I BOB Qidiruv va saralash algoritmlari

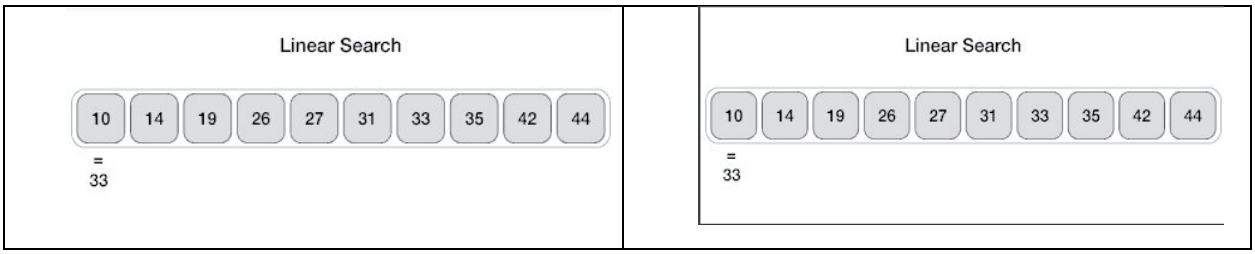
Qidiruv algoritmlari elementning mavjudligini tekshirish yoki u saqlanadigan har qanday ma’lumotlar tuzilmasidan elementning joylashgan joyini tekshirish uchun mo‘ljallangan. Qidiruv operatsiyalari turiga qarab, ushbu algoritmlar odatda ikkita toifaga bo‘linadi.

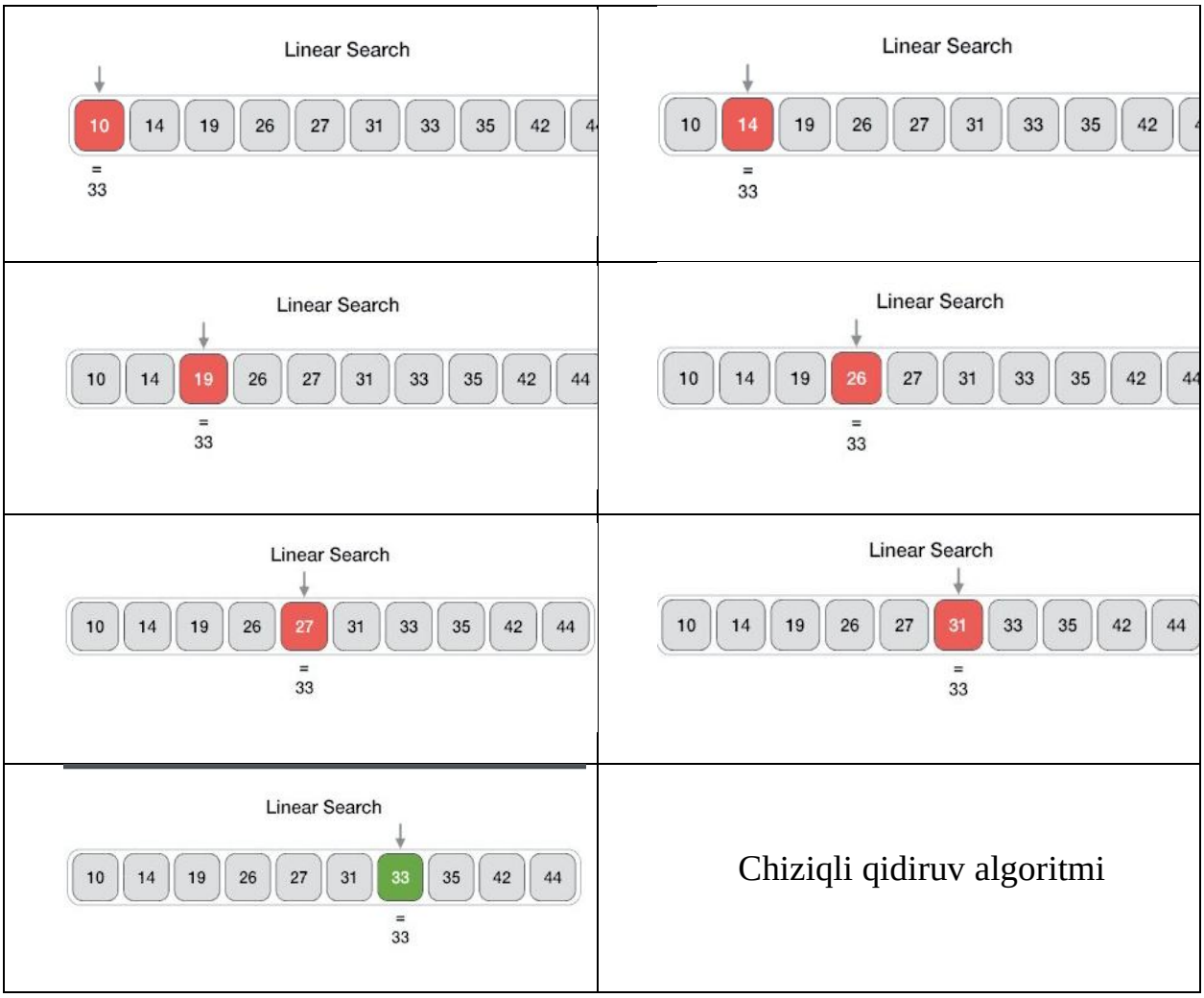
1) Ketma-ket qidirish: bunda ro‘yxat yoki massiv ketma-ket o‘tadi va har bir element tekshiriladi. Masalan: chiziqli qidiruv.
2) Intervalli qidiruv: ushbu algoritmlar saralangan ma’lumotlar tuzilmalarida qidirish uchun maxsus ishlab chiqilgan. Ushbu turdagi qidirish algoritmlari chiziqli qidiruvga qaraganda ancha samarali, chunki ular qidiruv strukturasining markazini qayta-qayta nishonga oladi va qidiruv maydonini yarmiga ajratadi. Masalan: Binar qidiruv.

1. Qidiruv algoritmlari

1.1 Chiziqli qidiruv algoritmi

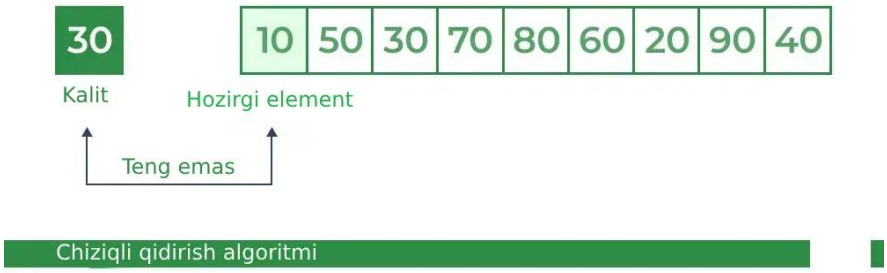
Chiziqli qidiruv ketma-ket qidirish algoritmi sifatida aniqlanadi, u bir uchidan boshlanadi va kerakli element topilmaguncha ro‘yxatning har bir elementidan o‘tadi, aks holda qidiruv ma’lumotlar to‘plamining oxirigacha davom etadi.Chiziqli qidiruv algoritmida har bir element kalit uchun potentsial bir xilligi tekshiriladi.Agar kalitga teng biron bir element topilsa, qidiruv muvaffaqiyatli bo‘ladi va ushbu elementning indeksi qaytariladi. Agar kalitga teng element topilmasa, qidiruv false javobini qaytaradi.



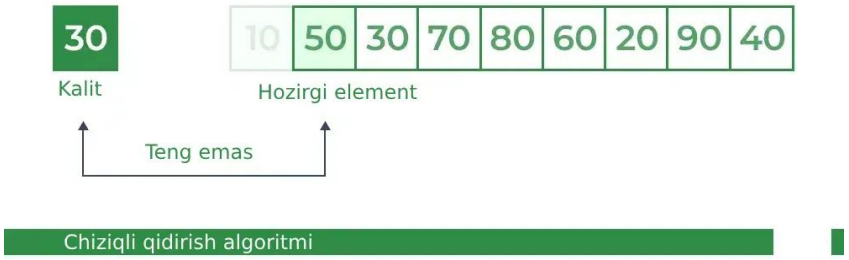


Masalan: `arr[] = {10, 50, 30, 70, 80, 20, 90, 40}` va `kalit = 30`

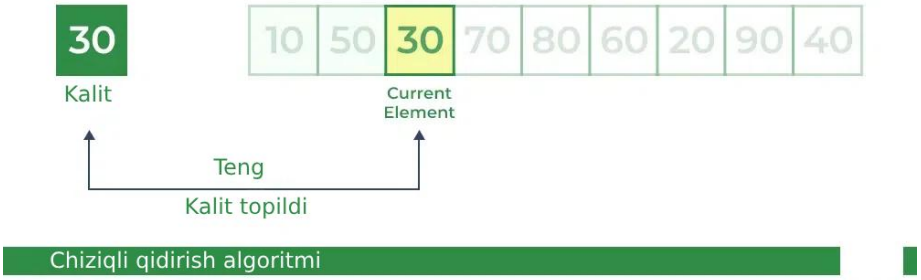
1-qadam: birinchi element bilan taqqoslashni boshlash (indeks 0) .



Arr[0] bo‘lmaganligi sababli, iterator keyingi elementga o‘tadi. Keyingi element arr[1] bilan asosiy taqqoslash.



2-qadam: endi arr[2] ni kalit bilan taqqoslaganda, qiymat mos keladi. Shunday qilib, chiziqli qidiruv algoritmi muvaffaqiyatli xabar beradi va kalit topilganda element indeksini qaytaradi (bu erda 2).



Chiziqli qidiruv algoritmining implementatsiyasi.

Quyida chiziqli qidiruv algoritmining C++ dasturlash tilidagi implementatsiyasi keltirilgan.

```
#include <iostream>
using namespace std;
// Chiziqli qidirish algoritmining funktsiyasi
int search(int arr[], int N, int x)
{
   for (int i = 0; i < N; i++)
        if (arr[i] == x)
             return i;
   return -1;
}
```

```
// Bosh funktsiya
int main(void)
{
   int arr[] = { 2, 3, 4, 10, 40 };
   int x = 10;
   int N = sizeof(arr) / sizeof(arr[0]);

   // Funktsiyani chaqirish
   int result = search(arr, N, x);
   (result == -1)
        ? cout << "Kalit elementi massivda topilmadi !"
        : cout << "Kalit elementining indexi " << result;
   return 0;
}
```

Chiziqli qidiruvning murakkabligi tahlili:

Vaqtning murakkabligi:

- Eng yaxshi holat: eng yaxshi holatda kalit birinchi indeksda bo‘lishi mumkin. Shunday qilib, eng yaxshi holat murakkabligi O(1) ga teng;
- Eng yomon holat: eng yomon holatda, kalit oxirgi indeksda, ya’ni ro‘yxatda qidiruv boshlangan oxiriga qarama-qarshi bo‘lishi mumkin. Shunday qilib, eng yomon murakkablik O(n) bu erda n - ro‘yxatning hajmi;

Xotira murakkabligi:O(1) ro‘yxat orqali takrorlash uchun o‘zgaruvchidan tashqari, boshqa hech qanday o‘zgaruvchi ishlatilmaydi.

Chiziqli qidiruvning afzalliklari:

- Chiziqli qidiruv massiv saralangan yoki yo‘qligidan qat’iy nazar ishlatilishi mumkin. U har qanday ma’lumot turidagi massivlarda ishlatilishi mumkin;
- Qo‘shimcha xotirani talab qilmaydi;
- Bu kichik ma’lumotlar to‘plamlari uchun juda mos algoritm;

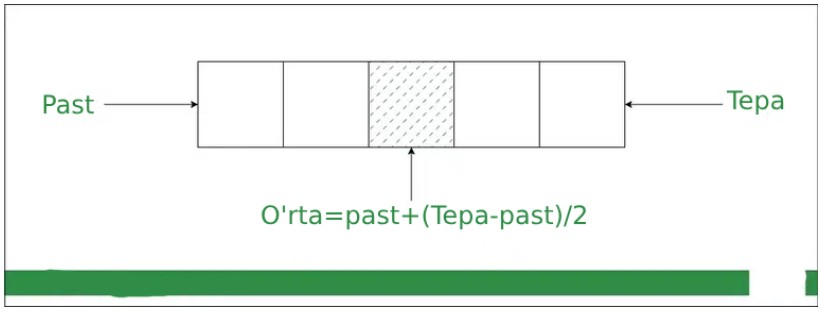
Chiziqli qidiruvning kamchiliklari:

- Chiziqli qidiruv O(n) vaqt murakkabligiga ega, bu esa o‘z navbatida katta ma’lumotlar to‘plamlari uchun uni sekinlashtiradi;

- Katta massivlar uchun mos emas;

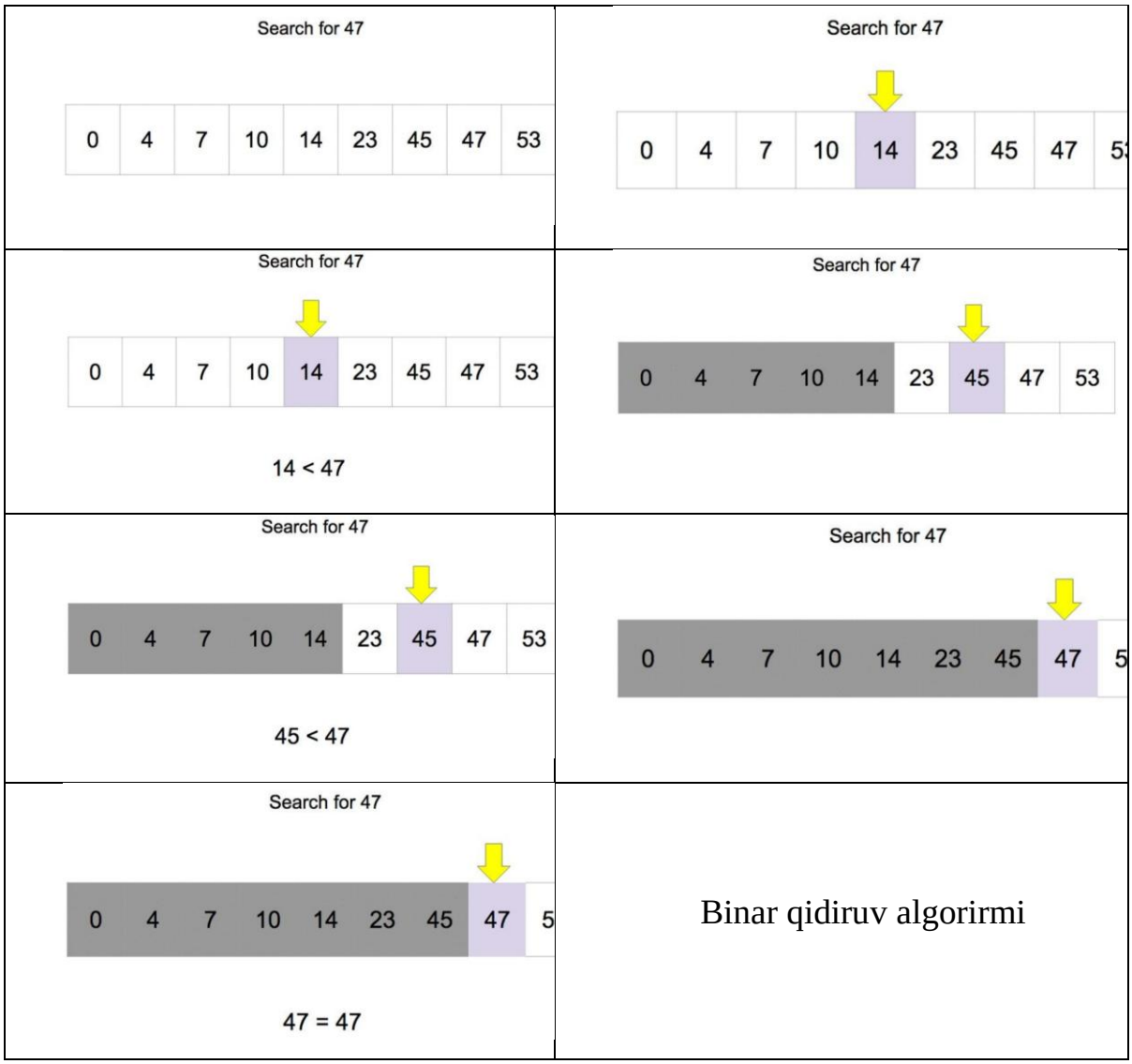
1.2 Binar (ikkilik) qidiruv algoritmi

Binar qidiruv massivni oralig'ini qayta-qayta yarmiga bo‘lish orqali va faqat saralangan massivda ishlatiladigan qidirish algoritmi hisoblanadi. Binar qidiruv g'oyasi massiv saralangan ma'lumotlardan foydalanish va vaqt murakkabligini $O(\log_2 n)$ ga kamaytirishdir. Ushbu algoritmda o‘rta indeksini topib, qidiruv maydonini ikkiga bo‘linadi.

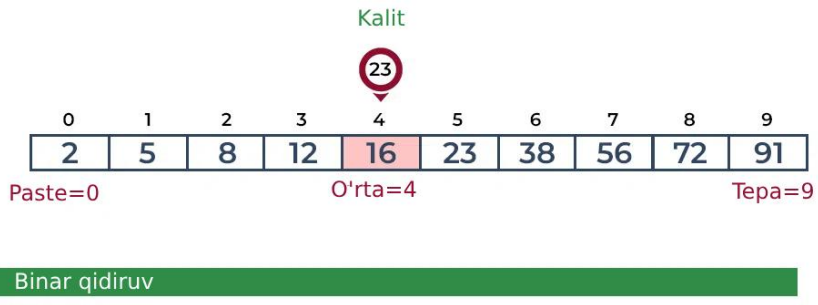


Rasm- 1 Binar qidiruv algoritmining formulasi

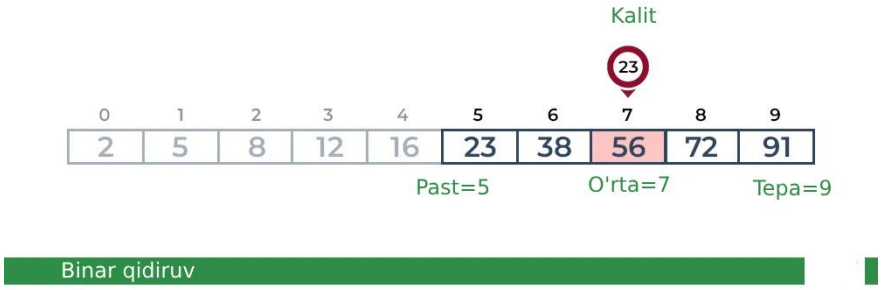
- Qidiruv maydonining o‘rta elementini kalit bilan solishtiriladi;
- Agar kalit topilsa o‘rta element, jarayon tugatiladi;
- Agar kalit topilmasa o‘rta element, keyingi qidiruv maydoni sifatida qaysi yarmi ishlatilishini tanlanadi;
- Agar kalit o‘rta elementdan kichikroq bo‘lsa, chap tomon keyingi qidirish uchun ishlatiladi;
- Agar kalit o‘rta elementdan kattaroq bo‘lsa, keyingi qidirish uchun o‘ng tomon ishlatiladi;
- Ushbu jarayon kalit topilmaguncha yoki umumiy qidiruv maydoni tugamaguncha davom etadi;



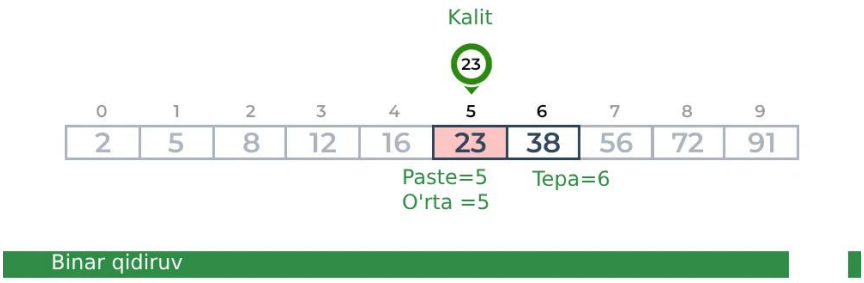
Binar qidiruvning ishlashini tushunish uchun quyidagi misolni ko‘rib chamiz: Keling bizga `arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91},` va maqsad shu massivda 23 kalitini qidirish.



Birinchi qadam: massivning o‘rta elementini topish va o‘rta elementi bilan kalitni taqqoslash. Agar kalit o‘rta elementdan kichik bo‘lsa, chapga, agar u o‘rtadan katta bo‘lsa, qidiruv maydonini o‘ngga o‘tkazing. Kalit (ya’ni, 23) hozirgi o‘rta elementdan katta (ya’ni, 16). Qidiruv maydoni o‘ngga siljiydi.



Kalit joriy o‘rtadan kam 56. Demak qidiruv maydoni chapga siljiydi.



- Ikkinchi qadam: agar kalit o‘rta elementning qiymatiga mos keladigan bo‘lsa, element topiladi va qidirishni to‘xtatadi.

Binar qidiruv algoritmi quyidagi ikki usulda amalga oshirilishi mumkin

- Takrorlash operatori yordamida;
- Rekursiya yordamida;

Quyida yondashuvlar uchun psevdokodlar berilgan.

Takrorlash operatori yordamida binar qidiruv algoritmi: Bu erda biz kalitni taqqoslash va qidiruv maydonini ikkiga bo‘lish jarayonini davom ettirish uchun bir muncha vaqt va xotiradan foydalanamiz.

C++ dasturlash tilida implementatsiyasi

```
#include <iostream>
using namespace std;
// Binar qidiruv funktsiyasi
int binarySearch(int arr[], int l, int r, int x)
{
   while (l <= r) {
        int m = l + (r - l) / 2;

        // o‘rta elementi kalit bilan tengligini tekshirish
        if (arr[m] == x)
             return m;

        //  agar  kalit  katta  bo‘lsa  chap  tarafni  olib
tashlash
        if (arr[m] < x)
             l = m + 1;

        //  agar  kalit  kichkina  bo‘lsa  o‘ng  tomonni  olib
tashlash
        else
             r = m - 1;
   }

   // agar element topilmasa
   return -1;
}

// Bosh funktsiya
int main(void)
{
   int arr[] = { 2, 3, 4, 10, 40 };
   int x = 10;
   int n = sizeof(arr) / sizeof(arr[0]);
   int result = binarySearch(arr, 0, n - 1, x);
   (result == -1)
        ? cout << "Massivda element topilmadi "
        : cout << "Elementning indexi: " << result;
   return 0;
}
```

Dastur na’tijasi:

```
/tmp/24rXQlZ2zd.o
Elementning indexi: 3
```

Vaqt murakkabligi: O (log n)

Xotira murakkabligi: O (1)

Rekursiv binar qidiruv algoritmi: Rekursiv funktsiyani yaratiladi va qidiruv maydonining o‘rtasini kalit bilan solishtiriladi.Natija asosida kalit topilgan indeksni qaytariladi yoki keyingi qidiruv maydoni uchun rekursiv funktsiyani chaqiriladi.

C++ dasturlash tilida implementatsiyasi

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

// binar qidiruv rekursiyasi
//kalit arr[......] borligi tekshiriladi
// aks holda -1
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l) {
        int mid = l + (r - l) / 2;

        // agar element o‘rtada mavjud bo‘sa
        // o‘zini qaytaradi
        if (arr[mid] == x)
             return mid;

        // Agar kalit o‘rtadan kichik bo‘sa
        // faqat chap tarafni ko‘rsatish
        if (arr[mid] > x)
             return binarySearch(arr, l, mid - 1, x);

        // aks holda element
        // faqat o‘ng tarafta bo‘ladi
        return binarySearch(arr, mid + 1, r, x);
   }
```

```
    // aks holda -1 ni qaytarish
    return -1;
}

// Bosh funktsiya
int main()
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1)
         ? cout << "Element topilmadi"
         : cout << "Kalit elementi " << result;
    return 0;
}
```

Vaqt murakkabligi:

Eng yaxshi holat: O (1)

O‘rtacha holat: O (logn)

Eng yomon holat: O (logn)

Xotira murakkabilgi: O(1), agar rekursiv chaqiruvlar to‘plami ko‘rib chiqilsa, yordamchi maydon bo‘ladi O(logn).

Binar qidiruvning afzalliklari:

- Binar qidiruv chiziqli qidiruvga qaraganda tezroq, ayniqsa katta massivlar uchun;
- Shunga o‘xshash vaqt murakkabligi bo‘lgan boshqa qidirish algoritmlariga qaraganda samaraliroq interpolatsiya qidirish yoki eksponensial qidirish;
- Binar qidiruv tashqi xotirada, masalan, qattiq diskda yoki bulutda saqlanadigan katta ma’lumotlar to‘plamlarini qidirish uchun juda mos keladi;

Binar qidiruvning kamchiliklari:

- Massivni saralash kerak;

- Binar qidiruv qidirilayotgan ma’lumotlar tuzilishini qo‘shni xotira joylarida saqlashni talab qiladi;
- Binar qidiruv massiv elementlarini taqqoslashni talab qiladi, ya’ni ularni murojat qilish imkoniyatiga ega bo‘lishi kerak;

Binar qidiruv algoritmini qo‘llash:

- Ikkilik qidiruv neyron tarmoqlarni o‘rgatish algoritmlari yoki model uchun optimal giperparametrlarni topish kabi mashinani o‘rganishda ishlatiladigan murakkabroq algoritmlar uchun qurilish bloki sifatida ishlatilishi mumkin.
- Binar qidirish kompyuter Grafikasi uchun algoritmlar ishlatilishi mumkin
- Binar qidirish ma’lumotlar bazasini qidirish uchun ishlatilishi mumkin.

1.3 Uchlik qidiruv

Uchlik qidiruv-bu massivdagi elementni topish uchun ishlatilishi mumkin bo‘lgan bo‘lish va zabt etish prinsipida ishlaydi. Bu binar qidiruvga o‘xshaydi, bu erda biz massivni ikki qismga ajratamiz, ammo ushbu algoritmda biz berilgan massivni uch qismga ajratamiz va qaysi kalit (qidirilgan element) borligini aniqlaymiz. Biz quyida ko‘rsatilgandek hisoblab mid1 va mid2 olib uch qismga qator ajratish mumkin. Dastlab, l va r mos ravishda 0 va n-1 ga teng bo‘ladi, bu erda n-massiv uzunligi.

Bu binar qidiruv bilan bir xil. Faqatgina farq shundaki, bu vaqt murakkabligini biroz ko‘proq kamaytiradi. algoritm N qadamlarni o‘z ichiga oladi. Qidiriladigan diapazon hajmi = 3^N bo‘ladi. aksincha, elementni topish uchun zarur bo‘lgan qadamlar soni to‘plam hajmining jurnalidir. Shunday qilib, ish vaqti O(logn).

Uchlamchi qidiruv uchun vaqt o‘rtacha murakkabligi O (logn) ga teng. Eng yaxshi ish vaqtining murakkabligi O(1) va eng yomon murakkablik O (logn). Ikkilik qidiruvning vaqt murakkabligi uchlik qidiruvdan kattaroqdir, ammo bu uchlik qidirish yaxshiroq degani emas. Aslida, uchlik Qidiruvdagi taqqoslashlar soni ancha ko‘p, bu uni ikkilik qidiruvga qaraganda sekinroq bo‘lishiga sabab bo‘ladi.

```
mid1 = l + (r-l)/3

mid2 = r - (r-l)/3
```

Eslatma: uchlik qidiruvni amalga oshirish uchun massiv saralangan bo‘lishi kerak yoki avval massivni saralab olish kerak.

Faraz qilaylik, bizda n o‘lchamdagi “NUM” nomli massiv bor va biz bu massivda “kalit” elementini topishimiz kerak. Uchlamchi qidiruv algoritmi quyidagicha:

Step 1: Start

Step 2: joriy etish left =0, right=n-1, mid1=0, va mid2=0.

Step 3:mid1 va mid2 ni forumladan foydalanib topish :

mid1 = left + (right - left) /3

mid2 = right - (right - left) / 3

Step 4: Agar arr[mid1] = kalit

Agar true bo‘lsa mid1 elementini qaytarish.

Step 5: If arr[mid2] = kalit

Agar true bo‘lsa mid2 elemenini qaytarish.

Step 6: If key < arr[mid1]

Unda mid1-1 va step 3 ga qaytish.

Step 7: If key > array[mid2]

Unda mid2+1 va step 3 ga qaytish.

Step 8: If key > arr[mid1] && key < arr[mid2]

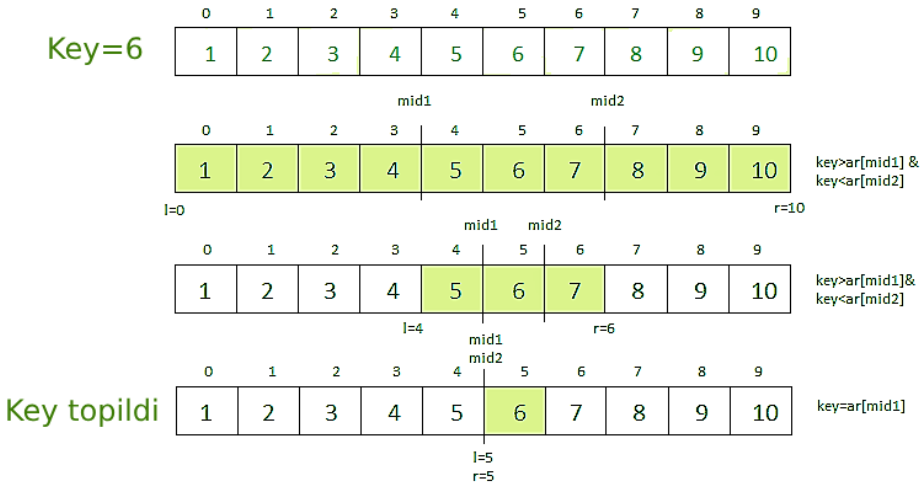
Unda left = mid1+1 va right = mid2-1, step 3 ga qaytish.

Step 9: Repeat steps 3 to 8 till Left <= Right.

Step 10: If Left > Right

Kalit topilmasa -1 ni qaytarish

Step 11: STOP



Rasm- 2Uchlamchi qidiruv algoritmi

C++ dasturlash tilida uchlik qidiruvni rekursiv amalga oshirish.

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

// Uchlik qidirish funktsiyasi
int ternarySearch(int l, int r, int key, int ar[])
{
   if (r >= l) {

        // mid1 va mid2 ni topish
        int mid1 = l + (r - l) / 3;
        int mid2 = r - (r - l) / 3;

        // tekshirish agar kalit mid da topilsa
        if (ar[mid1] == key) {
             return mid1;
        }
        if (ar[mid2] == key) {
             return mid2;
        }

        // kalit mid ta topilmasa
        // qaysi oraliqta bo‘lishini tekshirish
        // keyin qidirishni davom ettirish
```

```
          // shu oraliqta
          if (key < ar[mid1]) {

               // Kalit l va mid1 orasida
               return ternarySearch(l, mid1 - 1, key, ar);
          }
          else if (key > ar[mid2]) {

               // Kalit mid2 va r orasida
               return ternarySearch(mid2 + 1, r, key, ar);
          }
          else {

               // Kalit mid1 va mid2 orasida
               return ternarySearch(mid1 + 1, mid2 - 1, key, ar);
          }
     }

     // Kalit topilmadi
     return -1;
}

// Bosh funktsiya
int main()
{
     int l, r, p, key;

     // Massivni kiritish
     // Massivni saralash agar saralanmagan bo‘sa
     int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

     // boshlanish indexi
     l = 0;

     // oxirgi elementi
     r = 9;

     // 5 ni qidirish

     // kalit=5
     key = 5;
```

```
    // uchlik qidiruvni amalga oshirish
    p = ternarySearch(l, r, key, ar);

    // Javobni chiqarish
    cout  << key
         << " ning indexi " << p << endl;

    // kalit=50
    key = 50;

    // uchlik qidiruvdan foydalanish
    p = ternarySearch(l, r, key, ar);

    // Javobni chiqarish
    cout  << key
         << " ning indexi " << p << endl;

}
```

Afzalliklari:

- Uchlamchi qidiruv $O(\log_3 n)$ vaqt murakkabligiga ega, bu chiziqli qidiruvga qaraganda samaraliroq;
- Taqqoslashlar soni kamayadi;
- Katta ma’lumotlar to‘plamlari uchun yaxshi ishlaydi;
- Optimallashtirish muammolariga qo‘llash;
- Uchlik qidiruv rekursiv bo‘lmagan algoritmdir, shuning uchun funktsiya chaqiruvlari to‘plamini saqlash uchun qo‘shimcha xotira talab qilinmaydi, shuning uchun u bo‘sh joyni tejaydi;

Kamchiliklari:

- Uchlik qidiruv faqat buyurtma qilingan ro‘yxatlar yoki massivlarga taalluqlidir va tartibsiz yoki chiziqli bo‘lmagan ma’lumotlar to‘plamlarida ishlatilishi mumkin emas;
- Rekursiyani chuqur tushunishni talab qiladi;

- Implementatsiya oson emas;
- Uchlik qidiruv uzluksiz funktsiya uchun mos emas, chunki u qidiruv maydonini 3 qismga bo‘lishga asoslangan;

Mavzu yuzasidan savollar:

1. Chiziqli qidiruv algoritmini tushuntiring.
2. Binar qidiruv algoritmini tushuntiring
3. Uchlik qidiruv algoritmini tushintiring.
4. Chiziqli, binar va uchlik qidiruv algoritmlarining afzalliklari va kamchiliklarini aytib bering.
5. Chiziqli, binar va uchlik qidiruv algoritmlarining vaqt murakkabligiga to‘qtalib o‘ting.