ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
МЭДЭЭЛЭЛ, ХОЛБООНЫ ТЕХНОЛОГИЙН СУРГУУЛЬ

F.CS202
ОБЪЕКТ ХАНДЛАГАТ ПРОГРАМЧЛАЛ

Лекц №15

Эрэмбэлэлт, хайлтын энгийн алгоритмууд

2021 он

Агуулга

- Эрэмбэлэх алгоритмууд
  - Энгийн / Simple sort
  - Сонгох / Selection sort
  - Оруулах / Insertion Sort
  - Бөмбөлөг / Bubble Sort
  - Нэгтгэх / Merge Sort
  - Хурдан / Quick Sort
- Хайлтын алгоритмууд
  - Шугаман хайлт
  - 2-тын хайлт

Хайлт хийх

- Өгөгдлийн цуглуулгатай ажиллаж байх үед хэрэгтэй өгөгдлөө богино хугацаанд хайж олох хэрэгцээ гардаг.
- Даалгавар:

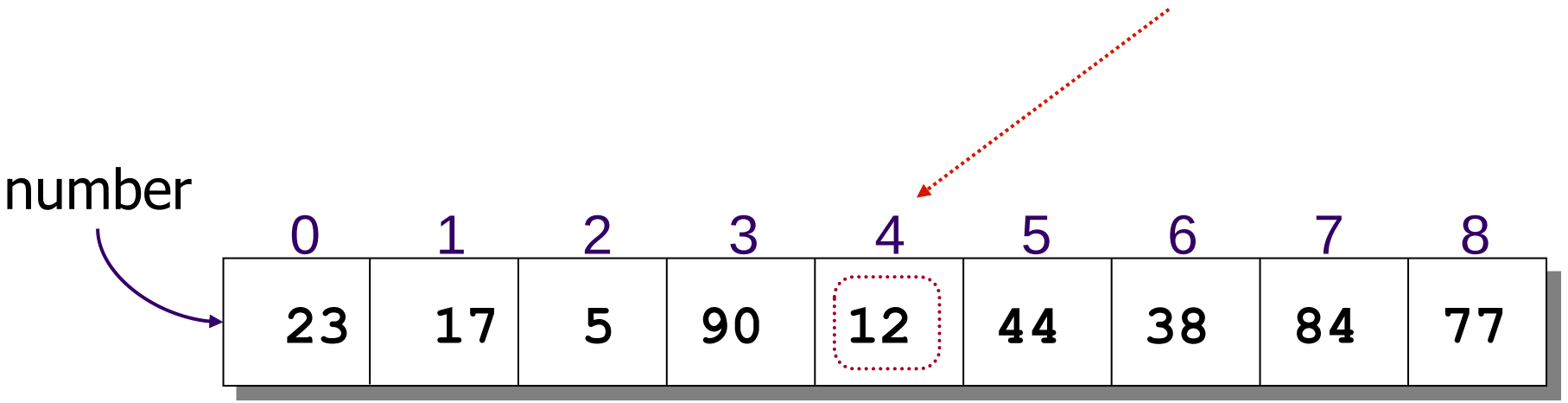
  Х утга өгөгдөхөд, хэрэв Х нь массивт байвал түүний массив дахь индексийг буцаах, . Үгүй бол, NOT_FOUND (-1) буцаана. Массивт давхардсан утга байхгүй гэж үзье.

- Алгоритмуудын гүйцэтгэлийг үнэлэхдээ тэдгээрийн харьцуулалт хийж буй тоог авч үзнэ.
  - Хайж буй өгөгдлийг олоход хамгийн цөөн тооны харьцуулалт хийж буй алгоритмыг хамгийн төгс гэж үзнэ.
  - Гүйцэтгэлийн үнэлгээг хийхдээ хайлт амжилттай болон амжилтгүй байх тохиолдлуудыг аль алиныг нь авч үзнэ.

Хайлтын үр дүн

Хайлт амжилтгүй: `search( 45 )` → **NOT_FOUND**

Хайлт амжилттай: `search( 12 )` → **4**



Шугаман хайлт

- Массивын эхнээс төгсгөл хүртэл шугаман байдлаар хайлт хийнэ.

```
public int linearSearch ( int[] number, int searchValue ) {
    int   loc = 0;
    while (loc < number.length     &&  number[loc] != searchValue) {
        loc++;
    }
    if (loc == number.length) { //олдоогүй бол
        return NOT_FOUND;
    } else {
        return loc;    //олдсон бол индексийг буцаана
    }
}
```

Шугаман хайлтын ажиллагаа

- Амжилттай болон амжилтгүй тохиолдлуудыг тус бүр тооцно.
- Хайж буй утгыг массивын хэдэн элементтэй харьцуулсаныг тоолно.

- Хайлт амжилттай
  - Хамгийн сайн – 1 харьцуулалт
  - Хамгийн муу – N харьцуулалт (N – массивын хэмжээ)

- Хайлт амжилтгүй
  - Хамгийн сайн =
    Хамгийн муу – N харьцуулалт

Эрэмбэлэх алгоритмууд

- Энгийн / Simple sort
- Сонгох / Selection sort
- Оруулах / Insertion Sort
- Бөмбөлөг / Bubble Sort
- Нэгтгэх / Merge Sort
- Хурдан / Quick Sort

Энгийн / Simple sort

- Эрэмбэлсэн, эрэмбэлээгүй гэсэн 2 хэсэгт хуваана,
- Эхнээс нь эхлэн хамгийн багуудыг хамгийн эхэнд байрлуулна.

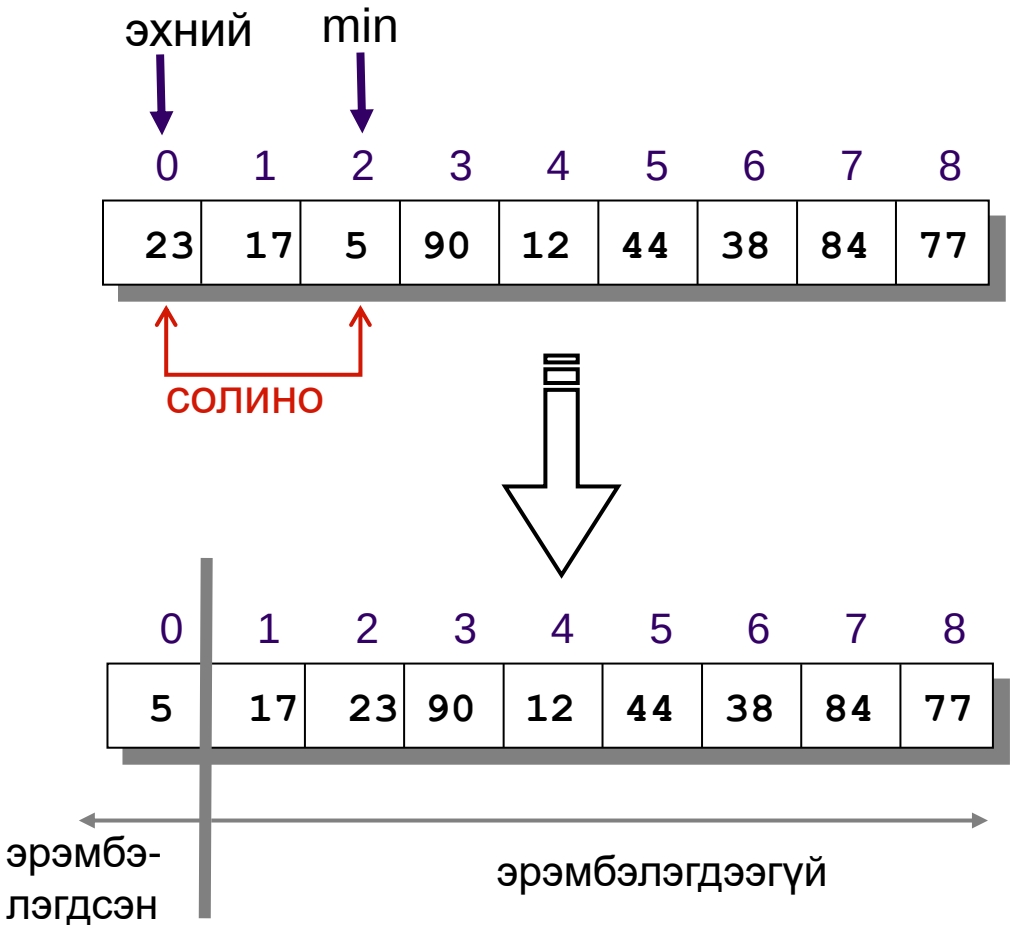
```
private static void simpleSort(int a[])
{
    for (int i = 0; i < a.length-1; i++)
        for (int j = i+1; j < a.length; j++)
            if (a[i] > a[j])
                swap(a,i,j);
}
private static void swap(int c[], int a, int b)
{
    int tmp = c[a];
    c[a] = c[b];
    c[b] = tmp;
}
```

Сонгох / Selection sort

- Эрэмбэлсэн, эрэмбэлээгүй гэсэн 2 хэсэгт хуваана,
- Эрэмбэлээгүй хэсгээс хамгийн бага утгыг олж, өмнөх элементтэй нь солино.

```
private static void selectionSort(int a[])
{
    for (int i = 0; i < a.length-1; i++) {
        int min = i;
        for (int j = i+1; j < a.length; j++)
            if (a[j] < a[min]) min = j;
        swap(a,i,min);
    }
}
```

Сонгох - Selection sort



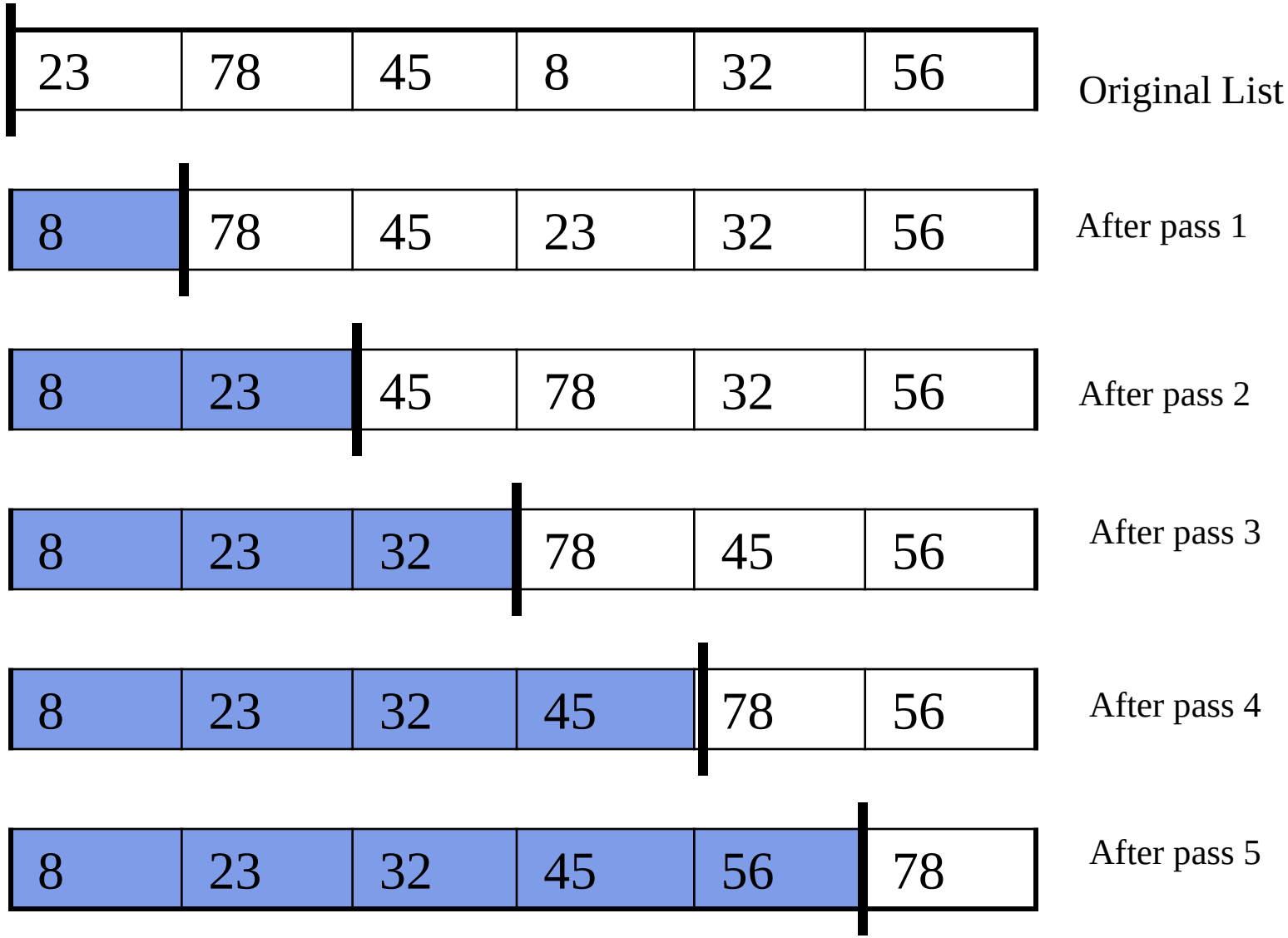
Эхний солилтын дараа. ▶▶▶

#	0	1	2	3	4	5	6	7	8
1	5	17	23	90	12	44	38	84	77

эрэмбэлэгдсэн

2	5	12	23	90	17	44	38	84	77
3	5	12	17	90	23	44	38	84	77
●●●									
7	5	12	17	23	38	44	77	84	90
8	5	12	17	23	38	44	77	84	90

Эрэмбэлэгдсэн **Эрэмбэлэгдээгүй**



Оруулах / Insertion Sort

- Цөөн элементтэй байхад ашиглах энгийн алгоритм
- Эрэмбэлсэн, эрэмбэлээгүй гэсэн 2 хэсэгт хуваана
- Эрэмбэлэгдээгүй хэсгийн эхний элементийг авч, эрэмбэлэгдсэн хэсгийн тохирох газар байрлуулна.

```
private static void insertionSort(int a[])
{
    for (int i = 1; i < a.length; i++)
    {
        int tmp = a[i];
        int j=i;
        for (;j>0 && tmp < a[j-1]; j--)
            a[j] = a[j-1];
        a[j] = tmp;
    }
}
```

Sorted **Unsorted**

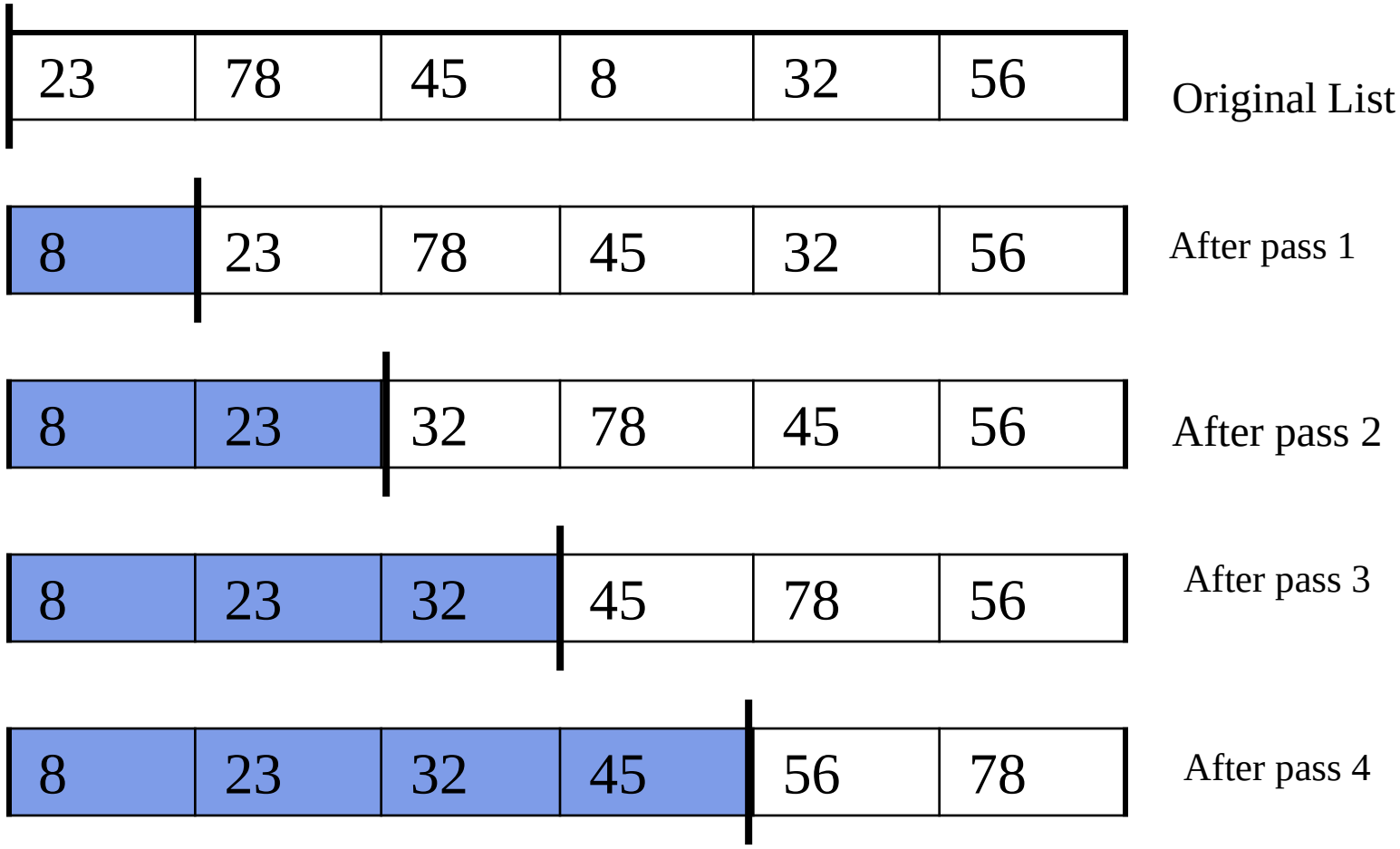
23	78	45	8	32	56	Original List
23	78	45	8	32	56	After pass 1
23	45	78	8	32	56	After pass 2
8	23	45	78	32	56	After pass 3
8	23	32	45	78	56	After pass 4
8	23	32	45	56	78	After pass 5

Бөмбөлөг / Bubble Sort

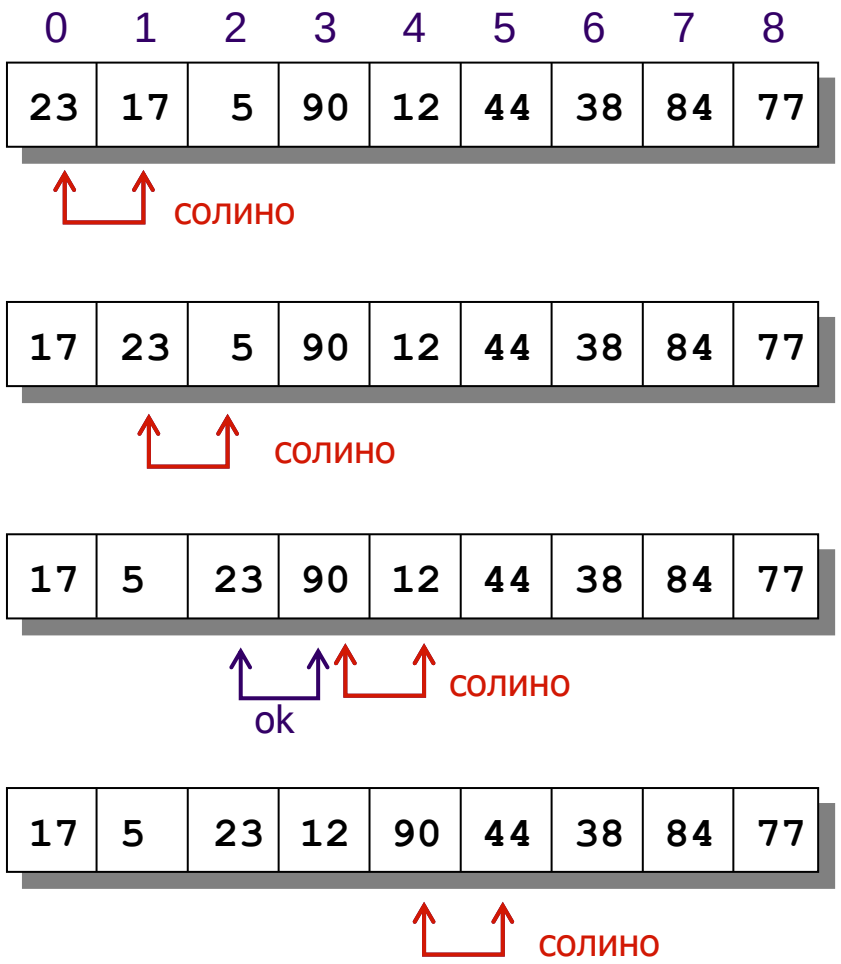
- Эрэмбэлсэн, эрэмбэлээгүй гэсэн 2 хэсэгт хуваана,
- Эрэмбэлэгдээгүй хэсгийн хамгийн бага элементийг эрэмбэлэгдсэн хэсэгт байрлуулна.
- Нэг элементээр урагшилсны дараа эрэмбэлэгдсэн элементүүдийн тоо нэгээр нэмэгдэж, эрэмбэлэгдээгүй элементийн тоо нэгээр багасна. Эрэмбэлэгдээгүй хэсгээс эрэмбэлсэн хэсэг рүү элемент шилжих замаар эрэмбэлэгдэнэ.

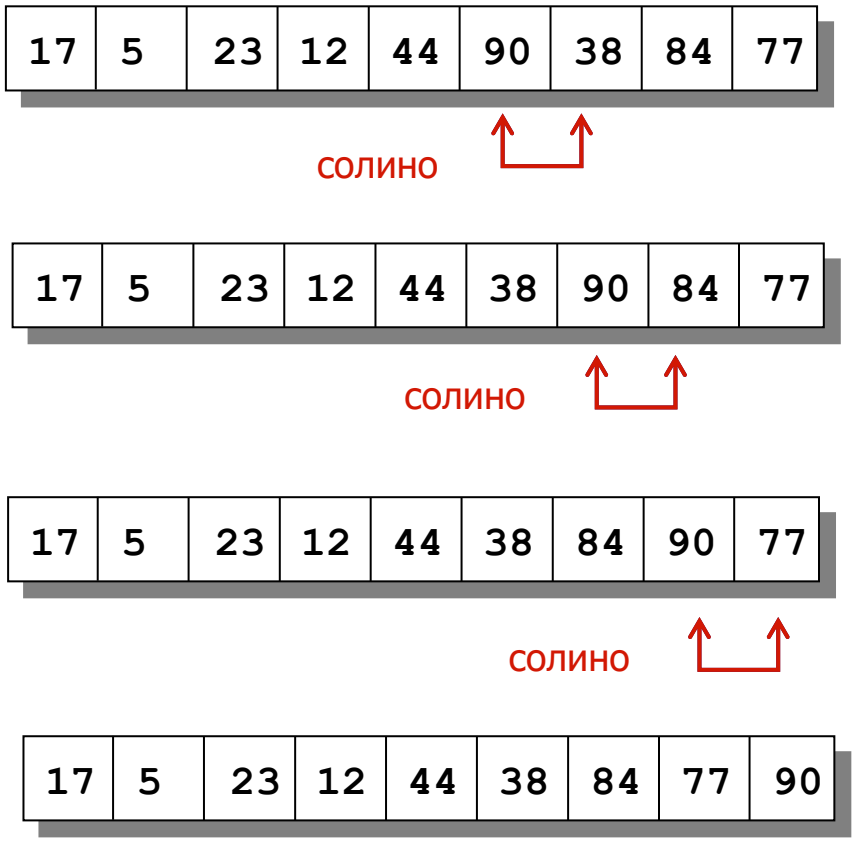
```
private static void bubleSort(int a[])  {
    Boolean sorted = false;
    for (int i = 0; i < a.length-1 && !sorted; i++) {
        sorted = true;
        for (int j= a.length-1; j > i; j--)
            if (a[j-1] > a[j]) {
                swap(a,j,j-1);
                sorted = false;      // signal exchange
            }
    }
}
```

Bubble Sort


23 78 45 8 32 56
Original List
8 23 78 45 32 56
After pass 1
8 23 32 78 45 56
After pass 2
8 23 32 45 78 56
After pass 3
8 23 32 45 56 78
After pass 4

Bubble Sort алгоритмын нэг алхам





Хамгийн их утга 90 нь жагсаалтын сүүлд байрлана.

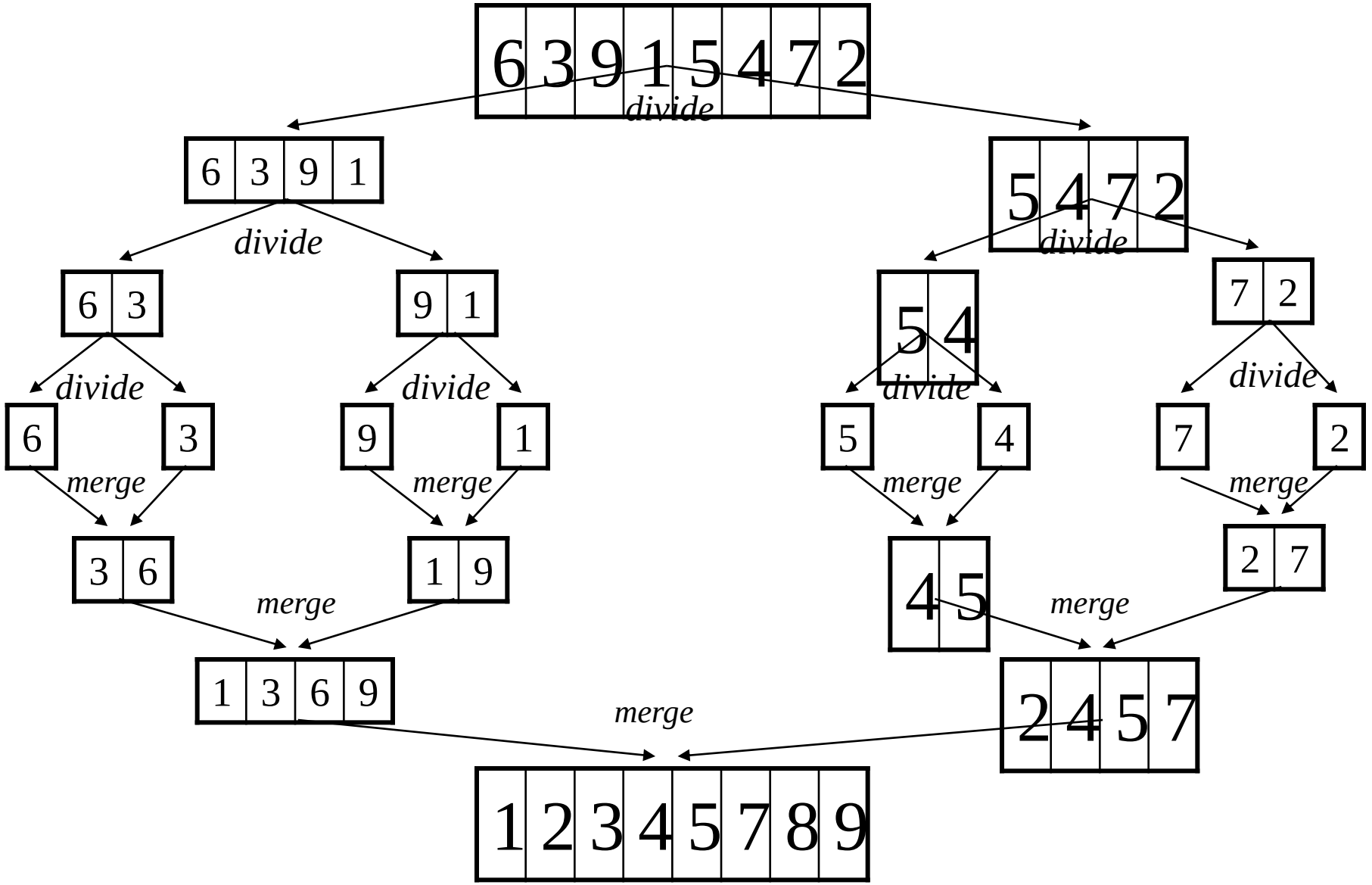
Bubble Sort Routine

```
public void bubbleSort(int[] number) {

    int           temp, bottom, i;
    boolean       exchanged = true;

    bottom = number.length - 2;

    while (exchanged)  {
        exchanged = false;

        for (i = 0; i <= bottom; i++) {
           if (number[i] > number[i+1]) {
                temp = number[i];     //солино
                number[i] = number[i+1];
                number[i+1] = temp;

                exchanged = true; //солих үйлдэл хийгдсэн
           }
        }
        bottom--;
    }
}
```

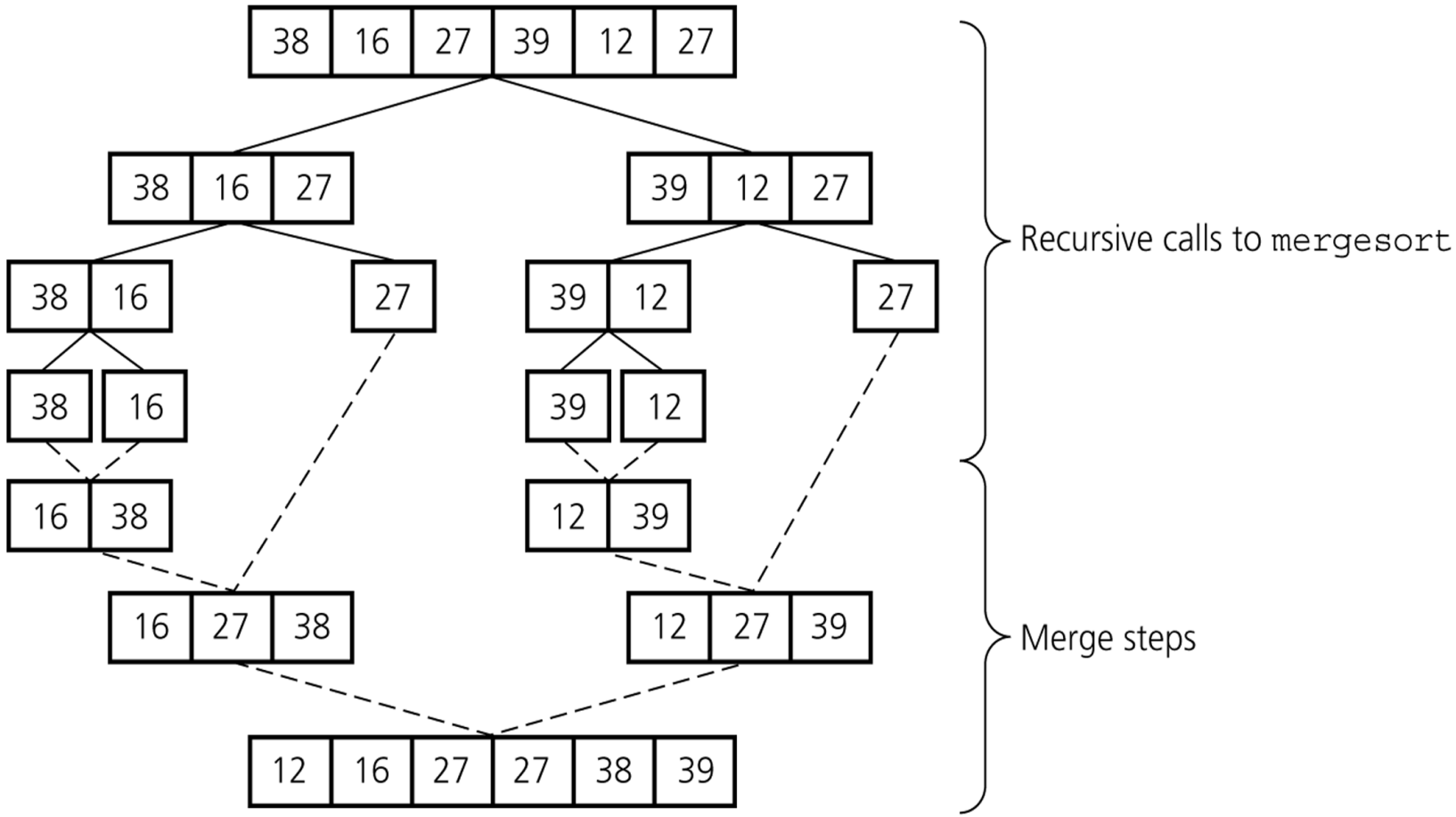
Нэгтгэх / Merge Sort

- Энэ эрэмбэлэлтийн алгоритм нь хуваагаад, нэгтгэх арга хэрэглэдэг эрэмбэлэлтийн аргын нэг бөгөөд рекурс алгоритм юм. Үүнд:
- 2 хувааж
- Хэсэг тус бүрийн тусад нь эрэмбэлнэ
- Дараа нь эрэмбэлсэн нэг массивт нэгтгэнэ
- Их хэмжээний өгөгдлийг эрэмбэлэхэд тохиромжтой

Merge sort - Жишээ



Merge sort – Жишээ 2



```
private static int[] mergeSort(int[] list)
{
    //жагсаалт хоосон бол юу ч хийхгүй
    if (list.length <= 1) {
        return list;
    }

    //Массивыг хоёр тэнцүү хэсэгт хуваах
    int[] first = new int[list.length / 2];
    int[] second = new int[list.length - first.length];
    System.arraycopy(list, 0, first, 0, first.length);
    System.arraycopy(list, first.length, second, 0, second.length);

    //Хуваасан хоёр хэсгээ тус бүр рекурсээр эрэмбэлэх
    mergeSort(first);
    mergeSort(second);

    //эх массивыг дарж хоёр хэсгээ нэгтгэж массив болгоно
    merge(first, second, list);
    return list;
}
```

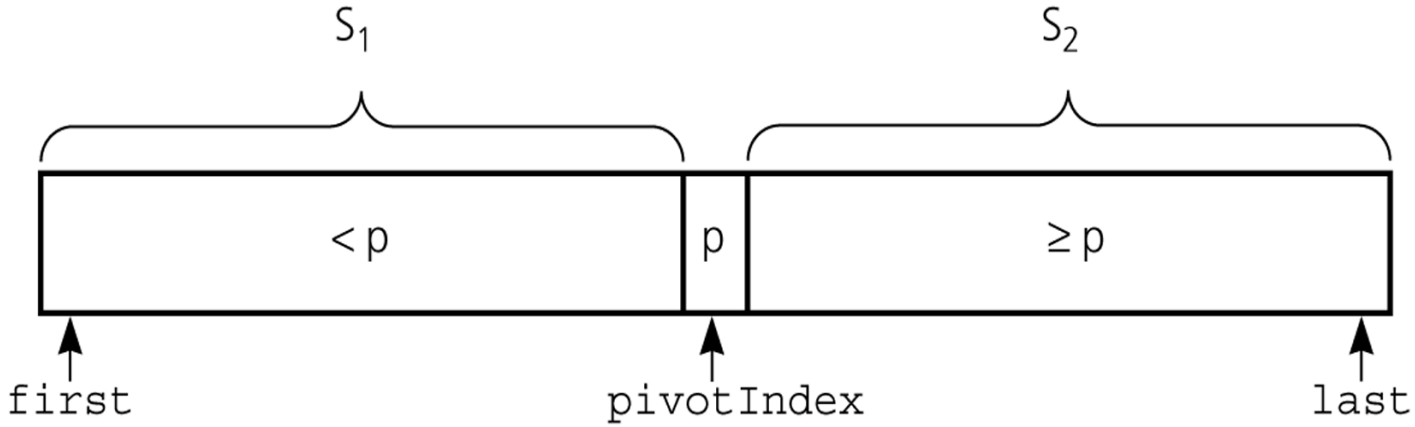
```
private static void merge(int[] first, int[] second, int[] result)
{
    int iFirst = 0;
    int iSecond = 0;
    int iMerged = 0;
    while (iFirst < first.length && iSecond < second.length)
    {
        if (first[iFirst] < second[iSecond]){
            result[iMerged] = first[iFirst];
            iFirst++;
        }
        else{
            result[iMerged] = second[iSecond];
            iSecond++;
        }
        iMerged++;
    }
    System.arraycopy(first, iFirst, result, iMerged, first.length - iFirst);
    System.arraycopy(second, iSecond, result, iMerged, second.length - iSecond);
}
```

Хурдан / Quick Sort

- Mergesort шиг хувааж, нэгтгэх арга хэрэглэнэ.
- Гэхдээ рекурсээр дуудахаасаа өмнө эрэмбэлэлт хийнэ.
- Дараах байдлаар ажиллана. Үүнд:
  - Эхлээд, массивыг 2 хэсэгт хуваана.
  - Дараа нь, хэсгүүдийг тусад нь эрэмбэлнэ.
  - Эцэст нь эрэмбэлэгдсэн хэсгүүдийг энгийн залгах аргаар нэгтгэнэ.

Хуваах

- Хуваахдаа массив дахь зөв байрлалыг сонгож хуваана. Түүнийг pivot гэнэ.




```
//    S1 = theArray[first..pivotIndex-1] < pivot
//    theArray[pivotIndex] == pivot
//    S2 = theArray[pivotIndex+1..last] >= pivot
```

Хуваах

```
private static int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // бага элементийн индекс
    for (int j=low; j<high; j++)
    {
        // хэрэв тухайн элемент нь pivot-с бага эсвэл тэнцүү бол
        if (arr[j] <= pivot)
        {
            i++;
            swap(arr,i,j);

        }
    }
    swap(arr,i+1, high);
    return i+1;
}
```

Нэгтгэх

```
private static void sort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi хуваах индекс, arr[pi] нь зөв байрлал */
        int pi = partition(arr, low, high);

        // хуваалтын хэсэг бүрийг рекурсээр эрэмбэлнэ
        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}
```

Эрэмбэлэлтийн алгоритмуудын харьцуулалт

	Удаандаа	Дунджаар
Selection sort	n^2	n^2
Bubble sort	n^2	n^2
Insertion sort	n^2	n^2
Merge sort	n*log n	n*log n
Quick sort	n^2	n*log n

Шугаман хайлт

```
private static int simpleSearch(int a[], int val)
{
    for(int i = 0; i< a.length; i++)
        if (a[i] == val)
            return i;
    return -1;
}

public static void main()
{
    int b[] = {23, 78, 45, 8, 32, 56};
    int ind = simpleSearch(b, 32);
    if(ind == -1)
        System.out.print("not found");
    else
        System.out.print(ind);
}
```

2-тын хайлт

- Эрэмбэлэгдсэн массив байх ёстой.
- Хуваагаад нэгтгэх арга хэрэглэдэг.
- Голын элемент хайж байгаа элементээс их бол дээд хагасаас, үгүй бол доод хагасаас хайдаг рекурс алгоритм юм.

```
public static int binarySearch(int a[], int first, int last, int key){
  if (last>=first){
    int mid = first + (last - first)/2;
    if (a[mid] == key)
      return mid;
    if (a[mid] > key)
      return binarySearch(a, first, mid-1, key);
    else
      return binarySearch(a, mid+1, last, key);
  }
  return -1;
}
public static void main()
{
  int b[] = {23, 78, 45, 8, 32, 56};
  sort(b, 0, b.length-1);
  int ind = binarySearch(b, 0, b.length-1, 32);
  if(ind == -1)
    System.out.print("not found");
  else
    System.out.print(ind);
}
```

Давталт ашигласан жишээ

```
public static void binarySearch(int a[], int first, int last, int key){
   int mid = (first + last)/2;
   while( first <= last ){
      if ( a[mid] < key ){
        first = mid + 1;
      }else if ( a[mid] == key ){
        System.out.println("Element is found at index: " + mid);
        break;
      }else{
         last = mid - 1;
      }
      mid = (first + last)/2;
   }
   if ( first > last ){
      System.out.println("Element is not found!");
   }
 }
```

ДҮГНЭЛТ

- Массивын элементүүдтэй ажиллаж байх үед шаардлагатай элементийг маш богино хугацаанд хайж олох хэрэгцээ гардаг.
- Хайлтыг эрэмбэлэгдсэн массив дээр хийх нь илүү хурдан.
- Хайлт болон эрэмбэлэлт нь програмчлалын хоёр чухал үйлдэл.
  - Энгийн / Simple sort
  - Сонгох / Selection sort
  - Оруулах / Insertion Sort
  - Бөмбөлөг / Bubble Sort
  - Нэгтгэх / Merge Sort
  - Хурдан / Quick Sort
- Хайлтын алгоритмууд
  - Шугаман хайлт – шугаман дарааллаар хайлт хийнэ
  - 2-тын хайлт – эрэмбэлэгдсэн жагсаалтын голоос эхлэн харьцуулалт хийх замаар хайлт хийнэ