TOSHKENT AMALIY FANLAR UNIVERSITETI

Ma’lumotlar tuzilmasi va algoritmlar fani

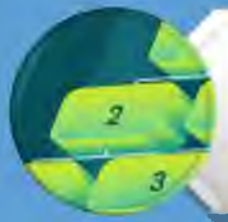
“Kompyuter injiniring” kafedrasi
Katta o’qituvchi Kendjayeva Dildora Xudayberganovna

Tez saralash algoritmi.

MA’RUZA REJASI

Tezkor saralash (Quick Sort).

Umumiy tavsif



Saralashning umumiy mexanizmi.

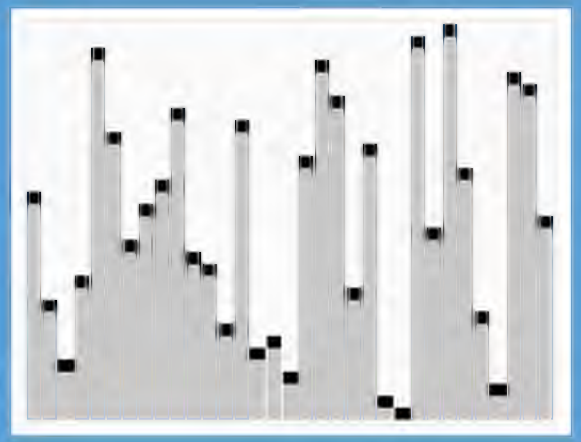
QuickSort algoritmi tahlili.

Tezkor saralash (Quick Sort).

Amaliy nuqtai nazardan Quicksort algoritmi raqobatbardosh bo‘lib, ko‘pincha MergeSort algoritmidan ustun turadi va shu sababli bu ko‘plab dasturlash kutubxonalarida standart tartiblash usuli hisoblanadi.

Quicksort algoritmining MergeSort algoritmidan katta ustunligi shundaki, u bir joyda ishlaydi - u kirish massivi bilan faqat elementlarning juft to‘g‘ridan-to‘g‘ri almashinuvini takrorlash orqali ishlaydi va shu sababli oraliq uchun faqat ozgina qo‘shimcha tezkor xotira kerak bo‘ladi.

Tezkor saralash (Quick Sort).

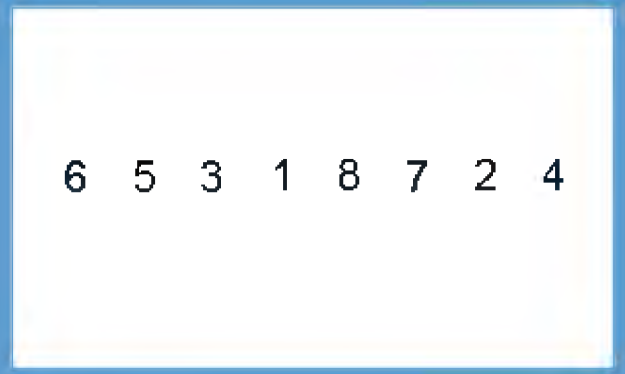


- Tezkor saralash (Quick sort – Xoara metodi) ko‘pincha qsort deb nomlanadi (uning nomi C standart kutubxonasida) - bu ingliz kompyuter olimi Toni Xoara tomonidan 1960-yilda Moskva davlat universitetida ishlab yurgan paytlarida yaratilgan saralash algoritmi hisoblanadi.

Massivlarni saralash bo‘yicha eng tez ma‘lum bo‘lgan universal algoritmlardan biri: n elementni saralashda o‘rtacha O (nlogn) almashinuv bo‘ladi. Bir qator kamchiliklar mavjudligi sababli amalda odatda ba‘zi bir o‘zgartirishlar bilan qo‘llaniladi.

Umumiy tavsif.

QuickSort - bu to‘g‘ridan-to‘g‘ri almashinuvni saralash algoritmining (Bubble Sort va Shaker Sort algoritmlari) sezilarli darajada takomillashtirilgan variant bo‘lib, u past samaradorligi bilan ham tanilgan. Asosiy farq shundaki, birinchi navbatda, almashtirishlar mumkin bo‘lgan masofada amalga oshiriladi va har bir o‘tishdan keyin elementlar ikkita mustaqil guruhga bo‘linadi.



Algoritmning umumiy g‘oyasi quyidagicha:

1)Birinchi qadam

1)Massivdan “tayanch” elementni tanlang. Bu massivdagi har qanday element bo‘lishi mumkin. Algoritmning to‘g‘riligi “tayanch” elementini tanlashga bog‘liq emas, lekin ba’zi hollarda uning samaradorligi kuchli bog‘liq bo‘lishi mumkin.

1)Ikkinchi qadam

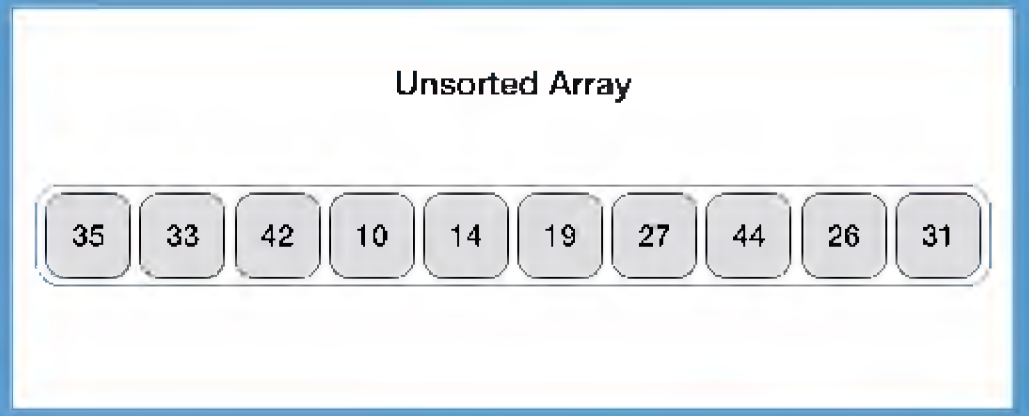
1)Qolgan barcha elementlarni "tayanch” elementi bilan taqqoslang va ularni massiv ichida tartiblang, shunday qilib massivni ketma-ket uchta doimiy segmentga bo‘ling: "tayanch elementdan *kichikroq* elementlar, "tayanch elementga teng elementlar" va "tayanch elementdan katta elementlar".

1)Uchinchi qadam

"Kichik" va "katta" qiymatlar segmentlari uchun segmentning uzunligi birdan katta bo‘lsa, bir xil amallar ketma-ketligini bajaring.

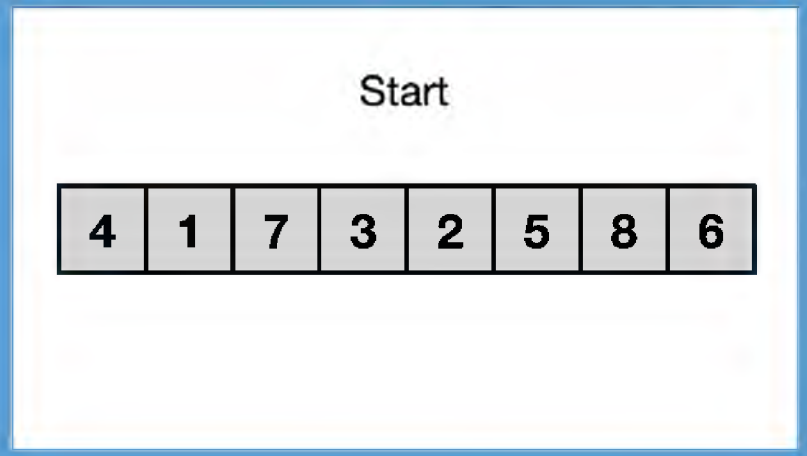
Saralashning umumiy mexanizmi

- **Quicksort** - bu ham “bo‘lib tashla va hukmronlik qil” prinsipiga asoslanuvchi algoritmdir.
- Eng umumiy ko‘rinishida psevdokod algoritmi quyida berilgan. (bu yerda A - saralanadigan massiv, low va high esa, mos ravishda, ushbu massivning saralangan qismining pastki va yuqori chegaralari)



Psevdokod nima?

- **Psevdokod** - bu imperativ dasturlash tillarining kalit so‘zlaridan foydalanadigan algoritmlarni tavsiflash uchun ixcham, ko‘pincha norasmiy til, ammo algoritmni tushunish uchun zarur bo‘lmagan tafsilotlar va o‘ziga xos sintaksisni chiqarib tashlaydi.
- Algoritmni kompyuterga tarqatish va dasturni keyinchalik bajarish uchun emas, balki odamga taqdim etish uchun mo’ljallangan.



Rekursiv QuickSort funksiyasi uchun psevdokod:

```
/* low  --> boshlang‘ich index,  high  --> yuqori index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        / * pi - bu qismlarni ajratish ko‘rsatkichi, arr [pi] endi kerakli joyda *
        /
pi = partition(arr, low, high);

         quickSort(arr, low, pi - 1);  // Pi oldin
        quickSort(arr, pi + 1, high); // pi keyin
    }
}
```

“Bo’lib tashlash” algoritmi

“Bo‘lib tashlash”ni amalga oshirishning ko‘plab usullari bo‘lishi mumkin, psevdokoddan so‘ng quyidagi algoritm qo‘llaniladi. Mantiqan sodda, biz eng chap elementdan boshlaymiz va kichik (yoki teng) elementlarning indeksini i sifatida kuzatamiz. Tekshirish paytida kichik element topsak, joriy elementni arr [i] bilan almashtiramiz. Aks holda biz joriy elementni e‘tiborsiz qoldiramiz.

```
quickSort(arr[], low, high)
{
    if (low < high)
    {

        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
```

“Bo‘lib tashlash” algoritmning psevdokodi.

- Ushbu funksiya so‘nggi elementni "tayanch” sifatida qabul qiladi, "tayanch” elementni tartiblangan qatorga to‘g‘ri holatiga qo‘yadi va kichikroq (burilishdan kichikroq) burilishning chap tomoniga va barcha katta elementlarni "tayanch element” ning o‘ng tomoniga joylashtiradi.

```
partition (arr[], low, high)
{
   // pivot (Element to‘g‘ri joyga joylashtiriladi)
   pivot = arr[high];

   i = (low - 1)  // Kichikroq element ko‘rsatkichi va tayanch
                  //to‘g‘ri holatini ko‘rsatadi
  for (j = low; j <= high- 1; j++)
   {
      // Agar joriy element “tayanch” elementdan kichikroq bo‘lsa
      if (arr[j] < pivot)
      {
         i++;    // kichik elementning o‘sish ko‘rsatkichi
         swap arr[i] and arr[j]
      }
   }
   swap arr[i + 1] and arr[high])
   return (i + 1)
}
```

“Bo‘lib tashlash” algoritmining ishlashini quyidagi misolda qarab chiqish mumkin:

arr[] = {10, 80, 30, 90, 40, 50, 70}

Indekslar: 0 1 2 3 4 5 6

low = 0, high = 6, pivot = arr[h] = 70

Kichik element indeksini initsializatsiya qilish, i = -1

j = low to high-1

j = 0 : arr[j] <= pivot, shart bajarilsa, i++ va swap(arr[i], arr[j])

i = 0

arr[] = {10, 80, 30, 90, 40, 50, 70} //Massivda o‘zgarish bo‘lmaydi

j = 1 : arr[j] > pivot, bajarilsa, hech nima o‘zgarmaydi

// i va arr [] da o‘zgarish yo‘q

j = 2 : arr[j] <= pivot, shart bajarilsa i++ va swap(arr[i], arr[j])

i = 1

arr[] = {10, 30, 80, 90, 40, 50, 70} // 80 va 30 almashdi

j = 3 : arr[j] > pivot, shart bajarilsa, hech nima o‘zgarmaydi

// No change in i and arr[]

j = 4 : arr[j] <= pivot, shart bajarilsa i++ va swap(arr[i], arr[j])

i = 2

arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 va 40 almashadi

j = 5 : arr[j] <= pivot, shart bajarilsa i++ va swap(arr[i], arr[j])

i = 3

arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 va 50 almashadi

So‘nggi natija

arr[i+1] va arr[high]

arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 va 70 almashtiriladi

“Tayanch element” hisoblanuvchi 70 ham o‘z o‘rnida. Undan kichik elementdan boshida, kattalari esa undan tepada

Quicksort algoritmi tahlili.

Massivni „tayanch“ elementiga nisbatan ikki qismga bo‘lish jarayoni $O(\log_2 n)$ vaqtni oladi. Bir xil rekursiya darajasi bajariladigan barcha bo‘linish amallari hajmi doimiy bo‘lgan boshlang‘ich massivning turli qismlarini qayta ishlagani uchun, har bir rekursiya darajasida jami $O(n)$ amallar ham talab qilinadi.

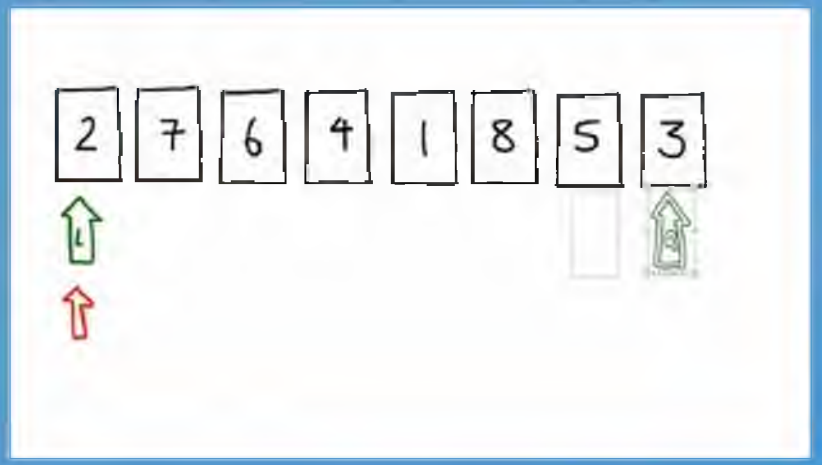
Shuning uchun algoritmning umumiy murakkabligi faqat bo‘linishlar soni, ya‘ni rekursiya darajasi bilan belgilanadi. Rekursiyaning darajsi, o‘z navbatida, kirishlarning kombinatsiyasiga va "tayanch element” qanday aniqlanishiga bog‘liq.

Eng yaxshi holat.

Eng yaxshi holatda har bir bo‘linish paytida massiv ikkita bir xil (+/- bitta element) qismlarga bo‘linadi, shuning uchun qayta ishlangan ichki massivlarning o‘lchamlari 1 ga yetadigan maksimal rekursiya darajasi $\log_2 n$ bo‘ladi. Natijada, quicksort tomonidan taqqoslash soni $O(n\log_2(n))$ algoritmining umumiy murakkabligini beradigan

$$C_n = 2C_{n/2} + n$$

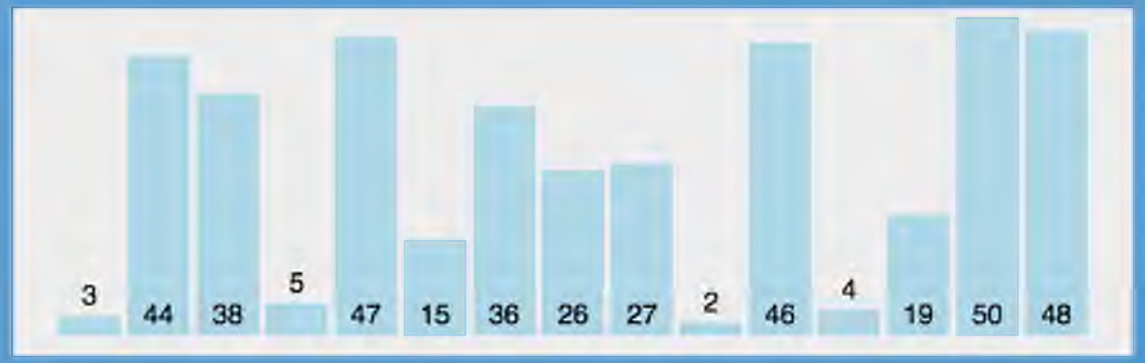
rekursiv ifodasining qiymatiga teng bo‘ladi.



O‘rtacha holat.

Kirish ma’lumotlarini tasodifiy taqsimlash uchun o‘rtacha murakkablik **faqat ehtimollik bilan** baholanishi mumkin.

Avvalo shuni ta’kidlash kerakki, aslida, "tayanch” elementi har safar massivni ikkita teng qismga ajratishi shart emas. Masalan, agar har bir bosqichda dastlabki massivning **75%** va **25%** uzunlikdagi massivlarga bo‘linish bo‘lsa, rekursiya darajasi $log_{\frac{4}{3}} n$ ga teng bo‘ladi va O(nlogn) murakkablikni beradi.



Yomon holat

Eng yomon holatda har bir "tayanch” 1 va n-1 o‘lchamdagi ikkita kichik massivni beradi, ya’ni har bir rekursiv chaqiriq uchun kattaroq massiv oldingi vaqtga nisbatan 1 ta qisqa bo‘ladi. Agar har bir ishlov berilgan elementlarning eng kichigi yoki eng kattasi mos yozuvlar sifatida tanlansa, bu sodir bo‘lishi mumkin.

Bunday holda, **n-1** “bo‘linish” amallar talab qilinadi va umumiy ishlash muddati

$$\sum_{i=0}^{n}(n-1) = O(n^2)$$

ta operatsiyani tashkil qiladi, ya‘ni saralash kvadratik vaqt ichida amalga oshiriladi. Ammo almashtirishlar soni va shunga ko‘ra ish vaqti uning eng katta kamchiligi emas. Bundan ham yomoni, bu holda algoritmni bajarish paytida rekursiya darajasi **n** ga yetadi.

Mavzu yuzasidan savollar:

1. Saralash algoritmlari va ularning tahlili haqida gapiring
2. Eng sodda algoritmlar va ularning murakkabligi
3. QuickSort va Merge Sort algoritmlarining biri-biridan farqli jihatlari.
4. Eng sodda algoritmlarning eng yaxshi va eng yomon holatdagi ishlash vaqtlarini tahlil qilish
5. Quick Sort algoritmining eng yaxshi va eng yomon holatdagi bahosini tahlil qiling.

Do you have any questions?