броеви (позитивни и негативни) | 8 бајти | -9223372036854775808 до 9223372036854775807 |
| unsigned long long unsigned long long int | позитивни цели броеви | 8 бајти | 0 до 18446744073709551615 |
| float | децимални броеви | 4 бајти | околу 7 точни цифри |
| double | децимални броеви, двојна прецизност | 8 бајти | околу 15 точни цифри |
| long double | децимални броеви | 10 (или 12) бајти | околу 22 точни цифри |

Главен дел

**Еднодимензионални низи (полиња) од цели броеви**

**Низа претставува група од мемориски локации кои имаат исто име и чии елементи се од ист тип.**

За да означиме специфичен елемент на низата, го задаваме името на низата и позицијата на тој елемент во низата. Доколку сакаме да пристапиме до елемент од низата тоа се врши преку името на низата и редниот број на елементот во низата. Со оглед на големината на низите тие можат да бидат еднодимензионални и повеќедимензионални низи.

Еднодимензионални полиња или низи претставуваат множество од елементи кои се зачувани и може да се внесуваат, печатат и обработуваат. Низите можат да бидат од различен тип на податок и тоа целоброен тип на податок, децимален тип на податок и знаковен тип на податок.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 6 | 7 | 8 | 9 | 2 | 3 | 1 | 12 | 76 |

a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]

Редни броеви таканаречени индекси на елементите од низата се: 0,1,2,3,4,5,6,7,8,9

***Декларација на низа***

**декларација значи резервирање на простор во RAM меморијата за елементите од низата.**

Тоа е слично како кога сакаме да одиме во кино, треба да си купиме карти со што си резервираме столче. Ако сме спречени може и да не отидеме, нашето столче ќе остане празно. Но ако не купиме карта никако не можеме да влеземе во кино салата.

Така е и со меморијата, може да се резервира простор за повеќе елементи, а да внесеме помалку, но не можеме да резервираме помалку, а да сакаме да внесеме повеќе елементи во низата. Непополнетите места ќе имаат вредност 0.

Исто така мораме да кажеме каков ќе биде типот на податоците во низата. Сите податоци мора да се од ист тип.

**општа Наредба за декларација**

**tip ime\_niza[вредност];**

**Декларацијата може да биде:**

* **статична**
* **динамична**

**Пример за статична декларација:**

|  |  |  |
| --- | --- | --- |
| **Декларација на 10 цели броеви** | **Декларација на 5 децимални броеви** | **Декларација на 6 знаци** |
| **int a[10];** | **double b[5];** | **char c[6]** |

**Пример за динамична декларација:**

|  |  |  |
| --- | --- | --- |
| **Декларација на n цели броеви** | **Декларација на 5 децимални броеви** | **Декларација на 6 знаци** |
| **int n;**  **cin>> n;**  **int a[n];** | **int n=5;**  **double b[n];** | **int n;**  **cin>>n;**  **char c[n];** |

**Од примерот забележуваме дека во секој случај пред да ја декларираме низата треба процесорот да знае точно колку елементи ќе има таа.**

**Најчесто се користи статичната декларација.**

## **Иницијализација И ВНЕСУВАЊЕ НА ПОДАТОЦИ ВО НИЗА**

**Иницијализација на низа значи доделување на вредности на елементите од низата.**

Кај основните типови податоци со декларирањето не се доделува вредност на променливата, но кај низите со декларирање се доделува вредност 0 на секој елемент од низата.

Подоцна можеме да внесеме вредности по желба. За оние елемeнти за кои нема да внесеме вредности, нивната вредност ќе остане 0.

На елементите може да им доделиме вредност по креирање на низата (како во примерите дадени погоре), или да им доделиме вредност при самата декларација на низата. Во C++, ова се прави со набројување на вредностите:

Доделувањето на вредности(иницијализација) се врши со наредбата:

**int array[10] = {4, 6, 7, 8, 9, 2, 3, 1, 12, 76};**

**Броењето почнува од реден број 0, затоа што 0 се рачуна за позитивен број.**

Можно е и креирање на низи без задавање на нивна големина (int array[]). Тогаш, компајлерот сам ќе ја одреди големината на низата според бројот на елементи во листата за иницијализација:

**float array[ ] = {4.2, 8.1, 5.66, 4.455, 5.1}; //niza od 5 decimalni t.e. realni broevi**

**ВНЕСУВАЊЕ НА НИЗА ПРЕКУ ТАСТАТУРА**

|  |  |
| --- | --- |
| for(i=0;i<10;i++)  cin>>a[i]; | внесување на низа од десет елементи (циклусот се движи од 0 до 9) |
| for(i=0;i<n;i++)  cin>>a[i]; | внесување на низа од n елементи (циклусот се движи од 0 до n-1) |

**ПЕЧАТЕЊЕ НА НИЗА НА ЕКРАН**

|  |  |
| --- | --- |
| for(i=0;i<10;i++)  cout<<a[i]; | печатење на низа од десет елементи (циклусот се движи од 0 до 9) |
| for(i=0;i<n;i++)  cout<<a[i]; | печатење на низа од n елементи (циклусот се движи од 0 до n-1) |

Еднодимензионалните полиња познати како низи најлесно се внесуваат, печатат, пребаруваат и пребројуваат со користење на структура за повторување со броење на циклусите. Во овој пример треба да се искористат два циклуса и тоа еден за внесување и еден за печатење на низата.

|  |  |
| --- | --- |
| ***Графички алгоритам*** | ***Текстуален алгоритам (со псевдо-јазик)*** |
|  | Почеток на програмата  За вредност на бројачот i од 0 до помалку од 10 т.е. до 9, зголемувај го i за чекор 1  Внесувај еден по еден елемент од низата:  за i=0 внеси го a[0],  зголеми го i за 1,  потоа внеси го a[1] и т.н. се додека i не добие вредност помала од 10 т.е. до 9.  За вредност на бројачот i од 0 до 9, зголемувај го i за чекор 1.  за i=0 печати го a[0],  зголеми го i за 1, потоа печати го a[1] и т.н. се додека i не добие вредност помала од 10.  Крај на програмата |

|  |
| --- |
| ***Програмски код во C++*** |
| #include<iostream>  using namespace std;  int main()  {  int a[10],i;    cout<<"\n Vnesete gi elementite na nizata "<<endl;  for(i=0;i<10;i++)  {  cout<<"\n a["<<i<<"]=";  cin>>a[i];  }  cout<<"\n Vnesenata niza e : " ;  for(i=0;i<10;i++)  cout<<" "<<a[i];  return 0;  } |

**Задача 2:**

Да се внесе низа од 10 цели броеви и да се печатат само **парните** броеви од низата.

***Анализа:***

Во оваа задача дополнително треба да се искористи контролна структура со еден избор, за проверка кои елементи од низата се парни.

|  |  |
| --- | --- |
| ***Графички алгоритам*** | ***Текстуален алгоритам (со псевдо-јазик)*** |
|  | Почеток на програмата  За вредност на бројачот i од 0 до 9, зголемувај го i за чекор 1  Внесувај еден по еден елемент од низата,  за i=1 внеси го a[1],  зголеми го i за 1, потоа внеси го вториот елемент од низата a[2] и т.н. се додека i не добие вредност помала од 10.  За вредност на бројачот i од 0 до 9, зголемувај го i за чекор 1.  Испитувај дали секој елемент од низата е парен, и ако е парен печати го на екран. |

***Програмски код во C++***

#include <iostream> //povikaj ja bibliotekata za vlez i izlez

using namespace std; //koristi go standardniot proctor za iminja

int main() //deklaracija na glavnata funkcija(programa)

{ //pocetok na glavnata funkcija(programa)

int a[10],i; //deklaracija na nizata a, deklaracija na promenlivata i

cout<<"\n Vnesete gi elementite na nizata "<<endl; //pecatenje na poraka na ekran

for(i=0;i<10;i++) // struktura za povtoruvanje so broenje na ciklusite

{ //pocetok na strukturata za povtoruvanje

cout<<"\n a["<<i<<"]="; //pecatenje na poraka na ekran

cin>>a[i]; //naredba za vnesuvanje na elementite od nizata

} //kraj na strukturata za povtoruvanje

cout<<"\n Vnesenata niza e : " ; //pecatenje na poraka na ekran

for(i=0;i<10;i++) //struktura za povtoruvanje so broenje na ciklusite

cout<<" "<<a[i]; //pecatenje na vrednostite od nizata

for(i=0;i<10;i++) //struktura za povtoruvanje so broenje na ciklusite

if(a[i]%2==0) cout<<a[i]; // ispituvanje ako a[i] e delivo so 2 da se pecati

return 0;

}

**Задача 3:**

Да се напише програма со која ќе се внесе низа од n реални броеви. Потоа да се најде најголемиот и најмалиот елемент од низата.

***Анализа:***

Во оваа задача треба да се употребат два циклуса , еден за наоѓање на најголемиот , и еден за наоѓање на најмалиот елемент од низата броеви.

***Програмски код во C++***

//Naogjanje na najgolem element od edna niza broevi

#include <iostream>

using namespace std;

int main()

{

int n,i,k,m;

float a[20],max,min,tmp;

cout<<"Vnesi broj na elementi od nizata-->";

cin>>n;

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

{

cout<<"a["<<i<<"]=";

cin>>a[i];

}

max=a[0];

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

{

if (max<a[i])

{

max=a[i];

k=i;

}

}

cout<<"Najgolemiot element od dadenata niza broevi e "<<max<<endl;

min=a[0];

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

{

if (min>a[i]) {

min=a[i];

m=i;

}

}

cout<<"Najmaliot element od dadenata niza broevi e "<<min<<endl;

tmp=a[k];

a[k]=a[m];

a[m]=tmp;

cout<<" Dadenata niza vo koja se zameneti najgolemiot i najmaliot element sega izgleda vaka"<<endl;

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

cout<<a[i]<<endl;

return 0;

}

**Завршен дел(Рефлексија)**

**Заклучок:**

Најбитно е да се направи разлика помеѓу вредност и позиција.

a[i] – преставува вредност во низата, тоа се сите елементи кои ги внесува корисникот преку тастатурата. i - преставува позиција т.е. реден број на елементите од низата, имено позициите се фиксни од нула до бројот на елементи минус еден.

**Домашна задача:**

1. Да се напише програма за внесување на низа од од n реални броеви, а потоа да се печатат елементите со парни индекси.

Алгоритми за пребарување.

Линеарно пребарување и негова сложеност.

Што значи „сложеност на алгоритми“?

Секој алгоритам се состои од конечен број инструкции. Колку повеќе иструкции, толку подолго тој ќе се извршува. Еден начин е да се бројат инструкциите (операциите) од кој е составен алгоритмот за решавање одреден проблем. Потребно е да се знае колку пати се извршува инструкцијата и која е “нејзината цена” т.е. времето (честота) на едно извршување на инструкцијата.

Нека се дадени следниве програмски сегменти:

|  |  |
| --- | --- |
| а) x=x+y; | а) честота на извршување е 1 |
| б) for(i=1;i<=n;i++)  x=x+y; | б) честота на извршување е n |
| в) for(i=1;i<=n;i++)  for(ј=1;ј<=n;ј++)  x=x+y; | в) честота на извршување е n2 |

Пример

|  |  |
| --- | --- |
| //Da se presmeta zbir na prvite n prirodni broevi so koristenje na naredbata for  #include <iostream>  using namespace std;  int main()  { int i,n,s;  cout<<"Do koj broj da se soberat prvite n prirodni broevi"<<endl;  cin>>n;  s=0;  for(i=1;i<=n;i++)  {  s=s+i;  }  cout<<"s="<<s<<endl;  return 0;  } | 1  1  1  n  1  1  1  1 |

Функцијата што го преставува бројот на инструкции за овој алгоритам може да се запише како f(n)=n+7.

Оваа функција на сложеност е линеарна, што значи и сложеноста на овој алгоритам е линеарна. Но мора да напоменеме дека оваа функција е зависна од сметачот и програмскиот јазик во кој се извршува.

Бројот на инструкции ќе варира зависно од бројот на внесените податоци.

Пример во програма за пресметка на плати, за влез од 100 лица ќе изврши повеќе инструкции одколку за 10 лица. Затоа може да го изразиме бројот на инструкции како функција на сложеност од бројот на внесени податоци.

Нека f(n) е функција на сложеност што зависи од компјутерот на кој се извршува алгоритмот, а g(n) функција на сложеност што не зависи од тоа на каков компјутер се извршува алгоритмот. Тогаш важи f(n)=O(g(n)). Кога ќе се каже ...Алгоритмот има O(g(n)) време на пресметување (извршување)... Тоа значи дека ако алгоритмот се извршува повеќе пати на иста машина, на ист тип податоци, но за различно (растечко) n, времето на извршување секогаш ќе биде помало од некоја константна вредност |g(n)|. Значи функцијата на сложеност која не зависи од тоа на каков сметач се извршува алгоритмот во примерот 5 е g(n)=n, и велиме алгоритмот има сложеност О(n) (се чита “од ред n ”).

Но дали треба да ги земеме во предвид сите инструкции во алгоритмот?

Најчесто алгоритмите се состојат од циклус и инструкции што се извршуваат во него или надвор од него. Бројот на инструкциите надвор од циклусот во алгоритми што се однесуваат за ист проблем незначително се разликува. Од нив не зависи дали алгоритмот ќе има помала или поголема сложеност. Ако бројот на влезни податоци расте тогаш бројот на инстукции што се извршуваат во циклусот го надминува бројот на оние што се надвор од него. За да ја упростиме нашата пресметка нема да ги броиме инструкциите надвор од циклусот. Дали треба да ги земеме во предвид сите инструкции во циклусот? Бројот на инструкции во циклусот незначително се разликуваат во 2 алгоритми напишани за ист проблем. Но, тоа што се разликува е колку пати циклусот се извршува.

Што значи линеарно пребарување?

Линеарно пребарување на низа значи пребарување на низата од почетниот елемент до последниот и наоѓање на вредноста која се бара. Можно е да не се најде.

Задача 1:

Да се напише програма со која ќе се внесе низа од n реални броеви. Потоа да се внесе број и да се најде дали бројот е во низата или не со линеарно пребарување.

|  |
| --- |
| //**Linearno prebaruvanje na niza - dali postoi vrednosta V vo nizata**  #include <iostream>  using namespace std;  int main()  { int n,i,v,m;  int a[10];  cout<<"Vnesi broj na elementi na nizata ";  cin>>n;  for(i=0;i<n;i++)  {  cout<<"a["<<i+1<<"]=";  cin>>a[i];  }  cout<<"Vnesi ja vrednosta sto se bara ";  cin>>v;  i=0;  while (i<n)  {  if (a[i]==v)  {  m=i;  break; //pri prvoto naogjanje na brojot ke prekine ciklusot  }  i++;  }  if (v==a[m]) cout<<"Vrednosta sto ja baravte e a["<<m+1<<"]="<<a[m]<<endl;  else cout<<"Vo nizata ne postoi vrednosta sto ja baravte "<<endl;    return 0;  } |

|  |
| --- |
| Тест пример: низата има 7 елементи, и тие се: 5,8,20,2,11,15,10  Бројот 11 ќе биде најден по 5 итерации (вртежи) на циклусот.  Сложеноста на алгоритмот е 5. |

Забелешка: Овој алгоритам е брз бидејќи прекинува постапката кога бројот ќе се најде. Но што ако бројот се повторува во низата. Тогаш треба да се пребарува низата до крај, за да се најдат и другите и на кои позиции се.

Задача 2:

Да се напише програма со која ќе се внесе низа од n реални броеви. Потоа да се внесе број и да се најде на која позиција е и колку пати бројот се повторува во низата.

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int i,n,br=0;  float a[20],x;  cout<<"Kolku elementi ima nizata n=? "<<endl;  cin>>n;  cout<<"Vnesi ja nizata "<<endl;  for(i=0;i<n;i++)  cin>>a[i];  cout<<"Koj broj se bara x=? ";  cin>>x;  for(i=0;i<n;i++)  if (a[i]==x)  {  cout <<"Najden e brojot "<<x<<" na pozicija "<<i<<endl;  br++;  }  if (br!=0) cout<<"Brojot "<<x<<" se sodrzi "<<br<<" pati vo nizata "<<endl;  else cout<<"Brojot "<<x<<" ne se sodrzi vo nizata"<<endl;  return 0;  } |

Забелешка:

Овој алгоритам е побавен но мора да се пребара целата низа за да се најдат сите броеви кои повторуваат.

Домашна работа:

Да се напише програма за линеарно пребарување на број во низа со користење на функција за линеарно пребарување.

**БИНАРНО ПРЕБАРУВАЊЕ (Binary search)**

Пребарување е една од основните операции кои се среќаваат при решавањето на разни проблеми. Во ова предавање, ќе разгледаме еден од основните и најкористените алгоритми од оваа област – т.н. бинарно пребарување. Иако алгоритамот сам по себе е доста едноставен, тој наоѓа примена во многу подрачја, и се користи во разни компјутерски програми и системи.

Наједноставниот начин да го сфатиме алгоритамот е да го замислиме следниот проблем: погодување на замислен број. Имено, нека некое лице одбере произволен број од 1 до 100, и нека треба ние да го погодиме бројот кој е замислен. Притоа, при секој наш обид, добиваме одговор дали бројот кој што сме го кажале е помал од замислениот, поголем, или пак доколку веќе сме го погодиле точниот број.

Како можеме да го решиме овој проблем? Нормално, кажувајќи ги сите броеви почнувајќи од 1, 2, 3, ..., па се додека не погодиме, е едно можно решение - но, секако, не е најдоброто кое што можеме да го постигнеме. Дали постои начин да ја искористиме повратната информација што ја добиваме во однос на тоа дали нашиот кажан број е помал или поголем од замислениот? Се разбира дека можеме - со т.н. алгоритам за бинарно пребарување. Притоа, како што ќе видиме од повеќе примери во ова предавање, самата постапка е исклучително едноставна – при секој чекор, го разгледуваме средниот број од интервалот (на пример, ако разгледуваме броеви од 1 до 100, средниот број е 50). Доколку тој број е поголем од замислениот, го спроведуваме истиот алгоритам на левата половина од преостанатиот интервал (1 - 49); а, ако е помал од замислениот, на десната половина од преостанатиот интервал (51 – 100). Така продолжуваме се додека не го погодиме точниот број. Притоа, овој алгоритам има логаритамска сложеност O(logN), и е исклучително едноставен. На пример, доколку со истиот се обидуваме да најдеме број во телефонски именик од сите луѓе на планетата (каде имињата се подредени, па можеме, на сличен начин, да го делиме интервалот на половина), ќе ни биде потребно да направиме само околу 30 проверки. Тука, ја гледаме моќта на бинарното пребарување.

Сега, да разгледаме уште поконкретен пример. Нека имаме подредена низа од цели или реални броеви. Користејќи ја истата идеја која што ја видовме погоре, можеме со логаритамска сложеност **O(logN)** да пронајдеме дали еден елемент постои во низата. Притоа, алгоритамот е исклучително едноставен за реализација. Забележете дека, во кодот даден во продолжение, со left и right ги означуваме границите на интервалот кој го разгледуваме – на почеток, тој ја опфаќа целата низа, а како што споредуваме елементи, го делиме истиот (во секој чекор) на половина. Ова е идејата која овозможува бинарното пребарување да функционира толку ефикасно.

**#include <iostream>**

**using namespace std;**

**int bsearch(int n, int a[10], int K)**

**{**

**int left = 0, right = n - 1;**

**while (left <= right)**

**{**

**int middle = (left + right) / 2;**

**if(a[middle] == K) return middle; //prvo se ispituva dali vrednosta se sovpagja so sredniot element od nizata**

**else**

**if(K < a[middle]) right = middle - 1; //prodolzhi so levata polovina na intervalot**

**else**

**left = middle + 1; //prodolzhi so desnata polovina na intervalot**

**}**

**return; //ako vrednosta ne e najdena vo nizata da vrati -1**

**}**

**int main()**

**{**

**int n, i, K, x,a[10];**

**cout<<"vnesete go brojot na elementi na nizata n=";**

**cin>>n;**

**cout<<"Vnesete gi elementite na nizata vo rastecki redosled"<<endl;**

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

**cin>>a[i];**

**cout<<"Vnesete ja vrednosta na elementot sto se bara ";**

**cin>>K;**

**x=bsearch(n,a,K);**

**if (x==-1) cout<<"Vrednosta "<<K <<" ne e vo nizata"<<endl;**

**else**

**cout<<"Vrednosta sto ja baravte e na pozicija "<<x<<endl;**

**return 0;**

**}**

Алгоритми за сортирање

1. Алгоритам меурче (Bubble sort)

I начин: Подредување со споредување на два соседни елементи од низата

<https://www.geeksforgeeks.org/bubble-sort/>

**Методот на меурче(bubble sort) се состои во следното:**

Овој алгоритам работи преку замена на соседни елементи кои се наоѓаат во погрешен редослед. Името на алгоритмот доаѓа од таму што поголемите елементи испливуваат (како меурчиња) на крајот на низата.

**Постапка:**

**Се споредуваат соседните елементи од низата a[j] со a[j+1] , почнувајки од почетниот елемент a[0] и завршувајќи со претпоследниот т.е. се споредуваат a[n-2] со a[n-1].**

**Пример:**

* **За i=0 и j=0 т.е. во првата итерација од циклусот по j , се споредуваат**

**a[0] и a[1]. Aко а[0] > a[1] си ги заменуваат местата, ако не, нема замена на местата на a[0] и a[1].**

**Потоа за i=0 и ј=1 се споредува a[1] со a[2] и ако a[1]>a[2] тие си ги заменуваат местата.**

**Постапката на споредување продолжува се до претпоследниот елемент**

**a[j]=а[n-2-i] т.е. ако a[n-2-0]>a[n-2-0+1] т.е. ако a[n-2]>a[n-1] тие си ги заменуваат местата. На тој начин на последно место доаѓа најголемиот и тој повеќе не се разгледува.**

* **За i = 1, j почнува пак од 0 и се движи до n-2-i т.е. до n-2-1=n-3.**

**Се споредува а[0] со a[1], потоа a[1] со a[2] и т.н. се до претпретпоследниот (за еден помалку, бидејќи i=1) односно затоа што последниот е веќе фиксиран на крајот на низата и тој не треба да се разгледува.**

**Вгнездените циклуси се следните:**

**for(i=0;i<=n-2;i++)**

**for(j=0;j<=n-2-i;j++)**

**{ if(a[j]>a[j+1] //ако се бара во опаѓачки редослед само знакот „>“ се заменува со знак „<“**

**{ //zameni im gi mestata**

**tmp=a[j];**

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

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

**}**

**}**

|  |
| --- |
| **I начин**  /\*Da se sortira(podredi) vo opagacki redosled nizata od broevi so metodot na meurche  i so koristenje na funkcija koja se povikuva od glavnata funkcija main \*/  #include <iostream>  using namespace std;  void bubble\_sort(int n, float a[])  { int i,j,tmp;  for(i=0; i<=n-2; i++)  for(j=0; j<=n-2-i; j++)  {  if (a[j]>a[j+1])  {  tmp=a[j];  a[j]=a[j+1];  a[j+1]=tmp;  }  }  cout<<"Sortiranata niza e: "<<endl;  for(i=0;i<=n-1;i++)  cout<<a[i]<<" ";  }  int main()  {  int i,n,j;  float a[20],tmp;  cout<<"Vnesi broj na elementi od nizata ";  cin>>n;  cout<<"Vnesi ja nizata so edno prazno mesto megju broevite \n";  for(i=0;i<=n-1;i++)  cin>>a[i];  bubble\_sort(n,a);  return 0;  } |

**II начин: Подредување со споредување на секој елемент со сите останати елементи**

**Се споредуваат елементите почнувајки од првиот со сите останати и ако првиот е помал од вториот си ги заменуваат местата. Потоа пак се споредува првиот со третиот елемент и ако третиот е поголем си ги заменуваат местата. Постапката на споредување на првиот со останатите елементи продолжува се до последниот елемент. На тој начин на прво место доаѓа најмалиот. Потоа се споредува вториот со третиот , четвртиот и т.н. се додека не се споредат и подерадат сите елементи.**

**За таа цел се користат два циклуса вгнездени еден во друг.**

|  |
| --- |
| **II начин**  /\*Da se sortira(podredi) vo opagacki redosled nizata od broevi so metodot na meurche  i so koristenje na funkcija koja se povikuva od glavnata funkcija main \*/  #include <iostream>  using namespace std;  void bubble\_sort(int n, float a[])  { int i,j,tmp;  for(i=0;i<=n-2;i++)  for(j=i+1;j<=n-1;j++)  {  if (a[i]>a[j])  {  tmp=a[i];  a[i]=a[j];  a[j]=tmp;  }  }  cout<<"Sortiranata niza e: "<<endl;  for(i=0;i<=n-1;i++)  cout<<a[i]<<" ";  }  int main()  {  int i,n,j;  float a[20],tmp;  cout<<"Vnesi broj na elementi od nizata ";  cin>>n;  cout<<"Vnesi ja nizata so edno prazno mesto megju broevite \n";  for(i=0;i<=n-1;i++)  cin>>a[i];  bubble\_sort(n,a);  return 0;  } |

Анализата на временската сложеност на овој алгоритам е малку потешка - бидејќи сега имаме циклус за кој не знаеме колку пати ќе се изврши.

Всушност, почетната позиција на елементите има многу значајна улога при одредувањето на времето на извршување на овој метод. Доколку низата е веќе подредена, сложеноста е линеарна O(N), и тоа е најдоброто време на извршување на алгоритамот. Доколку имаме големи елементи на почетокот [5, 1, 2, 3, 4, 5], тие не претставуваат сериозен проблем бидејќи брзо се преместуваат до нивната крајна позиција. Но, доколку имаме мали елементи на крајот од низата [2, 3, 4, 5, 5, 1], тие многу бавно се движат до почетокот.

На пример, за да се подреди низата [2, 3, 4, 5, 5, 1] потребен е квадратен број на операции.

Значи сложеноста на Алгоритмот Bubble sort е О(n2).

**Во просек, подредувањето со методот на меурче е еден од најбавните алгоритми за подредување - околу 40% е побавен од алгоритамот подредување преку одбирање.**

**Прашање:**

**Што ќе се промени во програмата ако треба да се сортира низата во опаѓачки редослед од поголем кон помал број?**

**Домашна задача:**

1. Да се напише програма за кружно поместување на елементите од една низа броеви за едно место надесно, така што 9 ќе дојде на почетна позиција а сите останати за едно место ќе се поместат надесно.

**Пример:**

Од низата 2 6 5 1 4 9 да се добие низата: 9 2 6 5 1 4

2

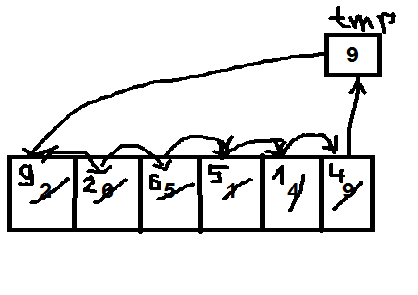
6

5

1

9

4



Циклусот е следниот:

tmp=a[n-1];

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

{

a[n-i]=a[n-i-1];

}

a[0]=tmp;

Напишете целосна програма со потпрограма-функција со овој циклус која ќе се повикува од главната функција main().

На екран треба да се печати новодобиената низа.

Сортирање со селекција и негова сложеност

(анг. Selection sort )

**Задача 1.**

1. Да се напише програма со која ќе се внесе една низа од n броеви, а потоа да се сортираат (подредат) во растечки редослед броевите од низата.

подредување преку одбирање (анг. selection sort)

За секоја позиција во низата (почнувајќи од 0 и завршувајќи со n-1) пронајди го елементот кој треба да се наоѓа на таа позиција во низата и смести го таму. За време на извршување на алгоритамот, низата (виртуелно) ќе се подели на два дела: подреден дел (на почетокот на низата) и неподреден дел (на крајот од низата - тоа се всушност елементите кои сеуште не сме ги сместиле на точната позиција). Во литературата, овој алгоритам е познат под името "подредување преку одбирање" (анг. selection sort).

|  |
| --- |
| **//Selection sort so funkcija**  //Da se sortira(podredi) vo rastecki redosled nizata od broevi so metodot na odbiranje  #include <iostream>  using namespace std;  void selection\_sort(int n, int a[])  {  int i,m,tmp;  for(i=0;i<=n-1;i++)  {  m = i;  for (int j=i+1; j<=n-1; j++)  if (a[m] > a[j]) m = j; //nov minimum    if (m != i) //zameni  {  int tmp = a[i];  a[i] = a[m];  a[m] = tmp;  }  }  cout<<"Sortiranata niza e: "<<endl;  for(i=0;i<=n-1;i++)  cout<<a[i]<<" ";  }  int main()  {  int i,n,j,m;  int a[20],tmp;  cout<<"Vnesi broj na elementi od nizata ";  cin>>n;  cout<<"Vnesi ja nizata so edno prazno mesto megju broevite \n";  for(i=0;i<=n-1;i++)  cin>>a[i];    selection\_sort(n, a);  return 0;  } |

**Истата програма да се напише со користење на функција како потпрограма.**

|  |
| --- |
| **//Selection sort so funkcija**  //Da se sortira(podredi) vo rastecki redosled nizata od broevi so metodot na odbiranje  #include <iostream>  using namespace std;  void selection\_sort(int n, int a[])  {  int i,m,tmp;  for(i=0;i<=n-1;i++)  {  int m = i;    for (int j=i+1; j<=n-1; j++)  if (a[m] > a[j]) m = j; //nov minimum    if (m != i) //zameni  {  int tmp = a[i];  a[i] = a[m];  a[m] = tmp;  }  m=i;  }  cout<<"Sortiranata niza e: "<<endl;  for(i=0;i<=n-1;i++)  cout<<a[i]<<" ";  }  int main()  {  int i,n,j,m;  int a[20],tmp;  cout<<"Vnesi broj na elementi od nizata ";  cin>>n;  cout<<"Vnesi ja nizata so edno prazno mesto megju broevite \n";  for(i=0;i<=n-1;i++)  cin>>a[i];    selection\_sort(n, a);  return 0;  } |

Анализата на временската сложеност е едноставна: за секоја од N-те позиции во низата треба да направиме споредба на тековниот елемент со околу N/2 елементи (оние елементи кои се наоѓаат десно од него). За првиот елемент (niza[0]) треба да направиме N-1 споредба, за вториот (niza[1]) треба да направиме N-2 споредби, итн. до последниот елемент за кој треба да направиме 0 споредби (тој сигурно е на точната позиција, бидејќи е последен елемент и нема со кого да го споредуваме)

Збирот е 0 + 1 + 2 + 3 + ... + (N-1), што е збир на елементите на позната аритметичка прогресија (првите X природни броеви) и изнесува 0 + 1 + 2 + 3 + ... + (N-1) = (N\*(N-1)) / 2. Значи, алгоритамот има временска сложеност O(N2).

-Која е разликата помеѓу методот на меурче и методот на сортирање со селекција?

Алгоритам „Кофа сортирање (Bucket sort)“

## **Вовед и активирање на претходни знаења**

## **Повторување за „сортирање на низа со метод на меурче”**

Прашања:

-Како се сортира низа од броеви со метод на меурче?

-Која е временската сложеност на алгоритмот?

-Како се сортира низа од броеви со метод на селекција?

-Која е временската сложеност на алгоритмот?

-Кој од двата алгоритми е побрз?

**Задача 1.**

1. Да се напише програма со која ќе се внесе една низа од n броеви, а потоа да се сортираат (подредат) во растечки редослед броевите од низата со алгоритам

**Кофа сортирање (Bucket sort)**.

трет начин за подредување-сортирање

1. алгоритам кофа сортирање (анг. bucket sort)

Овој алгоритам е сортирање со распределба.

Пример: Да се сортира низата 3,1,4,1,5,9,2,6 со методот Bucket sort!

Се дели низата на поднизи (кофи). Секоја кофа –подниза се сортира поединечно.

Чекори:

1. се дефинира низа од празни кофи-сегменти (поднизи)
2. се пребарува почетната низа и се сместуваат елементите во соодветна кофа
3. сортирање на секоја кофа поединечно
4. се посетува секоја кофа и се ставаат елементите во првата низа.

|  |
| --- |
| //Da se podredi nizata a vo rastecki redosled primer a={3,6,1,5,2,9} ako elementite ne se povtoruvaat  //bez funkcija  #include <iostream>  using namespace std;  int main()  {  int n,i,k=0,m,a[10],b[100];  cout<<"n=";  cin>>n;    cout<<"Vnesete ja nizata"<<endl;  for(i=0;i<n;i++)  cin>>a[i];    m=0;  for(i=0;i<n;i++)  if (a[i]>m) m=a[i]; //naogjanje na najgolemiot element vo nizata a, ako m=9 togas nizata b ke ima 8 elementi bidejki se pocnuva so broenje od 0    cout<<"Najgolemiot vo nizata a e "<<m<<endl;  for(k=0;k<m;k++)  b[k]=0;    i=0;  while(i<n)  {  k=a[i]; //vo j se vnesuva vrednosta na a[0], primer ako a[0]=3 sleduva j=3  b[k-1]=a[i]; //vo b[2] ke vleze 3  i++;  }  //pecatenje na nizata od korpi (buckets) ke vidime deka ima nuli  for(k=0;k<m;k++)  cout<<b[k]<<' ';  //Vrakanje nazad na elementite od nizata b koi ne se 0 vo nizata a  i=0;  for(k=0;k<m;k++)  {  if (b[k]!=0) {  a[i]=b[k];  i++;  }  }    cout<<"\nSortiranata niza e:"<<endl;  for(i=0;i<n;i++)  cout<<a[i]<<' ';  return 0;  } |

|  |
| --- |
| /\*Da se podredi nizata a vo rastecki redosled primer a={3,6,1,5,2,9} ako elementite ne se povtoruvaat (so funkcija)\*/  #include <iostream>  using namespace std;  void bucket\_sort(int n, int m, int a[ ],int b[ ])  {  int k,i;  i=0;  while(i<n)  {  k=a[i]; //vo j se vnesuva vrednosta na a[0], primer ako a[0]=3 sleduva j=3  b[k-1]=a[i]; //vo b[2] ke vleze 3  i++;  }  //Vrakanje nazad na elementite od nizata b koi ne se 0 vo nizata a  i=0;  for(k=0;k<m;k++)  {  if (b[k]!=0)  {  a[i]=b[k];  i++;  }  }  cout<<"\nSortiranata niza e:"<<endl;  for(i=0;i<n;i++)  cout<<a[i]<<' ';  }  int main()  {  int n,i,k=0,m,a[10],b[100];  cout<<"n=";  cin>>n;    cout<<"Vnesete ja nizata"<<endl;  for(i=0;i<n;i++)  cin>>a[i];    m=0;  for(i=0;i<n;i++)  if (a[i]>m) m=a[i]; //naogjanje na najgolemiot element vo nizata a, ako m=9 togas nizata b ke ima 8 elementi bidejki se pocnuva so broenje od 0    cout<<"Najgolemiot vo nizata a e "<<m<<endl;    for(k=0;k<m;k++)  b[k]=0;  bucket\_sort(n,m,a,b);  return 0;  } |

|  |
| --- |
| //I nacin Bucket\_sort ako elementite se povtoruvaat  #include <iostream>  using namespace std;  void bucketSort(int a[],int n,int m)  {  int k,b[100], i, j;  for (i=0; i<m; i++) b[i]=0;  k=0;  for (i=0; i<=m; i++)  for (j=0; j<n; j++)  {  if(a[j]==i)  {  b[k]=a[j];  k++;  }  }  cout<<"\n Sortiranata niza e : "<<endl;  for(int k=0;k<n;k++)  cout<<b[k]<<' ';  }  int main()  {  cout<<"Vnesi broj na elementi na niza:";  int n,m;  cin>>n;  int a[n],i;  cout<<"Vnesete ja nizata "<<endl;  for(int i=0;i<n;i++)  cin>>a[i];    m=a[0];  for(i=0;i<n;i++)  if (a[i]>m) m=a[i]; //m e najgolemiot element vo nizata a  cout<<"Najgolem element e "<<m<<endl;  bucketSort(a,n,m);  return 0;  } |

|  |
| --- |
| // II nacin Bucket sort (ako elementite se povtoruvaat)  //BUCKET sort  #include <iostream>  using namespace std;  void bucketSort(int a[],int n,int m)  {  int k,b[100], i, j,pom;  for (i=0; i<m; i++) b[i]=0;    for (i=0; i<n; i++) // se formira niza b koja sodrzi kolku pati se pojavuva sekoj broj od 0 do maximalniot element m  {  pom=a[i];  b[pom]= b[pom]+1; //b e niza od brojot na povtoruvanjanja na sekoj element vo nizata  }  cout<<"Nizata od broj na pojavuvanja na sekoj broj od 0 do maksimalniot broj m vo nizata e "<<endl;  for(k=0;k<=m;k++)  cout<<b[k]<<" ";    k=0;  for (i=0; i<=m; i++) //gi vrti site broevi od 0 do maksimumot m  for (j=0; j<b[i]; j++) // gi vrti broevite od 0 do brojot na povtoruvanja na dadeniot broj vo nizata  {  a[k]=i;  k++;  }  cout<<"\n Sortiranata niza e : "<<endl;  for(int k=0;k<n;k++)  cout<<a[k]<<' ';  }  int main()  {  cout<<"Vnesi broj na elementi na niza:";  int n,m;  cin>>n;  int a[n],c[100],i;  cout<<"Vnesete ja nizata "<<endl;  for(int i=0;i<n;i++)  cin>>a[i];    m=a[0];  for(i=0;i<n;i++)  if (a[i]>m) m=a[i]; //m e najgolemiot element vo nizata a  cout<<"Najgolem element e "<<m<<endl;  bucketSort(a,n,m);  return 0;  } |

Доколку се работи за големи броеви, тие се сместуваат во групи во посебни корпи. Пример ако треба да се подредат броевите од 1 до 100 , може да се сместат во групи по 10 елементи во 10 корпи и потоа секоја корпа да се подреди посебно, а потоа да се спојат елементите од корпите.

**Домашна работа**

- Да се истражи на Интернет за сортирање со алгоритмот Quick sort и да се објасни со зборови како работи .

- Да се истражи на Интернет за сортирање со алгоритмот Merge sort и да се објасни со зборови како работи .

Сортирање со спојување и негова сложеност

(merge sort)

Основна идеја:

Идејата за сортирање кај овој алгоритам е слична на идејата на алгоритмот за брзо сортирање. И овој алгоритам е од типот раздели па владеј, најпрво низата која треба да се сортира ја разделува на половина т.е. на две поднизи и потоа секоја подниза се сортира независно, на крај двете сортирани поднизи се мешаат со што се добива оригиналното решение.

Сложеност на алгоритамот за подредување со спојување е O(N\*logN).

Прочитај на <http://mendo.mk/Lecture.do?id=18>

*Пример:*

Со помош на метод на спојување (merge sort) да се сортира низата 38,27,43,3,9,82,10.

Во секој чекор низата се разделува на половина и рекурзивно се повикува процедурата за сортирање за левата и десната подниза, и потоа процедурата за спојување.

Алгоритам

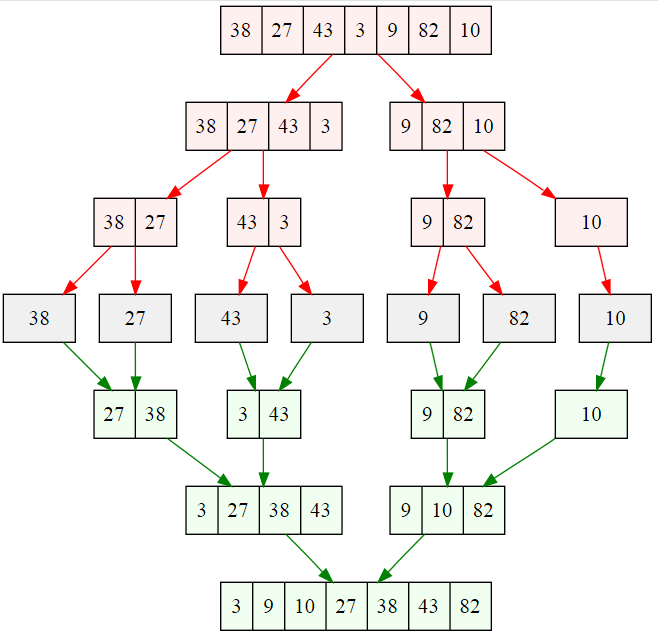
Алгоритмот може да се опише со следните чекори:

1.Раздели ја низата на две поднизи со должина n/2 ,n/2

2.Рекурзивно сортирај ја секоја од двете поднизи.

3.Измешај (спои) ги сортираните поднизи за да се добие крајниот резултат.

Базен случај за алгоритмот е подниза со точно еден елемент и таа се зема за сортирана.



Имплементација

Постојат повеќе начини за имплементација на процедурата за мешање (спојување = merge) , но таа најчесто се имплементира на следниот начин:

Двете поднизи кои се мешаат најпрво се копираат во една помошна низа, потоа двете поднизи сега составен дел од помошната низа со помош на индекси кои покажуваат кои се нивни делови се заменуваат и помалиот елемент од поднизите се копира назад во почетната низа.

Може да настане ситуација така што едниот индекс да дојде до крајот а другиот не, што означува дека во едната подниза има уште елементи и тогаш природно е преостанатите елементи да се ископираат на крајот од почетната низа.

Следниот програмски код е една имплементација на алгоритмот за сортирање со мешање.

|  |  |
| --- | --- |
| //msLR.cpp  #include <iostream>  using namespace std;  void merge(int a[], int l, int s, int d)  {  int i, j, k;  int n1 = s - l + 1;  int n2 = d - s;  int L[n1], R[n2];  /\* Copy data to temp arrays L[] and R[] \*/  for (i = 0; i < n1; i++)  L[i] = a[l + i];  for (j = 0; j < n2; j++)  R[j] = a[s + 1+ j];  /\* Merge the temp arrays back into arr[l..r]\*/  i = 0; // Initial index of first subarray  j = 0; // Initial index of second subarray  k = l; // Initial index of merged subarray  while (i < n1 && j < n2)  {  if (L[i] <= R[j])  { a[k] = L[i];  i++;  }  else  { a[k] = R[j];  j++;  }  k++;  }  /\* Copy the remaining elements of L[], if there are any \*/  while (i < n1)  { a[k] = L[i];  i++;  k++;  }  /\* Copy the remaining elements of R[], if there are any \*/  while (j < n2)  { a[k] = R[j];  j++;  k++;  }  }  //rekurzivno sortiranje na dvete podnizi  void mergeSort(int a[], int l, int d)  {  if (l < d)  { int s = l+(d-1)/2;  mergeSort(a, l, s);  mergeSort(a, s+1, d);  merge(a, l, s, d);  }  } | //glavna funkcija  int main()  {  int i,n,a[10];  cin>>n;  for (i=0; i<n; i++)  cin>>a[i];  mergeSort(a, 0, n - 1);  cout<<"\n Sortiranata niza e \n";  for (i=0; i<n; i++)  cout<<a[i]<<" ";  return 0;  } |

Домашна работа:

Да се разгледа објаснувањето на следните линкови:

<http://mendo.mk/Lecture.do?id=18>

<https://www.youtube.com/watch?v=TzeBrDU-JaY>

<https://www.youtube.com/watch?v=JSceec-wEyw>

<https://www.youtube.com/watch?v=Pr2Jf83_kG0>

<https://www.youtube.com/watch?v=jlHkDBEumP0>

<https://www.interviewbit.com/tutorial/merge-sort-algorithm/>

**Вежби со низи, пребарување и сортирање на низа**

**Во сите програми да се користи потпрограма – функција која ќе се повикува од главната функција main**

1. Да се напише програма со која ќе се најдат сите елементи во една низа од реални броеви со вредност v и на кои позиции се наоѓаат во низата.

2. Да се напише програма со која ќе се најдат сите елементи колку пати се повторуваат во низата од цели броеви.

3. Да се напише програма со која циклично ќе се поместат

а) за едно место надесно елементите од една низа.

б) за едно место налево елементите од една низа.

<https://www.loom.com/share/a338fd8656bd4d6ca0ff7558ae67f759>

4. Да се напише програма со која ќе се преврти низата во обратен редослед на своите елементи.

for(i=0;i<(n-1)/2;i++)

{

t=a[i];

a[i]=a[n-i-1];

a[n-i-1]=t;

}

5. Да се напише програма со која ќе се соберат две низи со еднаков број на елементи.

Упатство: збир на две низи a[] Iи b[] е низа c[] чии елементи се добиени како збир на соодветните елементи од поединечните низи.

Пример c[0]=a[0]+b[0], c[1]=a[1]+b[1], c[2]=a[2]+b[2] … c[n-1]=a[n-1]+b[n-1]

6. Да се напише програма со која ќе се сортираат елементите во една низа од броеви со метод на Меурче Bubble sort кога секои два соседни елементи си ги заменуваат местата.

7. Да се напише програма со која ќе се подредат елементите во опаѓачки редослед со алгоритам –Bucket sort.

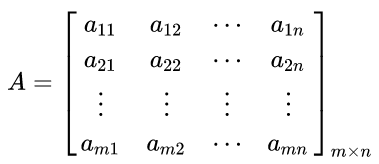
8. Да се напише програма со која ќе се подреди една низа од цели броеви , а потоа ќе се најде дадена вредност v со помош на бинарно пребарување.

**Дводимензионални низи – матрици**

**Матрицата е еден вид низа која може да има две или три димензии т.е. да биде претставена во рамнина или во простор.**

**Ние сега ќе ги претставиме дводимензионалните матрици кои уште се наречени и дводимензионални низи.**

**Во математиката сте запознаени со поимот матрица и знаете дека се запишува на следниот начин:**



**Каде m е бројот на редици, а n е бројот на колони на матрицата.**

**Во информатиката матрицата се обележува како правоаголна шема од m редици и n колони исто како и во математиката, само се почнува со броење од 0 наместо од 1, и индексите се поставени во средни загради.**

**Пример за матрица со 3 редици и 4 колони е претставена на следниот начин:**

|  |  |  |  |
| --- | --- | --- | --- |
| **a[0][0]** | **a[0][1]** | **a[0][2]** | **a[0][3]** |
| **a[1][0]** | **a[1][1]** | **a[1][2]** | **a[1][3]** |
| **a[2][0]** | **a[2][1]** | **a[2][2]** | **a[2][3]** |

Како и еднодимензионалните низи така и дводимензионалните низи т.е. матриците мора да се декларираат во програмата со што се резервира мемориски простор за нејзините елементи.

Дводимензионална низа т.е. матрица во С++ се декларира со следната наредба:

**тип име[m][n];**

каде што m го означува бројот на редиците, а n бројот на колоните во матрицата. И при тоа матрицата има вкупно m× nелементи.

**Пример:**

**Матрица a од цели броеви со 10 редици и 20 колони ќе ја декларираме на следниот начин:**

**int a[10][20];**

Елементот кој припаѓа на i-тата редица и j-тата колона на матрицата **а** се претставува со: **а[i][j],** каде што: i =0,1,2,3....m-1 j=0,1,2,3,…,n-1

**Пример**: Декларација на матрица.

1. int a[3][4]; //декларирана е матрица од цели броеви со 3 редици и 4 колони

a[0][0] a[0][1] a[0][2] a[0][3]

a[1][0] a[1][1] a[1][2] a[1][3]

a[2][0] a[2][1] a[2][2] a[2][3]

б) Димензиите на матрицата можат да се зададат и преку константи:

const int m=5;

const int n=5;

int a[m][n];

**Индексите** на елементите на некоја низа (матрица) можат да бидат: константи, променливи или изрази од **целоброен тип**:

c[2][3], c[7][i], c[i][j+2, c[j][i+j], c[i+j-1][j-i+1] итн.,

каде што i и j се целобројни променливи.

**Доделување вредност и иницијализација на матрица**

1. Наредба за доделување:

float a[2][2];

a[0][0]=1.0;

a[0][1]=2.0;

a[1][0]=3.0;

a[1][1]=4.0;

б) Декларирање со иницијализирачка листа:

int b[2][3]={1, -2, -3, 6, 3,5}; // (доделувањето е по редици)

еквивалентно е со:

int b[2][3]={{1, -2, -3},{6, 3,5}};

в) Со

int b[2][3]={{1},{6}};

се иницијализираат само елементите: b[0][0]=1 и b[1][0]=6, а останатите елементи

добиваат вредност 0.

г) Со

int b[2][3]={1, -2, -3};

се иницијализираат само елементите од првата редица: b[0][1]=1, b[0][2]=-2 и

b[0][3]=-3, а останатите елементи добиваат вредност 0.

Внесување на елементите на матрицата преку тастатура мора да биде со два вгнездени циклуса, еден по i и еден по j.

Циклусот на внесување на елементите на матрицата **а** е следниот:

for(i=0;i<m;i++)

for(j=0;j<n;j++)

cin>>a[i][j];

Притоа треба да се внесуваат вредностите на следниот начин:

кога ќе се заврши со првиот ред се притиска Ентер па се пишуваат елементите од вториот ред и т.н.

Пример ако матрицата е:

|  |  |  |
| --- | --- | --- |
| **2** | **3** | **5** |
| **6** | **0** | **7** |

Треба да биде внесена на следниот начин:

-во прв ред се пишува 2 3 5 (Enter)

-во втор ред се пишува 6 0 7

Друг начин на внесување на елементите на матрицата е со нагласување кој елемент се внесува:

for(i=0;i<m;i++)

for(j=0;j<n;j++)

{

cout<<"a["<<i<<"]["<<j<<"]=";

cin>>a[i][j];

}

Побрз е првиот начин на внесување.

**Излез на матрицата на екран**

**За да се испечати матрицата во форма на матрица треба да се користат два вгнездени циклуси, едниот за да се печатат сите елементи од секоја редица.**

**Тоа се прави на следниот начин:**

**Прво се печатат елементите од првата редица, потоа треба да се спушти курсорот во следен ред па потоа да се печатат елементите од втората редица и т.н. за секоја наредна редица.**

**for(i=0; i<m; i++)**

**{**

**cout<<endl;**

**for(j=0;j<n;j++)**

**cout<<a[i][j]<<' ';**

**}**

**Задача 1:**

**Да се напише програма со која ќе се внесе една mxn матрица и истата ќе се испечати на екран.**

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int i,j,m,n,a[20][10];  cout<<"Vnesete broj na redici na matricata m= ";  cin>>m;  cout<<"Vnesete broj na koloni na matricata n= ";  cin>>n;  cout<<"Vnesete ja matricata"<<endl;  for(i=0;i<m;i++)  for(j=0;j<n;j++)  cin>>a[i][j];    cout<<"Matricata sto ja vnesovte e "<<endl;  for(i=0;i<m;i++)  {  cout<<endl;  for(j=0;j<n;j++)  cout<<a[i][j]<<' ';  }  return 0;  } |

**Задача 2:**

**Да се напише програма со која ќе се соберат елементите на една општа mxn матрица од реални броеви.**

|  |
| --- |
| **//Zbir na elementi na edna opsta mxn matrica od realni broevi**  #include <iostream>  using namespace std;  int main()  {  float a[10][10], s=0;  int i,j,m,n;  cout<<"Vnesi go brojot na koloni vo matricata ";  cin>>n;  cout<<"Vnesi go brojot na redici vo matricata ";  cin>>m;  cout<<"Vnesi ja matricata "<<endl;  for(i=0;i<=m-1;i++)  for(j=0;j<=n-1;j++)  {  cin>>a[i][j];  s=s+a[i][j];  }    cout<<" Matricata sto ja vnesovte e "<<endl;  for(i=0;i<=m-1;i++)  {  cout<<endl;  for(j=0;j<=n-1;j++)  cout<<a[i][j]<<" ";  }    cout<<" Zbirot od elementite na dadenata matrica e "<<s<<endl;  return 0;  } |

**ЗАДАЧИ СО МАТРИЦИ**

**Да се напише програма со која ќе си ги заменат местата редиците со реден број h и k. h и k ги внесува корисникот.**

<https://www.loom.com/share/280437a2e03f424193756208650b9427>

|  |
| --- |
| **//zamena na redicite so reden broj h i k na edna opsta matrica. h i k gi vnesuva //korisnikot.**  #include <iostream>  using namespace std;  int main()  {  float a[20][10],tmp;  int m,n,i,j,h,k;  cout<<"Broj na koloni na matricata= ";  cin>>n;  cout<<"Broj na redici na matricata= ";  cin>>m;  cout<<"Vnesete ja matricata "<<endl;  for(i=0;i<m;i++)  for(j=0;j<n;j++)  cin>>a[i][j];  cout<<"Vnesete gi rednite broevi na redicite koi sakate da si gi zamenat mestata "<<endl;  cin>>h>>k;  for(j=0;j<n;j++)  {  tmp=a[h-1][j];  a[h-1][j]=a[k-1][j];  a[k-1][j]=tmp;  }  cout<<" Novata matrica e "<<endl;  for(i=0;i<m;i++)  {  cout<<endl;  for(j=0;j<n;j++)  cout<<a[i][j]<<" ";  }  return 0;  } |

**Да се напише програма со која ќе си ги заменат местата колоните со реден број p и q (p и q ги внесува корисникот) .**

[**https://www.loom.com/share/e7556f7716924cabb76b024cd4187dcf**](https://www.loom.com/share/e7556f7716924cabb76b024cd4187dcf)

|  |
| --- |
| //zamena na kolonite so reden broj p i q na edna opsta matrica. p i q gi vnesuva korisnikot.  #include <iostream>  using namespace std;  int main()  {  float a[20][10],tmp;  int m,n,i,j,p,q;  cout<<"Broj na koloni na matricata= ";  cin>>n;  cout<<"Broj na redici na matricata= ";  cin>>m;  cout<<"Vnesete ja matricata "<<endl;  for(i=0;i<m;i++)  for(j=0;j<n;j++)  cin>>a[i][j];    cout<<"Vnesete gi rednite broevi na kolonite koi sakate da si gi zamenat mestata "<<endl;  cin>>p>>q;      for(i=0;i<m;i++)  {  tmp=a[i][p-1];  a[i][p-1]=a[i][q-1];  a[i][q-1]=tmp;  }    cout<<" Novata matrica e "<<endl;  for(i=0;i<m;i++)  {  cout<<endl;  for(j=0;j<n;j++)  cout<<a[i][j]<<" ";  }  return 0;  } |

Да се напише програма за наоѓање на најголем и најмал елемент на една општа матрица од реални броеви.

<https://www.loom.com/share/7399814dd1d043d992bfb2a28ab7a2ab>

|  |
| --- |
| //najgolem i najmal element na matrica od realni broevi  #include <iostream>  using namespace std;  int main()  {  float a[10][10],max,min,tmp;  int m,n,i,j,x,y,k,l,p,q;    cout<<"Broj na redici na matricite= ";  cin>>m;  cout<<"Broj na koloni na matricite= ";  cin>>n;  cout<<"Vnesete ja matricata "<<endl;  for(i=0;i<m;i++)  for(j=0;j<n;j++)  cin>>a[i][j];    max=a[0][0];    for(i=0;i<m;i++)  for(j=0;j<n;j++)  if (a[i][j]>max) max=a[i][j];    min=a[0][0];    for(x=0;x<m;x++)  for(y=0;y<n;y++)  if (a[x][y]<min) min=a[x][y];    cout<<"najgolemiot element od matricata e "<<max<<endl;  cout<<"najmaliot element od matricata e "<<min<<endl;    return 0;  } |

РАБОТА СО ДВЕ МАТРИЦИ

Да се напише програма за збир на елементите на две матрици.

<https://www.loom.com/share/29d6be47e125422884811622f89ab582>

|  |
| --- |
| //Zbir na elementite od dve matrici  #include<iostream>  using namespace std;  int main()  {  int m,n,i,j;  float a[10][20],b[10][20],c[10][20];  cout<<"Kolku redici i koloni imaat matricite?"<<endl;  cin>>m>>n;  cout<<"\n Vnesete gi elementite na prvata matrica "<<endl;  for(i=0;i<m;i++)  for(j=0;j<n;j++)  cin>>a[i][j];  cout<<"\n Vnesete gi elementite na vtorata matrica "<<endl;  for(i=0;i<m;i++)  for(j=0;j<n;j++)  cin>>b[i][j];  for(i=0;i<m;i++)  for(j=0;j<n;j++)  c[i][j]=a[i][j]+b[i][j];  cout<<"\n Zbirnata matrica e: "<<endl;  for(i=0;i<m;i++)  {  cout<<endl;  for(j=0;j<n;j++)  cout<<c[i][j]<<" ";  }  cout<<endl;  return 0;  } |

Да се напише програма за наоѓање на индексите на елементите кои се со еднакви вредности од две дадени матрици од реални броеви.

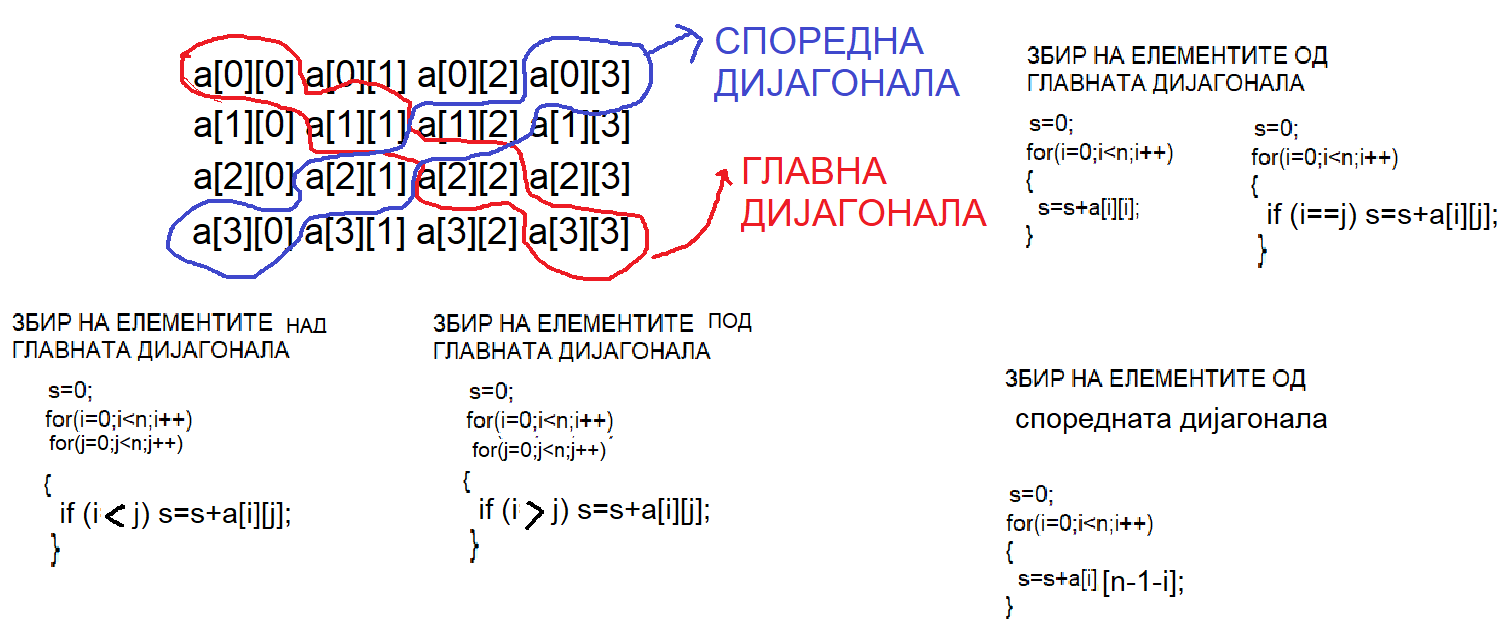
|  |
| --- |
| //Elementi so ednakvi vrednosti od dve matrici  #include<iostream>  using namespace std;  int main()  {  int m,n,i,j,p,q;  float a[10][20],b[10][20],c[10][20];  cout<<"Kolku redici i koloni imaat matricite?"<<endl;  cin>>m>>n;  cout<<"\n Vnesete gi elementite na prvata matrica "<<endl;  for(i=0;i<m;i++)  for(j=0;j<n;j++)  cin>>a[i][j];  cout<<"\n Vnesete gi elementite na vtorata matrica "<<endl;  for(i=0;i<m;i++)  for(j=0;j<n;j++)  cin>>b[i][j];    for(i=0;i<m;i++)  for(j=0;j<n;j++)  for(p=0;p<m;p++)  for(q=0;q<n;q++)  if (a[i][j]==b[p][q]) cout<<"a["<<i+1<<"]["<<j+1<<"]=b["<<p+1<<"]["<<q+1<<"]"<<endl;  cout<<endl;  return 0;  } |

Да се напише програма за производ на елементите на две матрици од реални броеви.

|  |
| --- |
| //Proizvod na elementite od dve matrici  #include<iostream>  using namespace std;  int main()  {  int m,n,i,j,p,k;  float a[20][20],b[20][20],c[20][20];  cout<<"Kolku redici i koloni ima prvata matrica?"<<endl;  cin>>m>>k;  cout<<"\n Vnesete gi elementite na prvata matrica "<<endl;  for(i=0;i<m;i++)  for(j=0;j<k;j++)  cin>>a[i][j];    cout<<"Brojot na redici na vtorata matrica e "<<k<<endl;  cout<<"Kolku koloni ima vtorata matrica?"<<endl;  cin>>n;  cout<<"\n Vnesete gi elementite na vtorata matrica "<<endl;  for(i=0;i<k;i++)  for(j=0;j<n;j++)  cin>>b[i][j];    for(i=0;i<m;i++)  for(j=0;j<n;j++)  {  c[i][j]=0;  for(p=0;p<k;p++)  c[i][j]+=a[i][p]\*b[p][j];  }  cout<<"\n Proizvodot na prvata i vtorata matrica e matricata: "<<endl;  for(i=0;i<m;i++)  {  cout<<endl;  for(j=0;j<n;j++)  cout<<c[i][j]<<" ";  }  cout<<endl;  return 0;  } |

**РАБОТА СО КВАДРАТНИ МАТРИЦИ**

**Квадратни матрици се матрици кај кои бројот на редиците е еднаков на бројот на колоните.**



Да се напише програма за замена на елементите од главната и споредната дијагонала на една квадратна матрица.

|  |
| --- |
| /\*zamena na elementite od glavnata i sporednata dijagonala na kvadratna matrica od broevi. \*/  #include <iostream>  using namespace std;  int main()  {  float a[20][10],tmp;  int n,i,j;  cout<<"Broj na koloni I redici na matricata= ";  cin>>n;  cout<<"Vnesete ja matricata "<<endl;  for(i=0;i<n;i++)  for(j=0;j<n;j++)  cin>>a[i][j];  for(i=0;i<n;i++)  {  tmp=a[i][i];  a[i][i]=a[i][n-i-1];  a[i][n-i-1]=tmp;  }  cout<<" Novata matrica e "<<endl;  for(i=0;i<n;i++)  {  cout<<endl;  for(j=0;j<n;j++)  cout<<a[i][j]<<" ";  }  return 0;  } |

Да се напише програма за наоѓање на збир на елементите над главната и збир на елементите под главната дијагонала на дадена квадратна матрица од реални броеви.

|  |
| --- |
| /\*zbir na elementite nad glavnata dijagonala i zbir na elementite pod glavnata dijagonala na kvadratna matrica od broevi. \*/  #include <iostream>  using namespace std;  int main()  {  float a[20][10];  int n,i,j;  cout<<"Broj na koloni I redici na matricata= ";  cin>>n;  cout<<"Vnesete ja matricata "<<endl;  for(i=0;i<n;i++)  for(j=0;j<n;j++)  cin>>a[i][j];    float s1=0;float s2=0;  for(i=0;i<n;i++)  for(j=0;j<n;j++)  if (i<j) s1=s1+a[i][j];  for(i=0;i<n;i++)  for(j=0;j<n;j++)  if (i>j) s2=s2+a[i][j];  cout<<" Zbirot nad glavnata dijagonala s1="<<s1<<endl;  cout<<" Zbirot pod glavnata dijagonala s2="<<s2<<endl;  return 0;  } |