O’ZBEKISTON RESPUBLIKASI AXBOROT TEXNOLOGIYALARI VA KOMMUNIKATSIYALARINI RIVOJLANTIRISH VAZIRLIGI

MUHAMMAD AL- XORAZMIY NOMIDAGI TOSHKENT AXBOROT TEXNOLOGIYALRI UNIVERSITETI SAMARQAND FILIALI

“Axborot texnologiyalari” kafedrasi

MA’LUMOTLAR TUZILMASI:

Saralash va qidiruv algoritmlari

Samarqand- 2018

Boynazarov I.M., To’xtayeva M.Sh. Ma’lumotlar tuzilmasi fanidan mustaqil ishlarni bajarish uchun uslubiy ko’rsatma. -Samarqand, TATU Samarqand filiali, 2018 y. 53 bet.

Taqrizchilar: TATU Samarqand filiali “Axborot texnologiyalari” kafedrasi mudiri, dotsent Z.M.Maxmudov

TATU Samarqand filiali “Dasturiy injiniring” kafedrasi katta o’qituvchisi R. Yusupov

Ushbu uslubiy ko’rsatma 5330500– Kompyuter injiniringi (Kompyuter injiniringi, AT-servisi), 5330600 – Dasturiy injiniring bakalavr ta’lim yo’nalishlari 1-kurs talabalari uchun mo’ljallangan bo’lib, o’quv rejasida ko’rsatilgan “Ma’lumotlar tuzilmasi” fanining “Saralash va qidiruv algoritmlari” bobi bo’yicha amaliy va laboratoriya mashg’ulotlarni o’tkazishga mo’ljallangan.

O’quv qo’llanma Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti Samarqand filialining uslubiy Kengashi tomonidan chop etishga tavsiya etilgan (2018-yil may).

Boynazarov I.M. – “Axborot texnologiyalari” kafedrasi dotsenti,

©To’xtayeva M.Sh. – Kompyuter injiniringi fakulteti 2-kurs talabasi.

MUNDARIJA

KIRISH

Ushbu uslubiy ko’rsatma 5330500– Kompyuter injiniringi (Kompyuter injiniringi, AT-servisi), 5330600 – Dasturiy injiniring bakalavr ta’lim yo’nalishlari 1-kurs talabalari uchun mo’ljallangan bo’lib, o’quv rejasida ko’rsatilgan “Ma’lumotlar tuzilmasi” fanining “Saralash va qidiruv algoritmlari” bobi bo’yicha nazariy bilimlar va amaliy mashg’ulotlarni o’tkazishga mo’ljallangan.

Kompyuterda ma’lumotlarni qayta ishlash uchun mo’ljallangan dasturlar tuzishda qo’yilgan masalaga mos ma’lumotlar tuzilmasini shakllantirib olish zarur bo’ladi. Ushbu ishda ma’lumotlar tuzilmasini shakllantirishning mantiqiy asoslari va dasturlash tillaridagi tadbiqi misollar yordamida berib o’tilgan.

Uslubiy ko’rsatmani o’rganib chiqish natijasida o’quvchi ma’lumotlarning tayanch va dinamik tuzilmalari ustida bajariladigan saralash va qidiruv amallarining bir necha usullarini amaliyotga tadbiq etish ko’nikmalariga ega bo’ladi. Shu bilan birgalikda egallangan bilimlar asosida amaliy masalalarni yechishda tuzilmalarni tadbiq etish ko’nikmalarini hosil qilishi mumkin.

1. Saralash algoritmlari: asosiy tushunchalar

Saralash (tartiblash)– bu berilgan ma’lumot elementlarining ba'zi bir xususiyatlariga ko'ra tartiblanishi (joylashtirilishi) hisoblanadi. Ko'p hollarda saralash mezoni (xusisiyati) sifatida aniq bir raqamli maydon qo’llaniladi va bu maydon kalit deb ataladi. Elementlarni kalit bo’yicha tartiblashda har bir keyingi elementning kaliti oldingisidan kichik bo’lsa kamayish tartibida, kalit maydon qiymati oldingisidan katta bo’lsa o’sish tartibida saralash deb ataladi.

Saralashdan asosiy maqsad - saralangan ma’lumotlarni qayta ishlash jarayonida zarur bo’ladigan elementni tez va oson qidirib topishni soddalashtirishdan iborat.

Mavjud saralash algoritmlarini ikki guruhga ajratish mumkin:

• ichki saralash algoritmlari (massivda saralash);

• tashqi saralash algoritmlari (faylda saralash).

Massivda saralash. Odatda massivlar ixtiyoriy jarayonlarni tez amalga oshirishni ta’minlovchi tezkor xotirada joylashadi. Massivlarni saralash algoritmlarining asosiy xususiyati tezkor xotirada ishlashni minimallashtirishdan iborat. Bunda elementlarni qayta joylashtirish jarayoni tezkor xotiraning o’zida bajarilishi shart.

Massivlarda saralash usullarini 3 ta sinfga ajratish mumkin:

• qo’yish orqali saralash;

• tanlash asosida saralash;

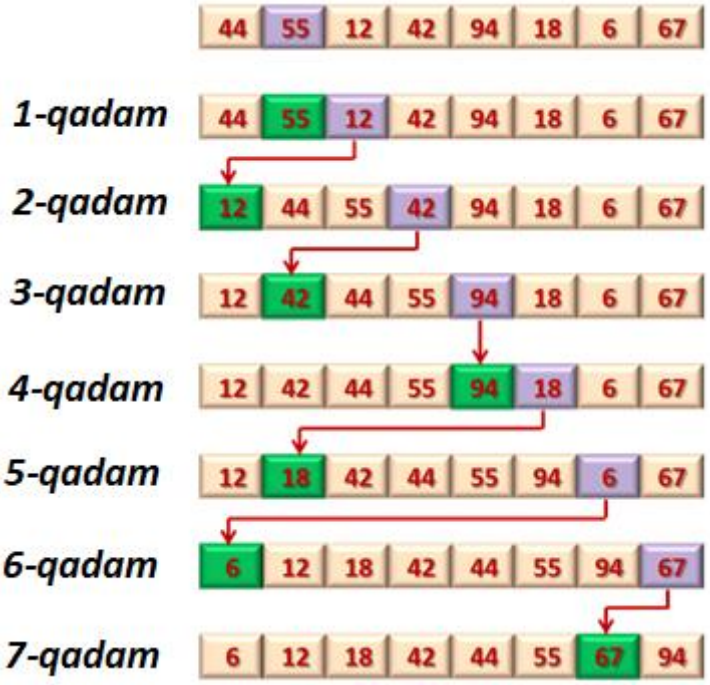
• almashtirish orqali saralash.

Faylda saralash. Fayllar sekin ishlovchi, lekin kattaroq hajmdagi tashqi xotirada saqlanadi. Agarda saralanadigan ma’lumotlar ketma-ket kirish mumkin bo’lgan tuzilmalarda saqlanayotgan bo’lsa, bunday tuzilmalarga massivda saralash algoritmlarini qo’llab bo’lmaydi. Chunki, ketma-ket kirishga ruxsat berilgan tuzilmalarda vaqtning har bir momentida faqat va faqat bitta komponentga murojaat qilish mumkin bo’ladi.

1.1. To’g’ridan-to’g’ri qo’yish orqali saralash algoritmi

Algorotmning asosiy g’oyasi: Massiv elementlari shartli ravishda oldindan tayyorlangan ketma-ketlik a_1, a_2, ..., a_{i-1} va kiruvchi ketma-ketlik a_i, a_{i+1}, ..., a_n kabi qismlarga ajratib olinadi.

Oldindan tayyor ketma-ketlikda har bir i-element qulay joyga joylashtiriladi.



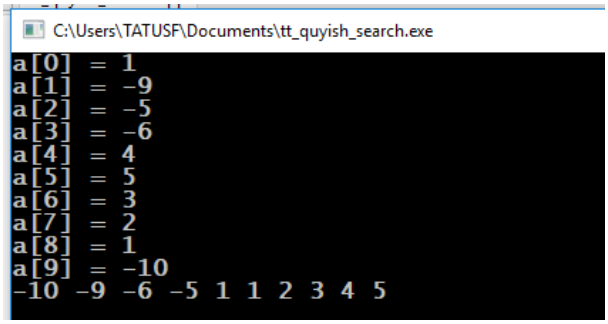
Ushbu algoritmning ishlashiga C++ dasturlash tilida misol. O’nta elementdan iborat butun sonli massiv berilgan. Massiv elementlarini o’sish tartibida saralang.

Yechimi:

```
#include <stdio.h>
void ttSort(int *num, int size) // saralash funksiyasi
{
  for (int i = 1; i < size; i++)
  {
    int value = num[i]; // element qiymati va uning indeksini saqlash
    int index = i;
    while ((index > 0) && (num[index - 1] > value))
    {
      num[index] = num[index - 1];
```

```
      index--;
    }
    num[index] = value;
  }
}
int main()
{
  int a[10];
  for (int i = 0; i < 10; i++)
  {
    printf("a[%d] = ", i);
    scanf("%d", &a[i]);
  }
  ttSort(a, 10);
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  getchar(); getchar();
  return 0;
}
```

Natija:



Algoritmning ishlash samaradorligi tahlili

C_i kalitlarni taqqoslashlar soni i-qadamda eng ko'p (i-1) marta, eng kamida 1 marta amalga oshiriladi. Agar n ta kalitning almashishi bir xil ehtimolli bo'lsa, u holda taqqoslashlar soni $\frac{n}{2}$ ga teng bo'ladi. Sijitishlar soni $M_i = C_i + 2$.

Shuning uchun taqqoslashlar va siljitishlar soni mos ravishda quyidagicha bo'ladi:

$C_{min} = n - 1; M_{min} = 3(n - 1);$

$C_{o'rt} = \frac{n^2+n-2}{4}; M_{o'rt} = \frac{n^2+9n-10}{4};$

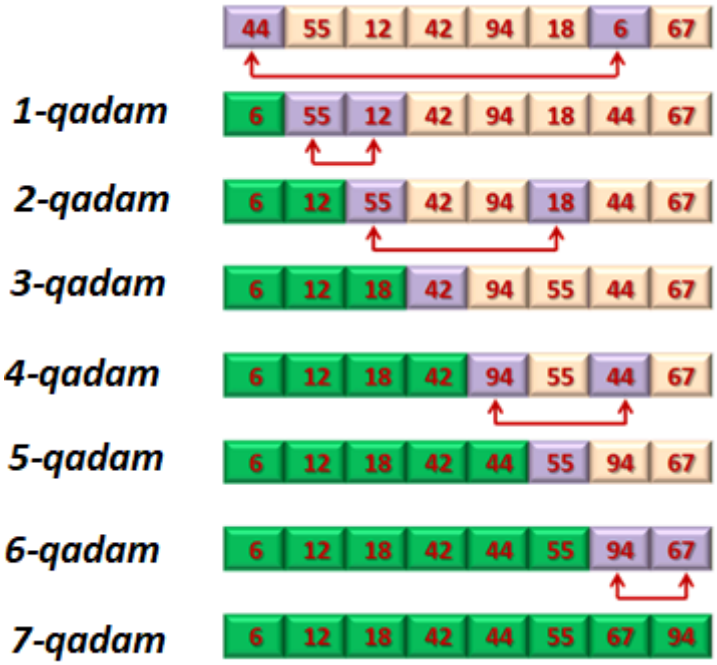
$C_{max} = \frac{n^2+n-4}{4}; M_{max} = \frac{n^2+3n-4}{2};$

Eng yaxshi holat dastlabki elementlarning tartiblangan holati. Eng yomon holat esa ularning teskari tartiblangan holati.

Xulosa: shunday qilib, to'g'ridan-to'g'ri qo'yish orqali saralash usuli kompyuter uchun unchalik ham ma'qul emas, chunki bir nechta elementlar guruhini birdaniga surish samarali bo'lmaydi.

1.2. To'g'ridan-to'g'ri tanlash usuli

To'g'ridan-to'g'ri tanlash usuli qandaydir ma'noda to'g'ridan-to'g'ri qo'yish usuliga ziddir. Bu yerda suriladigan elementlar faqat bitta bo'ladi va har bir surishdan keyin elementlarni taqqoslashlar soni bittaga kamayadi. Bu jarayon elementlar tugaguncha davom etadi.



Bizga ***n*** ta element berilgan bo'lsin. Biz shu elementlar ichidan eng kichigini topamiz va bu elementni massiv boshidagi element bilan almashtiramiz. Endi esa ikkinchi elementdan boshlab, ***n***-elementgacha bo'lgan ***n***-1 ta elementdan eng kichigini topamiz. Topilgan minimumni qaralayotgan elementlarning boshiga ya'ni ikkinchi element o'rniga

qo’yamiz. Ushbu jarayon massiv elementlari tugaguncha davom ettiriladi va natijada massiv elementlari o’sish tartibida saralangan bo’ladi.

To’g’ridan-to’g’ri tanlash orqali saralash usulining C++ dasturlash tilidagi algoritmi uchun misol.

```
#define _CRT_SECURE_NO_WARNINGS // scanf() ishlashi uchun
#include <stdio.h>
// to’g’ridan-to’g’ri tanlash bilan saralash uchun funksiya
void selectionSort(int *num, int size)
{
  int min, temp; // minimal elementni qidirish va almashtirish uchun
  for (int i = 0; i < size - 1; i++)
  {
    min = i; // joriy element indeksini saqlaymiz
    // i-o’ringa joylashtirish uchun minimum elementni topamiz
    for (int j = i + 1; j < size; j++)  // i-elementdan keyingilari uchun
    {
      if (num[j] < num[min]) // minimumdan kichigi bo’lsa,
        min = j;       // uning indeksini min da saqlaymiz
    }
    temp = num[i];      // minimum va i-elementlarni almashtiramiz
    num[i] = num[min];
    num[min] = temp;
  }
}
int main()
{
  int a[10]; // 10ta elementdan iborat massiv elementni e’lon qilamiz
  // massiv elementlari qiymatlarini kiritamiz
  for (int i = 0; i < 10; i++)
  {
    printf("a[%d] = ", i);
    scanf("%d", &a[i]);
  }
  selectionSort(a, 10);  // saralash funksiyasini chaqiramiz
```

```
  // massivning saralangan elementlarini chiqaramiz
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  getchar(); getchar();
  return 0;
}
```

Natija :

To'g'ridan-to'g'ri tanlash algoritmining tahlili

C kalitlarni taqqoslashlar soni:

$$C = \frac{n^2 - n}{2}$$

Minimal almashtirishlar soni:

$$M_{min} = 3(n-1)$$

Elementlar tartiblangan bo'lsa va teskari tartibda bo'lsa:

$$M_{max} = \frac{n^2}{4} + 3(n-1)$$

O'rtacha almashtirishlar soni:

$$M_{o'rt} \approx n(\ln n + g)$$

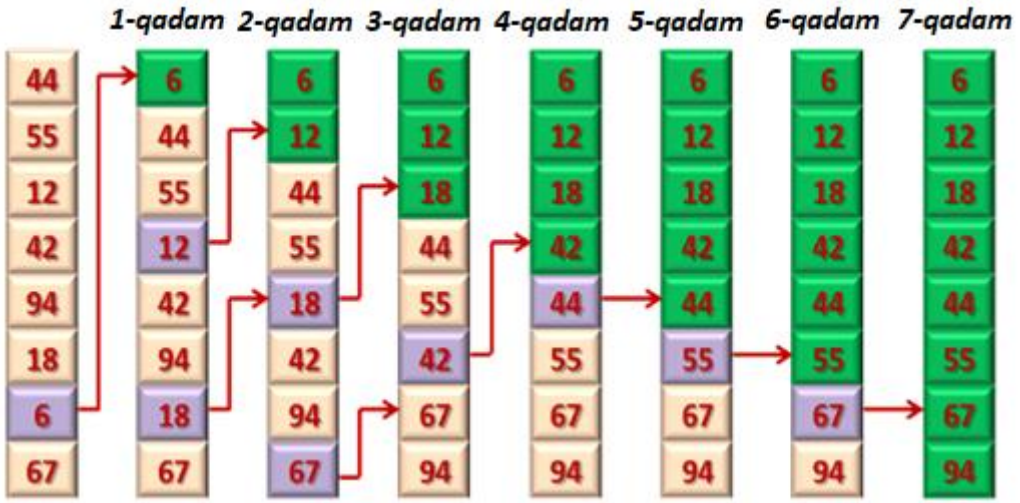
$g = 0.577216 \ldots$ - Eyler o'zgarmasi.

Xulosa: To'g'ridan-to'g'ri tanlash usuli to'g'ridan-to'g'ri qo'yish usulidan ustunroq. Lekin agar kalitlar dastlab saralangan bo'lsa, yoki qisman saralangan bo'lsa to'g'ridan-to'g'ri qo'yish usuli sal tezroq bo'ladi.

1.3. To'g'ridan-to'g'ri almashtirish usuli («Pufakcha» usuli)

To'g'ridan-to'g'ri almashtirish yoki pufakcha usuli – elementlar saralangunga qadar yonma-yon elementlarni saralashlar va almashtirishlar

jarayoni. Bu usulda xuddi to’g’ridan-to’g’ri saralash usuli kabi eng kichik element massiv boshiga ko’chiriladi. Agar massivni vertikal ko’rinishda deb tasavvur qilsak, minimal elementlarning tepaga ko’chishi xuddi suvdagi pufakchalarning tepaga ko’tarilishiga o’xshaydi. Shuning uchun ham bu usulni “pufakchali usul” deyish mumkin.



“Pufakcha” usulida saralashning C++ dastrulash tilidagi algoritmiga misol

```
#define _CRT_SECURE_NO_WARNINGS // scanf()to’g’ri ishlashi uchun
#include <stdio.h>
// To’g’ridan-to’g’ri almashtirish uchun funksiya ("pufakcha" usuli)
void bubbleSort(int *num, int size)
{
  // barcha elementlar uchun
  for (int i = 0; i < size - 1; i++)
  {
    for (int j = (size - 1); j > i; j--) // i-elementdan keyingi barcha elementlar uchun
    {
      if (num[j - 1] > num[j]) // joriy element oldingisidan kichik bo’lsa
      {
        int temp = num[j - 1]; // ularning joyini almashtiramiz
        num[j - 1] = num[j];
```

```
        num[j] = temp;
      }
    }
  }
}
int main()
{
  int a[10]; // 10ta elementdan iborat massiv e’lon qilamiz
  // massiv elementlarining qiymatlarini kiritamiz
  for (int i = 0; i < 10; i++)
  {
    printf("a[%d] = ", i);
    scanf("%d", &a[i]);
  }
  bubbleSort(a, 10);  // saralash funksiyasini chaqiramiz
  // saralangan massiv elementlarini chiqaramiz
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  getchar(); getchar();
  return 0;
}
```

Natija:

Algoritm tahlili

Taqqoslashlar soni:

$$C = \frac{n^2 - n}{2}$$

O’rin almashtirishlar soni (mos ravishda):

$$M_{min} = 0$$

$$M_{o'rt} = \frac{3(n^2 - n)}{2}$$

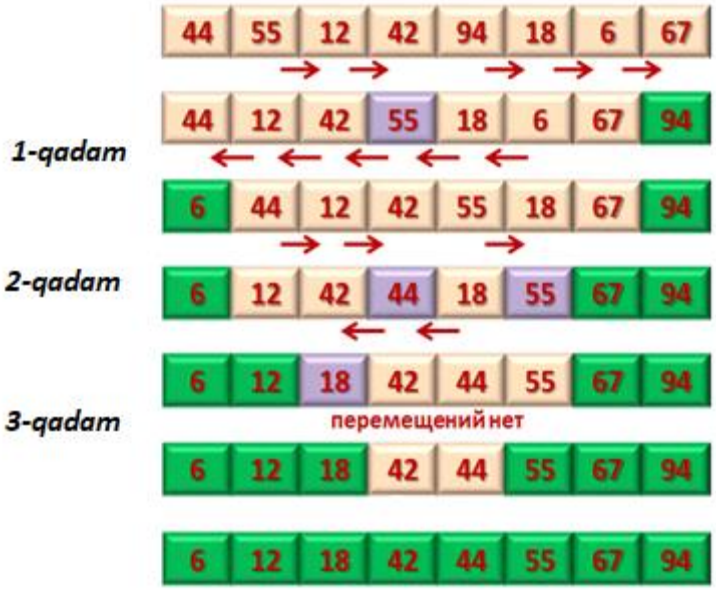
$$M_{max} = \frac{3(n^2 - n)}{4}$$

Xulosa: «Almashtirishli saralash» samaradorligi bo’yicha to’g’ridan-to’g’ri tanlash va to’g’ridan-to’g’ri qo’yish usullari o’rtasida desa ham bo’ladi.

1.4. Sheyker saralash algoritmi

Sheyker saralash usuli pufaksimon saralashning mukammallashgan usulidir. Pufakchali saralashda eng maksimal element massiv oxiriga boradi. Elementlarni bir marta to’liq ko’rib chiqganda elementlarning oxiridagisi saralangan bo’ladi. Shuning uchun massivni bir marta qarab chiqqanimizdan keyin uni to’liq tekshirmasdan n-1 elementigacha ko’rib chiqish kifoya. Ushbu jarayon elementlar tugaguncha davom etadi.

Sheyker – saralash algoritmini misolda ko’rib chiqamiz. Ketma-ketlik berilgan.



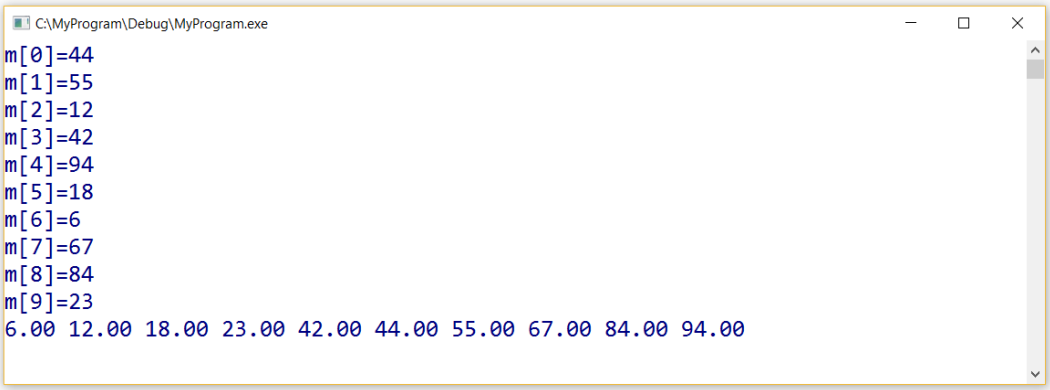
Sheyker-saralash algoritmining dasturi

Har bir while() siklining takrorlanishi saralash qadamini bildiradi.

```
#define _CRT_SECURE_NO_WARNINGS // scanf() to’g’ri ishlashi uchun
#include <stdio.h>
// sheyker-saralash funksiyasi
void shekerSort(double *mass, int count)
{
  int left = 0, right = count - 1; // saralanayotgan massivning chap va o’ng chegaralari
  int flag = 1;  // harakatni bildiruvchi bayroq
  // sikl toki chap chegara o’ng chegara bilan kesishguncha
  // yoki massivda harakatlanish mavjud bo’lsa
  while ((left < right) && flag > 0)
  {
    flag = 0;
    for (int i = left; i<right; i++)  //chapdan o’ngga yuramiz
    {
      if (mass[i]>mass[i + 1]) // agar keying element joriy elementdan kichik bo’lsa,
      {            // ularning joyini almashtiramiz
        double t = mass[i];
        mass[i] = mass[i + 1];
        mass[i + 1] = t;
        flag = 1;      // bu siklda harakatlanish bo’ldi
      }
    }
    right--; // o’ng chegarani oldingi elementga suramiz
    for (int i = right; i>left; i--)  //o’ngdan chapga yuramiz
    {
      if (mass[i - 1]>mass[i]) // joriy element oldingisidan kata bo’lsa,
      {            // ularning joyini almashtiramiz
        double t = mass[i];
        mass[i] = mass[i - 1];
        mass[i - 1] = t;
        flag = 1;    // bu siklda harakatlanish bo’lgan
```

```
    }
   }
   left++; // chap chegarani keying elementga suramiz
  }
}
int main() {
  double m[10];
  // massiv elementlarini kiritamiz
  for (int i = 0; i<10; i++) {
    printf("m[%d]=", i);
    scanf("%lf", &m[i]);
  }
  shekerSort(m, 10); // saralash funksiyasini chaqiramiz
  // massivning saralangan elementlarini chiqaramiz
  for (int i = 0; i<10; i++)
    printf("%.2lf ", m[i]);
  getchar(); getchar();
  return 0;
}
```

Natija:



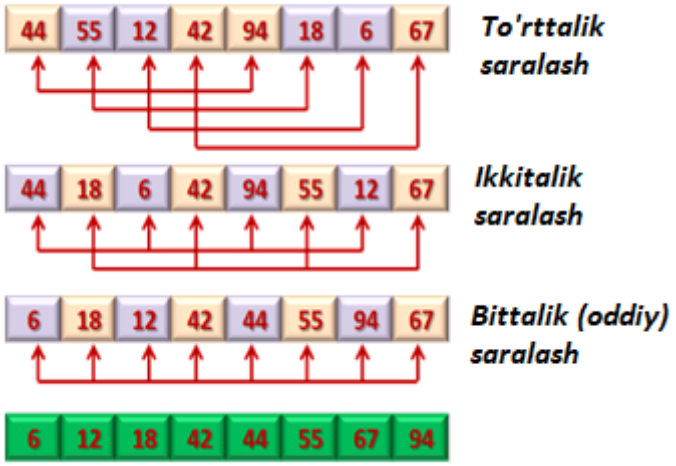
1.5. Shell saralash algoritmi

1959-yilda D.Shell tomonidan to’g’ridan-to’g’ri qo’yish usuli yordamida saralashning mukammallashtirilgan usulini taklif qildi.

Dastlab saralanayotgan har 4 ta pozitsiyadagi elementlar alohida guruhlanadi va saralanadi. Bu jarayon to’rttalik saralash deb nomlanadi.

Elementlar bir marta to’liq ko’rib chiqilgandan keyin ular yana qayta guruhlanadi- ya’ni saralanayotgan har 2 ta pozitsiyadagi elementlar alohida guruhlanadi va saralanadi (ikkitalik saralash).

Uchinchi to’liq ko’rib chiqilishda oddiy saralash jarayoni bo’ladi.



Har bir to’liq qarab chiqishda barcha elementlarni ishga tushirish mashinadan katta resurs talab qiladigandek ko’rinadi, lekin har bir to’liq qarab chiqishda kam elementlar saralanadi yoki yaxshi saralangan bo’lsa, ularni shunchaki taqqoslab chiqish yetarli bo’ladi.

Bu usul natijasida massiv saralangan bo’ladi va har bir to’liq qarab chiqish avvalgisiga nisbatan yaxshiroq bo’ladi. (chunki har bir i-saralash 2i-saralashda saralangan 2 ta guruhni birlashtiradi).

Shell saralash usulining C dagi algoritmlari

```
#define _CRT_SECURE_NO_WARNINGS // scanf() to’g’ri ishlashi uchun
#include <stdio.h>
// Shell saralash funksiyasi
void shellSort(int *num, int size)
{
  int increment = 3;    // saralashning dastlabki orttirishi
  while (increment > 0)  // toki inkrement mavjud
  {
    for (int i = 0; i < size; i++)  // massivning barcha elementlari uchun
    {
```

```
      int j = i;          // indeksi va elementini saqlaymiz
      int temp = num[i];
      // j-elementdan inkrement kattaligi bo'yicha orqada qolgan barcha
elementlarni ko'rib chiqamiz
      while ((j >= increment) && (num[j - increment] > temp))
      {  // toki ortda qolayotgan element joriy elementdan katta
        num[j] = num[j - increment]; // uni joriy pozitsiyaga ko'chiramiz
        j = j - increment;       // keyingi ortda qolayotgan elementga o'tamiz
      }
      num[j] = temp; // saqlangan elementni aniqlangan joyga qo'yamiz
   }
    if (increment > 1)      // inkrementni 2 ga bo'lamiz
      increment = increment / 2;
    else if (increment == 1)   // oxirgi ko'rib chiqish tugadi,
      break;  // sikldan chiqamiz
  }
}
int main()
{
  int m[10];
  // massiv elementlarini kiritamiz
  for (int i = 0; i<10; i++)
  {
    printf("m[%d]=", i);
    scanf("%d", &m[i]);
  }
  shellSort(m, 10); // saralash funksiyasini chaqiramiz
  // saralangan massiv elementlarini chiqaramiz
  for (int i = 0; i<10; i++)
    printf("%.2d ", m[i]);
  getchar(); getchar();
  return 0;
}
```

Natija:

Algoritm tahlili

Bu dastur qandaydir masofalar ketma-ketligiga mo'ljallanmagan. Barcha masofalar quyidagicha yoziladi:

h1, h2, ..., ht ,

ular uchun quyidagi shart bajariladi:

ht=1;

hi+1<hi.

Har bir h-saralash xuddi to'g'ridan-to'g'ri qo'yish usulidagi saralash yordamidagidek dasturlanadi. Bu algoritmni qanaqa masofalar eng zo'r natija berishi noma'lum.

Donal'd Knut quyidagi ketma-ketlikni taklif qilgan:

1, 3, 7, 15, 31, ...,

Ya'ni

hk-1=2hk+1,

ht=1 va t = [log2n] - 1.

1.6. Daraxt yordamida saralash

Daraxt yordamida saralash asosini binar qidiruv daraxti tashkil etadi.

Binar (ikkilik) qidiruv daraxti – quyidagi qo'shimcha shartlar bajariladigan binary daraxt hisoblanadi. (qidiruv daraxti xususiyatlari):

Ikkala shoxi ham – chap va o'ng ikkilik qidiruv daraxti hisoblanadi.

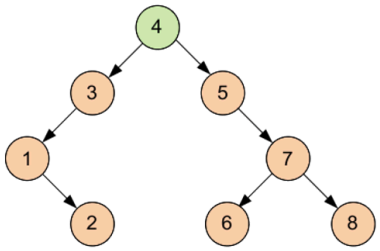
Istalgan chap shox kaliti o'zi chiqqan daraxtning kalitidan kichik.

Istalgan o'ng shox kaliti o'zi chiqqan daraxtning kalitidan kichik emas.

Daraxt yordamida saralash uchun berilgan ketma-ketlik daraxt ko'rinishidagi ma'lumotlar strukturasi bo'lishi kerak. Masalan, berilgan ketma-ketlik quyidagi ko'rinishga ega:

4, 3, 5, 1, 7, 8, 6, 2

Daraxrt ildizi ketma-ketlikning boshlang’ich elementi hisoblanadi. Qolgan elementlar ildizdan kichik bo’lsa chap shoxga joylashtiriladi, kattalari esa o’ng shoxga joylashtiriladi. Holbuki, bu shart barcha shoxlarda bajarilishi shart. Shundan keyin daraxt tuzilishiga joylashtirilgan barcha elementlarni infiks ko’rinishli aylanib chiqish yordamida chiqaramiz.



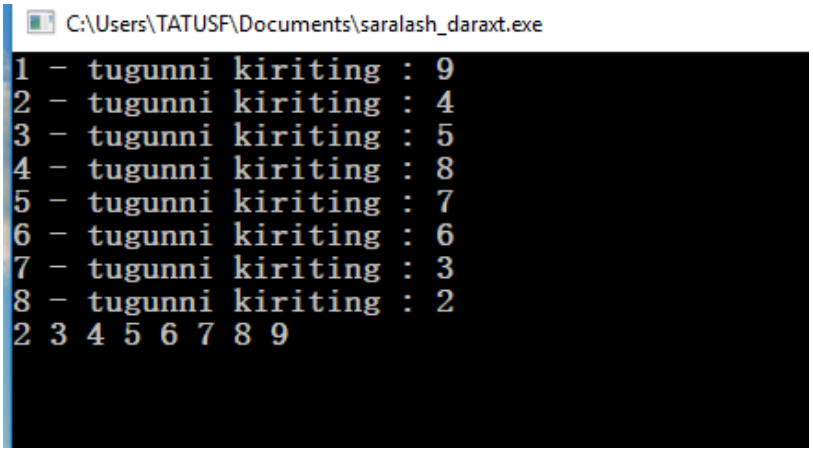
Daraxt yordamida saralashning C++ dagi dasturi

```
#include <iostream>
using namespace std;
// Tuzlima  - daraxt tarmog'i
struct tnode
{
  int field;          // ma'lumotlar maydoni
  struct tnode *left;  // chap avlod
  struct tnode *right; // o'ng avlod
};
// daraxt tarmoqlarini chiqarish (infiks ko'rinishli aylanib chiqish )
void treeprint(tnode *tree)
{
  if (tree != NULL) {      //toki bo'sh tarmoq uchramaguncha
    treeprint(tree->left);  //chap shoxning rekursiv chiqarish funksiyasi
    cout << tree->field << " "; //daraxt ildizini chiqaramiz
    treeprint(tree->right); // o'ng shoxning rekursiv chiqarish funksiyasi
  }
}
// daraxtga tarmoqlar qo'shish
struct tnode * addnode(int x, tnode *tree)
```

```
{
 if (tree == NULL)          //agar daraxt mavjud bo'lmasa, ildizni shakllantiramiz
{
   tree = new tnode; // tarmoq osti xotira
   tree->field = x;   //ma'lumotlar maydoni
   tree->left = NULL;
   tree->right = NULL; //shoxlarni bo'sh yaratamiz
 }
 else     // aks holda
   if (x < tree->field)   //agar x element ildizdan kichik bo'lsa chapga ketamiz
  tree->left = addnode(x, tree->left); //elementni rekursiv qo'shamiz
   else  //aks holda o'ngga ketamiz
     tree->right = addnode(x, tree->right); // elementni rekursiv qo'shamiz
   return(tree);
}
//daraxt xotirasini bo'shatish
void freemem(tnode *tree)
{
 if (tree != NULL)    // agar daraxt bo'sh bo'lmasa
 {
   freemem(tree->left);   // chap shoxni rekursiv o'chiramiz
   freemem(tree->right);  // o'ng shoxni rekursiv o'chiramiz
   delete tree;           // ildizni o'chiramiz
 }
}
// Ishni testlash
int main()
{
 struct tnode *root = 0;    // Daraxt tuzilishini e'lon qilamiz
 system("chcp 1251");    // konsolda rus tiliga o'tkazamiz
 system("cls");
```

```
    int a;            // joriy tarmoq qiymati
    // Siklga daraxtning 8 tarmog'ini kiritamiz
    for (int i = 0; i< 8; i++)
    {
      cout << i + 1 << " - tugunni kiriting "<< ": ";
      cin >> a;
      root = addnode(a, root); // daraxtga kiritilgan tarmoqni joylashtiramiz
    }
    treeprint(root);        // daraxt elementlarini chiqaramiz, saralangan massivga ega bo'lamiz
    freemem(root);      // ajratilgan xotirani o'chiramiz
    cin.get();  cin.get();
    return 0;
}
```

Natija

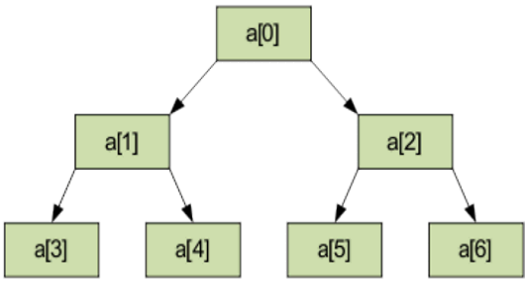


1.7. Piramidali saralash

D. Villyams tomonidan yaratilgan piramidali saralash usuli daraxt yordamida saralashning yaxshilangan variantidir. Piramidali (top’lam) bu ikki tomonlama daraxtdir

$a[i] \le a[2i+1]$;

$a[i] \le a[2i+2]$.



a[0] — piramidaning minimal elementi.

Piramidal tartiblashning asl g'oyasi, umumiy arifmetik elementlardan olingan piramidaning oldindan yasalishi va elementlarning tartiblashidir. Algoritmning bajarilishi 2 etapga bo'linadi.

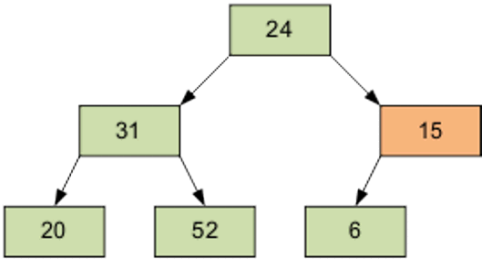
1 –bosqich. Piramidani qurish. n/2-1 dan boshlab, daraxting o'ng qismini aniqlab olamiz (daraxtning pastki darajasi). Massivning bu qismidan chaproqda turgan elementlarini olamiz va uni piramida bo'ylab joylashtiramiz, undan kichik elementlar kelib qolsa, ularning kichigi bo'ylab harakatlanamiz.

Masalan saralash uchun massiv

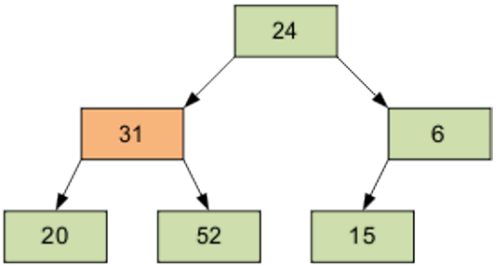
24, 31, 15, 20, 52, 6

Elementlarni dastlabki piramida ko'rinishida joylashtiramiz;

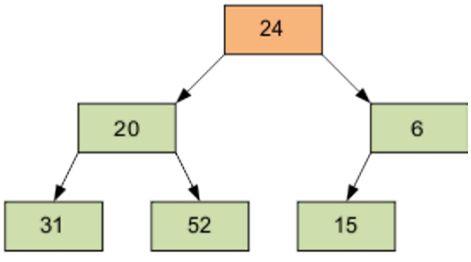
o'ng qism elementining nomeri (6/2-1)=2 – bu element 15.



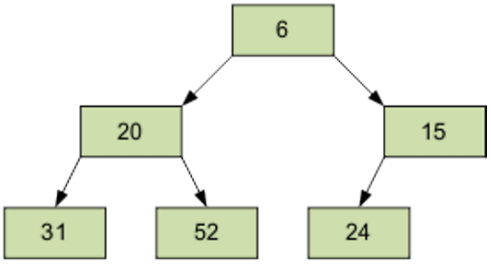
15 elementini piramida bo'ylab joylashtirish natijasi:



Keyingi joylashtirilayotgan element – 1, 31 ga teng

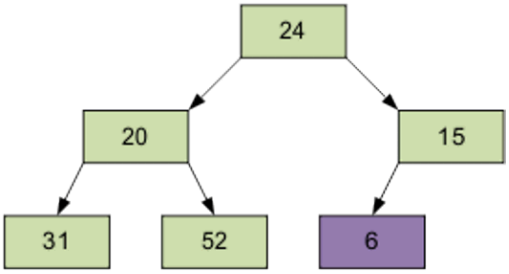


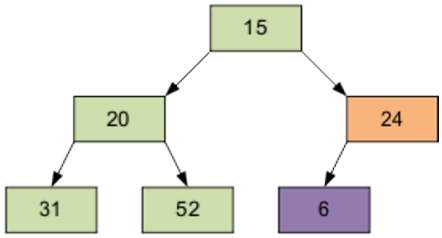
keyingi element –0, 24 ga teng.



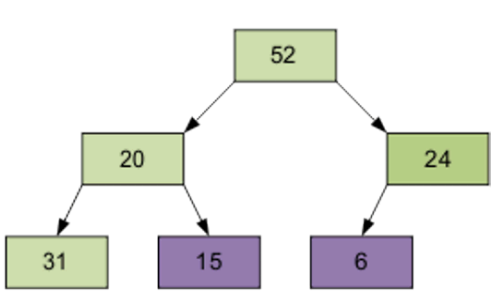
Albatta olingan massiv hali tartiblanmagan. Lekin joylashtirish jarayoni piramidali saralash asosi hisoblanadi. Joylashtirish natijasida eng kichik element piramida uchida bo’lib qoladi.

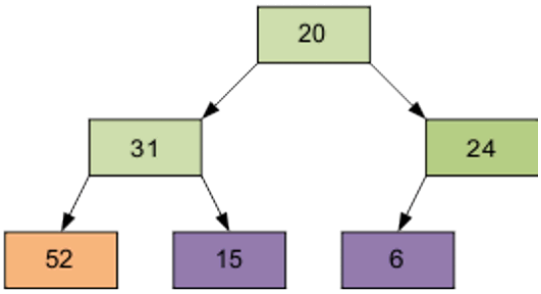
2– bosqich qurilgan piramidada saralash. Massivning eng oxirgi elementini joriy qilamiz. Joriy elementni uchidagi element(eng kichik) bilan joylarini almashtiramiz. Joriy elementni (uchidagi) n-1 elementli piramida bo’ylab joylashtiramiz. Keyin oxirgisidan bitta oldingi elementni tanlaymiz va h.k.

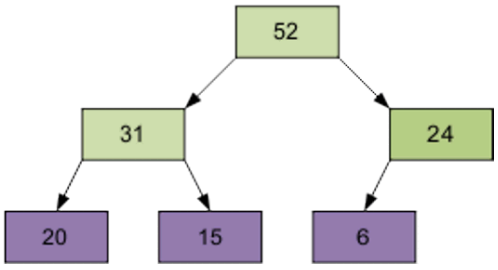


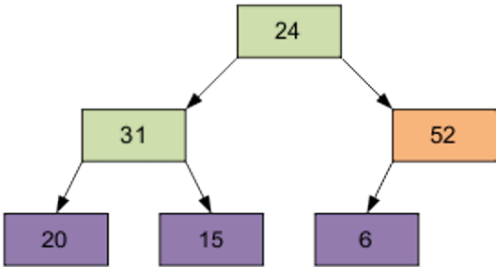


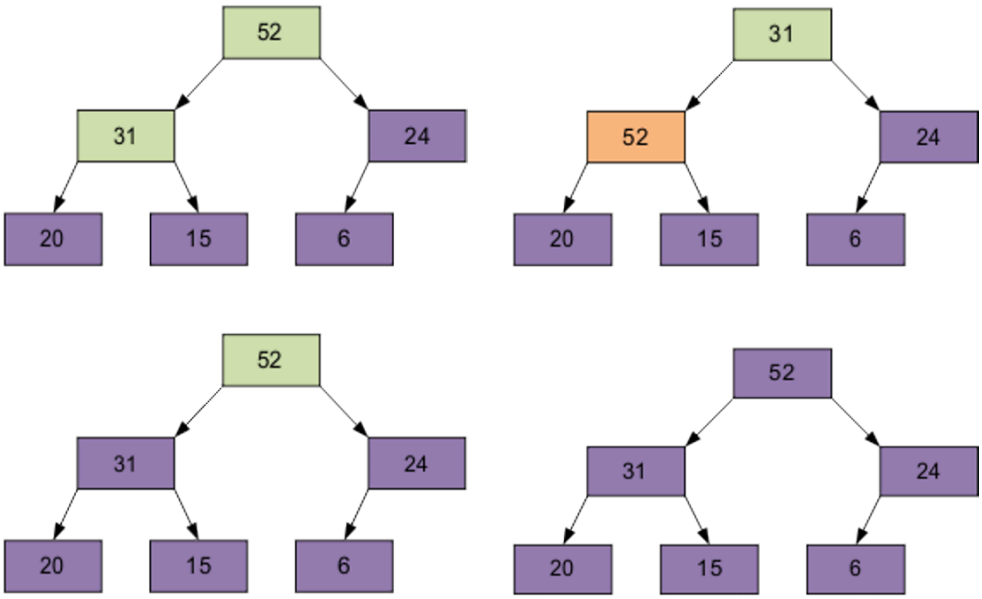
jarayonni davom ettiramiz. Natijada massiv kamayish tartibida saralangan bo’ladi.











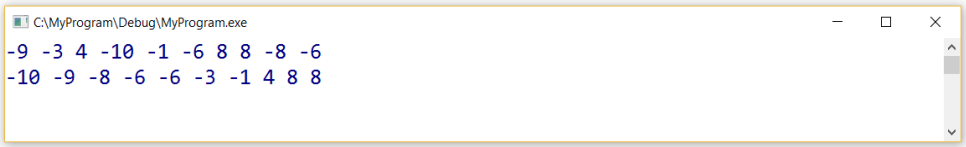
Piramidali saralashning C++ dagi dasturi

```
#include <stdio.h>
#include <stdlib.h>
// To’plamni shakllantirish – to’plam bo’yicha “joylashtirish”
void siftDown(int *numbers, int root, int bottom)
{
  int maxChild; // maksimal avlod indeksi
  int done = 0; // to’plam shakllanganligi bayrog’i
  // Toki oxirgi qatorga bormaganimizcha
  while ((root * 2 <= bottom) && (!done))
  {
    if (root * 2 == bottom)    // agar oxirgi qatorda bo’lsak,
      maxChild = root * 2;    // chap avlodni eslab qolamiz
    // aks holda, ulardan kattasini eslab qolamiz
    else if (numbers[root * 2] > numbers[root * 2 + 1])
      maxChild = root * 2;
    else
      maxChild = root * 2 + 1;
    // agar uchidagi element maksimal avloddan kichik bo’lsa
    if (numbers[root] < numbers[maxChild])
    {
      int temp = numbers[root]; // ularning joylarini almashtiramiz
      numbers[root] = numbers[maxChild];
```

```
      numbers[maxChild] = temp;
      root = maxChild;
    }
    else // aks holda
      done = 1; // piramida hosil bo'ladi
  }
}
// to'plamda saralash funksiyasi
void heapSort(int *numbers, int array_size)
{
  // piramidaning pastki qatorini shakllantiramiz
  for (int i = (array_size / 2) - 1; i >= 0; i--)
    siftDown(numbers, i, array_size);
  // qolgan elementlarni piramida bo'ylab joylashtiramiz
  for (int i = array_size - 1; i >= 1; i--)
  {
    int temp = numbers[0];
    numbers[0] = numbers[i];
    numbers[i] = temp;
    siftDown(numbers, 0, i - 1);
  }
}
int main()
{
  int a[10];
  // Massivni tasodifiy sonlar bilan to'ldiramiz
  for (int i = 0; i<10; i++)
    a[i] = rand() % 20 - 10;
  // massivning saralashgacha bo'lgan elementlarini chiqaramiz
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  printf("\n");
  heapSort(a, 10); // saralash funksiyasini chaqirish
  // Saralashdan keying massiv elementlarini chiqarish
```

```
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  printf("\n");
  getchar();
  return 0;
}
```

Natija:



Piramidali saralash algoritmining tahlili

Ba'zi tashqi murakkabliklarga qaramay, piramidali saralash eng samarali hisoblanadi. Saralash algoritmi katta n lar uchun samarali. Eng yomon holatda o'zgaruvchan elementlar qadamlari soni $n \log_2 n$. Almashtirishlarning o'rtacha soni taxminan quyidagiga teng: $\frac{n}{2} \log_2 n$

1.8. Tez saralash

Tez saralash - almashinish prinsipiga asoslangan mukammallashgan saralashning bir usuli. To'g'ridan-to'g'ri saralashlar ichida eng samarasizi pufakchali saralash hisoblanadi. Lekin mukammallashgani barcha ma'lum saralash algoritmlari ichida eng yaxshisi hisoblanadi. U shunday yorqin xususiyatlarga egaki, uning ixtirochisi Ch. Xoar uni tezkor saralash deb hisoblagan.

Katta samaraga erishish uchun katta masofadagi elementlarni almashtirish kerak. Massivda ruxsat beruvchi element tanalanadi. Undan keyin u shunday joyga joylashtiriladiki, saralangandan keyin o'z joyida turgan hisoblanadi. Bu jarayonda ruxsat beradigan element shunday joylashadiki, undan chapda joylashgan element undan kichik, undan o'ngda joylashgan element esa undan kata hisoblanadi. (massiv o'sish tartibida saralanadi) shunday qilib massiv 2 qismga bo'linadi:

Ruxsat beruvchi elementdan chapda joylashgan saralanmagan elementlar;

Ruxsat beruvchi elementdan o’ngda joylashgan saralanmagan elementlar;

Bu ikkita kichik massivlarni saralash uchun funksiya o’ziga rekursiv murojaat qiladi.

Agar bitta elementdan ko’proq elementni saralash kerak bo’lsa massivdan ruxsat beruvchi element tanlaymiz;

Elementni o’zining natijaviy joyiga joylashtirib, massivni qayta tartiblaymiz;

Ruxsat beruvchi elementdan chap tomondagi massivni rekursiv saralaymiz;

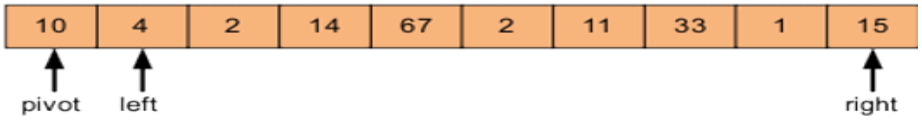
Ruxsat beruvchi elementdan o’ng tomondagi massivni rekursiv saralaymiz;

Tezkor saralashning asosiy elementi qayta tartiblash algoritmi hisoblanadi. Saralashni massiv misolida ko’rib chiqamiz:

10, 4, 2, 14, 67, 2, 11, 33, 1, 15.

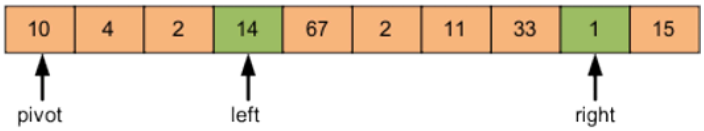
Qayta tartiblash algoritmini amalga oshirish uchun massivning chap elementi ya’ni left ko’rsatgichi olinadi. Ko’rsatgich o’ngga harakatlanadi toki, ruxsat beradigan elementdan kichik. Qayta tartiblash algoritmini amalga oshirish uchun massivning o’ng elementi ya’ni left ko’rsatgichi olinadi. Ko’rsatgich chapga harakatlanadi toki, ruxsat beradigan elementdan katta.

Massivning chapdagi elementi ruxsat beruvchi pivot bo’lsin. left ko’rsatgichini undan keying elementga o’rnatamiz; right — eng oxirgi. Algoritm 10 ning to’g’ri joyini aniqlab va ish jarayonida noto’g’ri joylashgan elementlarni joylarini almashtiradi.

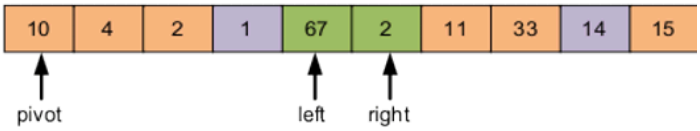


agar joylashishi ruxsat beruvci elementga nisbatan noto’g’ri bo’lsa, ko’rsatgichning harakati to’xtaydi.

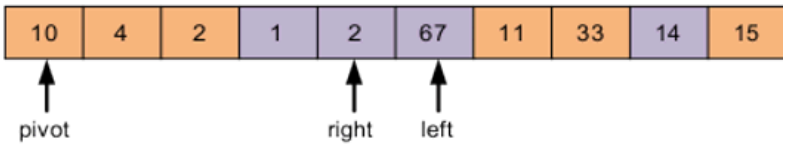
left ko’rsatgichi 10 dan kata element uchramaguncha harakatlanaveradi; right ko’rsatgichi 10 dan kichik element uchramaguncha harakatni davom ettiradi.



bu elementlar joylari bilan almashishadi va ko’rsatgichlar harakati davom etadi.

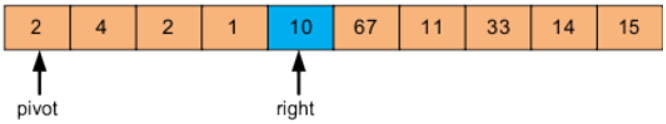


jarayon right left dan chapda bo’lib qolmaguncha davom etadi.



ruxsat beradigan elementning to’g’ri joyi topiladi.

ruxsat beradigan element bilan right ko’rsatayotgan elementlar bilan joy almashadi.



Ruxsat beradigan element kerakli joyda joylashgan: undan chapda joylashgan elementlar kichik qiymatga ega, o’ngdagilar- katta. Algoritm ruxsat beradiga elementning chap va o’ng tomonidagi massivlar uchun rekursiv chaqiriladi.

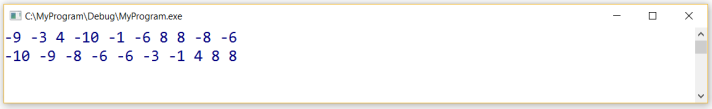
Tez saralash algoritmining S dagi dasturi

```
#include <stdio.h>
#include <stdlib.h>
// Tez saralash funksiyasi
void quickSort(int *numbers, int left, int right)
{
  int pivot; // ruxsat beradigan element
  int l_hold = left; //chap chegara
  int r_hold = right; // o’ng chegara
  pivot = numbers[left];
  while (left < right) // chegaralar yopiq bo’lmaguncha
{
    while ((numbers[right] >= pivot) && (left < right))
```

```
        right--; // [right] elementi [pivot] elementidan katta bo'lguncha o'ng chegara tomon yuramiz
        if (left != right) // agar chegaralar yopilmasa
        {
          numbers[left] = numbers[right]; // ruzsat beradigan element joyiga [right] elementni joylashtiramiz
          left++; // chap chegarani o'ngga suramiz
        }
        while ((numbers[left] <= pivot) && (left < right))
          left++; // [left] element [pivot] dan kichik bo'lguncha chap chegarani suramiz
        if (left != right) // chegaralar teng bo'lmaguncha
        {
          numbers[right] = numbers[left]; //  [left] elemntni joyiga [right] elementni joylashtiramiz
          right--; //o'ng chegarani o'ngga suramiz
        }
      }
      numbers[left] = pivot; // ruxsat beradigan elementni joyiga qo'yamiz
      pivot = left;
      left = l_hold;
      right = r_hold;
      if (left < pivot) // massivning chap va o'ng qismlari uchun saralashni rekursiv chaqiramiz
        quickSort(numbers, left, pivot - 1);
      if (right > pivot)
        quickSort(numbers, pivot + 1, right);
    }
    int main()
    {
      int a[10];
      // Massivni tasodifiy sonlar bilan to'ldiramiz
      for (int i = 0; i<10; i++)
```

```
    a[i] = rand() % 20 - 10;
  // saralashgacha bo’lgan massiv elementlarini chiqaramiz
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  printf("\n");
  quickSort(a, 0, 9); // saralash funksiyasini chaqirish
  // saralashdan keyingi massiv elementlarini
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  printf("\n");
  getchar();
  return 0;
}
```

Natija:



1.9. Birlashtirishli saralash

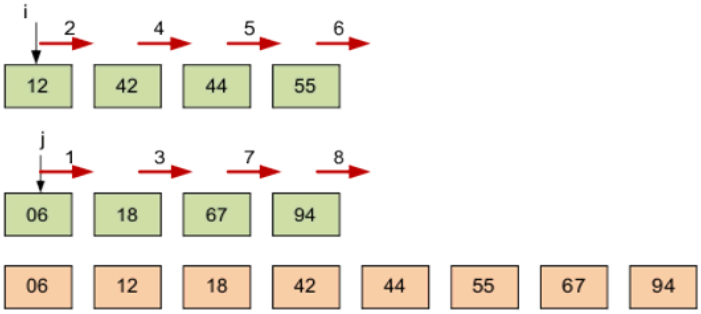
Saralash algoritmi har doim ham samarali bo’lavermaydi, agar ma’lum bir vaqt ichida ma’lumotlarga faqat bittasiga ruxsat bo’lgan ma’lumotlar tuzilmasi bo’lsa.

Fayllarni saralash asosiy metodi – birlashtirishli saralash. Birlashtirishli saralash — ma’lum bir ketma-ketlikdagi tartiblangan ma’lumotlar ro’yxatini (yoki boshqa tuzilma, elementlariga faqat ketma-ket murojaat qilsa bo’ladigan) saralash algoritmi. Siklik elementlarni tanlash bilan bir nechta ketma-ketlikni bitta tartiblangan ketma-ketlik qilish birlashtirish deyiladi. Birinchi asosiy vazifa bir nechta kichik vazifalarga bo’linadi va agar ularning hajmi yetarlicha kichik bo’lsa, vazifa rekursiv ravishda bajariladi. So’ng ularning yechimi kombinatsiyalashtiriladi va natijada dastlabki vazifa yechimi topiladi. Bir nechta ma’lumotlar ustida amal faza deyiladi.

Takrorlanayotgan eng kichik jarayon o’tish yoki etap deyiladi.

Birlashtirish jarayoni 2 ta n/2 o’lchamli ketma-ketlikni birlashtirib bitta n o’lchamli ketma-ketlik hosil qilishdir.bu ikkita tartiblangan ketma-

ketlik boshlang’ich elementlari tanlanadi va ulardan eng kichigi tanlanadi va o’sha ketma-ketlikdan shu element olib tashlanadi. Bu jarayon birirta ketma-ketlik tugaguncha davom etadi. Qolgan elementlar asosiy ketma-ketlikning oxiriga qo’yiladi.



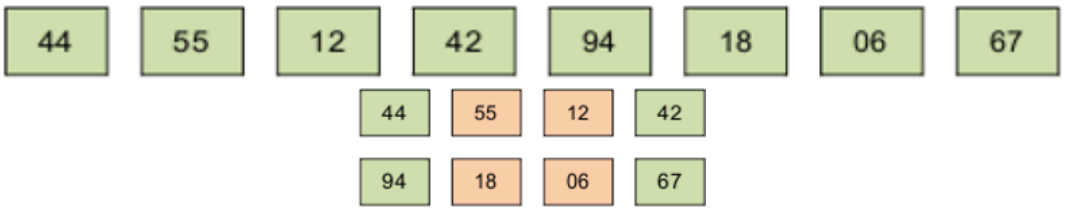
Birlashtirishli saralash ko’pincha tez saralash usuliga o’xshaydi. Birlashtirishli saralashning unumdorligi piramidali va tez saralash usulining o’rtasidadir. Lekin piramidali va tez saralashdan farqli ravishda birlashtirishli saralash, massivdagi elementlarni qayta joylashtirishga bog’liq bo’lmaganligi uchun birlashtirishli saralash bir xil ishlaydi.

Uning yana bir ustunligi shundan iboratki, u ketma-ket faqat bitta elementga ruxsat beriladigan ma’lumotlar tuzilmasi bilan ishlashga qulay hisoblanadi. Bularga tashqi qurilmadagi fayl va bog’langan ro’yxatlar misol bo’la oladi. Bu usul asosan tashqi saralashda ishlatiladi.

Bu usulning kamchiligi shuki, u xotirada fayl hajmiga teng katta joy talab qiladi. Shuning uchun katta fayllarni saralashda birlashmali saralash tezkor xotira uchun muammo tug’diradi. Agar saralash vaqti muhim bo’lib saralanayotgan elementlar tezkor xotiraga sig’ishi kafolatlansa, unda birlashtirishli saralash ishlatish ma’qulroq.

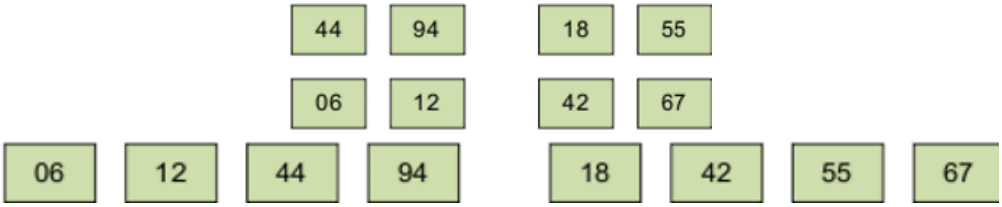
Ikki yo’lli birlashtirish algoritmi

Chiquvchi ketma-ketlik ikkita ketma-ketlikka bo’linadi:

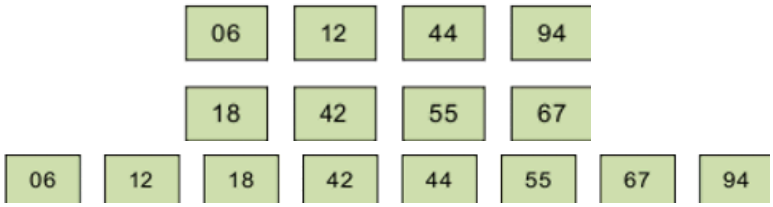


Bu ikki ketma-ketlik bittaga birlashtiriladi, tartiblangan juftliklardan iborat bo’ladi:

Olingan ketma-ketlik qaytadan ikkiga bo’linadi va tartiblangan to’rttaliklar yig’iladi.



Olingan ketma-ketlik qaytadan ikkiga bo’linadi va tartiblangan sakkizliklar yig’iladi.



Ushbu amal olingan tartiblangan ketma-ketlik saralangan ko’rinishga kelmaguncha takrorlanadi.

Asosiy amal birlashtirish hisoblanadi. Birlashtirish orqali fayllarni joylashtirish uchun qo’shimcha xotira ajratiladi. Chiziqli operatsiya e’lon qilingan massivdagi elementlarning soni bilan bog’liq.

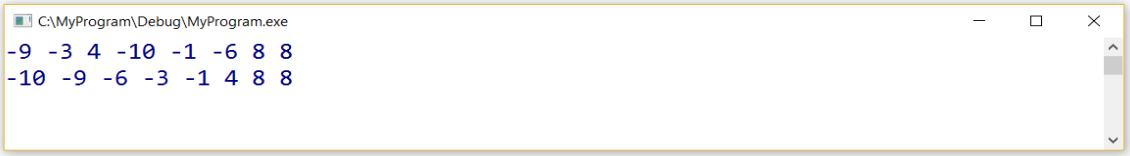
Ikki yo’lni birlashtirish usulining C dagi dasturi

```
#include <stdio.h>
#include <stdlib.h>
// ikki yo’lli saralash funksiyasi
void merge(int *a, int n)
{
  int mid = n / 2; // saralanayotgan ketma-ketlikning o’rtasiga o’tamiz
  if (n % 2 == 1)
    mid++;
  int h = 1; // qadam
  // shakllanayotgan ketma-ketlik uchun xotira ajratamiz
  int *c = (int*)malloc(n * sizeof(int));
  int step;
  while (h < n)
  {
    step = h;
    int i = 0;   // birinchi yo’l indeksi
```

```
        int j = mid; // ikkinchi yo'l indeksi
        int k = 0;   // natijaviy ketma-ketlik element indeksi
        while (step <= mid)
        {
          while ((i < step) && (j < n) && (j < (mid + step)))
          { // yo'lni oxiriga bormagunimizcha
            // shakllanayotgan ketma-ketlikning keying elementini to'ldiramiz
            // ikkalasidan eng kichigi ko'riladi
            if (a[i] < a[j])
            {
              c[k] = a[i];
              i++; k++;
            }
            else {
              c[k] = a[j];
              j++; k++;
            }
          }
          while (i < step)
          { // birinchi yo'lning qolgan elementlarini ko'chirib chiqamiz (agar 2-oldin tugagan bo'lsa)
            c[k] = a[i];
            i++; k++;
          }
          while ((j < (mid + step)) && (j<n))
          {  // ikkinchi yo'lning qolgan elementlarini ko'chirib chiqamiz (agar 1-oldin tugagan bo'lsa)
            c[k] = a[j];
            j++; k++;
          }
          step = step + h; // keyingi bosqichga o'tamiz
        }
        h = h * 2;
```

```
        // tartiblangan ketma-ketlikni dastlabki massivga ko’chiramiz(oraliq variant)
          for (i = 0; i<n; i++)
            a[i] = c[i];
        }
      }
      int main()
      {
        int a[8];
        // massivni tasodifiy sonlar bilan to’ldiramiz
        for (int i = 0; i<8; i++)
          a[i] = rand() % 20 - 10;
        // saralashgacha massiv elementlarini chiqarish
        for (int i = 0; i<8; i++)
          printf("%d ", a[i]);
        printf("\n");
        merge(a, 8); // saralash funksiyasini chaqirish
        // saralashdan keyingi massiv elementlarini chiqarish
        for (int i = 0; i<8; i++)
          printf("%d ", a[i]);
        printf("\n");
        getchar();
        return 0;
      }
```

Natija:



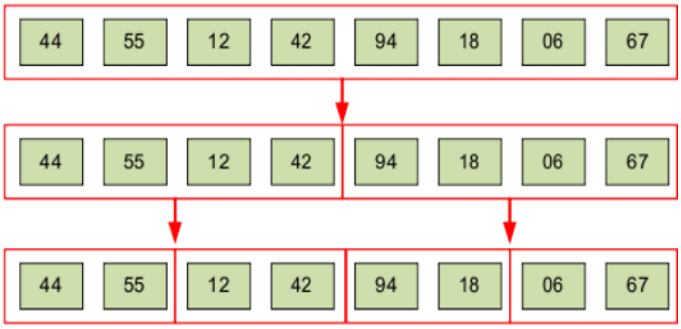
Pastga yo’nalgan birlashtirishli saralash

Chiquvchi ketma-ketlik 1 element bo’ylab ketma-ketlikni olmagunimizcha o’rtasidan rekursiv bo’linadi. Olingan ketma-ketlikdan

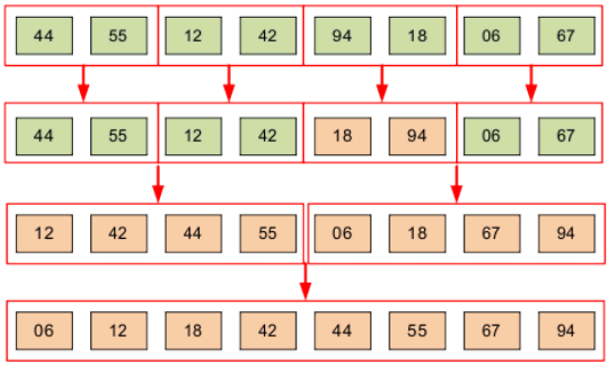
birlashtirish usulida tartiblangan juftliklarni, keyin choraklarni va h.k hosil qilamiz.

Ketma-ketlikni ko’rib chiqamiz.

Ketma-ketlikni 2 qismga bo’lamiz.



Har bir ketma-ketlikni birlashtirish usuli bilan tartiblaymiz va tayyor ketma-ketlikni olamiz.



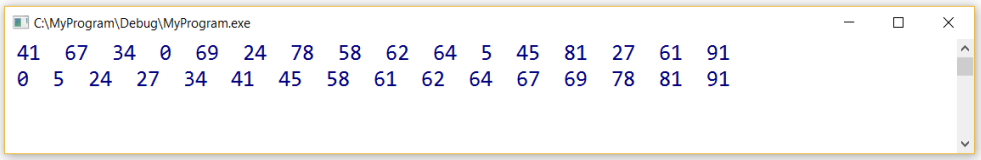
Pastga yo’nalgan birlashtirishli saralash usulining C dagi dasturi

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 16
// pastga yo’nalgan birlashtirishli saralanuvchi funksiya
void mergeSort(int *a, int l, int r)
{
  if (l == r) return; // chegaralar yopilguncha
  int mid = (l + r) / 2; // ketma-ketlik o’rtasini aniqlaymiz
  //va har bir qism uchun saralash funksiyasini rekursiv chaqiramiz
mergeSort(a, l, mid);
  mergeSort(a, mid + 1, r);
  int i = l;  // birinchi yo’l boshi
  int j = mid + 1; // ikkinchi yo’l boshi
```

```
    int *tmp = (int*)malloc(r * sizeof(int)); // qo’shimcha massiv
    for (int step = 0; step < r - l + 1; step++) // qo’shimcha massivning barcha elementlari uchun
    {
     // shakllanayotgan ketma-ketlikka ikkita yo’lning kichigini yozib qo’yamiz
     // agar  j > r bo’lsa, birinchining qoldig’i
     if ((j > r) || ((i <= mid) && (a[i] < a[j])))
     {
      tmp[step] = a[i];
      i++;
     }
     else
     {
      tmp[step] = a[j];
      j++;
     }
    }  // shakllangan ketma-ketlikni dastlabki massivga ko’chiramiz
    for (int step = 0; step < r - l + 1; step++)
     a[l + step] = tmp[step];
}
int main()
{
 int a[SIZE];
 // Massiv elementlarini to’ldirib chiqamiz
 for (int i = 0; i<SIZE; i++)
 {
  a[i] = (rand() % 100);
  printf(" %d ", a[i]);
 }
 mergeSort(a, 0, SIZE - 1); // saralash funksiyasini chaqiramiz
 printf("\n");
 // saralangan massivni chiqaramiz
 for (int i = 0; i<SIZE; i++)
```

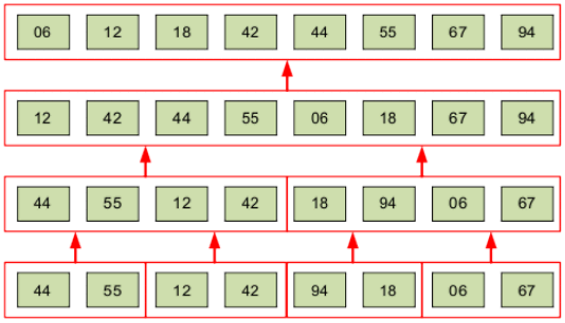
```
    printf(" %d ", a[i]);
  getchar();
  return 0;
}
```

Natija:



Yuqoriga yo'naltirib birlashtirishli saralash

Yuqoriga yo'naltirib birlashtirishli saralash usuli ketma-ketlikni teng ikkiga bo'lish protsedurasi orqali ifodalanadi. Chiquvchi ketma-ketlik xuddi ketma-ket olingan elementlar to'plami ko'rinishida bo'ladi.



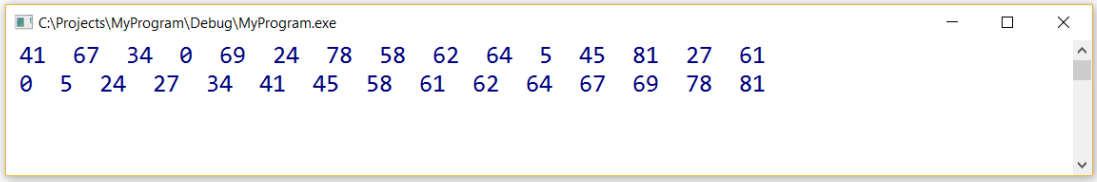
Yuqoriga yo'naltirib birlashtirishli saralash usulining C dagi dasturi

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 15
// Yuqoriga yo'naltirib birlashtirishli usuli
void mergeSort(int *a, int n)
{
  int step = 1;  //ketma-ketlikni bo'lish qadami
  int *temp = (int*)malloc(n * sizeof(temp)); // qo'shimcha massiv
  while (step < n)  // qadamlar massiv uzunligidan kichik
  {
    int index = 0;    //natijaviy massiv indeksi
    int l = 0;      // maydonning chap chegarasi
    int m = l + step;  // maydonning o'rtasi
```

```
        int r = l + step * 2;  // maydonning o’ng chegarasi
        do
        {
         m = m < n ? m : n;  // saralanayotgan maydon ketma-ketlik chegarasidan chiqmaydi
         r = r < n ? r : n;
         int i1 = l, i2 = m; // taqqoslanayotgan elementlar indeksi
         for (; i1 < m && i2 < r; ) // i1 markazga yetmaguncha va  i2 oxiriga yetmaguncha
         {
          if (a[i1] < a[i2]) { temp[index++] = a[i1++]; } // natijaviy ketma-ketlik maydonini to’ldiramiz
          else { temp[index++] = a[i2++]; }
         }
         //yoki i1 < m yoki  i2 < r – while operatorlaridan faqat bittasi bajarilishi mumkin.
         while (i1 < m) temp[index++] = a[i1++]; // saralanayotgan maydonning qolgan elementlarini kiritamiz
         while (i2 < r) temp[index++] = a[i2++]; //natijaviy massivda
         l += step * 2; // keyingi saralanayotgan maydonga o’tamiz
         m += step * 2;
         r += step * 2;
        } while (l < n); // toki saralanayotgan maydonning chap chegarasi ketma-ketlikning ichida
        for (int i = 0; i < n; i++) // shakllangan massivni a ga qaytaramiz
          a[i] = temp[i];
        step *= 2; // Bo'linish qadamini 2 barobar oshirish
      }
    }
    int main()
    {
     int a[SIZE];
     //Massiv elementlarini to’ldiramiz
     for (int i = 0; i<SIZE; i++)
```

```
  {
    a[i] = (rand() % 100);
    printf(" %d ", a[i]);
  }
  mergeSort(a, SIZE); // saralash funksiyasini chaqiramiz
  printf("\n");
  // saralangan massivni chiqaramiz
  for (int i = 0; i<SIZE; i++)
    printf(" %d ", a[i]);
  getchar();
  return 0;
}
```

Natija:



2. Qidiruv algoritmlari

Qidiruv – bu kompyuter xotirasida ma’lumotlarni qayta ishlash jarayonida qo’llaniladigan amallardan biri bo’lib, massiv elementlari orasidan berilgan elementga mos ma’lumot (element)ni aniqlash jarayonidir.

Barcha qidiruv algoritmlari quyidagi turlarga bo'linadi:

– saralangan ma'lumotlar tuzilmasidan qidirish uchun qo’llaniladigan algoritmlar;

– saralanmagan ma'lumotlar tuzilmasidan qidirish uchun qo’llaniladigan algoritmlar.

2.1. Ketma-ket qidiruv

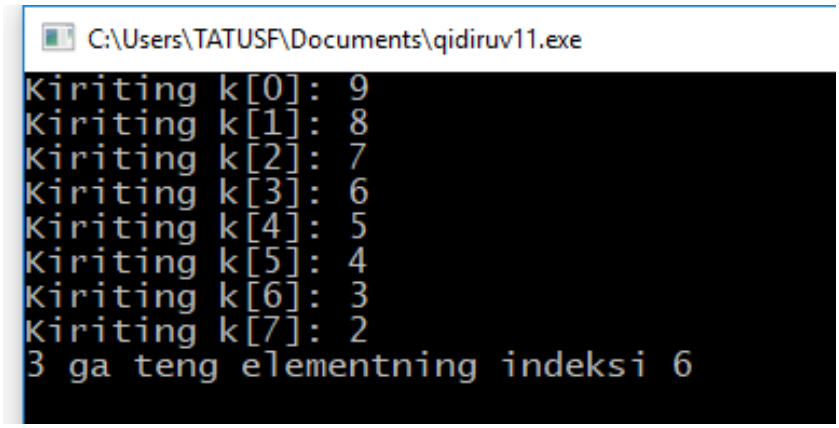
Ketma-ket qidiruv massiv kabi tashkil etilgan tuzilmaning barcha elementlarini ketma-ket qarab chiqishni talab qiladi.

Misol: Berilgan massivdan qiymati 3 ga teng bo’lgan elementini toping

```
#define _CRT_SECURE_NO_WARNINGS //scanf to’g’ri ishlashi uchun Visual Studioning oxirgi versiyalarida
#include <stdio.h>
#include <stdlib.h> // system() funktsiyasining rus tilidagi
                    //ma’lumotlar bilan ham ishlashi uchun
// n o’lchamli K massivda key qiymatli
//elementni qidiruv funksiyasi
int search(int *k, int n, int key)
{
  for (int i=0; i<n; i++)
  {
    if (k[i]==key) //key qiymatli elementni joylashtirsak,
      return i;    // uning indeksini qaytaradi
  }
  return -1;  // element topilmadi- -1 qaytaradi
}
int main()
```

```
{
  int k[8];    // 8 ta elementdan iborat massivni tasvirlaymiz
  int point; // element indeksi ko’rsatilgan qiymatga tengligi (3)
  system("chcp 1251"); // rus tilida qo’llash uchun 1251-sahifaga o’tamiz
  system("cls");       // konsol oynani tozalaymiz
  // siklda massiv elementlarini kiritamiz
  for (int i = 0; i<8; i++)
  {
   printf("Kiriting k[%d]: ", i);
   scanf("%d", &k[i]);
  }
  // massiv elementi 3 ga teng bo’lganda qidiruv funksiyasini chaqiramiz
  point = search(k, 8, 3);
  if (point == -1) // agar funksiya -1ga qaytsa massivda bunday element yo’q
    printf("Massivda qiymati 3 ga teng bo’lgan element yo’q!\n");
  else // aks holda element indeksini kiritamiz
    printf("3 ga teng elementning indeksi %d", point);
  getchar(); getchar();
  return 0;
}
```

Dastur natijasi:



2.2. Transpozitsiya usuli

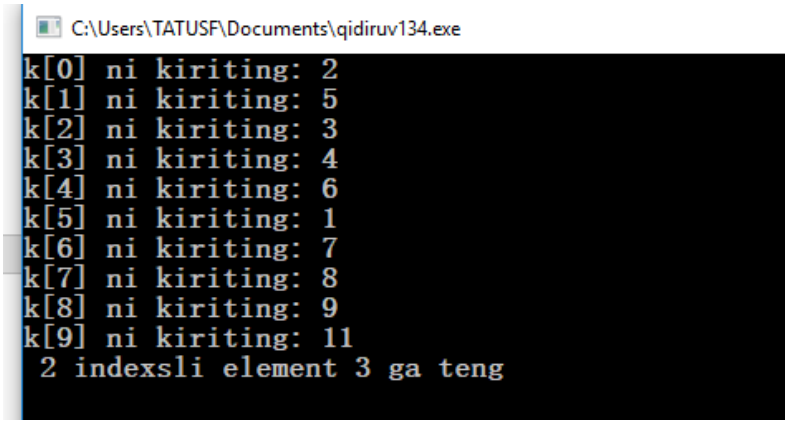
Quyidagi ko’rib chiqilgan usulning mukammallashgani transpozitsiya usulidir: har bir elementga bo’lgan murojaatda u o’zidan oldingi element bilan joy almashadi; natijada ko’p ishlatiladigan element jadval boshiga o’tadi va keyingi ularga bo’lgan murojaatlarda qisman tez topiladi.

```
#define _CRT_SECURE_NO_WARNINGS
// scanf() to’g’ri ishlashi uchun
#include <stdio.h>
#include <stdlib.h>
int search(int *k, int n, int key)
{
int temp; // almashtirish uchun yordamchi o'zgaruvchi
  for (int i = 0; i<n; i++) // sikldagi har bir elementlarni ko'rib chiqamiz
  {
    if (k[i] == key)      // agar topilgan element key qiymatga teng bo'lsa,
    {
      if (i == 0)    // agar indeks nolga teng bo'lsa, qaytamiz,
return i; // chunki massiv boshi yaqiniga ko'chishning imkoni yo'q
  temp = k[i];      // topilgan elementni oldingisi bilan almashtiramiz
     k[i] = k[i - 1];
     k[i - 1] = temp;
     return i;         // topilgan element indeksiga qaytamiz
    }
  }
  return -1;  // agar element topilmasa, -1 ga qaytamiz
}
int main()
{
  int k[10];    // 10 ta elementdan iborat massivni tasvirlaymiz
  int point;  //ko'rsatilgan qiymatga teng bo'lganda element indeksi (3)
```

```
    system("chcp 1251"); // rus tilida qo'llash uchun 1251-sahifaga o'tamiz
    system("cls");       // konsol oynani tozalaymiz
    // massiv elementlarni siklda kiritamiz
    for (int i = 0; i<10; i++)
    {
      printf("k[%d] ni kiriting: ", i);
      scanf("%d", &k[i]);
    }
       point = search(k, 10, 3); // qidiruv funksiyasini chaqirish
       if (point == -1)
       printf("Massivda 3 ga teng bo'lgan element yo'q!\n");
      else
       printf(" %d indexsli element 3 ga teng \n", point);

    getchar(); getchar();
    return 0;
}
```

Bajarish natijasi



2.3. Boshiga ko’chirish usuli

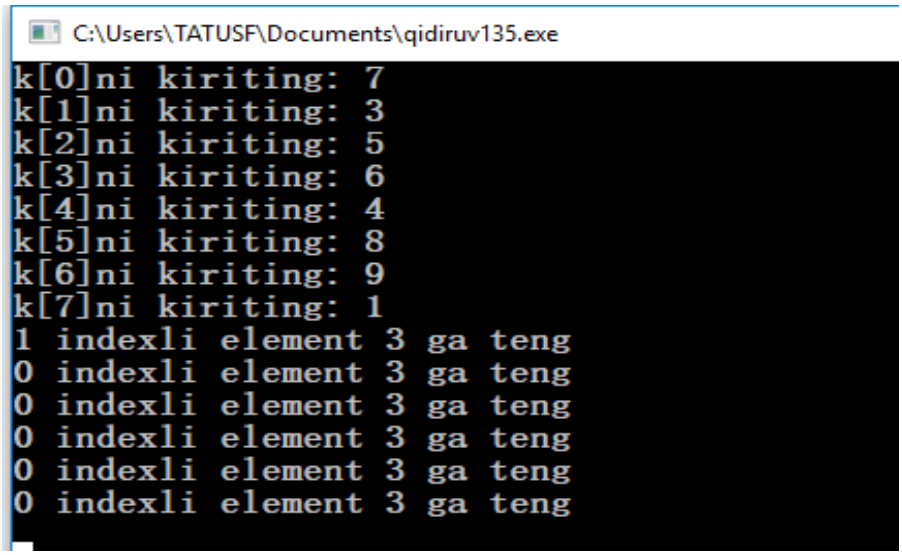
Bu usulda elementga bo’lgan har bir murojaat uni jadvalning boshiga ko’chirish bilan amalga oshadi. Natijada jadval boshida oxirida ishlatiladigan element paydo bo’ladi.

#define _CRT_SECURE_NO_WARNINGS // to’g’ri ishlashi uchun scanf()

```
#include <stdio.h>
#include <stdlib.h>        // system() funksiya-rus tilida foydalanish uchun
// n o'lchamli k massivda qidiruv funksiyasi
// foydalanilayotgan transpozitsiyada key qiymatidagi element
int search(int *k, int n, int key)
{
  int temp; // almashtirish uchun yordamchi o'zgaruvchi
  for (int i = 0; i<n; i++) // sikldagi barcha elementlarni ko'rib chiqamiz
  {
  if (k[i] == key)    // agar qiymat key ga teng bo'lsa,
    {
      temp = k[i];       // boshlang'ich element bilan topilgan elementni joyini almashtiradi
      k[i] = k[0];
      k[0] = temp;
      return i;         // topilgan element indeksini qaytaradi
    }
  }
  return -1;  // agar element topilmasa -1 qaytaradi
}
int main()
{
  int k[8];    // 8 ta elementdan iborat massivni e'lon qilamiz
 int point;  // ko'rsatilgan qiymatga teng bo'lgan elementning indeksi (3)
  system("chcp 1251"); // 1251-sahifaga o'tamiz, rus tilida qo'llash uchun
  system("cls");       // konsol oynani tozalaymiz
  // Massiv elementlarini sikl bilan kiritamiz
  for (int i = 0; i<8; i++)
  {
    printf("k[%d]ni kiriting: ", i);
```

```
    scanf("%d", &k[i]);
  }
  // qidiruv protsedurasini 6 marta takrorlaymiz (transpozitsiya hisobi
bilan)
  for (int j = 0; j < 6; j++)
  {
    point = search(k, 8, 3); // qidiruv funksiyasini chaqirish
    // 3ga teng element indeksini chiqaradi
    if (point == -1)
      printf("Massivda 3 ga teng element mavjud emas!\n");
    else
      printf("%d indexli element 3 ga teng \n", point);
  }
  getchar(); getchar();
  return 0;
}
```

Bajarish natijasi:

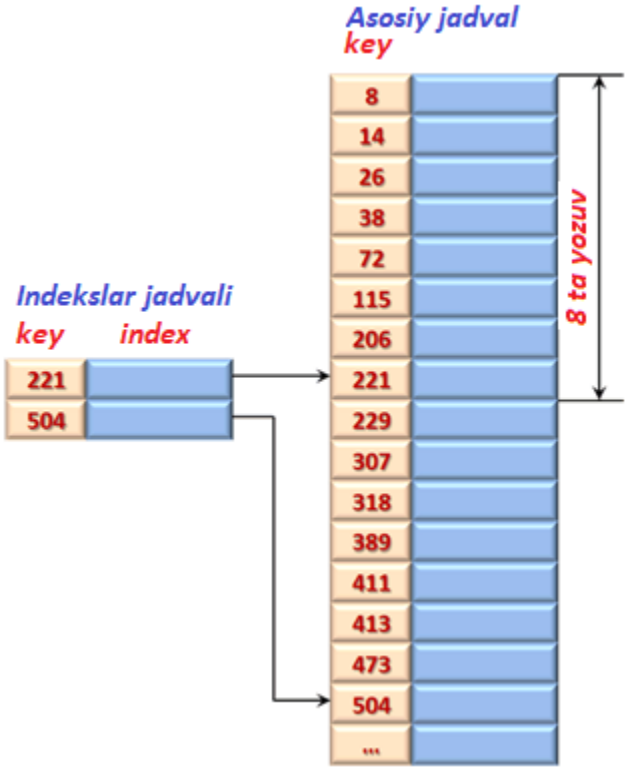


Ko’pchilik hollarda bir xil jadvallarga murojaat qilinganida qidiruvning bunday variantidan foydalanish maqsadga muvofiqdir.

2.4. Indeksli ketma-ket qidiruv

Indeksli ketma-ket qidiruv uchun saralangan jadvalga qo’shimcha ravishda indeksli jadval deb nomlanadigan yordamchi jadval chaqiriladi.

Indekslar jadvalining har bir elementi ushbu kalitga mos keladigan asosiy jadvaldagi kalit va ko'rsatgichlardan iborat. Asosiy jadvaldagi elementlar sifatida indeks jadvalidagi elementlar bu kalit bilan tartiblashtirilishi kerak.



Agar asosiy jadvalning o'lchami — ***n*** ga teng bo'lsa, indeks jadvalining o'lchami — ***ind_size = n/8*** ga teng bo'ladi.

Indeksli ketma-ket qidiruv algoritmining afzalligi qidirish vaqtini qisqartirishdan iborat, chunki ketma-ket qidiruv dastlab jadvalning asosiy jadvalidan kichikroq hajmli indekslar jadvalida amalga oshiriladi. Qachonki to'g'ri indeks topilsa, ikkinchi ketma-ketli qidiruv asosiy jadvalning kichik qismida bajariladi.

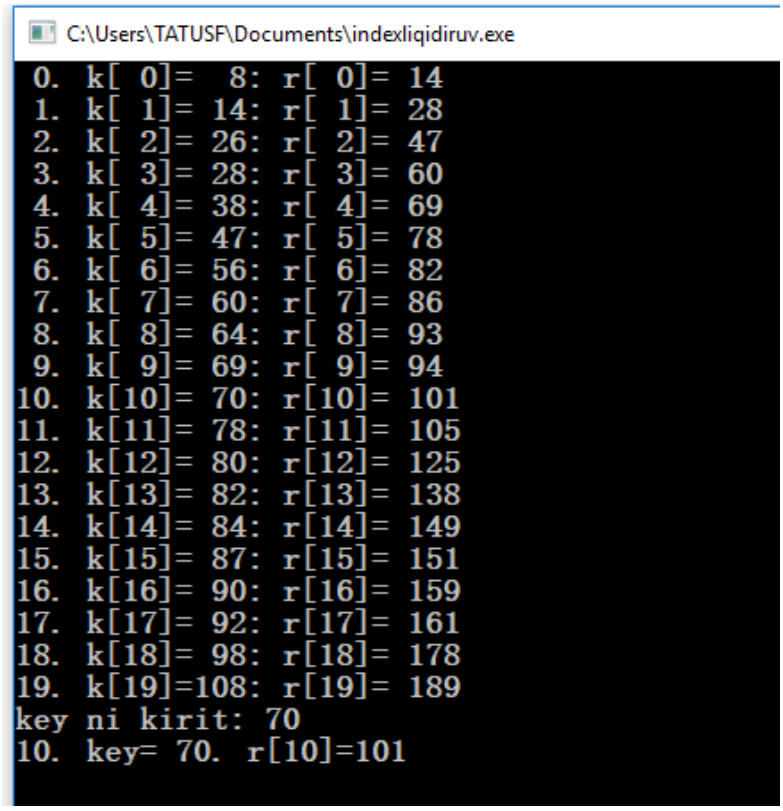
Indeksli ketma-ket qidiruvning C dagi dasturi:

```
#define _CRT_SECURE_NO_WARNINGS // scanf() to'g'ri ishlashi uchun
#include <stdio.h>
#include <stdlib.h> // system() funksiyasini qo'llash uchun
int main()
{
  int k[20]; // asosiy jadval kalitlari massivi
  int r[20]; // asosiy jadval qiymatlari massivi
```

```
  int i, j, ind_size;
  int key;
  int kindex[3]; // indeksli jadval kalitlari massivi
  int pindex[3]; // indeksli jadval indekslari massivi
  system("chcp 1251"); //konsolni rus tiliga o’tkazish uchun
system("cls");     // konsolni tozalash
  // qiymatlarni tartiblangan holda kalitlar maydonini ta’minlash
  k[0] = 8;  k[1] = 14;
  k[2] = 26;  k[3] = 28;
  k[4] = 38;  k[5] = 47;
  k[6] = 56;  k[7] = 60;
  k[8] = 64;  k[9] = 69;
  k[10] = 70; k[11] = 78;
  k[12] = 80; k[13] = 82;
  k[14] = 84; k[15] = 87;
  k[16] = 90; k[17] = 92;
  k[18] = 98; k[19] = 108;
  // qiymatlarni kiritish
  for (i = 0; i < 20; i++)
  {
    printf("%2d. k[%2d]=%3d: r[%2d]= ", i, i, k[i], i);
    scanf("%d", &r[i]);
  }
  // indeksli jadvalni shakllantirish
  for (i = 0, j = 0; i < 20; i = i + 8, j++)
  {
    kindex[j] = k[i]; // har bir 8-kalitni indeksli jadvalga o’tkazamiz
    pindex[j] = i;    // joriy indeksni saqlaymiz
  }
  ind_size = j;  // indeksli jadvalni o’lchamini eslab qolamiz
  pindex[j] = 20; // oxirgi indeks 20 ga teng
 // izlash
  Printf ("key ni kirit: "); // kalitli maydonni kiritamiz
  scanf("%d", &key);
```

```
        // indeksli jadval elementlarini ko'rib chiqamiz
        for (j = 0; j < ind_size; j++)
        {
          if (key < kindex[j]) // agar kiritilgandan kichik kalitni topsak,
            break;             // sikldan chiqamiz – biz asosiy jadvalning kerakli
maydonini topdik
        }
        if (j == 0)  i = 0;       // asosoiy jadvaldagi izlanayotgan oraliqning
boshlang'ich indeksini i ga ta'minlaymiz
        else  i = pindex[j - 1];
        for (i; i<pindex[j]; i++) // asosiy jadvalda qidiruvni tashkil etamiz
        {                         //indeksli jadvalning keying indeksigacha
          if (k[i] == key)  // agar kiritilgan qiymat topilsa, uni chiqaramiz
            printf("%2d. key=%3d. r[%2d]=%3d", i, k[i], i, r[i]);
        }
        getchar(); getchar();
        return 0;
      }
```

Natija:



2.5. Binar qidiruv

Binar qidiruv – tartiblangan massivda amalga oshiriladi. Binar qidiruvda qidirilayotgan kalit massivdagi o'rta element kaliti bilan solishtiriladi. Agar ular teng bo'lsa qidiruv (eng yaxshi) yakunlanadi. Aks holda massivning chap yoki o'ng qismlarida qidiruv jarayoni davom ettiriladi.

Algoritm rekursiv yoki rekursiv bo'lmagan ko'rinishlarda bo'lishi mumkin.

Binar qidiruvni oraliqni teng ikkiga bo'lish usuli deb ham atash mumkin.

Qidiruvda qadamlar soni $\log_2 n \uparrow$ ga teng.

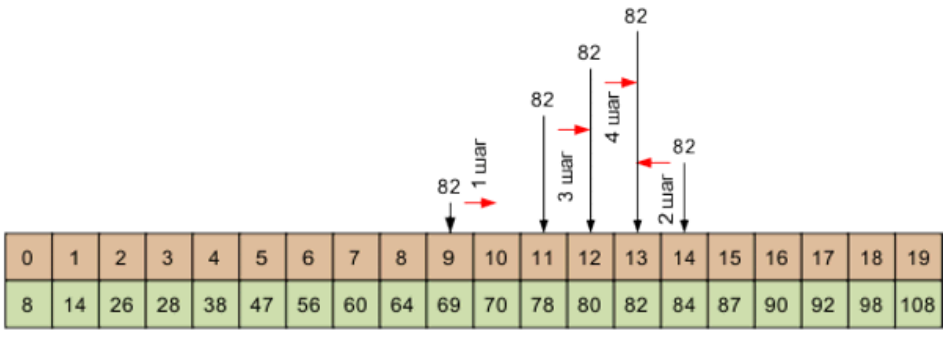
n – elementlar soni;

↑ - katta butun songa yaxlitlash.

Har bir qadamda o'rtasidan bo'lib qidiriladi va bunda quyidagi formuladan foydalaniladi:

$$mid = \frac{left - right}{2}$$

Agar qidirilayotgan element *mid* indeksli elementga teng bo'lsa, qidiruv yakunlanadi. Qidirilayotgan element *mid* elementdan kichik bo'lsa u holda bu qiymat *mid* dan oldin, aks holda (katta bo'lsa), *mid* qiymatdan keyin joylashtiriladi.



Massivning chap va o'ng chegara qiymatlarini aniqlab olamiz:

left = 0, right = 19

1-qadam. Massivning o'rta indeksini topamiz:

mid = (19+0)/2=9

Bu qiymatni qidirilayotgan indeks bilan taqqoslaymiz:

69 < 82

Chap chegarani siljitamiz:

left = mid = 9

2-qadam. Massivning o’rta qiymatini topamiz :

mid = (9+19)/2=14

bu qiymatni qidirilayotgan indeks bilan taqqoslaymiz:

84 > 82

O’ng chegaraga suramiz:

right = mid = 14

3-qadam. Massivning o’rta indeksini topamiz:

mid = (9+14)/2=11

bu qiymatni qidirilayotgan indeks bilan taqqoslaymiz:

78 < 82

Chap chegaraga siljitamiz:

left = mid = 11

4-qadam. Massivning o’rta indeksini topamiz:

mid = (11+14)/2=12

Bu qiymatni qidirilayotgan indeks bilan solishtiramiz:

80 < 82

Chap chegaraga suramiz:

left = mid = 12

5-qadam. Massivning o’rta indeksini topamiz:

mid = (12+14)/2=13

Bu qiymatni qidirilayotgan indeks bilan taqqoslaymiz:

82 = 82

Yechim topildi!

Qidiruv qadamlar sonini kamaytirish uchun darhol qidiruv chegaralarini segmentning o'rtasidan keyingi elementga o'tkazish mumkin:

left = mid + 1

right = mid - 1

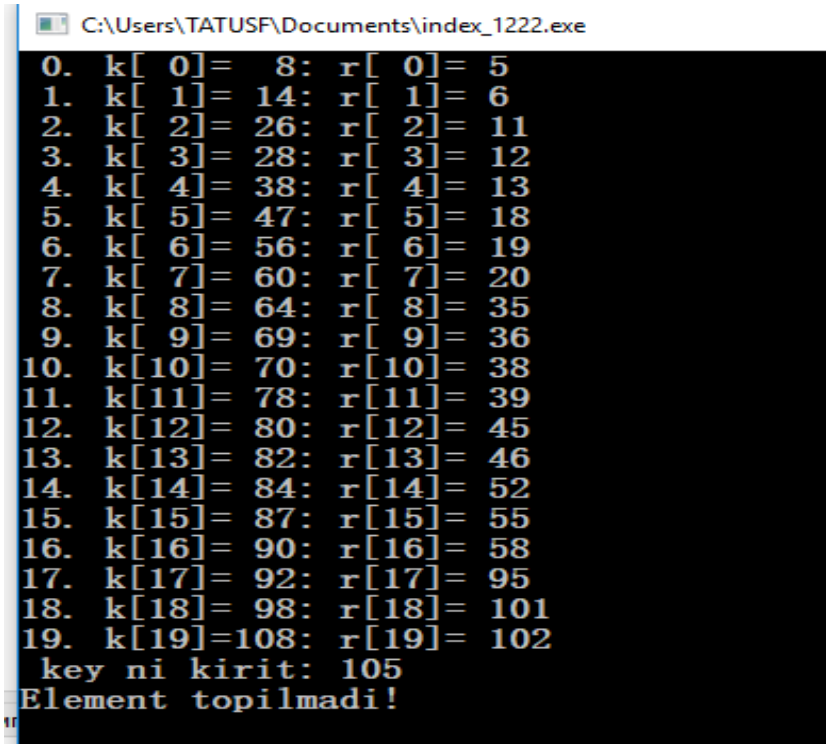
Binar qidiruvning C++ dagi dasturi:

```
#define _CRT_SECURE_NO_WARNINGS // scanf() ishlashi uchun
#include <stdio.h>
#include <stdlib.h> // system() funksiyadan foydalanish uchun
int main()
```

```
{
  int k[20]; // asosiy jadval kalitlari majmuasi
  int r[20]; // asosiy jadvaldagi yozuvlar majmuasi
  int key, i;
  system("chcp 1251"); // konsol oynani rus tiliga o’girish uchun
  system("cls");     // konsol oynani tozalash
  k[0] = 8;  k[1] = 14;
  k[2] = 26;  k[3] = 28;
  k[4] = 38;  k[5] = 47;
  k[6] = 56;  k[7] = 60;
  k[8] = 64;  k[9] = 69;
  k[10] = 70; k[11] = 78;
  k[12] = 80; k[13] = 82;
  k[14] = 84; k[15] = 87;
  k[16] = 90; k[17] = 92;
  k[18] = 98; k[19] = 108;
  // yozuvlarni kiritish
  for (i = 0; i < 20; i++)
  {
    printf("%2d. k[%2d]=%3d: r[%2d]= ", i, i, k[i], i);
    scanf("%d", &r[i]);
  }
  printf(" key ni kirit: "); //qidirilayotgan kalitni kiritamiz
  scanf("%d", &key);
  int left = 0; // qidiruvning chap va o'ng chegaralarini belgilaymiz
  int right = 19;
  int search = -1; // topilgan elementlar indeksi -1 (element topilmadi)
while (left <= right) // hali chap chegara o’ngga “sakrab” bormaydi
  {
    int mid = (left + right) / 2; // o’rtasini topamiz
    if (key == k[mid]) {   // agar qidirilayotgan element kalitga teng
bo’lsa
      search = mid;     // izlanayotgan elementni topdik,
      break;            // sikldan chiqamiz.
```

```
        }
        if (key < k[mid])     // kalit topilgan o’rta qiymatdan kichik bo’lsa
         right = mid - 1;  // o’ng tomonga surib, chap tomonda qidiruvni davom ettiramiz
        else                  // aks holda
          left = mid + 1;    // chap tomonga surib, o’ng tomonda qidiruvni davom ettirmiz
       }
       if (search == -1)     // element indeksi -1 bo’lsa, element topilmadi
        printf("Element topilmadi!\n");
       else          // aks holda elementning kaliti va qiymatini chiqaramiz
        printf("%d. key= %d. r[%d]=%d", search, k[search], search, r[search]);
       getchar(); getchar();
       return 0;
      }
```

Natija:



Adabiyotlar ro’yxati

1. Adam Drozdek. Data structures and algorithms in C++. Second edition. 2001 by Brooks/ Cole.

2. Narzullaev U.X., Qarshiev A.B., Boynazarov I.M. Ma’lumotlar tuzilmasi va algoritmlar. //O’quv qo’llanma. Toshkent: Tafakkur nashriyoti, 2013 y. – 192 b.