5 - AMALIY MASHG‘ULOT. *TEZKOR SARALASH ALGORITMI.*

Ishdan maqsad: Talabalarni tezkor saralash algoritmlar bilan tanishtirish. C++ dasturlash tilida tezkor saralash algoritmlarga oid masalalar ishlash va misollar yaratish.

Nazariy qism.Tezkor saralash algoritmi raqamlarni kiritish massivi uchun eng yomon ish vaqtiga ega. Bunday sekin bajarilish vaqtiga qaramay, tezkor saralash ko’pincha saralash uchun eng yaxshi amaliy tanlovdir, chunki u o’rtacha hisobda juda samarali: uning kutilgan bajarilish vaqti$\Theta(n^2)n$ $(n \lg n)$,va -notatsiyada yashiringan doimiy omillar juda kichik. Shuningdek, u joyida saralashning afzalligiga ega va hatto virtual xotira muhitida ham yaxshi ishlaydi.$\Theta(n \lg n)$

Saralash - ob’ektlarni ma’lum bir tartibda joylashtirish jarayoni. Bu kompyuter fanidagi eng fundamental operatsiyalardan biri bo’lib, ma’lumotlar bazalaridan tortib mashinani o’rganish algoritmlarigacha bo’lgan turli xil ilovalarda qo’llaniladi. Saralash yanada samarali kirish, qidirish va qayta ishlash uchun ma’lumotlarni tartibga solishga yordam beradi.

Ko’plab saralash algoritmlari mavjud bo’lib, ularning har biri maxsus dastur shartlariga qarab o’zining afzalliklari va kamchiliklariga ega. Ular orasida eng ko’p qo’llaniladiganlari qo’shish tartiblash, tanlash tartiblash, birlashtirish tartiblash va tez tartiblashdir.

Algoritmning asosiy bosqichlari:

1. Massivdan mos yozuvlar elementini tanlang.
2. Massiv ikkita kichik guruhga bo’linadi: birida mos yozuvdan kichik elementlar, ikkinchisida undan katta yoki unga teng elementlar mavjud.
3. Algoritm kichik guruhlarning har biriga rekursiv ravishda qo’llaniladi.
4. Kichik guruhlar birlashadi.

Afzalliklari:

- Katta ma’lumotlar to’plamlari uchun amalda juda samarali.
- Amalga oshirish oson.
- Qo’shimcha xotiradan kam foydalanadi.

Kamchiliklari:

- Katta qiymatlar yoki saralangan ma’lumotlarga chidamli emas.
- Kichik massivlarda yoki takroriy elementlarda sekin bo’lishi mumkin.

Tez tartiblash algoritmining murakkabligi qo’llab-quvvatlovchi elementni tanlashga va kiritilgan ma’lumotlarning tuzilishiga bog’liq. O’rtacha, uning murakkabligi O(n log n), bu erda n - massivdagi elementlar soni.

Tez tartiblash ko’plab sohalarda keng qo’llaniladi, jumladan:

- Ma’lumotlar bazalarini saralash.
- Qidiruv tizimlarida veb-sahifalarni saralash.

- Ma’lumotlarni tahlil qilish va mashinani o’rganishda katta hajmdagi ma’lumotlarni qayta ishlash.

Tezkor saralash algoritmining vaqt murakkabligi o’rtacha O(n log n) ni tashkil etadi, bu uni eng samarali saralash algoritmlaridan biriga aylantiradi. Biroq, agar pivot elementi noto’g’ri tanlangan bo’lsa, eng yomon murakkablik O(n^2) bo’lishi mumkin.

Tezkor saralash algoritmi turli sohalarda, jumladan, katta hajmdagi ma’lumotlarni saralash, ma’lumotlar bazalari va standart dasturlash kutubxonalarida keng qo‘llaniladi.

Amaliy qism:

Bu yerda yechimning batafsil tavsifi bilan C++ da tezkor saralash algoritmidan foydalanishingiz mumkin bo’lgan masalalar mavjud:

1-topshiriq: Butun sonlar massivini o‘sish tartibida tartiblang.

```
#include <iostream>
#include <vektor>
#include <algoritm>
using namespace std;
void quickSort (vektor<int>& arr, int left, int right) {
if (left < right) {
int pivot = arr [(left + right) / 2];
int i = left, j = right;
while ( i <= j) {
while ( arr [ i ] < pivot) i ++;
while ( arr [j] > pivot) j--;
if ( i <= j) {
swap( arr [ i ], arr [j]);
            i ++;
j--;
        }
}
      QuickSort ( arr, left, j);
      QuickSort ( arr, i, right);
       }
}
int main() {
vektor<int> arr = {7, 2, 1, 6, 8, 5, 3, 4};
   QuickSort ( arr , 0, arr.size () - 1);
for (int num: arr ) {
      cout << num << " ";
   }
   return 0;
```

```
}
```

Dastur ishlash natijasida:

Tez tartiblash algoritmi butun sonlar massivini saralash uchun ishlatiladi. Tez tartiblash funksiyasi bitta element qolguncha massivning har bir yarmida rekursiv chaqiriladi. Massivning o'rta elementi mos yozuvlar elementi sifatida tanlanadi. Elementlar mos yozuvlar bilan taqqoslanadi va mos yozuvdan kichikroq elementlar chapda, kattalari esa o'ngda bo'lishi uchun qayta tartibga solinadi.

2-topshiriq: Maxsus tuzilmani uning maydonlaridan biri bo'yicha saralash.

```
#include <iostream>
#include <vektor>
#include <algoritm>
Using namespace std;
struct Person {
string name;
int age;
};
bool compareAge (const Person & p1, const Person & p2) {
return p1.age < p2.age;
}
void quickSort (vektor<Person>& people, int left, int right) {
if (left < right) {
int pivotIndex = (left + right) / 2;
Person pivot = people[ pivotIndex ];
int i = left, j = right;
while ( i <= j) {
while (people[ i ].age < pivot.age ) i ++;
while (people[j].age > pivot.age ) j--;
if ( i <= j) {
swap(people[ i ], people[j]);
            i ++;
j--;
        }
}
      QuickSort (people, left, j);
      quickSort (people, i, right);
        }
}
int main() {
vektor<Person> people = {{"Elis", 25}, {"Bob", 30}, {"Charli", 20}};
```

```
    QuickSort (people, 0, people.size () - 1);
for (const auto & person: people) {
        cout << person.name << " - " << person.age << endl;
    }
    return 0;
}
```

Dastur ishlash natijasida:

Person yosh bo'yicha saralash uchun ishlatiladi. ' compareAge ' funksiyasi yosh bo'yicha tartiblash tartibini aniqlaydi. Yo'naltiruvchi element struktura massivining o'rta elementi sifatida tanlanadi. Keyin tuzilmalar solishtiriladi va ularning yoshiga qarab qayta tartibga solinadi.

3-topshiriq: Massivdagi eng kichik k elementni topish.

```
#include <iostream>
#include <vektor>
#include <algoritm>
Using namespace std;
int partition (vektor<int>& arr, int left, int right) {
int pivot = arr [right];
int i = left - 1;
for (int j = left; j < right; j++ ) {
if ( arr [j] < pivot) {
          i ++;
swap( arr [ i ], arr [j]);
        }
}
swap( arr [ i + 1], arr [right]);
return i + 1;
}
int kthSmallest (vektor<int>& arr, int left, int right, int k) {
if (k > 0 && k <= right - left + 1) {
int indeks = partition ( arr, left, right);
if (indeks - left == k - 1) {
return arr [indeks];
} else if (indeks - left > k - 1) {
return kthSmallest ( arr, left, indeks - 1, k);
} else {
return kthSmallest ( arr, indeks + 1, right, k - indeks + left - 1);
        }
}
return INT_MAX;
```

```
}
int main() {
vektor<int> arr = {7, 2, 1, 6, 8, 5, 3, 4};
int k = 3;
   cout << "" << k << " rd eng kichik element: " << kthSmallest ( arr , 0,
arr.size () - 1, k);
   return 0;
}
```

Dastur ishlash natijasida:

Bu kod massivdagi k-chi eng kichik elementni topish uchun tezkor tartiblash algoritmidan foydalanadi. “ Partition ” funksiyasi massivni mos yozuvlar elementiga nisbatan ikki qismga bo‘lish uchun ishlatiladi. ‘ kthSmallest‘ funksiyasi massivning mos keladigan qismidagi k-eng kichik elementni topish uchun rekursiv chaqiriladi.

4-topshiriq: kalit-qiymat juftlarini kalit bo‘yicha saralash.

```
#include <iostream>
#include <vektor>
#include <algoritm>
Using namespace std;
struct Pair {
int key;
string value;
};

int partition (vektor<Pair>& arr, int left, int right) {
int pivot = arr [right].key;
int i = left - 1;
for (int j = left; j < right; j++ ) {
if ( arr [j].key < pivot) {
          i ++;
swap(arr [ i ], arr [j]);
        }
}
swap(arr [ i + 1], arr [right]);
return i + 1;
}
void quickSortPairs (vektor<Pair>& arr, int left, int right) {
if (left < right) {
int index = partition ( arr, left, right);
       quickSortPairs ( arr , left, index - 1);
       quickSortPairs ( arr , index + 1, right);
```

```
        }
}
int main() {
vektor<Pair> pairs = {{3, "uch"}, {1, "bir"}, {2, "ikki"}};
   quickSortPairs (pairs, 0, pairs.size () - 1);
for (const auto & pair : pairs) {
      cout << pair.key << ": " << pair.value << endl;
   }
   return 0;
}
```

Dastur ishlash natijasida:

Tezkor saralash algoritmi kalit-qiymat juftliklarini kalit bo'yicha saralash uchun ishlatiladi. ' Partition ' funksiyasi massivni mos yozuvlar elementiga nisbatan ikki qismga ajratadi va ' quickSortPairs ' funksiyasi har bir kichik guruhlar uchun rekursiv chaqiriladi.

5-topshiriq: kasrlar massivini tartiblash.

```
#include <iostream>
#include <vektor>
#include <algoritm>
Using namespace std;
int partition (vektor<double>& arr, int left, int right) {
double pivot = arr [right];
int i = left - 1;
for (int j = left; j < right; j++ ) {
if ( arr [j] < pivot) {
          i ++;
swap( arr [ i ], arr [j]);
        }
}
swap( arr [ i + 1], arr [right]);
return i + 1;
}
void quickSort (vektor<double>& arr, int left, int right) {
if (left < right) {
int index = partition ( arr, left, right);
       QuickSort ( arr, left, index - 1);
       QuickSort ( arr, index + 1, right);
         }
}
int main() {
vektor<double> arr = {7,5, 2,3, 1,8, 6,9, 8,1, 5,4, 3,2, 4,7};
```

```
    QuickSort ( arr , 0, arr.size () - 1);
for (double num: arr ) {
        cout << num << " ";
}
    return 0;
}
```

Dastur ishlash natijasida:

Tezkor tartiblash algoritmi kasr sonlar massivini saralash uchun ishlatiladi. ‘ Partition ‘ funksiyasi massivni mos yozuvlar elementiga nisbatan ikki qismga ajratadi va ‘ QuickSort ‘ funksiyasi har bir kichik guruhlar uchun rekursiv chaqiriladi.

Mustaqil bajarish uchun topshiriqlar

1. Butun sonlar massivini kamayish tartibida tartiblang.
2. Satrlar massivini leksikografik tartibda saralash.
3. Maxsus sinf ob’ektlari massivini a’zolaridan biri bo’yicha saralash.
4. Saralanmagan massivda medianani topish.
5. Massivdagi maksimal elementni topish.
6. Massivdan dublikatlarni olib tashlash.
7. Kalit-qiymat juftliklari massivini kalit bo‘yicha saralash.
8. Kasr sonlar vektorini saralash.
9. Massivni uning elementlari moduli bo’yicha saralash.
10. Ob’ektlar massivini uzunligi/o’lchami bo’yicha saralash.
11. Ob’ektlarga ko’rsatgichlar massivini saralash.
12. Massivdagi k-eng kichik elementni qidiring.
13. Maxsus taqqoslash funksiyasidan foydalanib massivni tartiblang.
14. Indekslar diapazonida massivni saralash.
15. Massiv qismini aniq indeksdan boshlab saralash.
16. Butun sonlar massivini paritet bo‘yicha saralash (avval juft sonlar, keyin toq sonlar).
17. Sanalar massivini o‘sish tartibida tartiblang.
18. Satrlar massivini uzunligining o’sish tartibida tartiblash.
19. Bir nechta mezonlar bo’yicha maxsus sinf ob’ektlari massivini saralash.

20. Sonlar massivini absolyut qiymatlarning kamayish tartibida tartiblash.

21. Butun sonlar massivini o‘rnatilgan bitlar soni bo‘yicha saralash.

22. Satrlar massivini unlilar soni bo‘yicha saralash.

23. Maxsus sinf ob’yektlari massivini yaratilgan sana bo’yicha saralash.

24. Belgini e’tiborsiz qoldirib, sonlar massivini saralash (avval ijobiy, keyin salbiy).

25. Butun sonlar massivni bo‘luvchilar soni bo‘yicha saralash.

Ushbu muammolarning har biri muammoning o’ziga xos talablariga moslashtirilgan tezkor tartiblash algoritmi yordamida hal qilinishi mumkin.