Sorting

توصيف عمومي مسئله

n عنصر با كليدهاي $a_1, a_2, \ldots, a_n$ و رابطه $<$، $>$، يا $=$ بين هر دو عنصر داده شده، يك جايگشت از عناصر بايد پيدا شود به طوري كه

$$a_{\pi 1} \leq a_{\pi 2} \leq \ldots \leq a_{\pi n}$$

دسته بندي الگوريتمهاي مرتب سازي

- **نحوه مرتب كردن داده ها:**
  - الگوريتم هاي مبتني بر مقايسه (comparison sort)
    بر اساس مقايسه كليدهاي عناصر عمل مي كنند و اطلاعات ديگري از داده ورودي ندارند
  - الگوريتم هاي غير مبتني بر مقايسه (non-comparison)
    از اطلاعاتي كه از قبل در باره كليدها وجود دارد استفاده مي شود مقايسه كليدها كم مي شود يا حذف مي شود

- **حفظ ترتيب نسبي عناصر**
  - الگوريتمهاي متعادل(stable):

  - الگوريتمهای نامتعادل (unstable) :

دسته بندي الگوريتمهاي مرتب سازي

- **نحوه مرتب كردن داده ها:**
  - الگوريتم هاي مبتني بر مقايسه (comparison sort)
    بر اساس مقايسه كليدهاي عناصر عمل مي كنند و اطلاعات ديگري از داده ورودي ندارند
  - الگوريتم هاي غير مبتني بر مقايسه (non-comparison)
    از اطلاعاتي كه از قبل در باره كليدها وجود دارد استفاده مي شود مقايسه كليدها كم مي شود يا حذف مي شود

- **حفظ ترتيب نسبي عناصر**
  - الگوريتمهاي پايدار(stable): ترتيب نسبي عناصر با كليدهاي يكسان قبل و بعد از اجراي الگوريتم مرتب سازي **يكي است**
  - الگوريتمهای ناپايدار (unstable) : ترتيب نسبي عناصر با كليدهاي يكسان قبل و بعد از اجراي الگوريتم مرتب سازي **لزوماً يكي نيست**

دسته بندي الگوريتمهاي مرتب سازي

- **موقعيت داده ها (محل ذخيره سازی) در زمان مرتب كردن:**
  - الگوريتم هاي مرتب ساز داخلي (internal sorting)

همه داده ها در هنگام مرتب شدن در حافظه هستند و دسترسي به آنها سريع است

  - الگوريتم هاي مرتب ساز خارجي (external sorting)

همه داده ها در هنگام مرتب شدن در حافظه نيستند و دسترسي به آنها كند است. پس تعداد دسترسي به عناصر خيلي اهميت دارد.

پيچيدگي الگوريتمهاي مرتب سازي

الگوريتمهاي مرتب سازي مبتني بر مقايسه $\Omega(n \lg n)$ هستند.

الگوريتمهاي مرتب سازي غير مبتني بر مقايسه $\Omega(n)$ هستند.

چند الگوريتم مرتب سازي مبتني بر مقايسه

- Insertion Sort
- Selection Sort
- Bubble Sort
- Merge Sort
- Quick Sort
- Heap Sort

Insertion Sort

عناصر، يكي يكي به قسمت مرتب شده اضافه مي شوند و در محل مناسب قرار مي گيرند

InsertionSort (A [0..*n*–1]):

for *j* ←1 **to n-1 do**

key ← A[*j*]

i ← *j-1*

while *i* >= 0 **and** A[*i*] > *key* **do**

A[*i+1*] ← A[*i*]

i ← *i* – 1

A[*i+1*] ← *key*

0	*1*	*2*	*3*	*4*
8	2	4	9	7
2	8	4	9	7
2	4	8	9	7
2	4	8	9	7
2	4	7	8	9

Stable يا Unstable ؟

پيچيدگي الگوريتم:

بدترين حالت و حالت متوسط: $O(n^2)$

بهترين حالت : $O(n)$

Selection Sort

در i امین مرحله، i امین کوچکترین عنصر انتخاب مي شود و با عنصر بعد از محدوده مرتب شده تعويض مي شود.

```
SelectionSort(A [0..n–1]):
    for i ←0  to n-2 do
        minindex ← i
        minkey ← A[i]
        for j ←i+1  to n-1 do
            if A[j] < minkey then
                minkey ← A[j]
                minindex ← j
        swap(A[i], A[minkey])
```

0	1	2	3	4	5
7	4	6	5	1	2
1	4	6	5	7	2
1	2	6	5	7	4
1	2	4	5	7	6
1	2	4	5	7	6
1	2	4	5	6	7

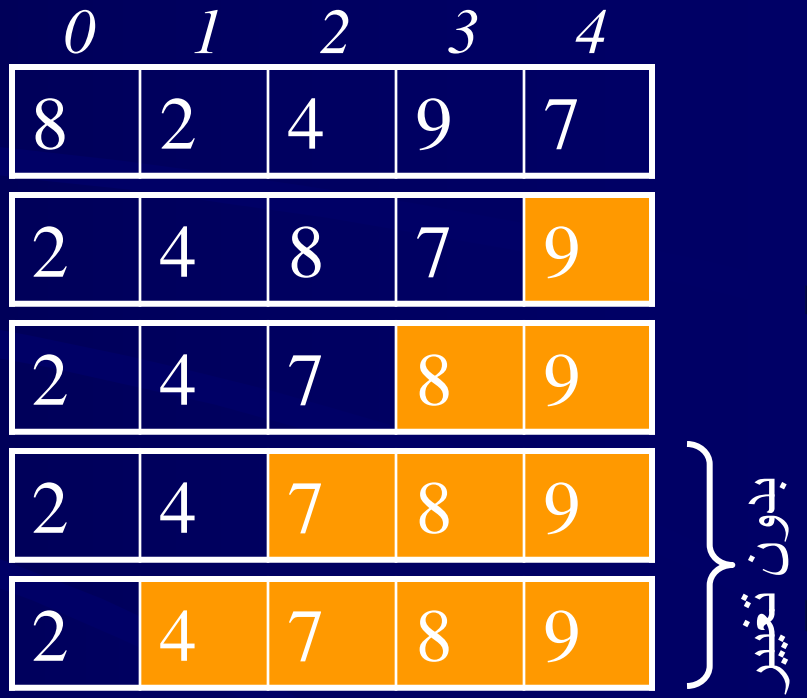
؟ Unstable يا Stable

پيچيدگي الگوريتم: $\Sigma\Sigma c = O(n^2)$

Bubble Sort

در هر تکرار (از n-1 تکرار)، هر دو عنصر مجاور با هم مقايسه مي شوند و احتمالا تعويض مي شوند

BubbleSort(A[0..*n*–1]):
    for *i* **from** n-1 **downto** *1* **do**
        for *j* **from** 1 **to** *i* **do**
            if A[*j*] < A[*j*–1] **then**
                A[*j*] ↔ A[*j*–1]



Stable يا Unstable ؟

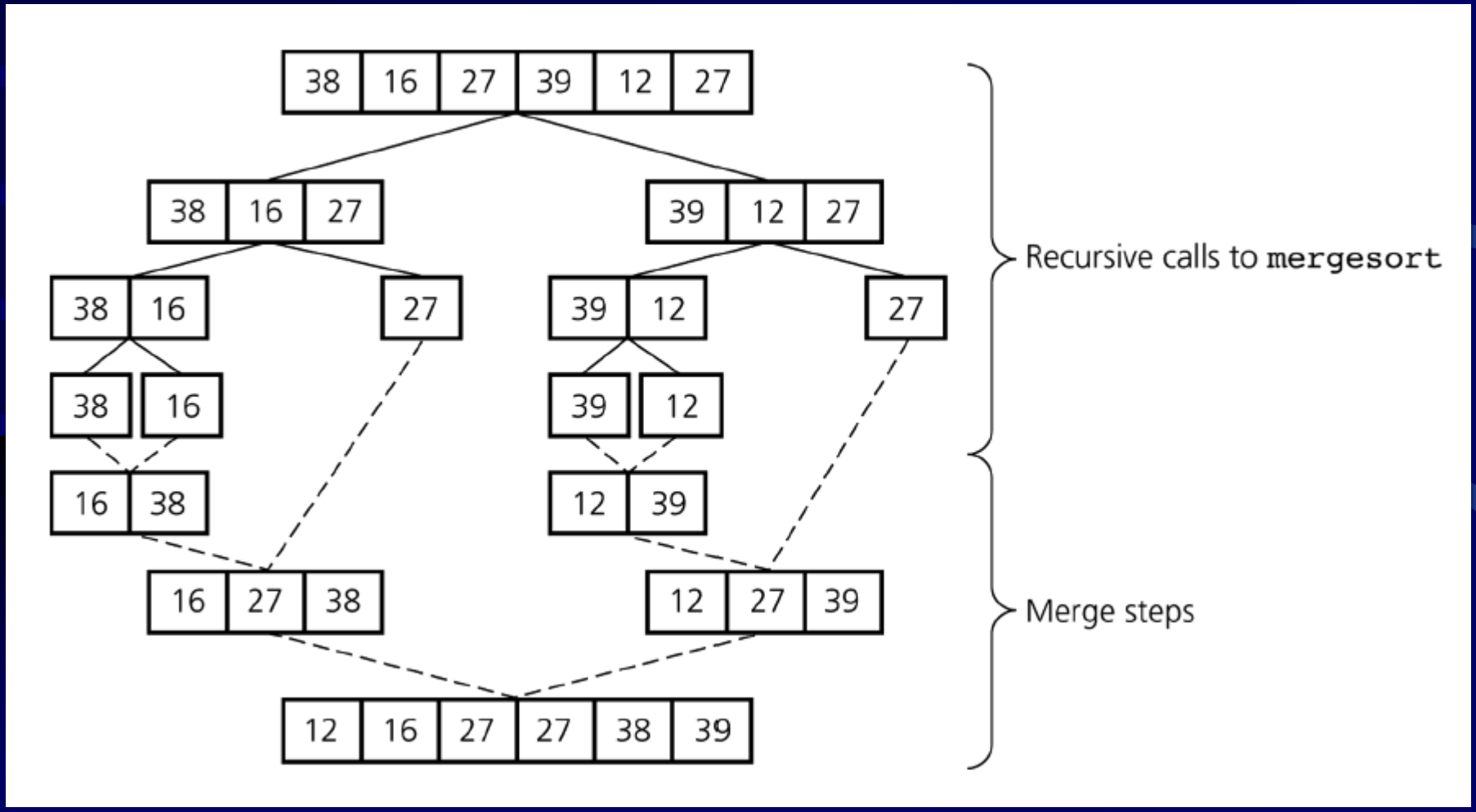
پيچيدگي الگوريتم: $\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} c = \Theta(n^2)$

Bubble Sort (version 2)

```
procedure BubbleSort(array A[0..n–1]):
    stop = 0
    while stop = 0 do
        stop = 1
        for j from 1 to n – 1 do
            if A[j] < A[j–1] then
                stop = 0
                A[j] ↔ A[j–1]
```

Merge Sort

تقسیم کلیدها به دو دسته، مرتب کردن هر دسته (بازگشتي) و ادغام (Merge) دو لیست مرتب شده



Merge Sort

```
void MergeSort(int A[], int left, int right)
{
    if (right<=left) return
    int middle = (left + right)/2;
    MergeSort(A, left, middle);
    MergeSort(A, middle+1, right);
    Merge(A, left, middle, right);
}
```

پیچیدگي الگوريتم:

$$T(n)= c_1 \qquad n=1$$

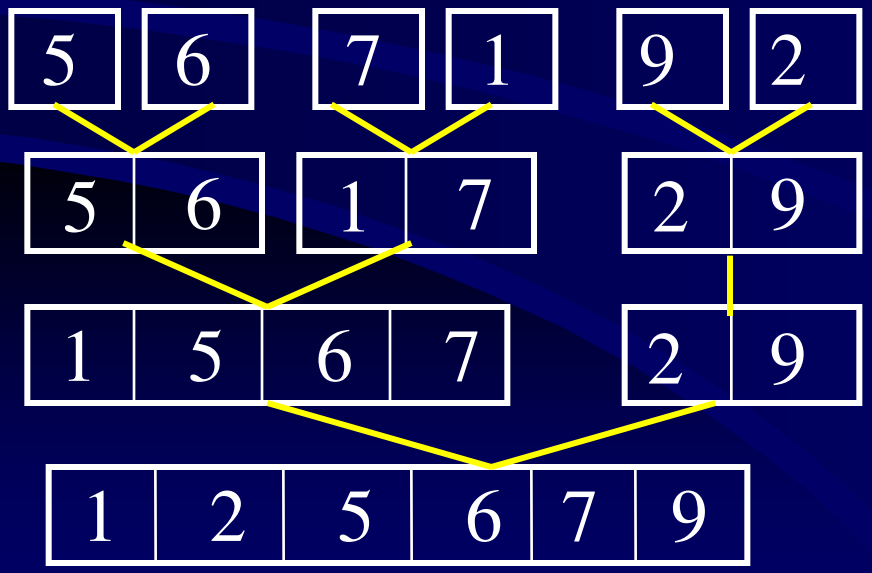
$$T(n)=2T(n/2)+c_2n$$

$$\rightarrow T(n) =O(nlgn)$$

Stable يا Unstable ؟

Iterative Merge Sort

```
void MergeSort(int A[], int left, int right)
{
    for(int m=1; m<=right-left; m+=m)
        for(int i=left; i<=right-m; i+=m+m)
            merge(A, i, i+m-1, min(i+m+m-1, right))
}
```



Quick Sort

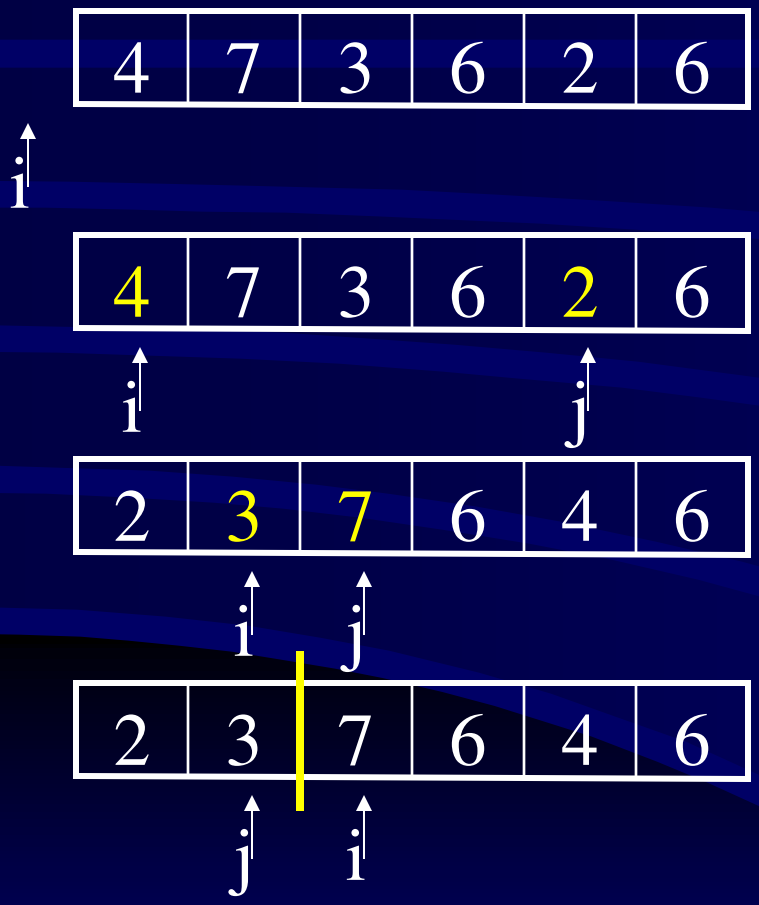
```
QuickSort(A, p, r)
    if p < r then
        q ← partition(A, p, r)
        QuickSort(A, p, q)
        QuickSort(A, q+1, r)

Partition(A, p, r)
    x ← A[p]
    i ← p
    j ← r
    while TRUE do
        while A[j] >= x
            do          j ← j-1
        while A[i]<x
            do          i ← i+1
        if  i<j  then exchange A[i] ↔ A[j]
        else return j
```

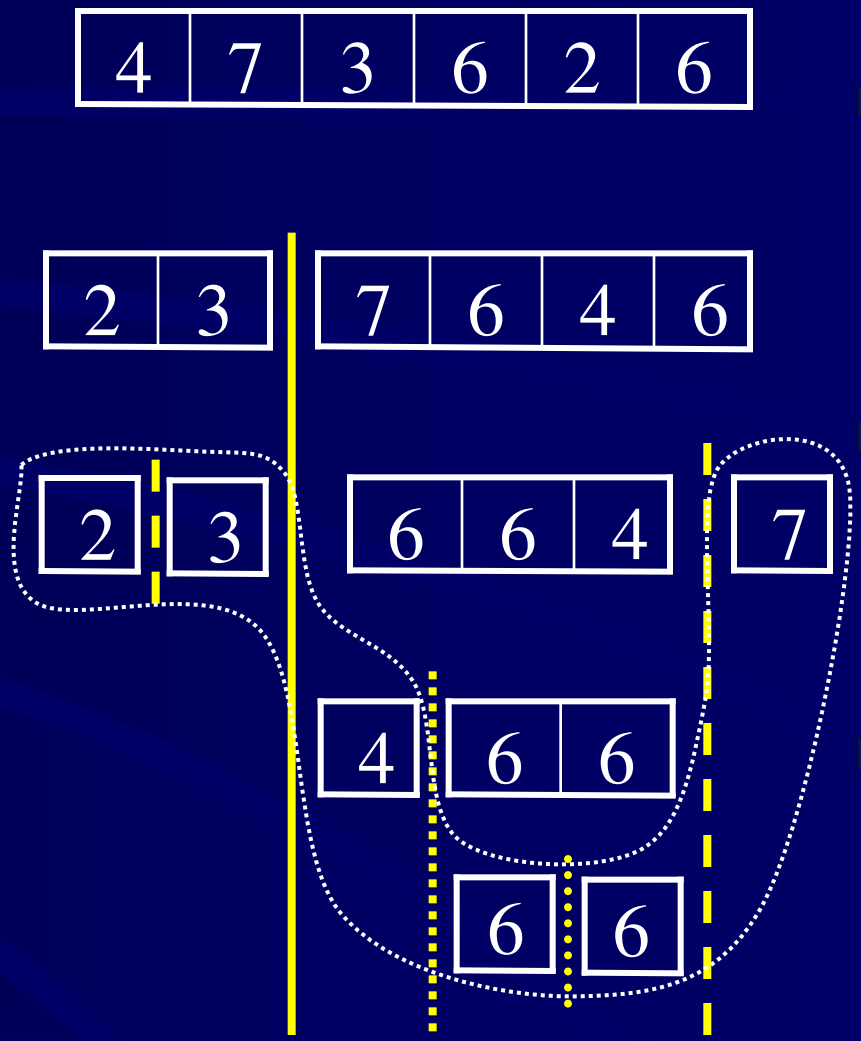
الگوريتم:

- يك عنصر را به عنوان محور (pivot) انتخاب كن
- عناصر را به دو جزء تقسيم كن: عناصر كوچكتر از محور و عناصر بزرگتر يا مساوي محور
- الگوريتم را بصورت بازگشتي تكرار كن

Quick Sort مثال

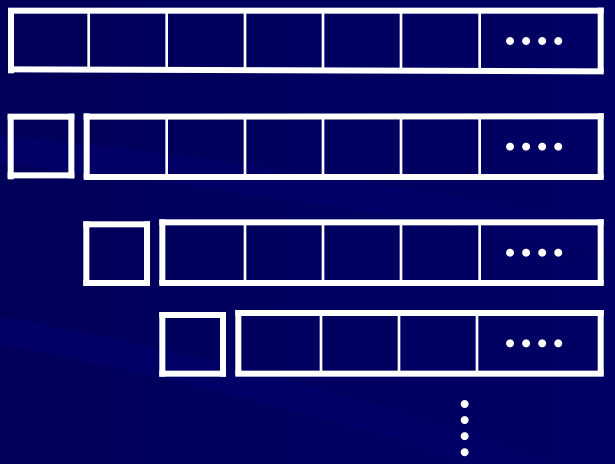
Partitioning, pivot=4





Quick Sort

- اگر محور انتخاب شده، بزرگترين كليد باشد، بدترين حالت اجراي الگوريتم اتفاق مي افتد



- روشهاي مختلف براي انتخاب محور مي توان استفاده نمود
- مثلاً انتخاب بزرگترين عنصر بين دو عنصر اول ليست

Quick Sort

Stable یا Unstable ؟

پیچیدگي الگوريتم:

هزينه تقسيم آرايه (partitioning): $\Theta(n)$

هزينه الگوريتم در بدترين حالت: $T(n)=T(n-1)+\Theta(n)=\mathbf{\Theta(n^2)}$

هزينه الگوريتم در بهترين حالت: $T(n)=2T(n/2)+\Theta(n)=\mathbf{\Theta(nlgn)}$

حالت متوسط به بهترين حالت نزديك است

مثلا اگر عناصر به نسبت 1و 9 تقسيم شوند

$T(n)=T(9/10\ n)+T(1/10\ n)+\Theta(n)=$ $\mathbf{\Theta(nlgn)}$

Heap Sort

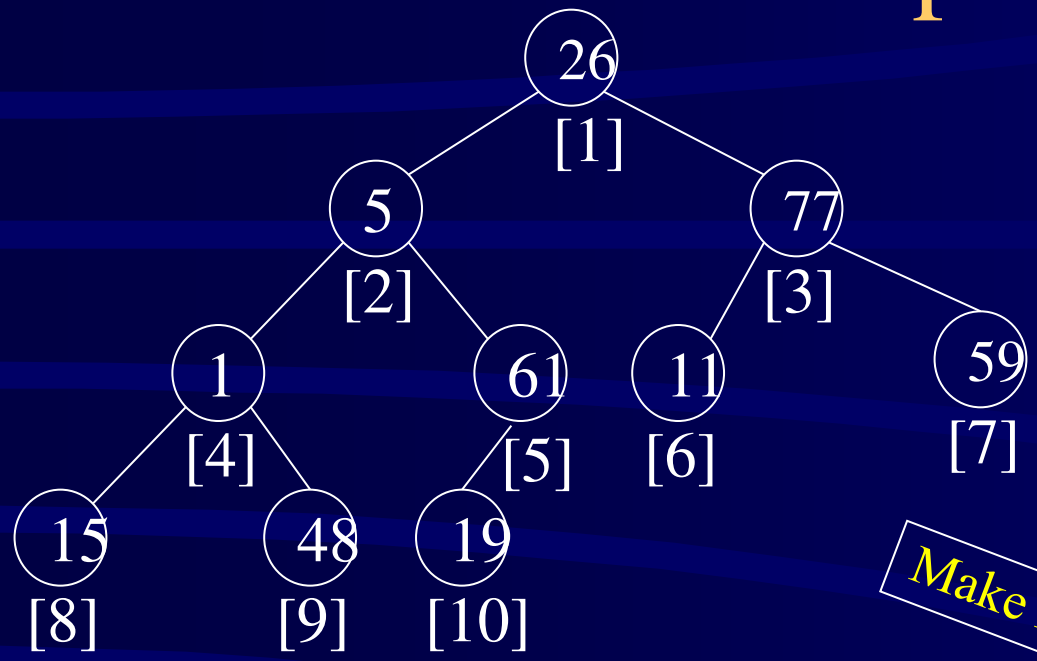
الگوریتم:

- با n عنصر یك max heap بساز
- n بار بزرگترین عنصر heap را حذف کن

Heap Sort

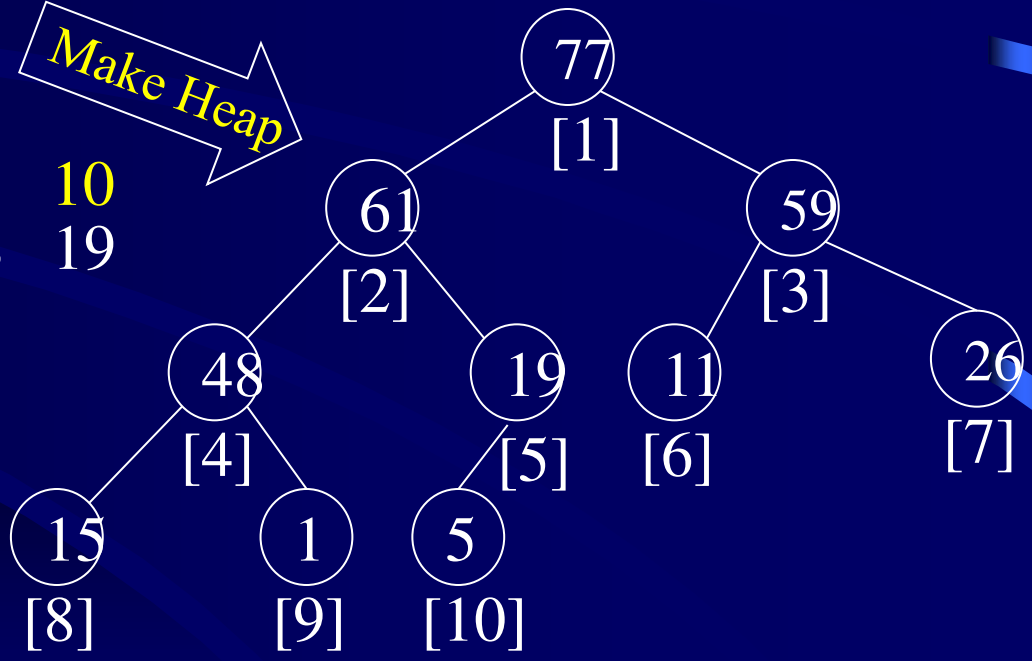
مرحله اول:

- ساختن max heap



1	2	3	4	5	6	7	8	9	10
26	5	77	1	61	11	59	15	48	19

Make Heap



77	61	59	48	19	11	26	15	1	5

مقايسه الگوريتمهاي مرتب سازي

	بهترين حالت	بدترين حالت	حالت متوسط	Stable
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$	yes
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	no
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	yes
Merge sort	O(nlgn)	O(nlgn)	O(nlgn)	yes
Quick sort	O(nlgn)	$O(n^2)$	O(nlgn)	no
Heap sort	O(nlgn)	O(nlgn)	O(nlgn)	no

Count Sort

Count Sort : کلیدهایي که باید مرتب شوند در محدوده 0..k-1 هستند.

الگوریتم:

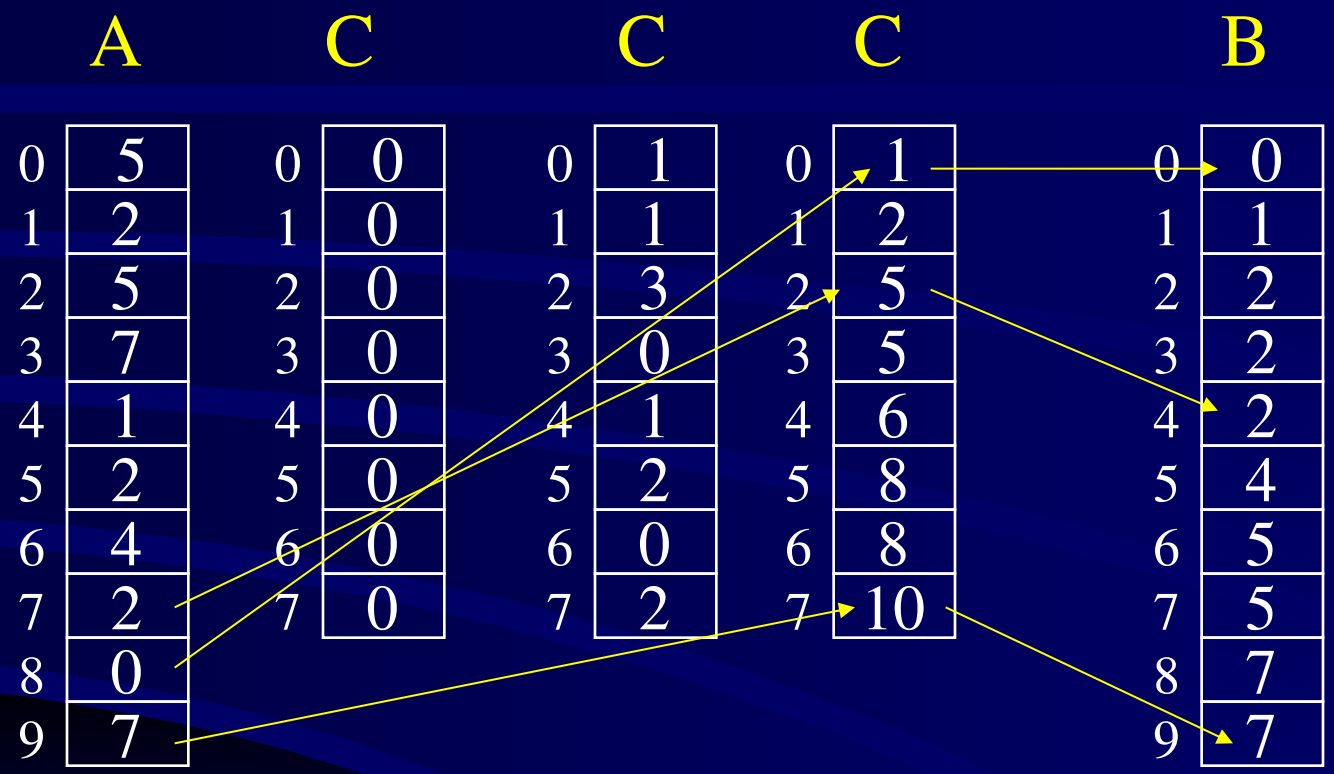
پیدا کردن تعداد کلیدهاي مساوي هر کلید (در محدوده 0..k-1)

پیدا کردن اندیس شروع هر مقدار کلید در ترتیب نتیجه

قرار دادن کلیدها در آرایه نتیجه

```
CountingSort(array A[0..n–1], B[0..n–1], C[0..k–1], k):
    for i from 0 to k – 1 do
        C[i] ← 0
    for i from 0 to n – 1 do
        C[A[i]] ← C[A[i]] + 1
    for i from 1 to k – 1 do
        C[i] ← C[i] + C[i – 1]
    for i from n – 1 downto 0 do
        B[C[A[i]] – 1] ← A[i]
        C[A[i]] ← C[A[i]] – 1
```

مثال


A
C
C
C
B
0 5
1 2
2 5
3 7
4 1
5 2
6 4
7 2
8 0
9 7
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
0 1
1 1
2 3
3 0
4 1
5 2
6 0
7 2
0 1
1 2
2 5
3 5
4 6
5 8
6 8
7 10
0 0
1 1
2 2
3 2
4 2
5 4
6 5
7 5
8 7
9 7

k=3

A					B					C		
2	1	2	0	1								
2	1	2	0	1						0	0	0
2	1	2	0	1						1	2	2
2	1	2	0	1						1	3	5
2	1	2	0	1	0	1	1	2	2	1	3	5

پيچيدگي الگوريتم:

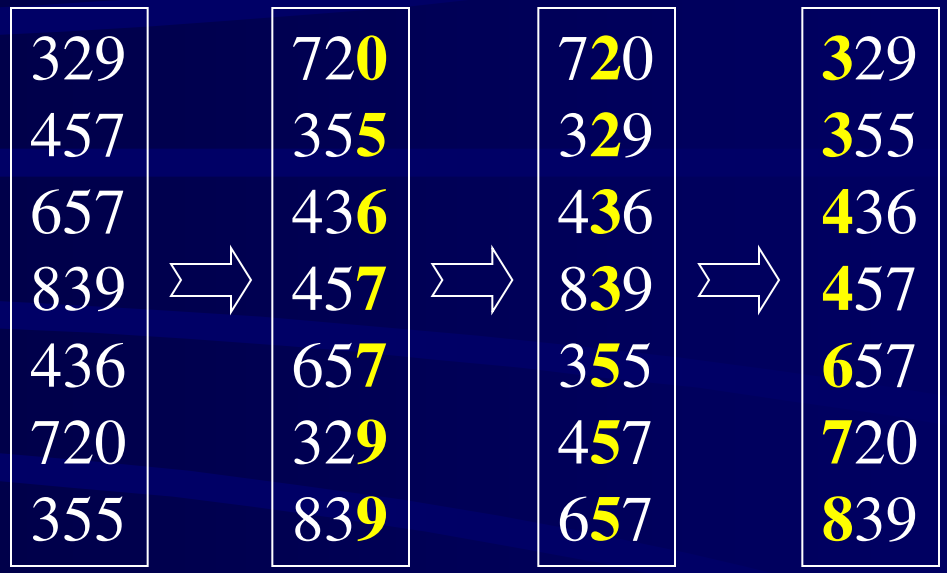
$$T(n) = \Theta(k) + \Theta(n) + \Theta(k) + \Theta(n) = \Theta(k+n)$$

Radix Sort

- براي مرتب كردن n عدد d رقمي استفاده مي شود
- اعداد بر حسب كم ارزش ترين رقم با يك الگوريتم پايدار مرتب مي شوند و سپس بر اساس دومين كم ارزش ترين عدد مرتب مي شوند، به همين ترتيب تا با ارزش ترين رقم
- اگر كليدها داراي k مولفه باشند برحسب هريك از اين مولفه ها و با در نظر گرفتن ارزش مكاني آنها مرتب سازي انجام مي شود

procedure *RadixSort* (**array** A[0..*n*–1], *d*):
    for *i* **from** 1 **to** *d* **do**
        use a *StableSort* to sort array A[0..*n*–1] on digit i
    /// digit 1 is the lowest-order digit and digit d is the highest order digit

Radix Sort



پيچيدگي الگوريتم:

تكرار d بار حلقه براي مرتب كردن n عنصر

اگر از الگوريتم count sort براي مرتب كردن در هر تكرار حلقه (با هزينه $\Theta(n+k)$) استفاده شود،

هزينه count sort : $\Theta(dn+dk)$

اگر $k=O(n)$ ، پيچيدگي الگوريتم خطي است.

مرتب‌سازی سطلی (Bucket Sort)

```
BUCKET-SORT(A)
    ▷ مرتب‌سازی سطلی که لیست A را مرتب می‌کند
 1  for i ← 1 to k
 2      do for each value v of type t_i
 3             do make B_i[v] empty
 4         for each record r in list A in order
 5             do let v be the value of f_i of the key for r
 6                move r from A to the end of B_i[v]
 7         for each value v of type t_i from lowest to highest
 8             do concatenate B_i[v] to the end of A
```

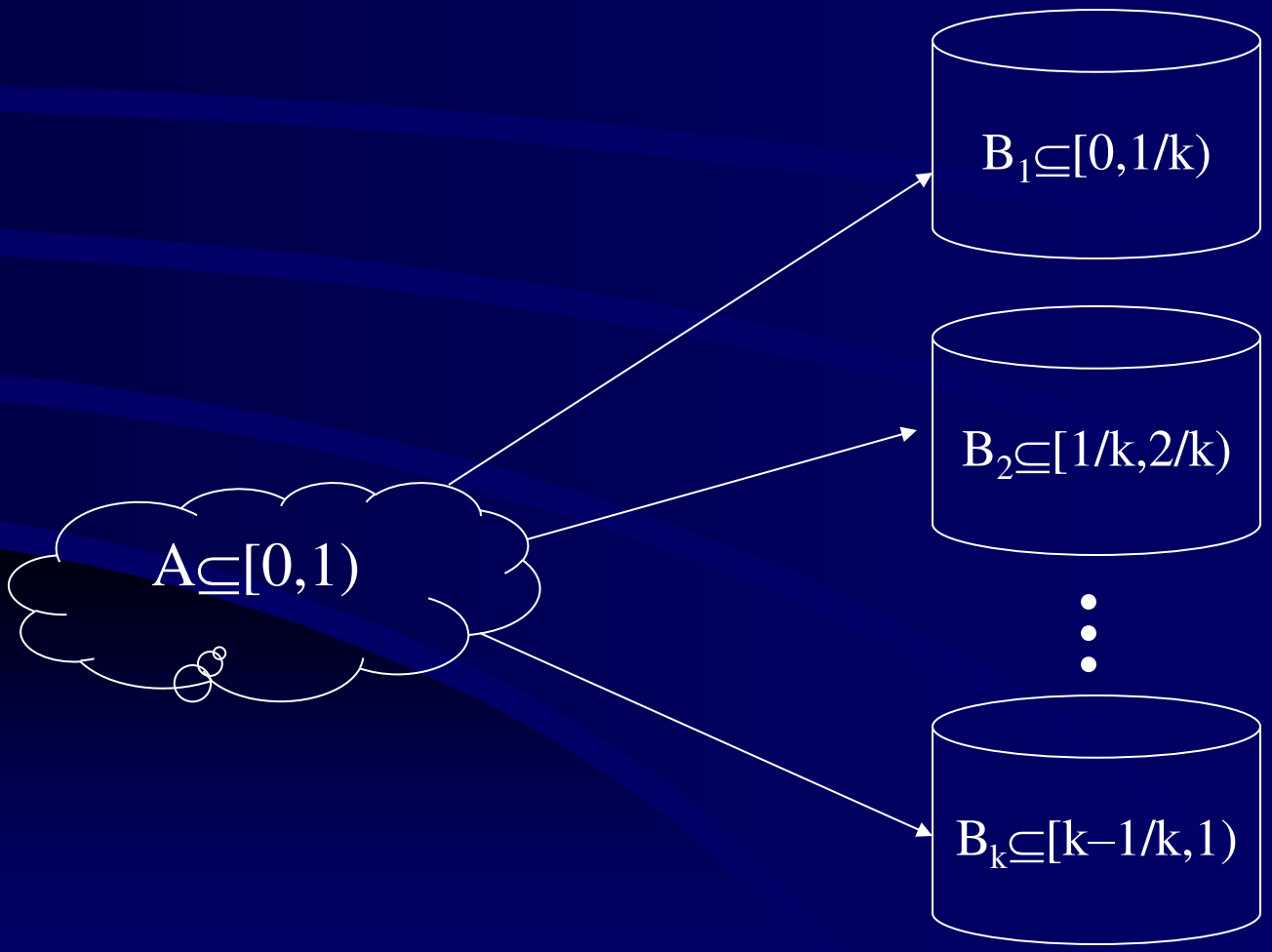
مثال: کلیدها شامل سه مؤلفه با مقادیر a..z، ۱۰۰ . . . ۱ و ۱۴۰۰ . . . ۱۳۰۰.

عنصر	$f_۳$	$f_۲$	$f_۱$
$a_۱$	a	۵	۱۳۲۰
$a_۲$	c	۱۲	۱۳۱۰
$a_۳$	b	۱۲	۱۳۰۵
$a_۴$	a	۸	۱۴۰۰
$a_۵$	z	۱۰	۱۳۰۸
$a_۶$	b	۱۲	۱۳۰۴
$a_۷$	a	۶	۱۳۱۰

عنصر	$f_۳$	$f_۲$	$f_۱$
$a_۱$	a	۵	۱۳۲۰
$a_۲$	c	۱۲	۱۳۱۰
$a_۳$	b	۱۲	۱۳۰۵
$a_۴$	a	۸	۱۴۰۰
$a_۵$	z	۱۰	۱۳۰۸
$a_۶$	b	۱۲	۱۳۰۴
$a_۷$	a	۶	۱۳۱۰

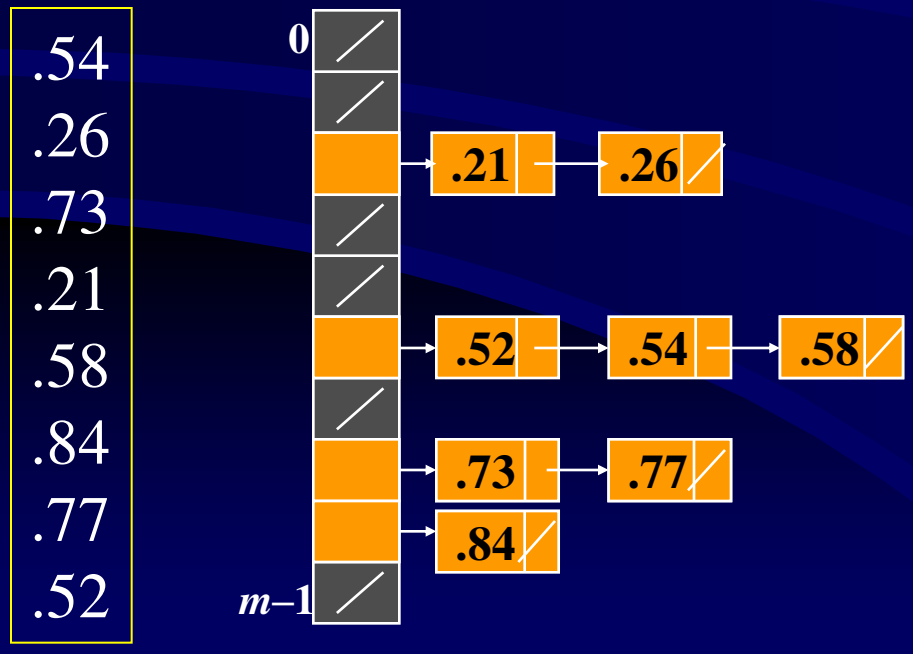
ورودی	$B_۱[.]$	خروجی ۱	$B_۲[.]$	خروجی۲	$B_۳[.]$	خروجی ۳
$a_۱$	$[۱۳۰۴]\ a_۶$	$a_۶$	$[۵]\ a_۱$	$a_۱$	$['a']\ a_۱, a_۷, a_۴$	$a_۱$
$a_۲$	$[۱۳۰۵]\ a_۳$	$a_۳$	$[۶]\ a_۷$	$a_۷$	$['b']\ a_۶, a_۳$	$a_۷$
$a_۳$	$[۱۳۰۸]\ a_۵$	$a_۵$	$[۸]\ a_۴$	$a_۴$	$['c']\ a_۲$	$a_۴$
$a_۴$	$[۱۳۱۰]\ a_۲, a_۷$	$a_۲$	$[۱۰]\ a_۵$	$a_۵$	$['z']\ a_۵$	$a_۶$
$a_۵$	$[۱۳۲۰]\ a_۱$	$a_۷$	$[۱۲] a_۶, a_۳, a_۲$	$a_۶$		$a_۳$
$a_۶$	$[۱۴۰۰]\ a_۴$	$a_۱$		$a_۳$		$a_۲$
$a_۷$		$a_۴$		$a_۲$		$a_۵$

Bucket Sort


$B_1 \subseteq [0,1/k)$
$B_2 \subseteq [1/k,2/k)$
$A \subseteq [0,1)$
$B_k \subseteq [k-1/k,1)$

Bucket Sort

procedure *BucketSort* (**array** A[0..*n*–1]):
    for *i* **from** 0 **to** *n*–1 **do**
        insert A[i] into list B[⌊n A[i]⌋]
    for *i* **from** 0 **to** *m*–1 **do**
        sort B[i] with *InsertionSort*
concatenate the lists B[0], B[1], ..., B[m–1] together in order



بدترین حالت:

$T(n) = \Theta(n^2)$

حالت متوسط با n بسته:

$T(n) = \Theta(n)$

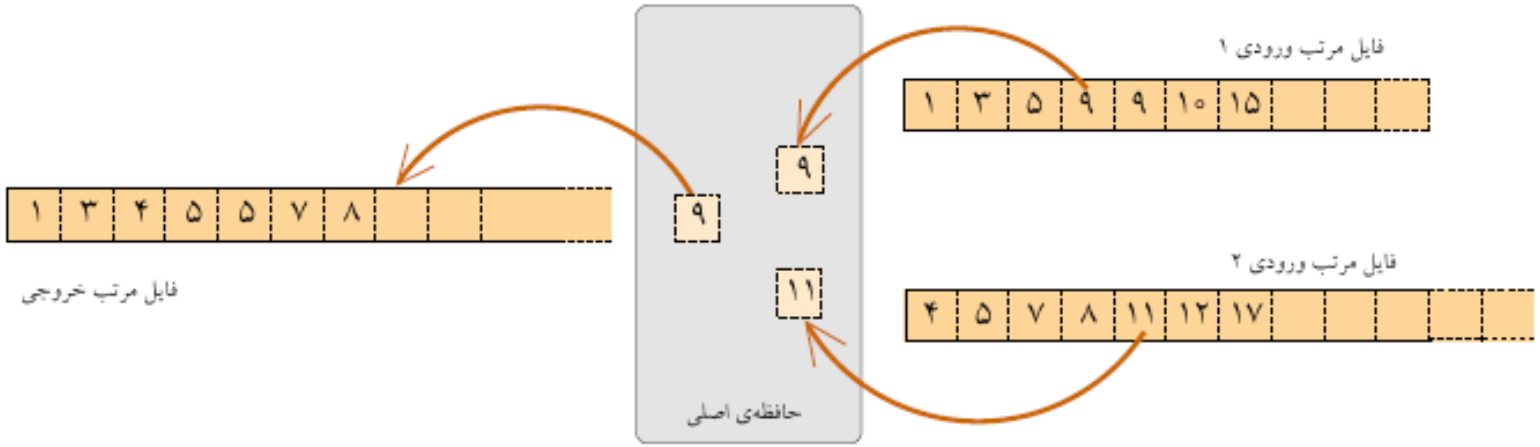
مرتب سازي خارجي

- ميزان حافظه كمتر از داده ها هستند.
- استفاده از حافظه جانبي براي مرتب سازي ...
- استفاده از ايده هاي MergeSort
- External MergeSort

مرتب‌سازی خارجی (فرض)

- اطلاعات بر روی فایل‌ها به‌صورت ترتیبی ذخیره شده است.
- هر فایل شامل n رکورد است. هر رکورد یک کلید دارد.
- می‌خواهیم در فایل خروجی رکوردها براساس کلیدهایشان مرتب باشند.
- با هر دسترسی به دیسک k رکورد خوانده می‌شود.
- تعداد فایل‌هایی که در یک زمان باز هستند r و محدود است.
- تعداد حافظه‌ی اصلی قابل استفاده ثابت است.
- عملیات مقایسه و محاسبات فقط می‌تواند در حافظه‌ی اصلی انجام شود.

ادغام دو قطعه‌ی مرتب



ادغام چند فایل مرتب