1. Saralash algoritmlari

Saralash bu odatda ma’lumotlar elementlarining ma’lum bir atributiga asoslanib, ma’lumotlar elementlari to‘plamini ko‘tarilish yoki pasayish tartibida tartibga solish jarayoni sifatida qarash hisoblanadi.

Saralashning xususiyatlari:

Vaqt murakkabligi: algoritmni ishlatish uchun qancha vaqt ketishini o‘lchash. jarayonning vaqt murakkabligini aniqlash uchun saralash algoritmining eng yomon, o‘rtacha va eng yaxshi ko‘rsatkichlaridan foydalanish mumkin.

Xotira murakkabligi: saralash algoritmlari xotira murakkablikka ham ega, bu algoritmni bajarish uchun zarur bo‘lgan xotira miqdori.

Barqarorlik: Agar saralashdan keyin teng elementlarning nisbiy tartibi saqlanib qolsa, saralash algoritmi barqaror deyiladi. Bu teng elementlarning asl tartibini saqlash kerak bo‘lgan ba’zi dasturlarda muhimdir.

Joyida saralash: joyida saralash algoritmi-bu ma’lumotlarni saralash uchun qo‘shimcha xotirani talab qilmaydigan algoritm. Bu mavjud xotira cheklangan yoki ma’lumotlarni ko‘chirish mumkin bo‘lmaganda muhimdir.

Moslashuvchanlik: adaptiv saralash algoritmi-bu ishlashni yaxshilash uchun ma’lumotlarda oldindan mavjud bo‘lgan tartibdan foydalanadigan algoritm.

1.1 Bubble sort (pufakchali saralash)

Bubble sort-bu qo‘shni elementlarni noto‘g’ri tartibda bo‘lsa, ularni qayta-qayta almashtirish orqali ishlaydigan eng oddiy saralash algoritmi.

6 5 3 1 8 7 2 4	
	
	
	

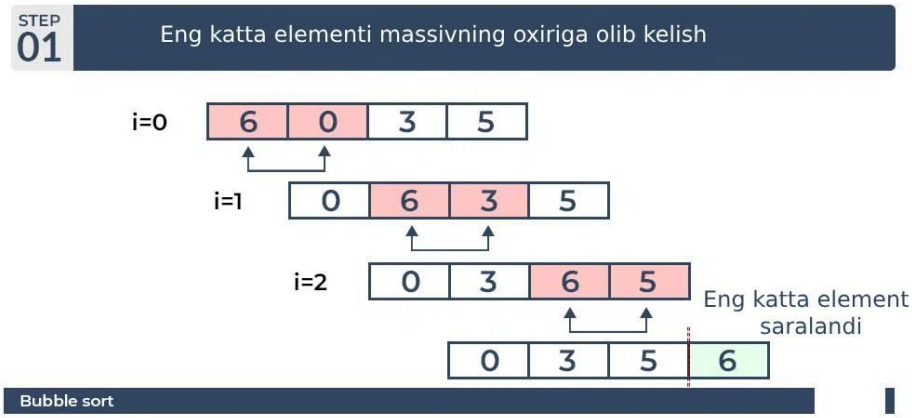
	
	
	Bubble sort algoritmi 1- qadam

Ushbu algoritmda,

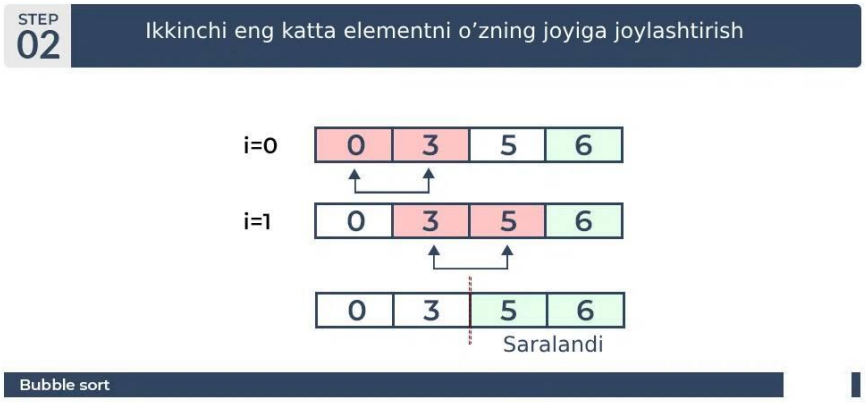
- chapdan o‘ting va qo‘shni elementlarni taqqoslanadi, kattasini esa o‘ng tomonga joylashtiramiz.
- Shunday qilib, eng katta element dastlab eng o‘ng uchiga ko‘chiriladi.
- Keyin bu jarayon ikkinchi eng kattasini topiladi va uni joylashtiramiz va hokazo ma’lumotlar saralanmaguncha davom ettiriladi.

Keling quydagi berilgan `arr[] = {6, 3, 0, 5}` massivini bubble sort yordamida saralashni ko‘rib chiqamiz.

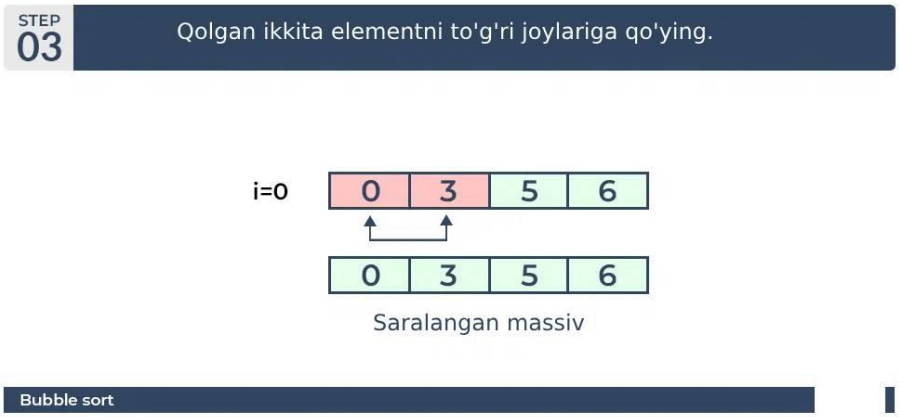
1-qadam: Berilgan massivning qo‘shi elementlari taqqoslanadi va kattasi o‘ng tomonda siljitib boriladi va oxirida eng katta elementi massivning eng chetiga olib kelamiz .



2-qadam: Ikkinchi eng katta elementni o‘zning joyiga joylashtirish.



3-qadam: Qolgan ikkita elementni to‘g’ri joylariga joylashtirish.



C++ dasaturlash tilida bubble sort saralash algoritmi implementatsiyasi.

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

//Bubble sort funktsiyasi
void bubbleSort(int arr[], int n)
{
   int i, j;
   bool swapped;
   for (i = 0; i < n - 1; i++) {
        swapped = false;
        for (j = 0; j < n - i - 1; j++) {
             if (arr[j] > arr[j + 1]) {
                  swap(arr[j], arr[j + 1]);
                  swapped = true;
             }
        }

        // agar ikki element almashsa
        // ichki tskil yordamida break
        if (swapped == false)
             break;
   }
}

// Massiv elementlarini chiqarish funktsiyasi
void printArray(int arr[], int size)
```

```
{
   int i;
   for (i = 0; i < size; i++)
        cout << " " << arr[i];
}

// Bosh funktsiya
int main()
{
   int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
   int N = sizeof(arr) / sizeof(arr[0]);
   bubbleSort(arr, N);
   cout << "Saralangan massiv: \n";
   printArray(arr, N);
   return 0;
}
```

```
/tmp/A7bfSq2dVJ.o
Saralangan massiv:
 11 12 22 25 34 64 90
```

Vaqt murakkabligi: O(n^2)

Xotira murakkabligi: O(1)

Bubble sortning afzalliklari:

- Bubble sort tushunish va amalga oshirish oson;
- Bubble sort har qanday qo‘shimcha xotira oraliq talab qilmaydi;
- Bubble sort barqaror saralash algoritmi, ya’ni bir xil kalit qiymatiga ega bo‘lgan elementlar saralangan chiqishda nisbiy tartibini saqlaydi;

Bubble sortning kamchiliklari:

- Bubble sort O(n^2) vaqt murakkabligiga ega, bu esa uni katta ma’lumotlar to‘plamlari uchun saralashni sekinlashtiradi;
- Bubble sort-bu taqqoslashga asoslangan saralash algoritmi, ya’ni kirish ma’lumotlari to‘plamidagi elementlarning nisbiy tartibini aniqlash uchun

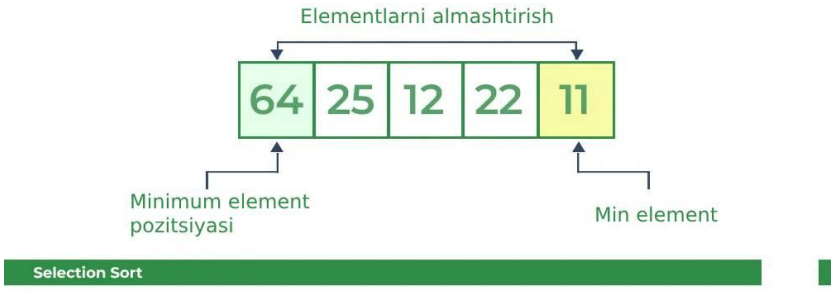
taqqoslash operatorini talab qiladi. Bu ba’zi hollarda algoritm samaradorligini cheklashi mumkin;

1.2 Selection sort (tanlab saralash)

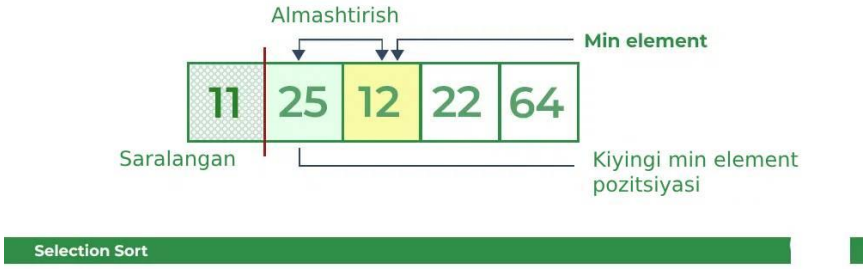
Selection sort-bu ro‘yxatning saralanmagan qismidan eng kichik (yoki eng katta) elementni qayta-qayta tanlash va uni saralanmagan qismning birinchi elementi bilan almashtiradi. Ushbu jarayon butun ro‘yxat saralanmaguncha qolgan saralanmagan qism uchun takrorlanadi.

Misol sifatida quyidagi massivni ko‘rib chiqaylik: `arr[] = {64, 25, 12, 22, 11}`

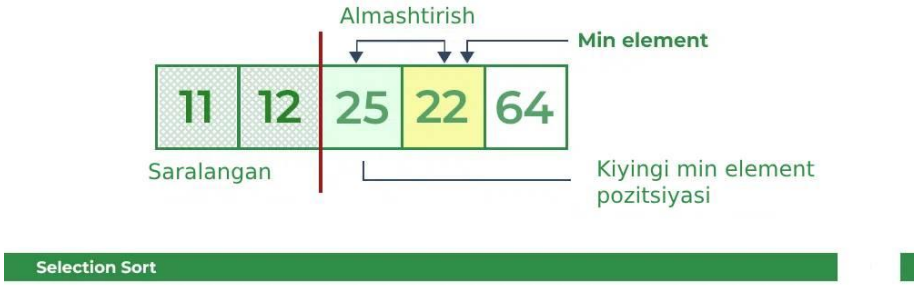
1-qadam: Tartiblangan massivdagi birinchi pozitsiya uchun indeks 0 dan 4 gacha bo‘lgan butun massivni ketma-ket chetlab o‘tish amalga oshiriladi. Hozirda 64 saqlanadigan birinchi pozitsiya, butun qatorni aylanib chiqqandan so‘ng, 11 eng kichik qiymat ekanligi aniq bo‘ladi. Shunday qilib, 64 ni 11 bilan almashtiring. Bir marta takrorlangandan so‘ng, massivdagi eng kichik qiymat bo‘lib chiqadigan 11 tartiblangan ro‘yxatning birinchi pozitsiyasida paydo bo‘ladi.



2-qadam: 25 mavjud bo‘lgan ikkinchi pozitsiya uchun yana ketma-ket qatorning qolgan qismiga o‘ting.Aylanib chiqqandan so‘ng, biz 12 massivdagi ikkinchi eng kichik qiymat ekanligini aniqladik va u massivda ikkinchi o‘rinda paydo bo‘lishi kerak va shu bilan bu qiymatlarni almashtiradi.



3-qadam:Endi, 25 mavjud bo‘lgan uchinchi joy uchun yana massivning qolgan qismiga o‘tiladi va massivda mavjud bo‘lgan uchinchi oxirgi qiymatni toping.O‘tish paytida 22 oxirgi uchinchi qiymat ekanligi aniqlandi va u massivda uchinchi o‘rinda paydo bo‘lishi kerak, bu 22 ni uchinchi o‘rinda element bilan almashtirishamiz.



4-qadam: Xuddi shunday, to‘rtinchi pozitsiya uchun massivning qolgan qismiga o‘tiladi va massivdagi to‘rtinchi oxirgi elementni topladi. 25 eng kichik qiymat bo‘lgani uchun u to‘rtinchi o‘rinni egallaydi.

5-qadam: Nihoyat massivda mavjud bo‘lgan eng yirik qiymati avtomatik ravishda qator oxirgi holatda joylashtirilgan.

Saralangan massiv

Selection Sort

C++ dasturlash tilida selectin sort salaralash algoritmining implementatsiyasi.

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

// Selection sort funktsiyasi
void selectionSort(int arr[], int n)
{
   int i, j, min_idx;

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

        // min elementni topish
        // saralanmagan massiv
        min_idx = i;
        for (j = i + 1; j < n; j++) {
             if (arr[j] < arr[min_idx])
                  min_idx = j;
        }

        // min elementni birinshi element bilan almashtirish
        if (min_idx != i)
             swap(arr[min_idx], arr[i]);
   }
}

// Massivni chiqarish funktsiyasi
void printArray(int arr[], int size)
{
   int i;
   for (i = 0; i < size; i++) {
        cout << arr[i] << " ";
```

```
            }
}

// Bosh funktsiya
int main()
{
   int arr[] = { 64, 25, 12, 22, 11 };
   int n = sizeof(arr) / sizeof(arr[0]);

   // Funktsiyasni chaqirish
   selectionSort(arr, n);
   cout << "Saralangan massiv: \n";
   printArray(arr, n);
   return 0;
}
```

Dastur natijasi:

```
/tmp/A7bfSq2dVJ.o
Saralangan massiv:
11 12 22 25 64
```

Vaqt murakkabligi: tanlov tartibining vaqt murakkabligi $O(n^2)$, chunki ikkita ichki sikl mavjud:

Xotira murakkabligi: O(1) massivdagi ikkita qiymatni almashtirishda vaqtinchalik o‘zgaruvchilar uchun ishlatiladigan yagona qo‘shimcha xotira ishlatiladi.

Selection sort algoritmining afzalliklari

- Oddiy va tushunish oson;
- Kichik ma’lumotlar to‘plamlari bilan yaxshi ishlaydi;

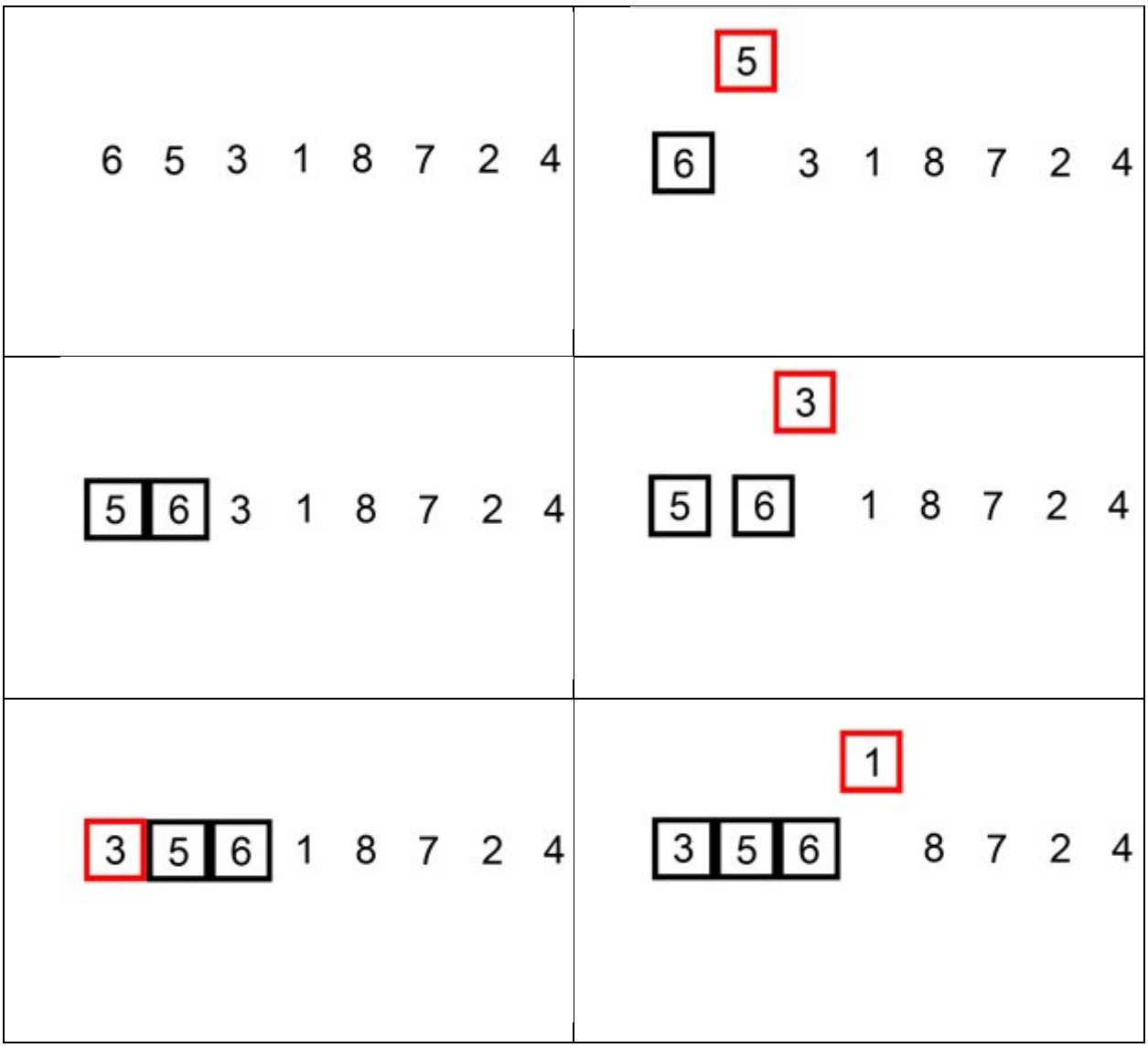
Selection sort algoritmining kamchiliklari

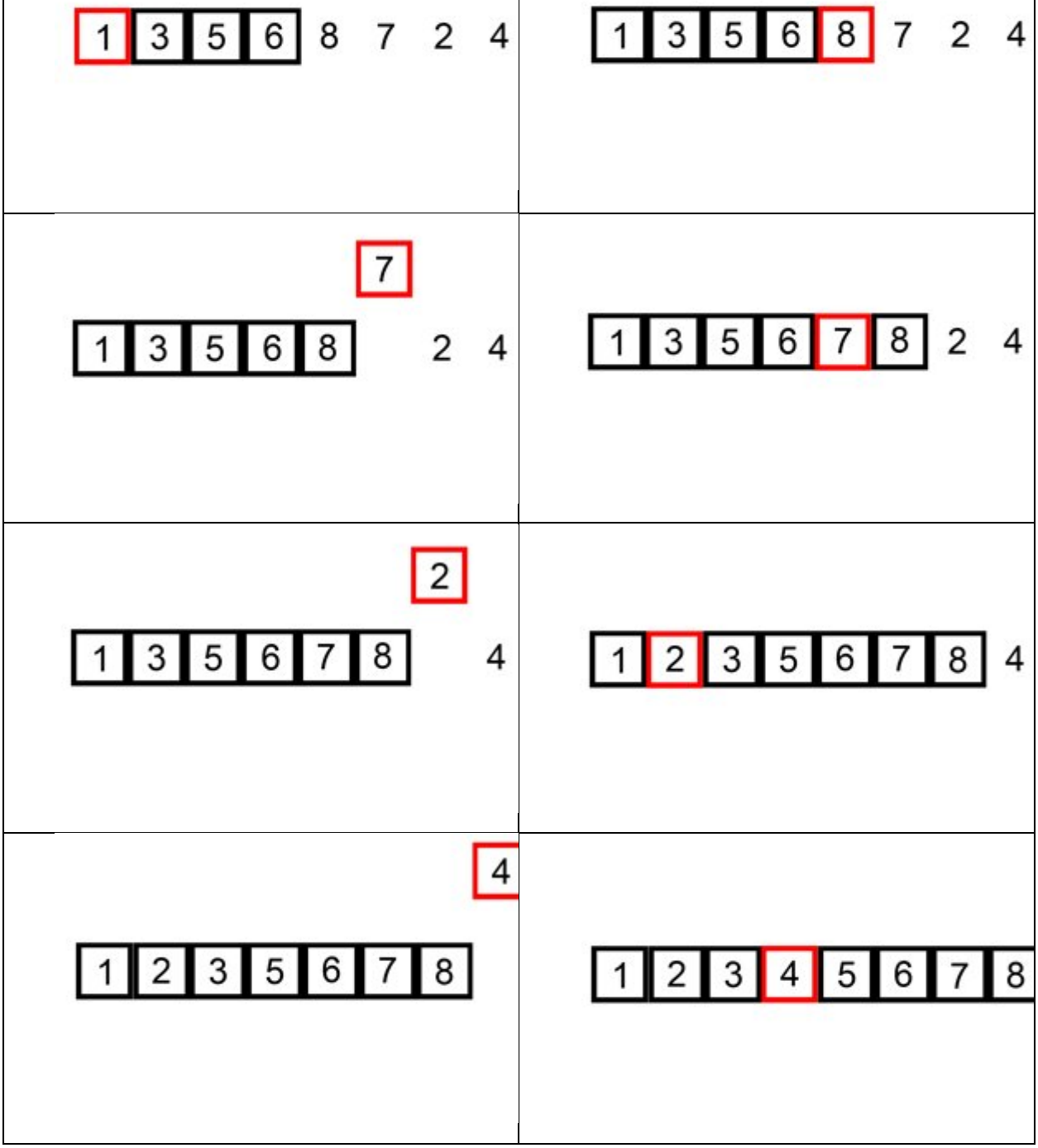
- Tanlov saralash eng yomon va o‘rtacha holatda $O(n^2)$ vaqt murakkabligiga ega;
- Katta ma’lumotlar to‘plamlarida yaxshi natija bermaydi;

- Teng kalitlarga ega elementlarning nisbiy tartibini saqlamaydi, ya’ni u barqaror emas;

1.3 Insertion sort

Selection sort- bu sizning qo‘lingizdagi kartalarni saralash uslubiga o‘xshash ishlaydigan oddiy saralash algoritmi. Massiv saralangan va saralanmagan qismga bo‘linadi. Saralanmagan qismdan qiymatlar tanlanadi va saralangan qismdagi to‘g’ri joyga joylashtiriladi.




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

	Selection sort (tanlab saralash)

n o‘lchamdagi massivni o‘sish tartibida saralash uchun massiv ustida takrorlang va joriy elementni (kalitni) avvalgisiga solishtiriladi, agar kalit element avvalgisidan kichikroq bo‘lsa, uni oldingi elementlar bilan solishtiriladi. Almashtirilgan element uchun joy ajratish uchun katta elementlarni bir joyga yuqoriga siljiting.

Algoritmining ishlashi: `arr[]: {12, 11, 13, 5, 6}`

12	***11***	***13***	***5***	***6***

1-qadam: Dastlab, massivning dastlabki ikkita elementi qo‘shish tartibida taqqoslanadi.

12	***11***	*13*	*5*	*6*

Bu erda 12 11 dan katta va ular o‘sish tartibida emas. Shunday qilib, 11 va 12 ni almashtiramiz.Shunday qilib, hozircha 11 saralangan kichik massivda saqlanadi.

11	***12***	*13*	*5*	*6*

2-qadam: Endi keyingi ikkita elementga o‘tamiz va ularni taqqoslaymiz.

11	***12***	***13***	*5*	*6*

Bu erda 13 soni 12 dan katta, shuning uchun ikkala element ham o‘sish tartibida joylashgan, shuning uchun hech qanday almashtirish sodir bo‘lmaydi. 12 shuningdek, 11 bilan birga saralangan kichik massivda saqlanadi

11	12	13	5	6

3-qadam: Endi tartiblangan kichik massivda 11 va 12 bo‘lgan ikkita element mavjud.13 va 5 bo‘lgan keyingi ikkita elementga o‘tiladi.

11	12	13	5	6

5 va 13 ikkalasi ham to‘g’ri joyda joylashmagan, shuning uchun ularninin o‘rinlarini almashtiramiz.

11	12	5	13	6

5 va 12 ikkalasi ham to‘g’ri joyda joylashmagan, shuning uchun ularninin o‘rinlarini almashtiramiz.

11	5	12	13	6

5 va 11 ikkalasi ham to‘g’ri joyda joylashmagan, shuning uchun ularninin o‘rinlarini almashtiramiz.

5	11	12	13	6

Endi 5 o‘zining saralangan joyiga keltirildi.

4-qadam: Endi saralangan subarrayda mavjud bo‘lgan elementlar 5, 11 va 12.Keyingi ikkita elementga o‘tish 13 va 6

5	11	12	13	6

Shubhasiz, ular saralanmagan, shuning uchun ikkalasi o‘rtasida almashtirish amalga oshiriladi

5	11	12	6	13

Endi 6 soni 12 dan kichik, shuning uchun yana almashtiring

5	11	6	12	13

Bu yerda ham 6 va 12 sonlari almashtiriladi.

5	6	11	12	13

Nihoyat, massiv to‘liq tartiblangan.

C++ dasturlash tilida insertion sort implementatsiyasi.

```
#include <iostream>
using namespace std;
// saralash uchun funktsiya
// insertion sort
void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

                  while (j >= 0 && arr[j] > key) {
             arr[j + 1] = arr[j];
             j = j - 1;
        }
        arr[j + 1] = key;
   }
}

// Massivni chop etish uchun yordamchi funktsiya
void printArray(int arr[], int n)
{
   int i;
   for (i = 0; i < n; i++)
        cout << arr[i] << " ";
   cout << endl;
}
```

```
// Bosh funktsiya
int main()
{
   int arr[] = { 12, 11, 13, 5, 6 };
   int N = sizeof(arr) / sizeof(arr[0]);

   insertionSort(arr, N);
   printArray(arr, N);

   return 0;
}
Dastur natijasi:
```

```
/tmp/Kib2adAnJ5.o
5 6 11 12 13
```

Vaqt murakkabligi:O(n^2);

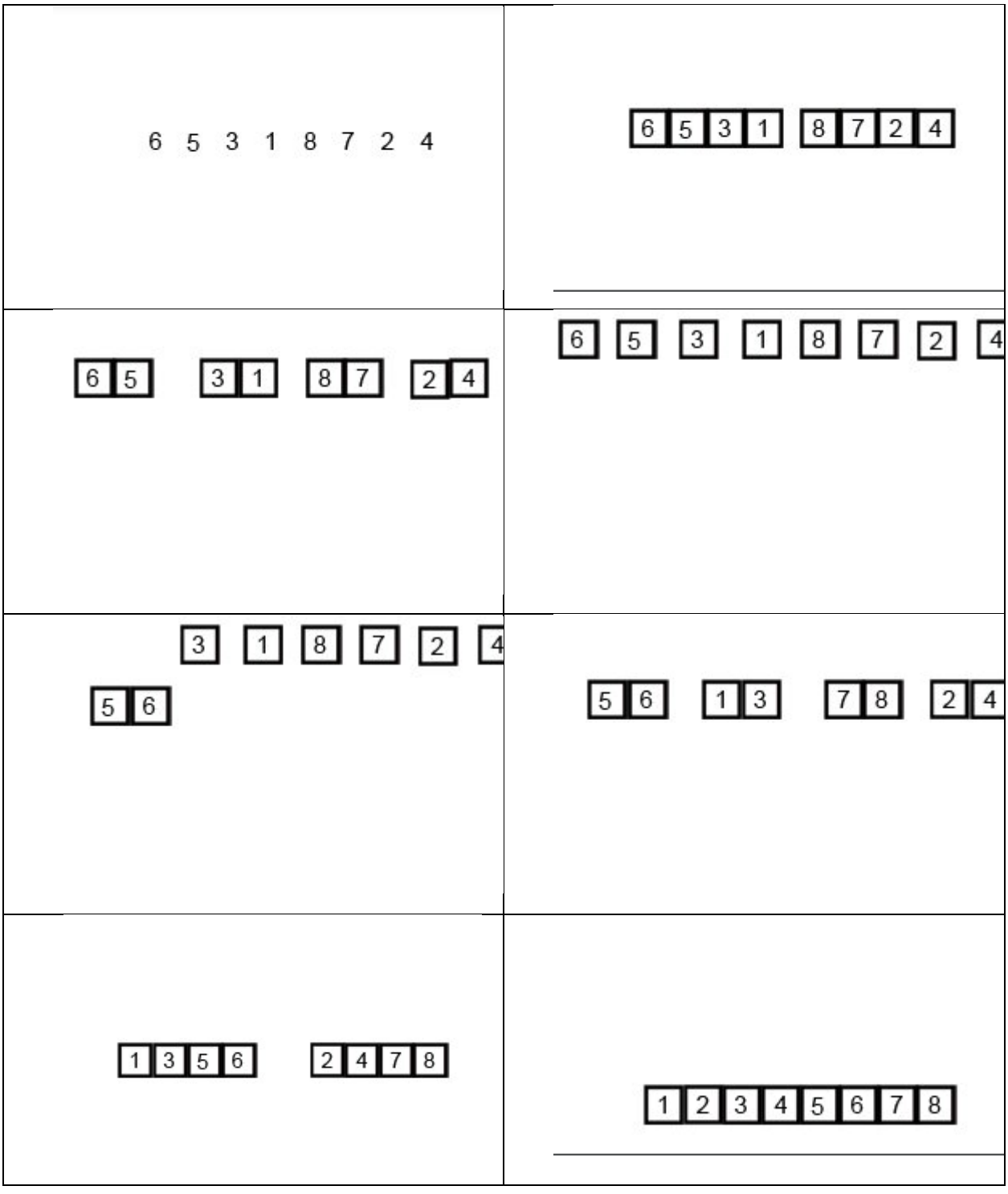
Xotira murakkabligi:O(1);

Insertion sort xususiyatlari

- Ushbu algoritm oddiy amalga oshiriladigan eng oddiy algoritmlardan biridir;
- Asosan, kiritish tartibi kichik ma’lumotlar qiymatlari uchun samarali;
- Insertion sort tabiatan moslashuvchan, ya’ni allaqachon qisman saralangan ma’lumotlar to‘plamlari uchun mos keladi;

1.4 Merge sort

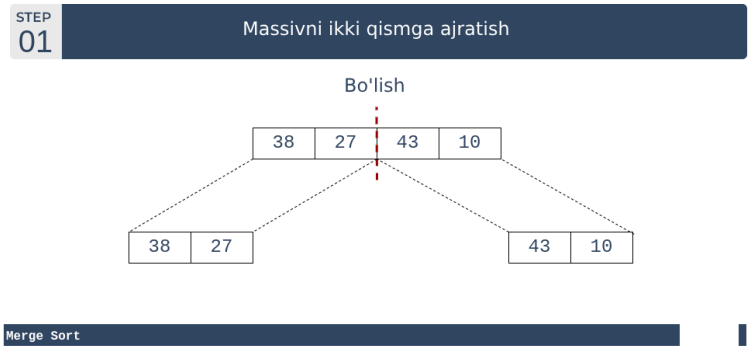
Merge sort-bu massivni kichikroq qismlarga bo‘lish, har bir kichik massivni saralash va so‘ngra tartiblangan kichik satrlarni birlashtirib, yakuniy tartiblangan massivni hosil qilish orqali ishlaydigan saralash algoritmi.



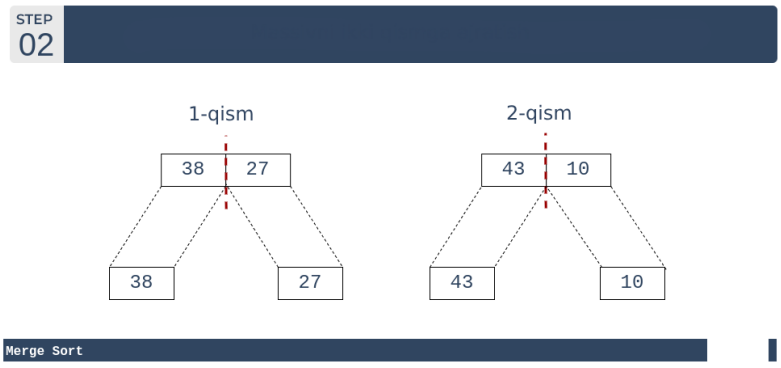
Merge sort - bu rekursiv algoritm bo‘lib, massiv elementlarini faqat bitta element qolmaguncha u massivni doimiy ravishda yarmiga bo‘ladi. Keyin tartiblangan

massivlar bitta tartiblangan massivga birlashtiriladi. Masalan: `array arr[] = {38, 27, 43, 10}`

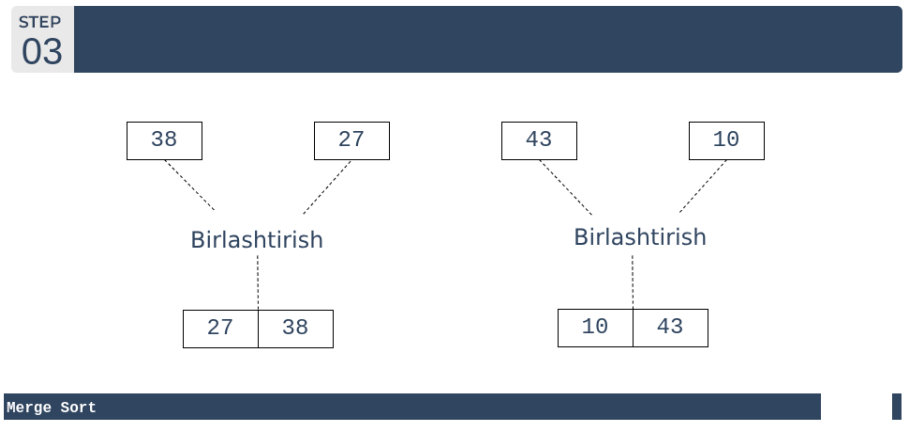
1-qadam: Dastlab massivni ikkita teng yarmiga bo‘ling.



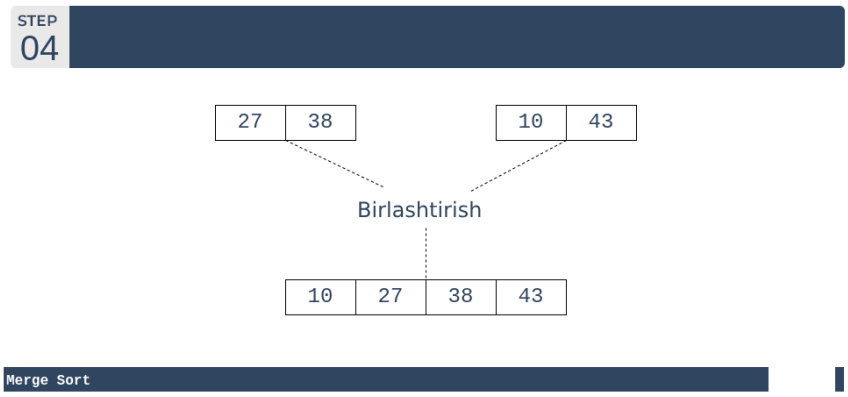
2-qadam: Ushbu kichik massivlar yana ikkiga bo‘linadi. Endi ular endi ajratib bo‘lmaydigan birlik uzunligi massiviga aylanadi va birlik uzunligi massivi saralanadi.



3-qadam: Ushbu tartiblangan sub-massivlar birlashadi va biz kattaroq tartiblangan sub-massivlarga ega bo‘lamiz.



4-qadam: Ushbu birlashish jarayoni kichikroq saralangan massivlardan to‘liq yig’ilguncha davom etadi.



C++dasturlash tilida merge sort implementatsiyasi.

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

// Ikkita massivni biriktirish
void merge(int array[], int const left, int const mid,
        int const right)
{
   int const subArrayOne = mid - left + 1;
   int const subArrayTwo = right - mid;

   // vaqtinchalik massivlarni yaratish
   auto *leftArray = new int[subArrayOne],
```

```
          *rightArray = new int[subArrayTwo];

     // ma'lumotlarni sub massivlarga nusxalash
     for (auto i = 0; i < subArrayOne; i++)
          leftArray[i] = array[left + i];
     for (auto j = 0; j < subArrayTwo; j++)
          rightArray[j] = array[mid + 1 + j];

     auto indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;
     int indexOfMergedArray = left;

     while (indexOfSubArrayOne < subArrayOne
          && indexOfSubArrayTwo < subArrayTwo) {
          if (leftArray[indexOfSubArrayOne]
               <= rightArray[indexOfSubArrayTwo]) {
               array[indexOfMergedArray]
                    = leftArray[indexOfSubArrayOne];
               indexOfSubArrayOne++;
          }
          else {
               array[indexOfMergedArray]
                    = rightArray[indexOfSubArrayTwo];
               indexOfSubArrayTwo++;
          }
          indexOfMergedArray++;
     }

     while (indexOfSubArrayOne < subArrayOne) {
          array[indexOfMergedArray]
               = leftArray[indexOfSubArrayOne];
          indexOfSubArrayOne++;
          indexOfMergedArray++;
     }

     while (indexOfSubArrayTwo < subArrayTwo) {
          array[indexOfMergedArray]
               = rightArray[indexOfSubArrayTwo];
          indexOfSubArrayTwo++;
          indexOfMergedArray++;
     }
     delete[] leftArray;
```

```
    delete[] rightArray;
}

void mergeSort(int array[], int const begin, int const end)
{
    if (begin >= end)
        return;

    int mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}

void printArray(int A[], int size)
{
    for (int i = 0; i < size; i++)
        cout << A[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "Berilgan massiv \n";
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    cout << "\nSaralangan massiv \n";
    printArray(arr, arr_size);
    return 0;
}
```

Dasturlash natijasi:

```
/tmp/Kib2adAnJ5.o
Berilgan massiv
12 11 13 5 6 7

Saralangan massiv
5 6 7 11 12 13
```

Birlashtirish orqali saralashning murakkabligini tahlil qilish:

Vaqt murakkabligi: O(nlog(n)), merge sort rekursiv algoritm bo‘lib, vaqt murakkabligi quyidagi takrorlanish nisbati bilan ifodalanishi mumkin.

$$T(n) = 2T(n/2) + \theta(n)$$

Xotira murakkabiligi: O(n), merge sortda barcha elementlar yordamchi qatorga ko‘chiriladi. Shunday qilib, birlashtirish uchun n yordamchi joy talab qilinadi.

Birlashtirish tartibining afzalliklari:

Barqarorlik: Merge sort-bu barqaror saralash algoritmi, ya’ni u kirish massividagi teng elementlarning nisbiy tartibini saqlaydi.

Kafolatlangan eng yomon ishlash: Merge sort O(nlogn) ning eng yomon vaqt murakkabligiga ega, ya’ni u hatto katta ma’lumotlar to‘plamlarida ham yaxshi ishlaydi.

Parallelizable: Merge sort-bu tabiiy ravishda parallellashtiriladigan algoritm, ya’ni bir nechta protsessor yoki iplardan foydalanish uchun uni osongina parallellashtirish mumkin.

Birlashtirish tartibining kamchiliklari:

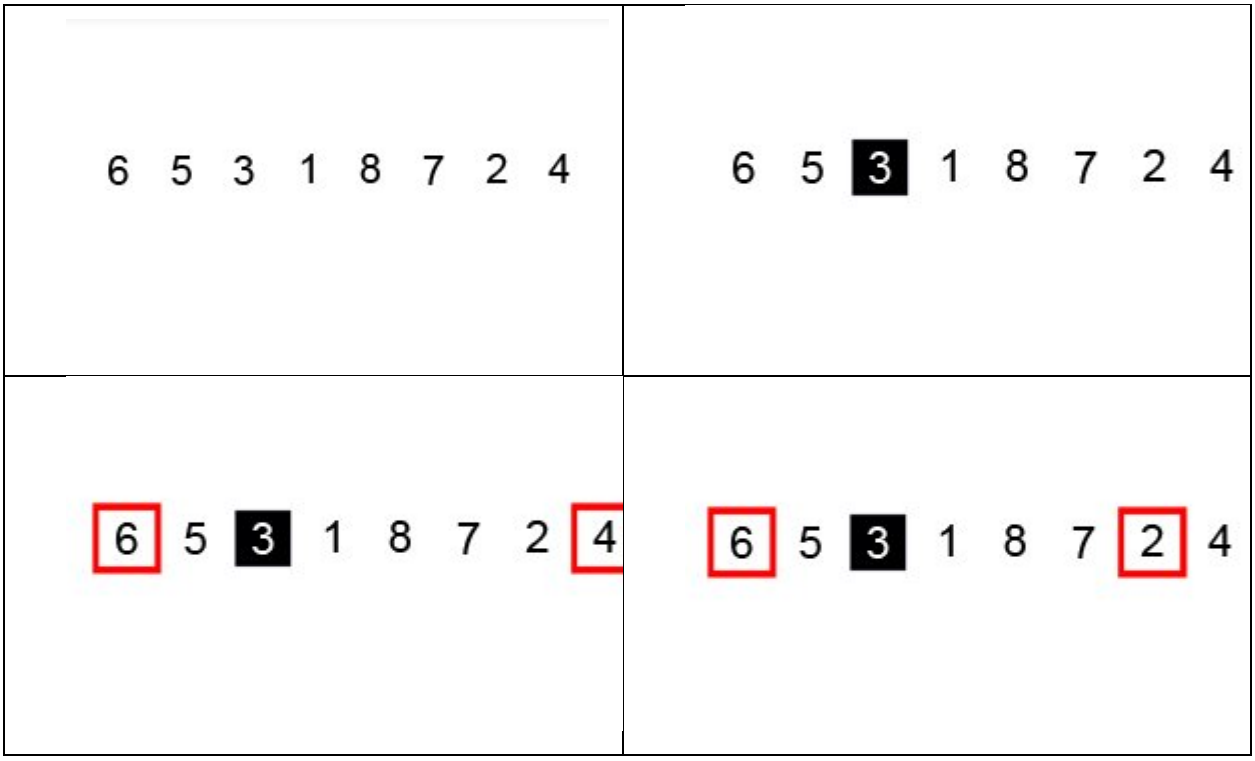
Xotiraviy murakkablik: Saralash jarayonida birlashtirilgan pastki massivlarni saqlash uchun qo‘shimcha xotirani talab qiladi.

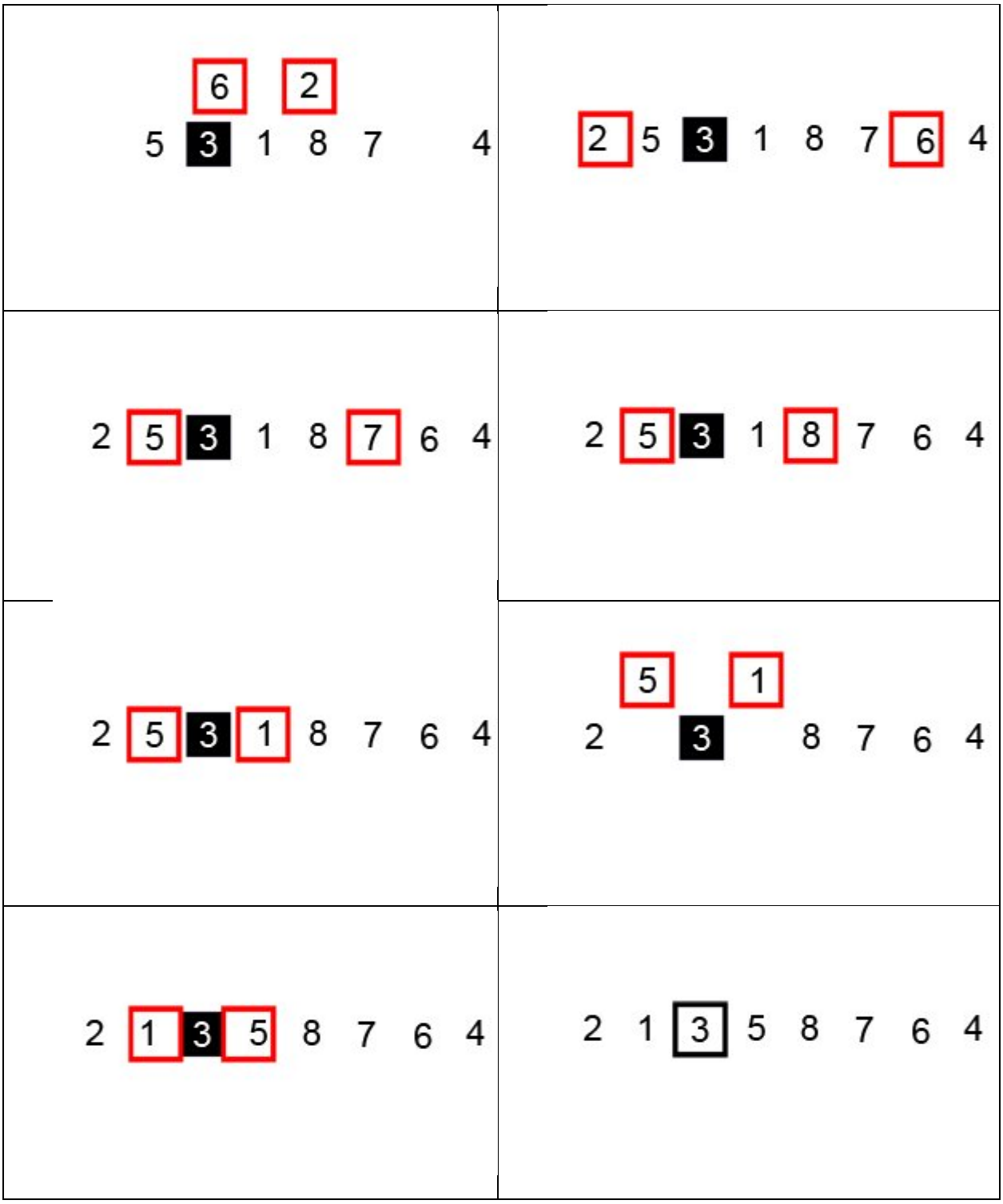
Joyida emas: birlashtirib saralash joyida saralash algoritmi emas, ya’ni saralangan ma’lumotlarni saqlash uchun qo‘shimcha xotira talab etiladi. Bu xotiradan foydalanish tashvish tug’diradigan ilovalarda kamchilik bo‘lishi mumkin.

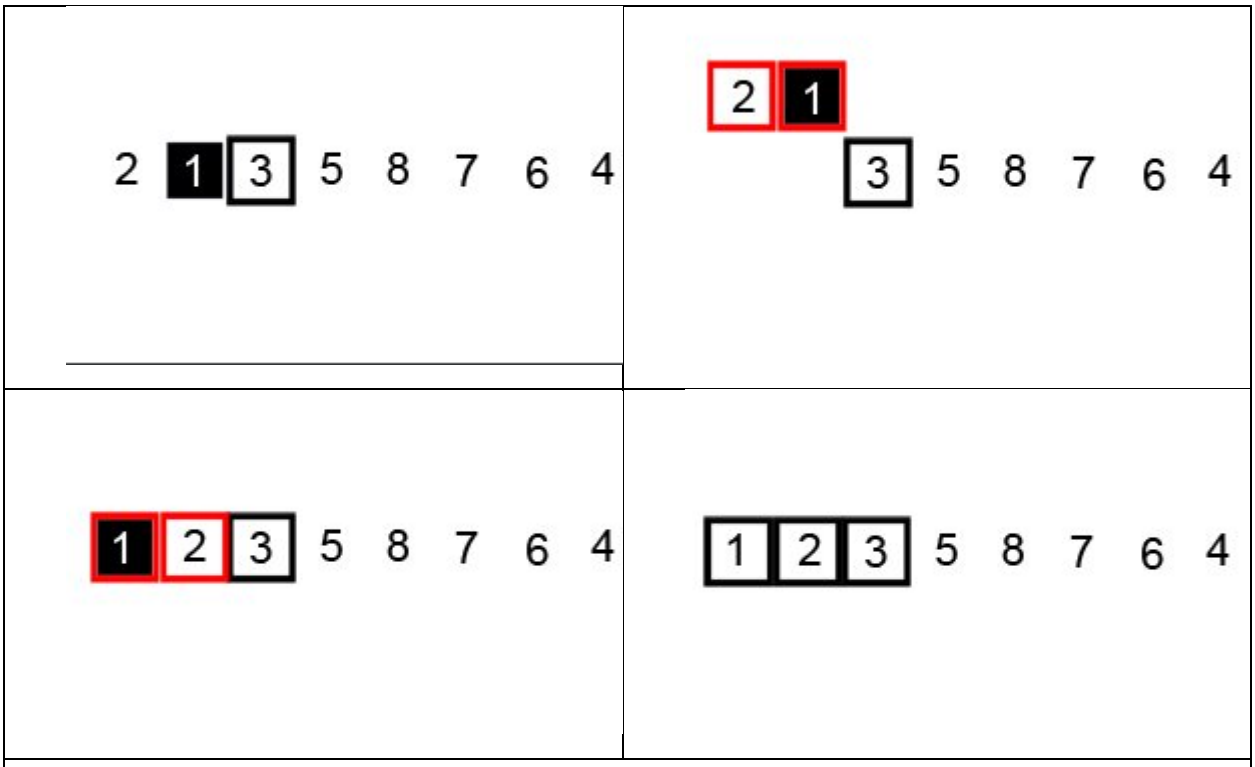
Kichik ma’lumotlar to‘plamlari uchun har doim ham maqbul emas: kichik ma’lumotlar to‘plamlari uchun Merge sort boshqa saralash algoritmlariga qaraganda yuqori vaqt murakkabligiga ega. Bu juda kichik ma’lumotlar to‘plamlari uchun sekinroq ishlashga olib kelishi mumkin.

1.5 Quick sort (tez saralash algoritmi)

Tez saralash-bu “bo‘linish va zabt etish” algoritmiga asoslangan saralash algoritmi bo‘lib, u elementni tayanch nuqtasi sifatida tanlaydi va ma’lum bir qatorni tanlangan tayanch nuqtasi atrofida sindirib, tayanch nuqtasini saralangan massivda to‘g’ri holatiga qo‘yadi.




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



Tez saralash algoritmi 1 va 2 - qadamlar

Pivotni tanlash (tayanch elementini)

Quicksort (tez saralash)-ni tezkor amalga oshirish uchun yaxshi pivotni (tayanch elementini) tanlash kerak. Pivotni tanlashning ba'zi usullari quyidagilar –

- Pivot tasodifiy bo‘lishi mumkin, ya’ni.berilgan qatordan tasodifiy pivotni tanlash.
- Pivot berilgan massivning eng chap elementining eng yoki o‘ng elementi tanlash mumkin.
- Pivot element sifatida medianni tanlash.

```
Algoritm
QUICKSORT (array A, start, end)
{
```

```
 1 if (start < end)
 2 {
3 p = partition(A, start, end)
4 QUICKSORT (A, start, p - 1)
5 QUICKSORT (A, p + 1, end)
6 }
}
```

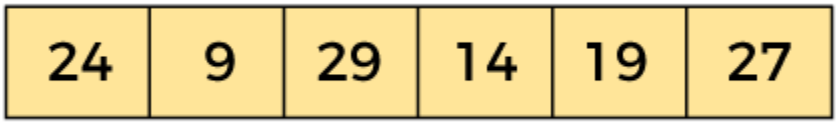
Bo‘lib tashlash algoritmi:

Algoritm

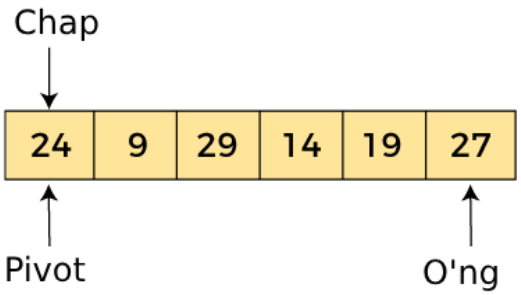
```
 PARTITION (array A, start, end)
{
 1 pivot ? A[end]
 2 i ? start-1
 3 for j ? start to end -1 {
 4 do if (A[j] < pivot) {
 5 then i ? i + 1
 6 swap A[i] with A[j]
```

```
  7   }}

  8 swap A[i+1] with A[end]

  9 return i+1

 }
```

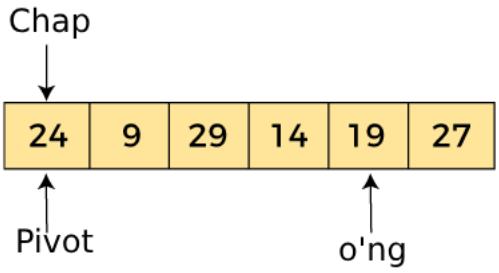
Keling, berilgan massivni tez saralash algoritmi bilan saralashni ko‘rib chiqamiz:



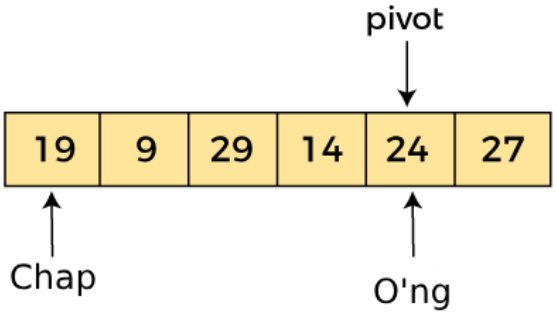
Berilgan massivda biz eng chap elementni pivot deb hisoblaymiz. Shunday qilib, bu holda, a[chap] = 24, a[o‘ng] = 27 va a[pivot] = 24. Pivot chapda bo‘lgani uchun algoritm o‘ngdan boshlanadi va chapga qarab harakatlanadi.



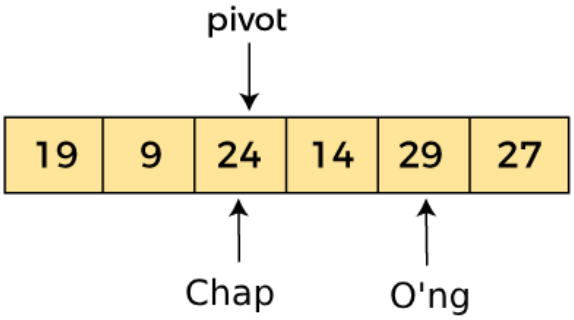
Endi, a[pivot] < a [o‘ng], shuning uchun algoritm chap tomonga bir pozitsiyani oldinga siljitadi, ya'ni. –



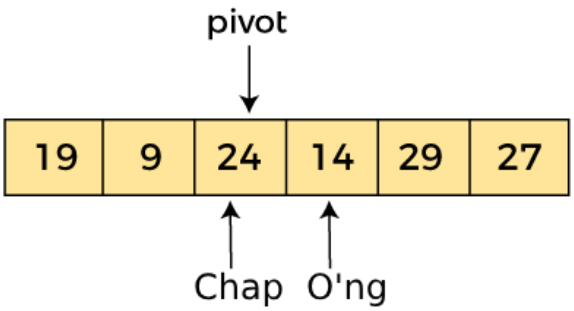
Endi, a[chap] = 24, a[o‘ng] = 19 va a[pivot] = 24. a[pivot] bilan a[o‘ng] o‘rinlarini almashtiradi , Chunki, a[pivot] > a [o‘ng].



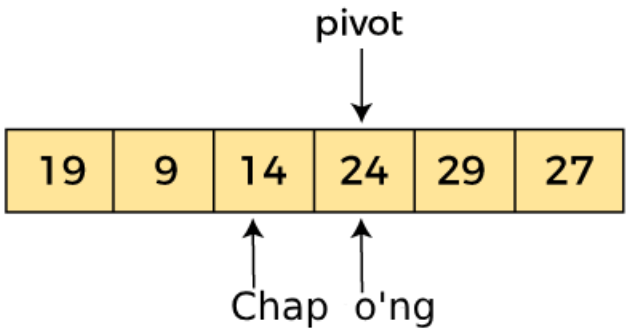
Endi, a[chap] = 9, a[o‘ng] = 24 va a[pivot] = 24. Sifatida[pivot] > a [chap], shuning uchun algoritm bitta pozitsiyani o‘ngga siljitadi –



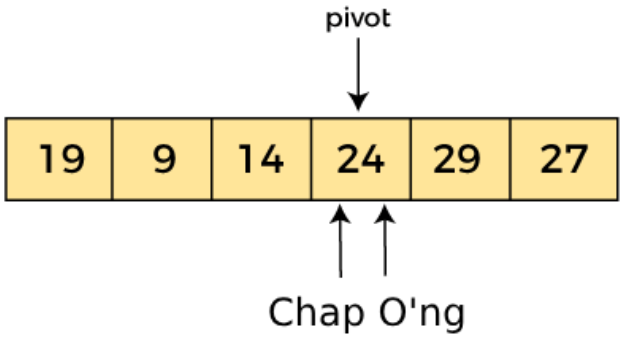
Endi, a[chap] = 24, a[o‘ng] = 29 va a[pivot] = 24. a[pivot] < a [o‘ng], shuning uchun algoritm bitta pozitsiyani chapga siljitadi -



Endi, a[pivot] = 24, a[chap] = 24 va a[o‘ng] = 14. a[Pivot] > a[o‘ng] sifatida, shuning uchun a[pivot] va a[o‘ng] ni almashtiring, endi pivot o‘ng tomonda, ya'ni. –

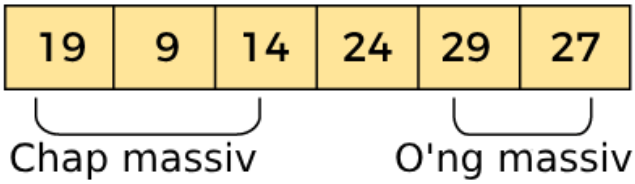


Endi, a[pivot] = 24, a[chap] = 14 va a[o‘ng] = 24. Pivot o‘ng tomonda, shuning uchun algoritm chapdan boshlanadi va o‘ngga o‘tadi.

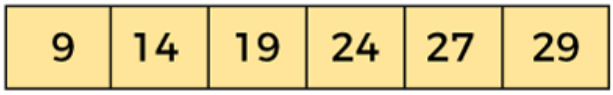


Endi, a[pivot] = 24, a[chap] = 24 va a [o‘ng] = 24. Shunday qilib, pivot, chap va o‘ng bir xil elementni ko‘rsatmoqda. Bu protsedurani tugatishni anglatadi. Markaz

element uning aniq holatda joylashtirilgan bo‘ladi Element 24. 24-elementning o‘ng tomoni bo‘lgan elementlar undan kattaroq va 24-elementning chap tomoni bo‘lgan elementlar undan kichikroq.



Endi, xuddi shunday tarzda, quicksort algoritmi chap va o'ng pastki massivlarga alohida qo‘llaniladi.



Tez saralash algoritmining murakkabligi.

Keling, eng yaxshi holatda, o‘rtacha holatda va eng yomon holatda quicksortning vaqt murakkabligini ko'rib chiqaylik.

Holat	Time Complexity
Eng yaxshi holat	O(n*logn)
O‘rtacha holat	O(n*logn)
Eng yomon holat	$O(n^2)$

Mavzu yuzasidan savollar

1. Nima uchun saralash algoritmlari muhim?
2. Ideal saralash algoritmi nima ekanligini tushuntiring?
3. Bubble sortning afzalliklari va kamchiliklari qanday?
4. Merge sortning afzalliklari va kamchiliklari qanday?
5. Quick sortning afzalliklari va kamchiliklari qanday?

6. Bubble sort algoritmininin dasturini tuzin.