ساختمان داده‌ها و الگوریتم‌ها

فصل پنجم (مرتب‌سازی)

صمد نجارقابل

S.Najjar.G@Gmail.com

S.Najjar.G@Gmail.com

فهرست مطالب

- مقدمه‌ای بر الگوریتم‌ها و مفاهیم پایه
- معرفی پیچیدگی زمانی و حافظه‌ای و روشهای تحلیل مسائل
- معرفی ساختمان داده‌های مقدماتی و الگوریتم‌های وابسته به آنها
  - آرایه
  - صف
  - پشته
  - لیست پیوندی
- تئوری درخت و گراف و الگوریتم‌های مرتبط
- **الگوریتم‌های مرتب‌سازی و تحلیل پیچیدگی مربوط به آنها**
- مباحث تکمیلی در ساختمان داده‌ها

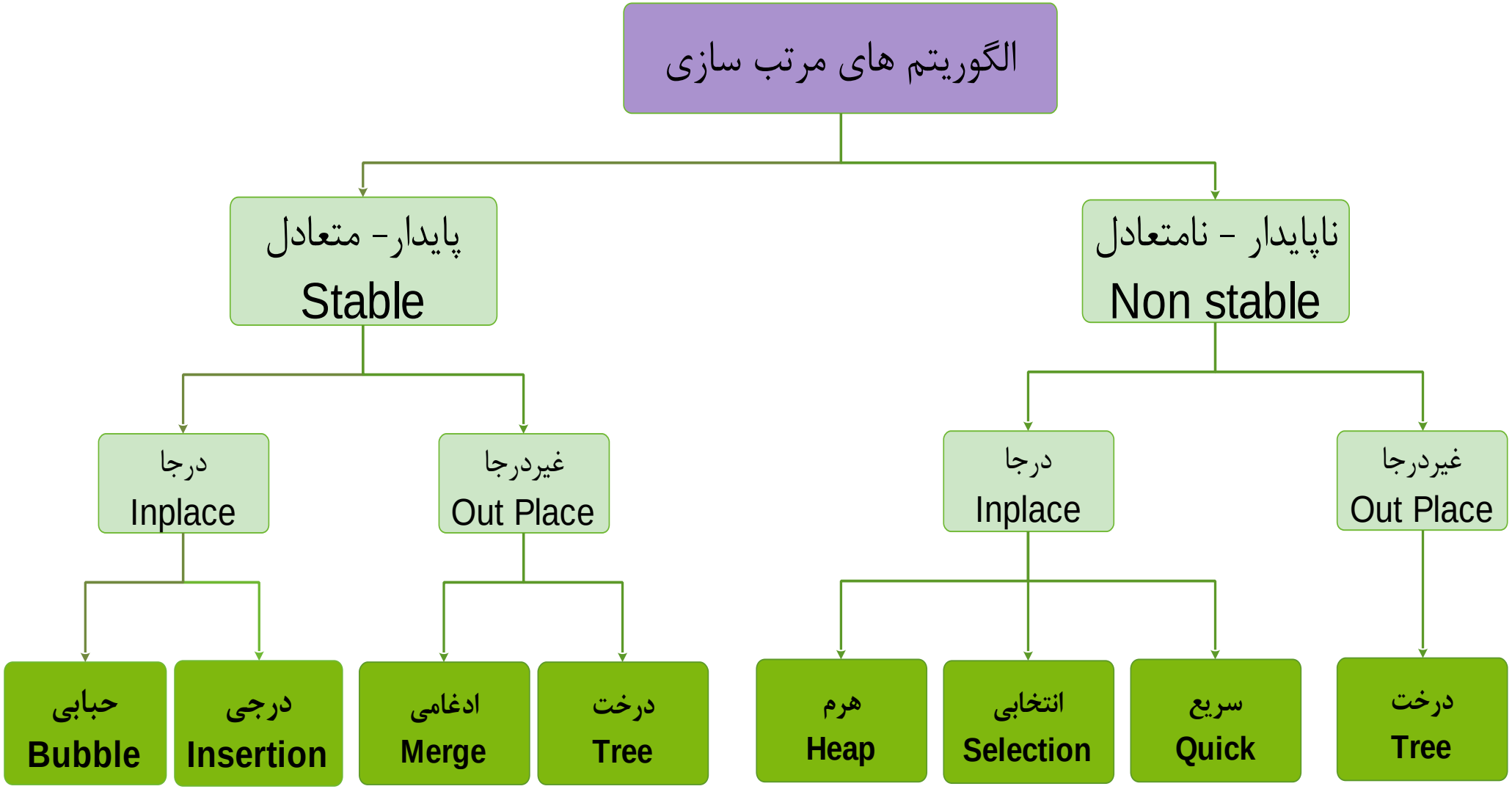
S.Najjar.G@Gmail.com

مرتب‌سازی

- فرض کنید لیستی از رکوردهای $(R_0, R_1, \ldots, R_{n-1})$ داده شده است و هر رکورد R_i دارای مقدار کلید K_i است. مساله مرتب کردن متناظر با پیدا کردن جایگشت σ است که با این جایگشت باید $K_{\sigma(i-1)} \leq K_{\sigma(i)}, 0<i \leq n-1$
- در مورد لیستی که چند کلید یکسان دارد جایگشت σ منحصر به فرد نیست.
- مرتب سازی که جایگشتی با خاصیت زیر ایجاد کند را پایدار گویند.
  - اگر $i < j$ و $K_i = K_j$ آنگاه در لیست مرتب شده R_i قبل از R_j باشد.
- روش مرتب سازی داخلی
  - برای مرتب کردن لیستی که به قدر کافی کوچک است از روشهایی استفاده می‌کنیم که بتوانیم تمام لیست را در حافظه اصلی مرتب کنیم
- روش مرتب سازی خارجی
  - روشهایی که روی لیست‌های بزرگتر به کار گرفته می شود.

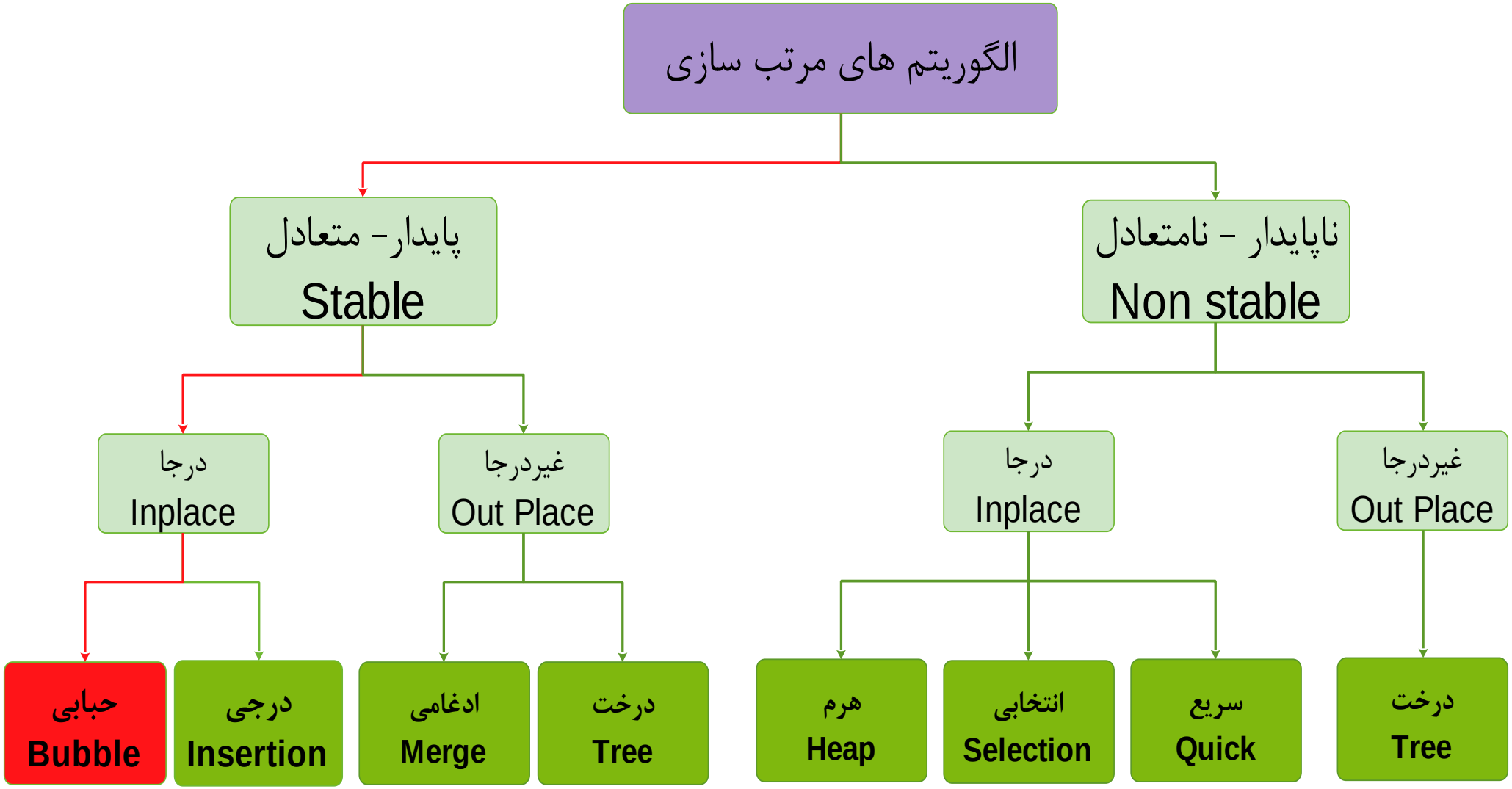
S.Najjar.G@Gmail.com

دسته‌بندی برخی از الگوریتم‌های مرتب‌سازی



S.Najjar.G@Gmail.com

دسته‌بندی برخی از الگوریتم‌های مرتب‌سازی



S.Najjar.G@Gmail.com

مرتب سازی حبابی (bubble sort)

- الگوریتم با تعویض مرتب عناصر مرتب سازی را انجام می دهد.
- در طول n بار حرکت در طول بردار عناصر، یک عنصر با عنصر بعدی مقایسه می شود و در صورت لزوم جابجا می شوند.
- در اولین بار طی کردن بردار بزرگترین (کوچکترین) عنصر در انتهای بردار قرار می گیرد.
- در i امین مرتبه عناصر [n-i,…, n] به صورت صحیح قرار گرفته اند.
- بهبود
  - در مرحله i ام می‌توان طی کردن بردار عناصر را تا n-i انجام داد.
  - در صورتی که در یکبار طی کردن هیچ تعویضی صورت نگیرد بردار مرتب شده است.

S.Najjar.G@Gmail.com

مرتب سازی حبابی (bubble sort) (ادامه)

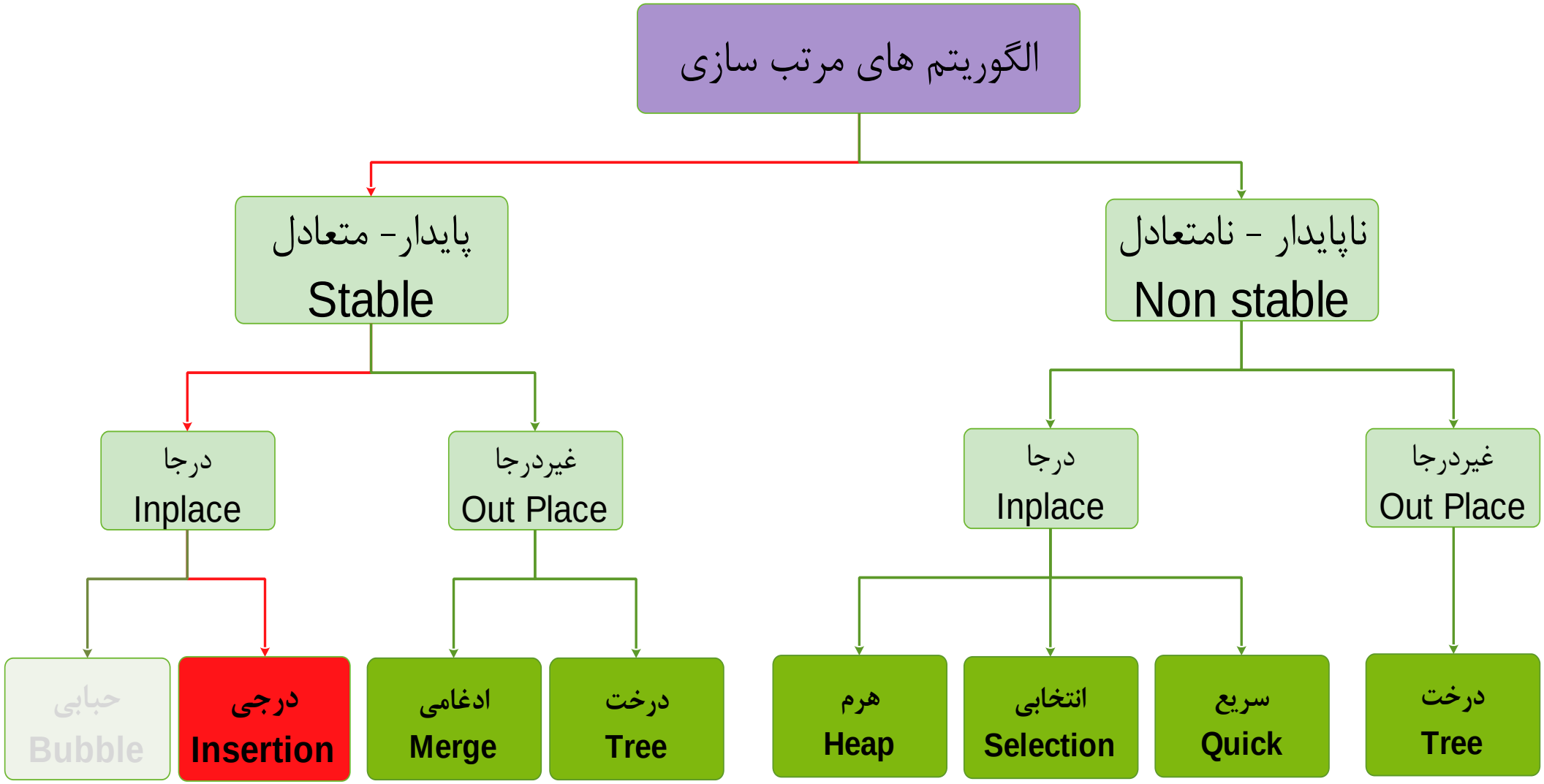
```
flag =true;
Pass= 1;
while  (pass <= n) && flag
{
    flag = false;
    for (j=1; j<= n- pass; j++)
            if  (x[j]> x[j+1] )
            {
                    flag=true;
                    swap( x[j], x[j+1]);
            }
    pass= pass+1;
}
```

در یک بردار مرتب بهترین عملکرد و در یک بردار نامرتب بدترین عملکرد را نشان می دهد.

پیچیدگی	بهترین حالت	حالت متوسط	بدترین حالت
	$O(n)$	$O(n^2)$	$O(n^2)$

S.Najjar.G@Gmail.com

دسته‌بندی برخی از الگوریتم‌های مرتب‌سازی



S.Najjar.G@Gmail.com

مرتب سازی درجی

- بر اساس درج یک عنصر در محل صحیح کار می کند
- مرحله ی اصلی در این روش اضافه کردن (درج) رکورد R در دنباله ای از رکوردهای مرتب شده $R_1, R_2, \ldots, R_i$ $(K_1 \leq K_2 \leq, \ldots, \leq K_i)$ است به گونه ای که دنباله ی حاصل به طول i+1 نیز مرتب باشد.
- قسمت اول ارایه مرتب در نظر گرفته می شود. برای اضافه کردن هر عنصر اخرین عنصر این قسمت و مقادیر قبل از آن تا جایی که این عنصر کوچکتر از آنها است به جلو رانده شده این عنصر در محل مناسب درج می شود.

S.Najjar.G@Gmail.com

مرتب سازی درجی (ادامه)

i = 2 next = 2

list	[0]	[1]	[2]	[3]	[4]
	3	5	6	4	7

```
void insertion_sort(element list[], int n)
/* perform a insertion sort on the list */
{
   int i,j;
   element next;
   for (i = 1; i < n; i++) {
      next = list[i];
      for (j = i-1; j >= 0 && next.key < list[j].key; j--)
         list[j+1] = list[j];
      list[j+1] = next;
   }
}
```

S.Najjar.G@Gmail.com

مرتب سازی درجی (ادامه)

- تحلیل مرتب سازی درجی
- زمان لازم جهت درج i امین عنصر به داخل لیست مرتب شده O(i) خواهد بود

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

پیچیدگی	بهترین حالت	حالت متوسط	بدترین حالت
	$O(n)$	$O(n^2)$	$O(n^2)$

- بهترین حالت مربوط به یک بردار کاملا مرتب و بدترین حالت مربوط به یک بردار مرتب معکوس است. الگوریتم از پیچیدگی $O(n^2)$ است ولی از الگوریتم حبابی بهتر است
- مرتب سازی درجی پایدار است
- در طول مرتب سازی تنها به یک حافظه اضافی برای یک رکورد نیاز داریم.

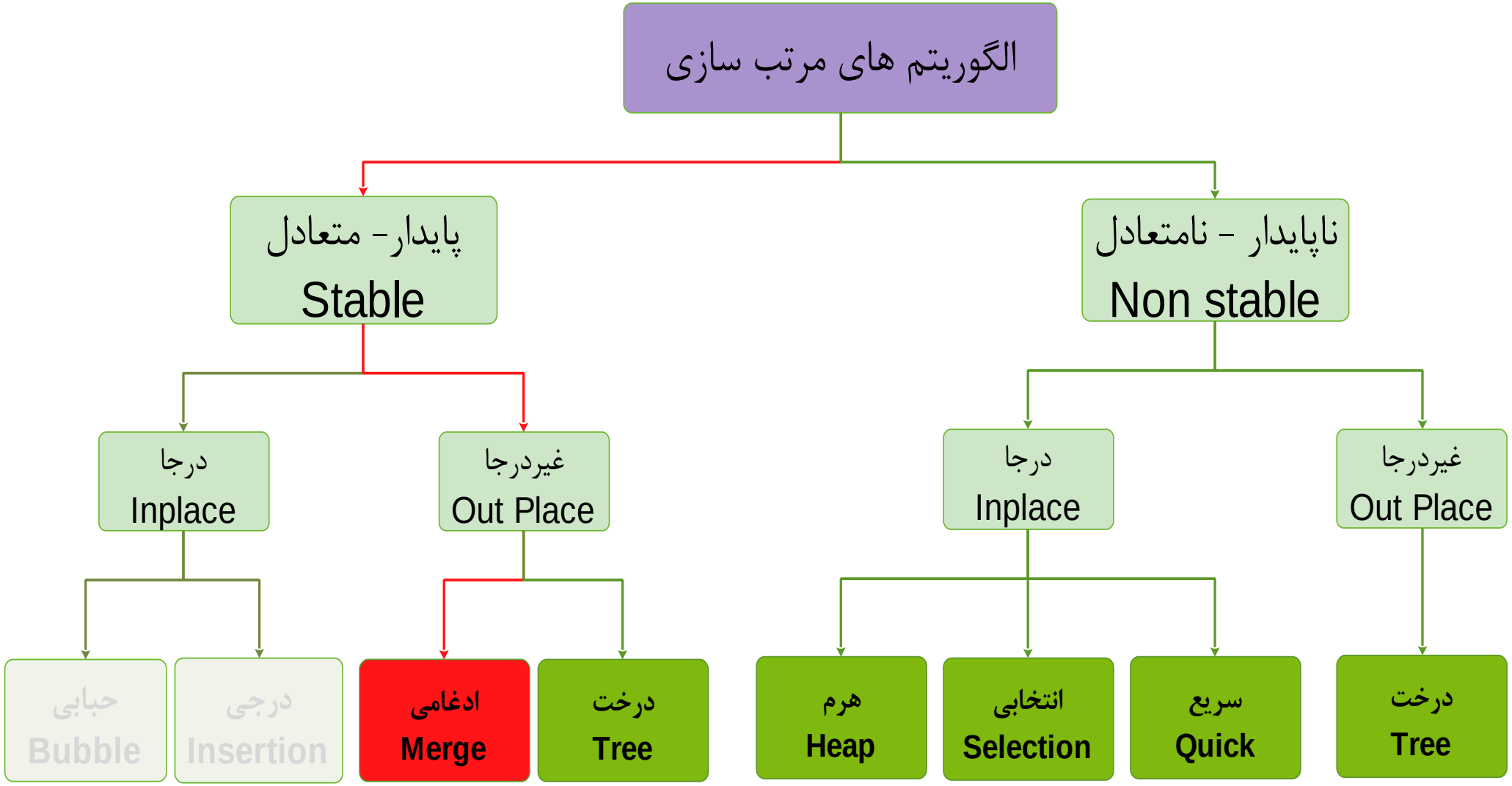
S.Najjar.G@Gmail.com

مرتب سازی درجی (ادامه)

- مرتب سازی درجی لیست
  - عضوهای لیست به جای آرایه به صورت لیست پیوندی نمایش داده شوند.
  - تعداد رکوردهای جابجا شونده برابر صفر است (تنها فیلدهای پیوند نیازمند تنظیم)
  - باید جستجوی ترتیبی برای یافتن محل عنصر به کار رود.

S.Najjar.G@Gmail.com

دسته‌بندی برخی از الگوریتم‌های مرتب‌سازی



S.Najjar.G@Gmail.com

مرتب سازی ادغام

- مرتب سازی ادغام معمولا بر روی عناصری که در فایلها قرار دارند اجرا می شود گرچه می توان این روش را در مورد بردارها نیز به کار برد.
- پیش از آنکه مرتب سازی ادغام را بیان کنیم باید ببینیم چگونه می توان دو لیست مرتب را با هم ادغام کرد تا یک لیست مرتب به دست آید.
- ادغام
  - الگوریتمی برای ادغام با حافظه اضافی O(n) وجود دارد. این الگوریتم دو لیست مرتب (list[m+1], …, list[n]) و (list[i], … , list[m]) را گرفته و آنها را در یک لیست (sorted[i], … , sorted[n]) مرتب می کند.
  - الگوریتمی برای ادغام با حافظه اضافی O(1) نیز وجود دارد.

S.Najjar.G@Gmail.com

ادغام با حافظه O(n)

```
void merge(element list[], element sorted[], int i, int m,
                                              int n)
/* merge two sorted files: list[i],...,list[m], and
list[m+1],..., list[n]. These files are sorted to
obtain a sorted list: sorted[i],..., sorted[n] */
{
   int j,k,t;
   j = m+1;          /* index for the second sublist */
   k = i;            /* index for the sorted list */

   while (i <= m && j <= n) {
      if (list[i].key <= list[j].key)
         sorted[k++] = list[i++];
      else
         sorted[k++] = list[j++];
   }
   if (i > m)
   /* sorted[k],..., sorted[n] = list[j],..., list[n] */
      for (t = j; t <= n; t++)
         sorted[k+t-j] = list[t];
      else
      /* sorted[k],..., sorted[n] = list[i],..., list[m] */
         for (t = i; t <= m; t++)
            sorted[k+t-i] = list[t];
}
```

i j

(list[i], … , list[m]) (list[m+1], …, list[n])

k

(sorted[i], … , sorted[n])

S.Najjar.G@Gmail.com

مرتب سازی ادغام به صورت تکراری

- مرتب سازی ادغام به صورت تکراری

۱) فرض می کنیم ورودی شامل n لیست مرتب شده است که طول هر یک از آنها 1 می باشد.

۲) این لیست ها دوبه دو با یکدیگر ادغام می شوند تا n/2 لیست که طول هر یک از آنها 2 است به دست آید. (اگر n فرد باشد آنگاه یک لیست به طول 1 داریم).

۳) این n/2 لیست دوبه دو با یکدیگر ادغام می شوند و این فرایند ادامه می یابد تا در انتها به یک لیست برسیم.

S.Najjar.G@Gmail.com

مرتب سازی ادغام به صورت تکراری

- تحلیل مرتب سازی ادغام به صورت تکراری
  - تعداد مراحل عبور از داده ها از $O(\log_2 n)$ است. زیرا

در مرحله اول طول لیستها=1
در مرحله دوم طول لیستها=2
در مرحله i ام طول لیستها= 2^{i-1}

در مرحله ؟ طول لیستها= n

$O(\log_2 n)$ =؟

  - ادغام دو لیست می تواند در زمان خطی انجام شود: $O(n)$
  - زمان اجرای کل الگوریتم $O(n \log n)$ است.
  - به سادگی ثابت می شود الگوریتم مرتب سازی ادغام پایدار است.

پیچیدگی	بهترین حالت	حالت متوسط	بدترین حالت
	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$

S.Najjar.G@Gmail.com

مرتب سازی ادغام به صورت تکراری

- تابع *merge_pass*

تابعی که زیرلیست های مجاور در هر مرحله را با هم ادغام می کند. مثال برای length=2

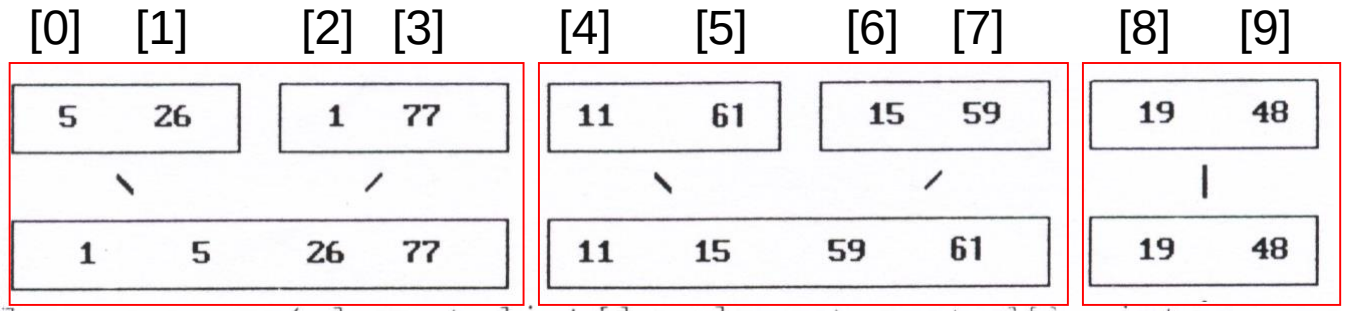
length=2

n=10

i= 0 5 7

list

sorted




```
void merge_pass(element list[], element sorted[], int n,
                int length)
{
  int i,j;
  for (i = 0; i <= n - 2 * length; i += 2 * length)
    merge(list,sorted,i,i + length - 1,i + 2 * length - 1);
  if (i + length < n)
    merge(list,sorted,i,i + length - 1,n - 1);
  else
    for (j = i; j < n; j++)
      sorted[j] = list[j];
}
```

طول زیر لیست ها

تعداد اعضای درون لیست

S.Najjar.G@Gmail.com

مرتب سازی ادغام

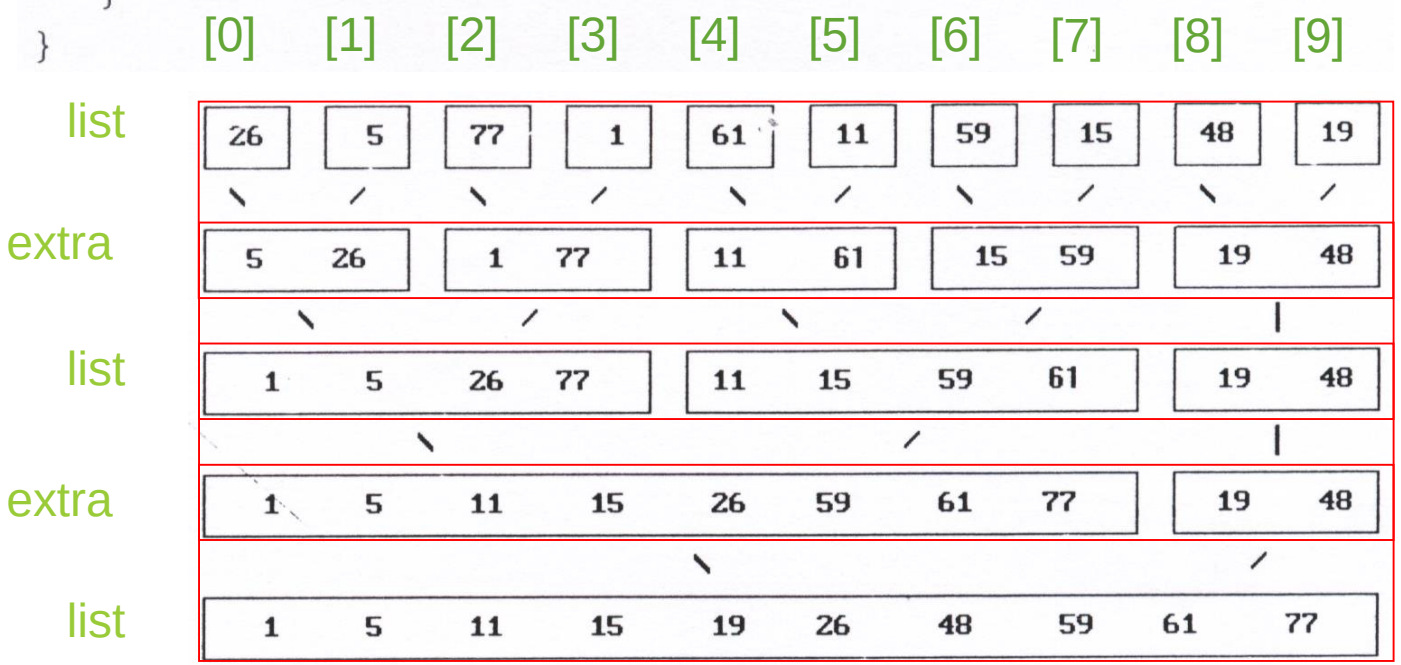
length= 1 2 4 8 16

n=10

```
void merge_sort(element list[], int n)
{
   int length = 1; /* current length being merged */
   element extra[MAX_SIZE];

   while (length < n) {
      merge_pass(list,extra,n,length);
      length *= 2;
      merge_pass(extra,list,n,length);
      length *= 2;
   }
}
```

مرتب سازی ادغام را بر روی merge_sort یک فایل (لیست) انجام می دهد.



S.Najjar.G@Gmail.com

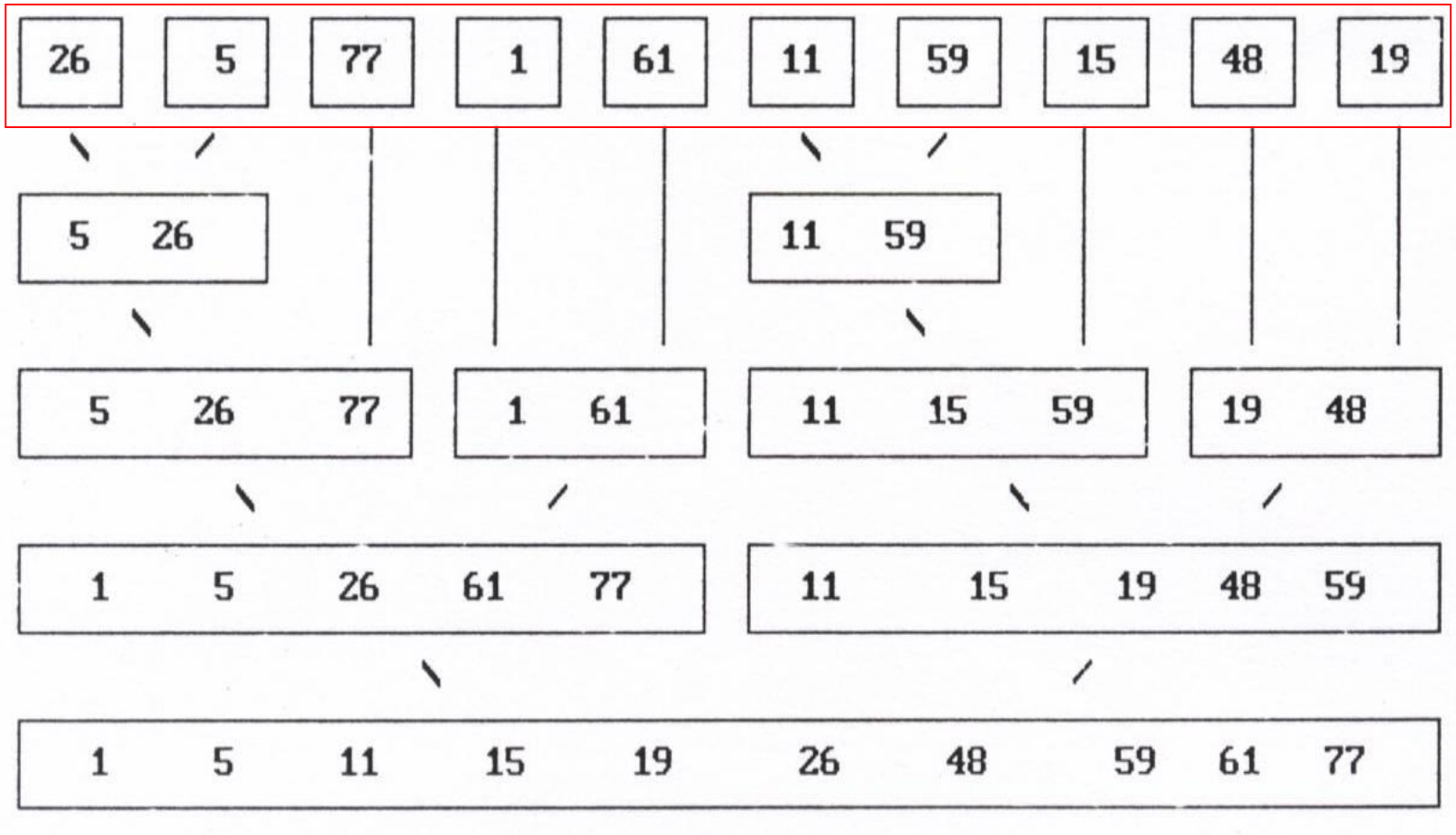
مرتب سازی ادغام به صورت بازگشتی

```
int rmerge(element list[], int lower, int upper)
  {
     int middle;
     if (lower >= upper)
        return lower;
     else {
        middle = (lower + upper) / 2;
        return listmerge(list,rmerge(list,lower,middle),
                               rmerge(list,middle+1,upper));
     }
  }
```

قرار دارند1-n تا 0اعداد از اندیس

مرتب سازی ادغام به صورت بازگشتی (ادامه)

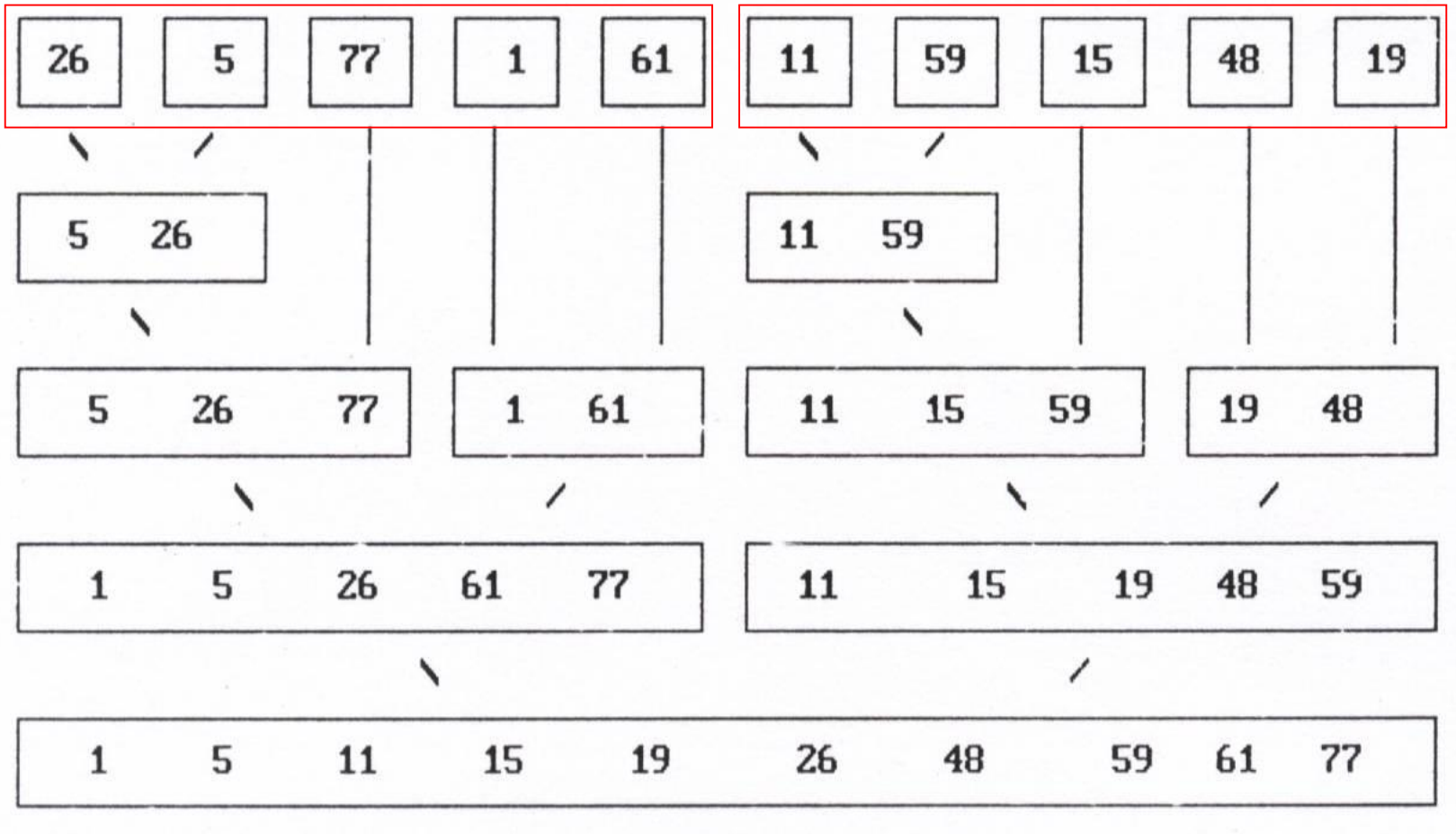
- مفهوم مرتب سازی ادغام بازگشتی



S.Najjar.G@Gmail.com

مرتب سازی ادغام به صورت بازگشتی (ادامه)

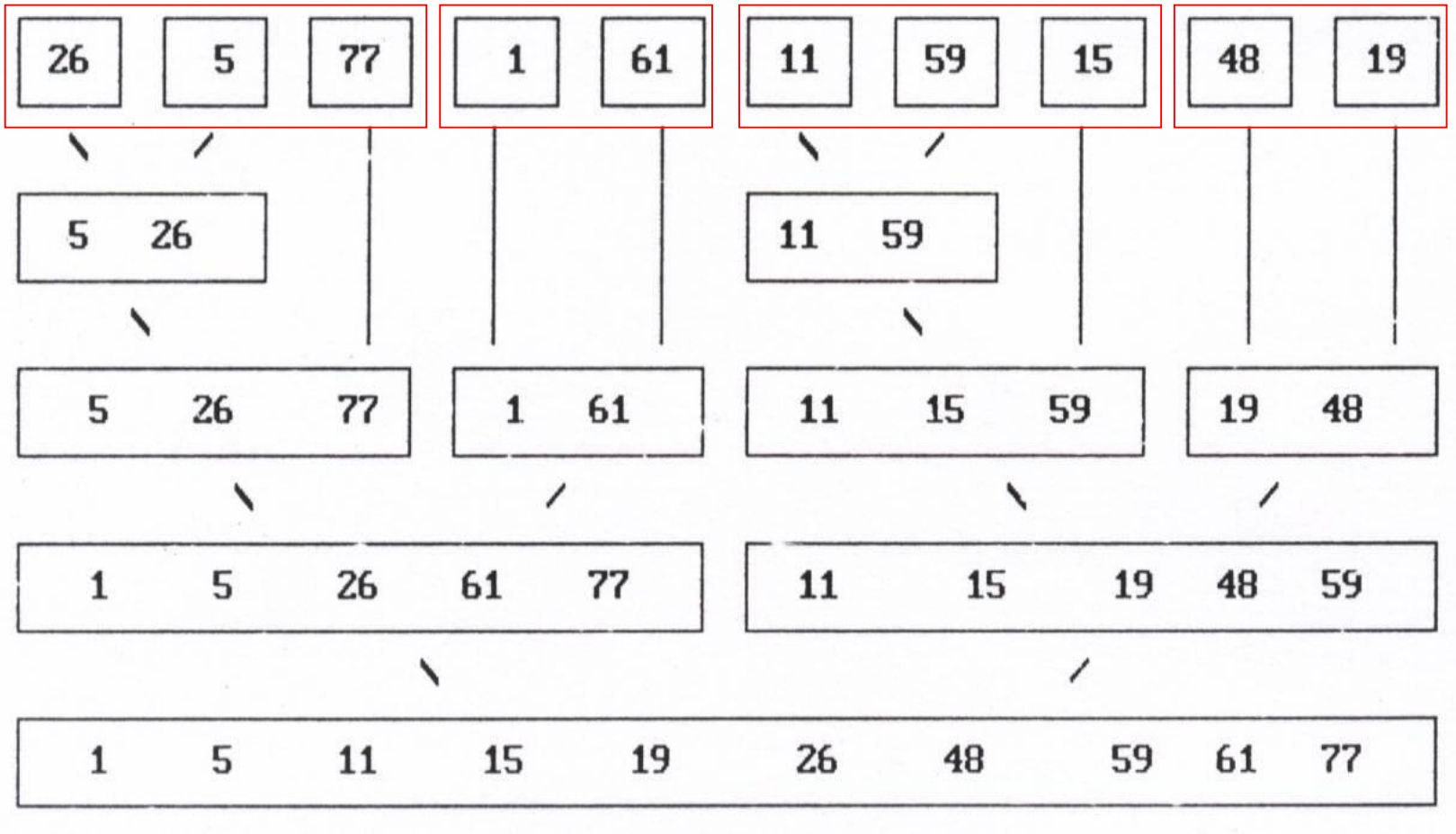
- مفهوم مرتب سازی ادغام بازگشتی



S.Najjar.G@Gmail.com

مرتب سازی ادغام به صورت بازگشتی (ادامه)

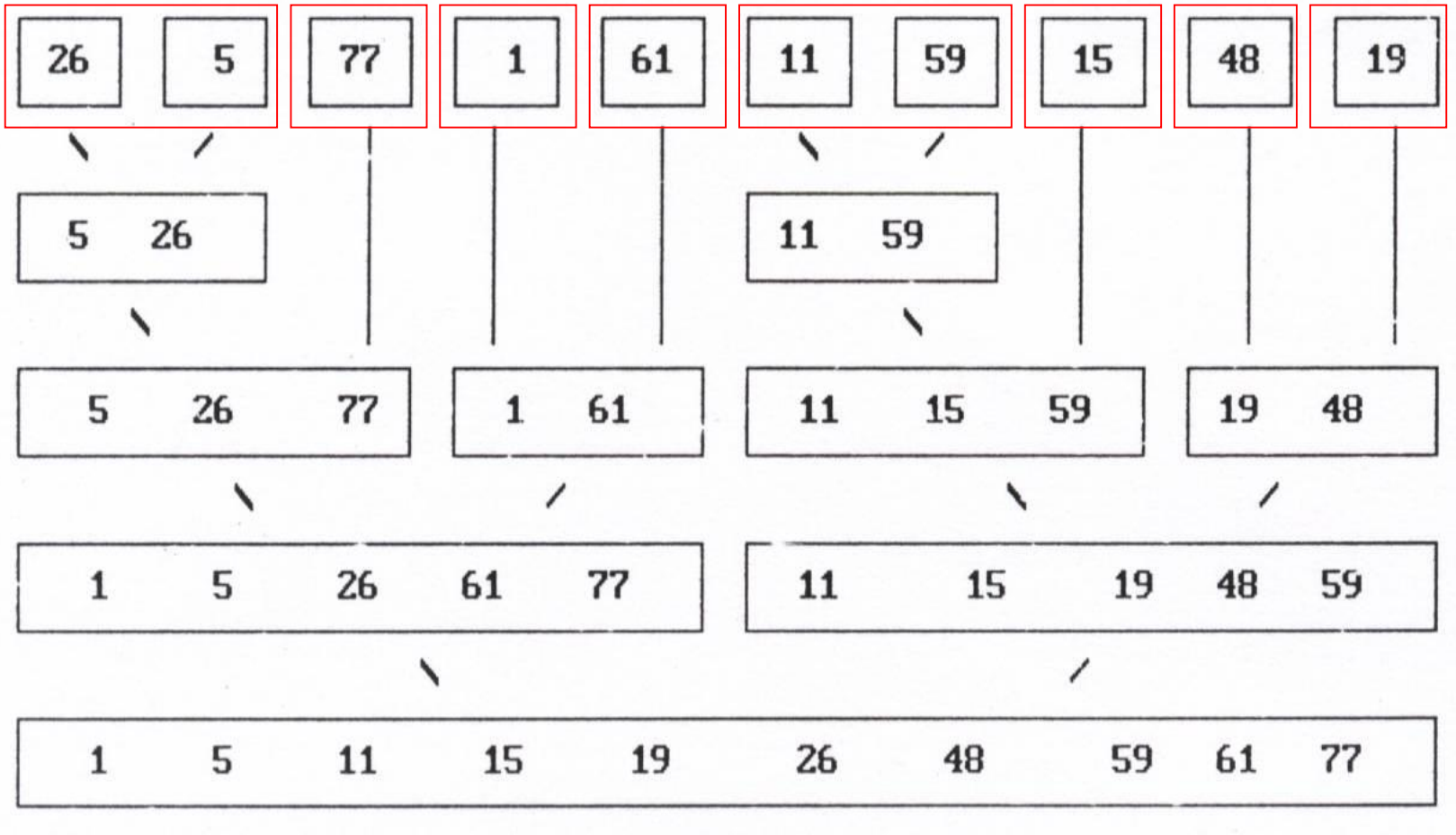
- مفهوم مرتب سازی ادغام بازگشتی



S.Najjar.G@Gmail.com

مرتب سازی ادغام به صورت بازگشتی (ادامه)

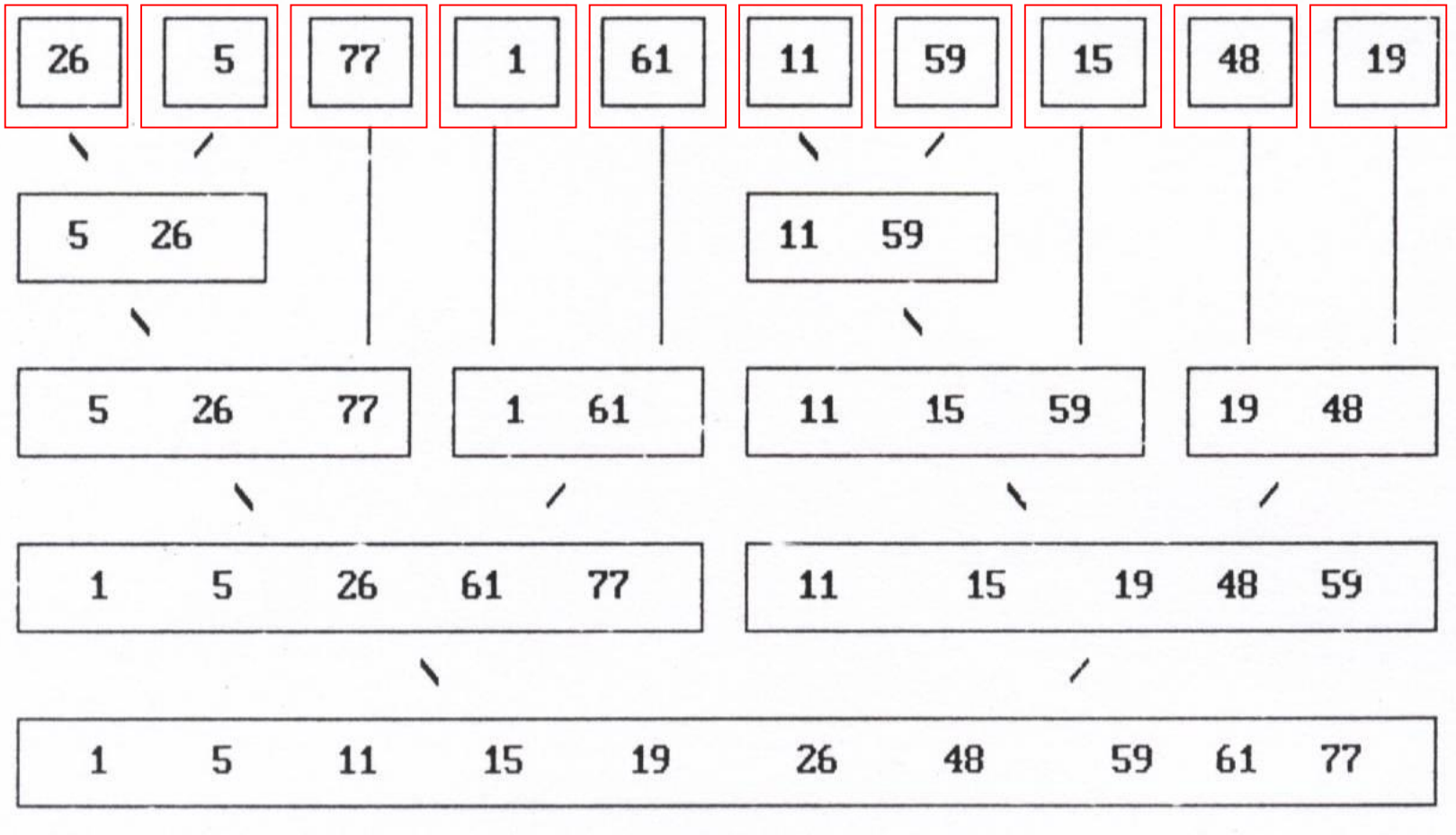
- مفهوم مرتب سازی ادغام بازگشتی



S.Najjar.G@Gmail.com

مرتب سازی ادغام به صورت بازگشتی (ادامه)

- مفهوم مرتب سازی ادغام بازگشتی



S.Najjar.G@Gmail.com

مرتب سازی ادغام به صورت بازگشتی (ادامه)

- *تابع ادغام*
  - اگر بخواهیم از تابع ادغامی که قبلا بیان شد برای ادغام زیرلیستهای مرتب شده از یک ارایه در ارایه دیگر استفاده کنیم آنگاه لازم است لیست ها کپی شوند.
  - برای جلوگیری از این کپی غیر ضروری زیرلیست ها، نمایش پیوندی برای آنها در نظر میگیریم.
  - برای هر رکورد یک فیلد link در نظر میگیریم که در ابتدا 1- است یعنی در ابتدا هر رکورد در زنجیری قرار دارد که فقط از خودش تشکیل شده است.
- *Listmerge*

فرض کنید first و second اشاره گرهایی به دو زنجیر از رکوردها باشند که هر یک از این زنجیرها به ترتیب غیر نزولی مرتب هستند. Listmerge(list, first, second) تابعی است که دو زنجیرکه از first و second شروع می شوند را در ارایه لیست ادغام میکند و عدد صحیحی را بر می گرداند که به شروع زنجیر حاصل از ادغام اشاره می کند.

S.Najjar.G@Gmail.com

Listmerge تابعی است که مکان شروع دو لیست مرتب غیر نزولی را می گیرد و عدد صحیحی را بر می گرداند که به شروع لیست حاصل از ادغام اشاره می کند.

در ابتدا برای linkفیلد است1-هر رکورد

n-1 تا 0چون رکوردها از قرار گرفته اند ما از برای ذخیره کردن list[n] استفاده می کنیمstart

```
int listmerge(element list[], int first, int second)
/* merge lists pointed to by first and second */
{
   int start = n;
   while (first != -1 && second != -1)
      if (list[first].key <= list[second].key) {
      /* key in first list is lower, link this element to
      start and change start to point to first */
         list[start].link = first;
         start = first;
         first = list[first].link;
      }
      else {
      /* key second list is lower, link this element into
      the partially sorted list */
         list[start].link = second;
         start = second;
         second = list[second].link;
      }
   /* move remainder */
   if (first == -1)
      list[start].link = second;
   else
      list[start].link = first;
   return list[n].link; /* start of the new list */
}
```

مرتب سازی ادغام به صورت بازگشتی (ادامه)

lower=

upper=

middle=

```
int rmerge(element list[], int lower, int upper)
{
   int middle;
   if (lower >= upper)
      return lower;
   else {
      middle = (lower + upper) / 2;
      return listmerge(list,rmerge(list,lower,middle),
                           rmerge(list,middle+1,upper));
   }
}
```

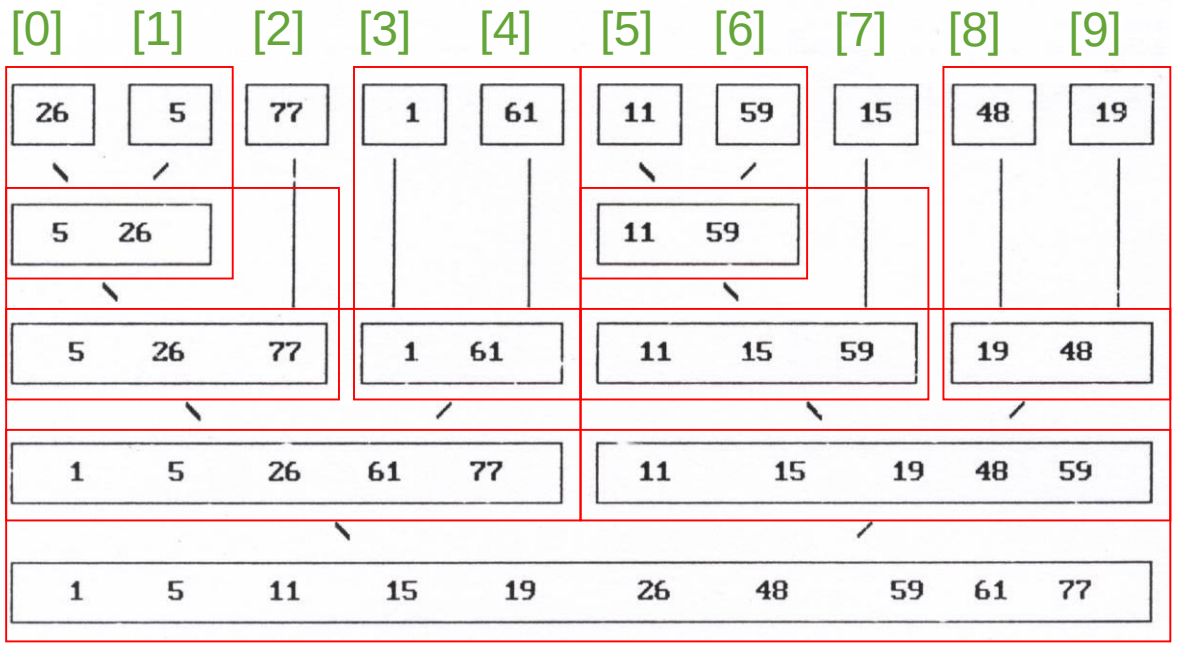
start = rmerge(list, 0, n-1);

= 0

0

rmerge تابع
list[lower], …, listلیست
list[upper] را
مرتب می کند

list



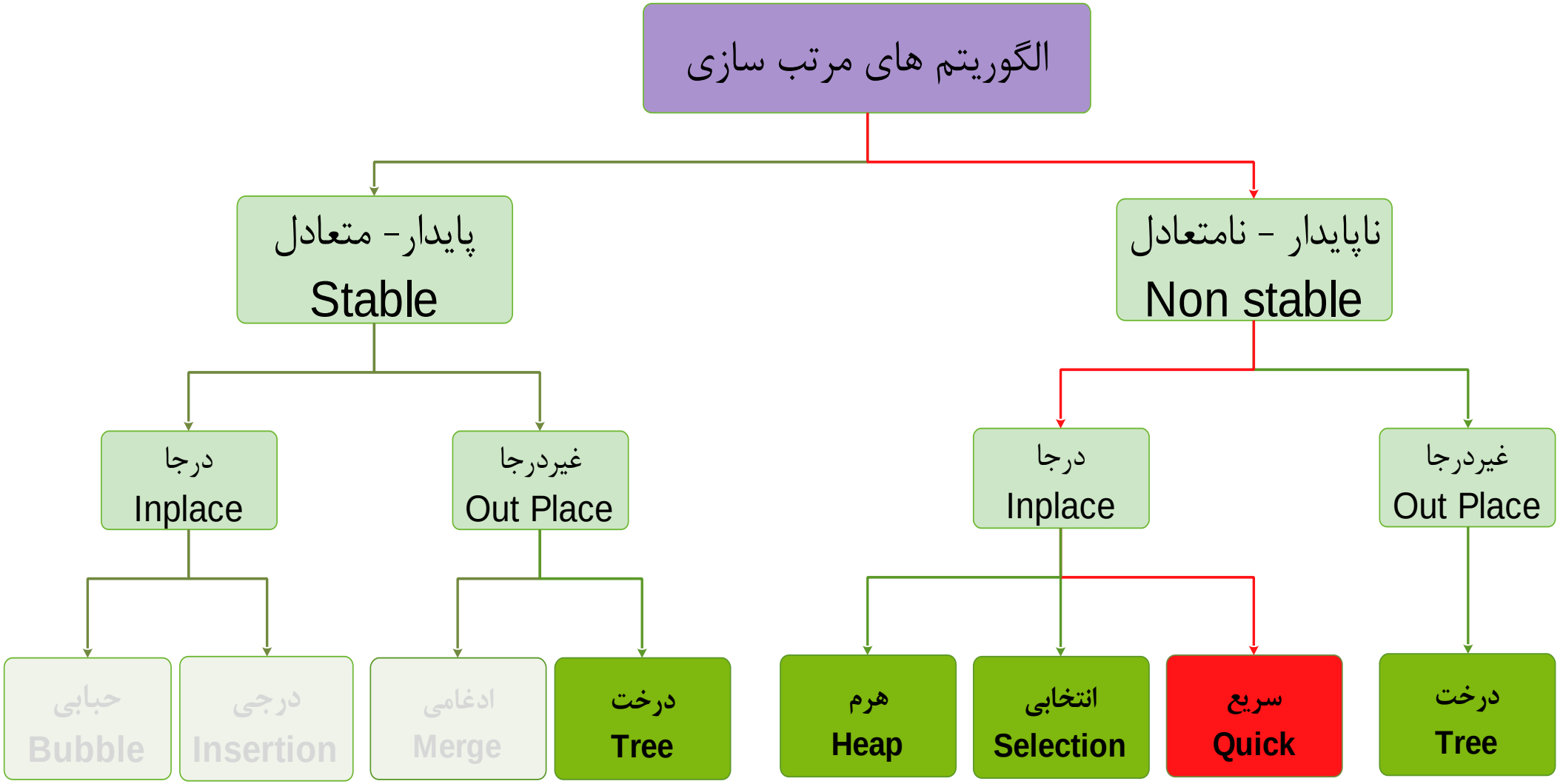
S.Najjar.G@Gmail.com

مرتب سازی ادغام

- پیچیدگی محاسباتی الگوریتم ادغام در بدترین حالت و حالت میانگین برابر با O(n log n) است.
- این الگوریتم نیازمند فضای اضافی متناسب با تعداد رکوردهایی که باید مرتب شوند است.
- مرتب سازی ادغام پایدار است.
- زمان مرتب سازی ادغام روی یک لیست مرتب O(n log n) است.

S.Najjar.G@Gmail.com

دسته‌بندی برخی از الگوریتم‌های مرتب‌سازی



S.Najjar.G@Gmail.com

مرتب سازی سریع

- بهترین رفتار را در حالت میانگین بین روشهای مرتب سازی دارد که مطالعه می کنیم. بر اساس جابجا کردن زیاد عناصر عمل می کند
- هر بار یک عنصر محوری pivot انتخاب شده و سایر عناصر در بردار به صورتی جابجا می شوند که کلیه عناصر کوچکتر از آن در یک طرف و عناصر بزرگتر از آن در طرف دیگر عنصر محوری قرار گرفته و سپس دو بردار دو طرف عنصر محوری به همین طریق مرتب می شوند (به صورت بازگشتی)

```
Quick (element *list, int left, int right)
{
    int    j;
    if (left< right)
    {
        j=partition (list,  left,  right);
        Quick (list, left, j-1);
        Quick (list,j+1, right);
    }
}
```

S.Najjar.G@Gmail.com

مرتب سازی سریع (ادامه)

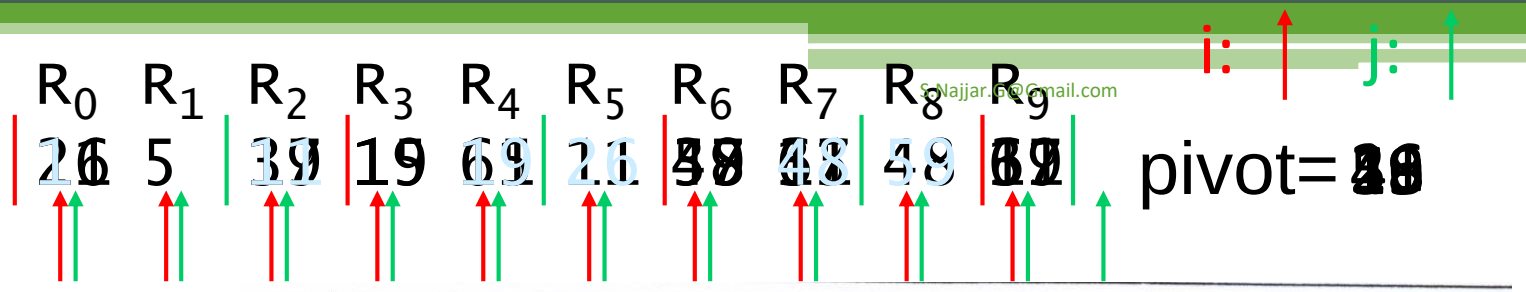
R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
26	5	37	1	61	11	59	15	48	19
11	5	19	1	15	26	59	61	48	37
1	5	11	19	15	26	59	61	48	37
1	5	11	19	15	26	59	61	48	37
1	5	11	15	19	26	59	61	48	37
1	5	11	15	19	26	48	37	59	61
1	5	11	15	19	26	37	48	59	61
1	5	11	15	19	26	37	48	59	61

S.Najjar.G@Gmail.com

مرتب سازی سریع (ادامه)

- در تابع partition دو متغیر i و j به ترتیب به حد پایین و بالای آرایه اشاره می کنند. در هر نقطه اجرا عناصری که بعد از j قرار دارند بزرگتر یا مساوی pivot و عناصری که قبل از i قرار دارند کوچکتر یا مساوی pivot هستند.

- i و j به صورت زیر به به طرف هم حرکت می کنند:
  - به i افزوده می شود تا زمانی که list[i]>pivot شود.
  - j کاهش می یابد تا زمانی که list[j]<pivot شود.
  - اگر i<j آنگاه list[i] با list[j] جابجا می شود.



left

right

j=partition (list, left, right);

```
void quicksort(element list[], int left, int right)
{
    int pivot,i,j;
    element temp;
    if (left < right) {
        i = left;     j = right + 1;
        pivot = list[left].key;
        do {
            do
                i++;
            while (list[i].key < pivot);
            do
                j--;
            while (list[j].key > pivot);
            if (i < j)
                SWAP(list[i],list[j],temp);
        } while (i < j);
        SWAP(list[left],list[j],temp);
        quicksort(list,left,j-1);
        quicksort(list,j+1,right);
    }
}
```

مرتب سازی سریع(ادامه)

- تحلیل مرتب سازی سریع
  - فرض کنیم هرگاه یک عنصر در مکان درست خود قرار می گیرد طول زیرلیست چپ و راست برابر باشد، آنگاه T(n) زمان لازم برای مرتب کردن یک لیست به طول n به صورت زیر به دست می آید:
  - زمان لازم برای قرار دادن یک رکورد در لیستی به طول n — O(n)
  - زمان لازم برای مرتب کردن زیر لیست چپ — T(n/2)
  - زمان لازم برای مرتب کردن زیر لیست راست — T(n/2)

$$T(n)<=cn+2T(n/2) \text{ for some } c$$
$$<=cn+2(cn/2+2T(n/4))$$
$$\ldots$$
$$<=cn\log_2 n+nT(1)=O(n\log n)$$

S.Najjar.G@Gmail.com

مرتب سازی سریع (ادامه)

- مرتب سازی سریع از نوع الگوریتم های تقسیم و غلبه است که با شکستن بازه به چند بازه مساله را با مرتبه اجرایی nlogn حل می کند.
- در بردارهای مرتب بدترین عملکرد و در بردارهای نامرتب بهترین عملکرد را نشان می دهد.

پیچیدگی	بهترین حالت	حالت متوسط	بدترین حالت
	$O(nlogn)$	$O(nlogn)$	$O(n^2)$

- از نظر میانگین زمان اجرا بهترین روش مرتب سازی داخلی است.
- روش مرتب سازی سریع پایدار نیست.

مرتب سازی سریع (ادامه)

- **قضیه:** فرض کنید $T_{avg}(n)$ زمان لازم برای مرتب سازی یک ارایه با n رکورد باشد، آنگاه ثابتی مانند k وجود دارد به طوریکه برای $n \geq 2$، $T_{avg}(n) \leq knlogen$

- حافظه مورد نیاز

  برای مرتب سازی سریع نیاز به فضای پشته داریم (بر خلاف مرتب سازی درجی که فقط به یک حافظه اضافی نیاز داشت).

  بهترین حالت و حالت متوسط

  در طول اجرا لیست ها به دو قسمت مساوی تقسیم شوند

  حداکثر عمق بازگشتی logn حافظه مورد نیازپشته O(logn)

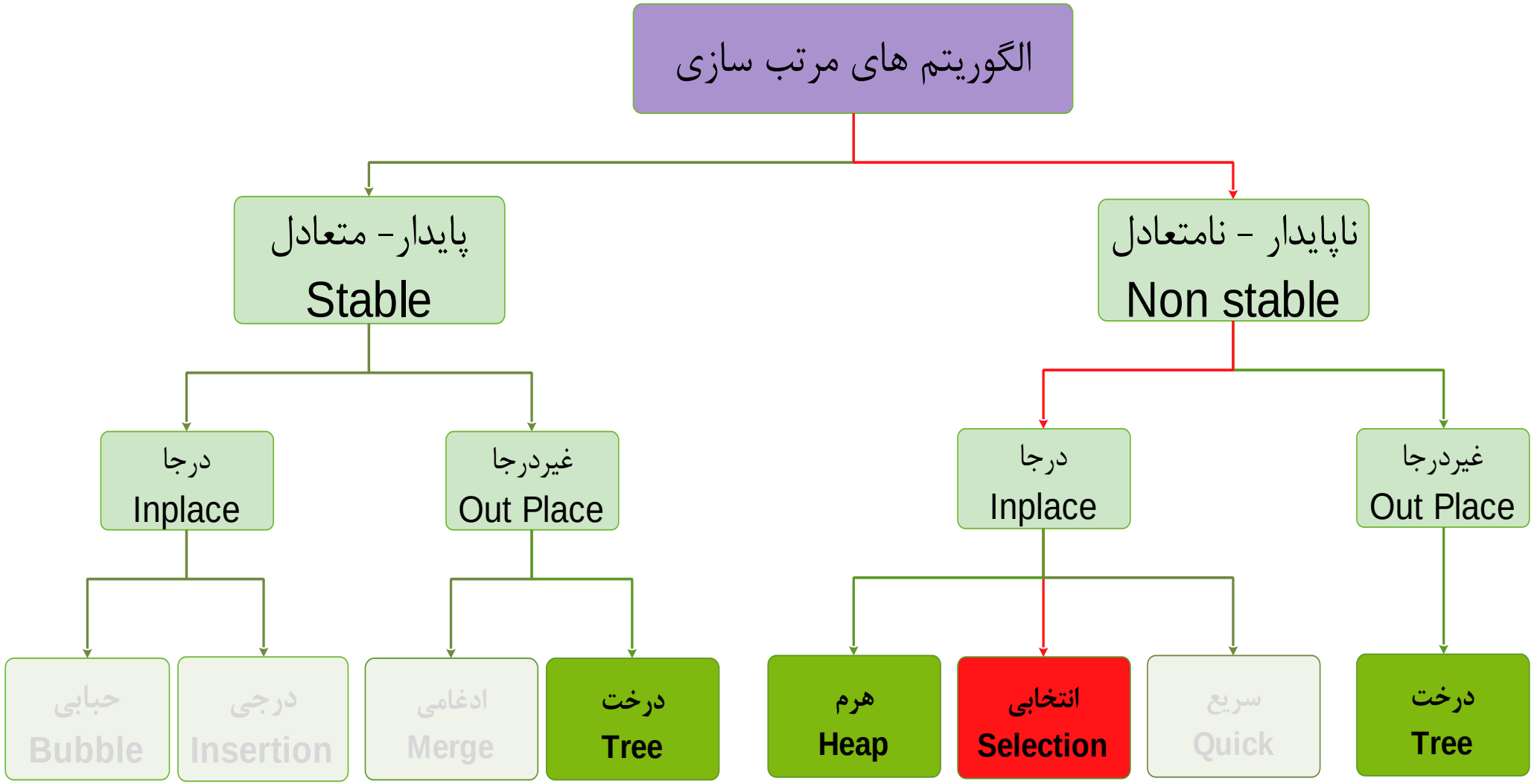
  بدترین حالت

  در هر سطح از اجرای بازگشتی، لیست به دو زیر لیست چپ (راست) به طول n-1 و زیر لیست راست (چپ) به طول صفر تجزیه شود.

  حداکثر عمق بازگشتی n حافظه مورد نیاز پشته O(n)

S.Najjar.G@Gmail.com

دسته‌بندی برخی از الگوریتم‌های مرتب‌سازی



S.Najjar.G@Gmail.com

مرتب سازی انتخابی (selection sort)

در این روش در هر مرتبه طی کردن طول بردار محل درست یک عنصر پیدا شده و سپس با یک جابجایی در جای خود قرار می گیرد.

```
for i=n to 1 do
begin
    large =x[1];
    index=1;
    for j=1 to i do
        if x[j]> large
        begin
                large =x[j];
                index=j;
        end
    x[index]= x[i];
    x[i]=large;
end
```

S.Najjar.G@Gmail.com

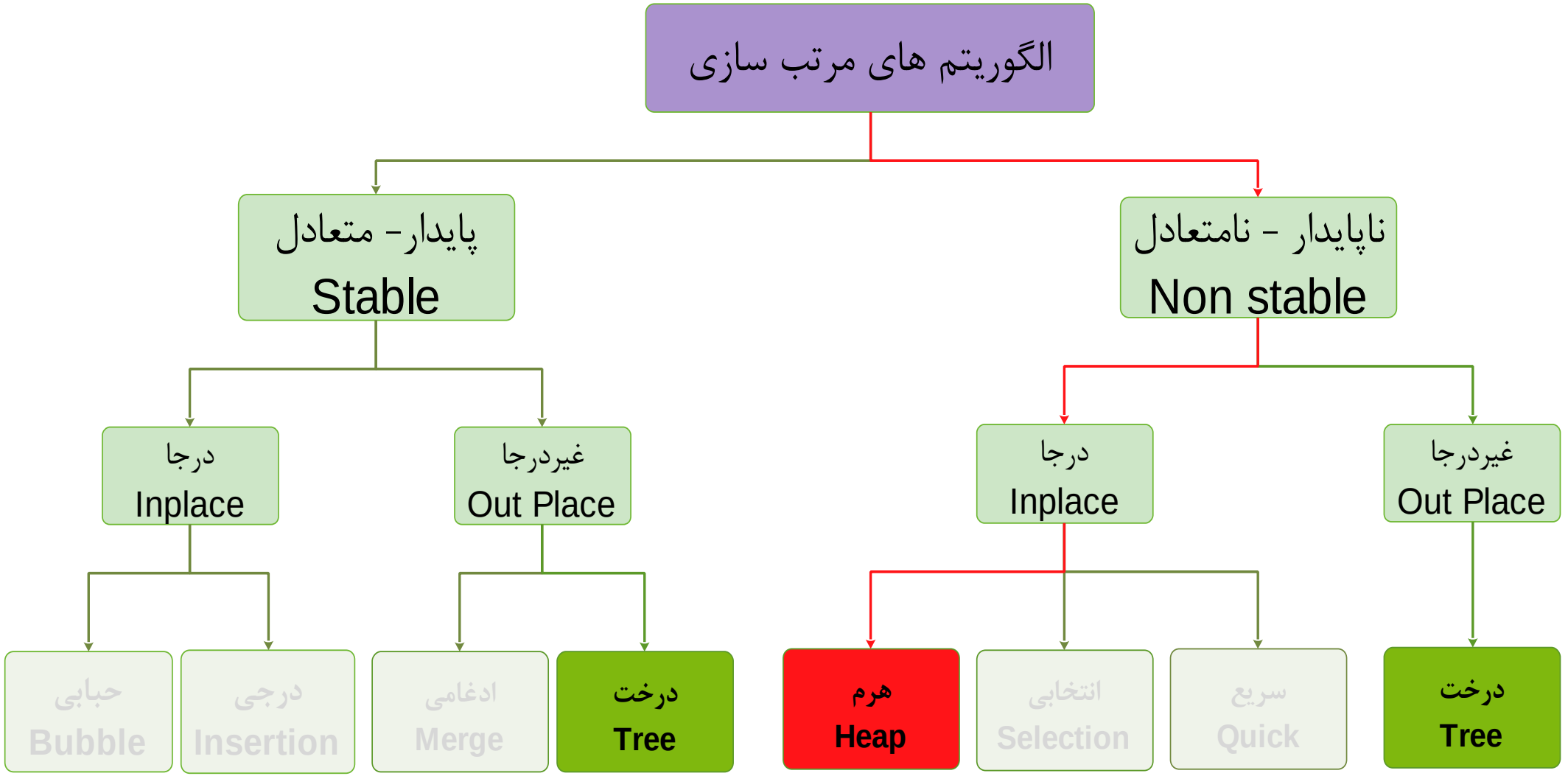
مرتب سازی انتخابی (selection sort) (ادامه)

- تحلیل مرتب سازی انتخابی
  - مرتبه اجرایی مشابه مرتب سازی حبابی $O(n^2)$ است هر چند با تعوض کمتر کار می کند.

	بهترین حالت	حالت متوسط	بدترین حالت
پیچیدگی	$O(n^2)$	$O(n^2)$	$O(n^2)$

S.Najjar.G@Gmail.com

دسته‌بندی برخی از الگوریتم‌های مرتب‌سازی



S.Najjar.G@Gmail.com

مرتب سازی درخت نیمه مرتب Heap

- مشکلات مرتب سازی ادغام

- این الگوریتم نیازمند فضای اضافی متناسب با تعداد رکوردهایی که باید مرتب شوند است.
- چنانچه در مرحله ادغام از روشی استفاده شود که به میزان حافظه از O(1) نیاز داشته باشد نیاز حافظه کل الگوریتم به O(1) کاهش می یابد ولی در اینصورت الگوریتم حاصل به صورت قابل توجهی کندتر از نسخه اصلی خواهد بود.

- مرتب سازی Heap

- نیازمند تنها یک مقدار حافظه اضافی ثابت است.
- کمی کندتر از الگوریتم مرتب سازی ادغام با حافظه اضافی O(n) است.
- سریعتر از الگوریتم ادغام با حافظه اضافی O(1) است.
- بدترین حالت و حالت میانگین آن مشابه با مرتب سازی ادغام از O(n log n) است.
- این مرتب سازی ناپایدار است.

S.Najjar.G@Gmail.com

مرتب سازی درخت نیمه مرتب Heap (ادامه)

- ابتدا n رکورد به یک Heap خالی اولیه اضافه می شوند
- آنگاه رکوردها یک به یک از Heap استخراج می شوند.
- توابع حذف و اضافه کردن مرتبط با MaxHeap مستقیما مرتب سازی با زمان O(n log n) را نتیجه می دهد.
- با اینحال امکان ایجاد Heap ای با n رکورد به صورت سریعتر وجود دارد:
  - با استفاده از تابع adjust .
  - این تابع یک درخت دودویی T را دریافت می کند که زیر درخت های چپ و راست آن در خاصیت Heap صدق می کند اما ممکن است این خاصیت در ریشه اش وجود نداشته باشد. این تابع T را طوری تنظیم می کند که تمام درخت دودویی در خاصیت Heap صدق کند.

S.Najjar.G@Gmail.com

تابع adjust

```
void adjust(element list[], int root, int n)
{
   int child,rootkey;
   element temp;
   temp = list[root];
   rootkey = list[root].key;
   child = 2 * root;          /* left child */
   while (child <= n) {
      if ((child < n) &&
      (list[child].key < list[child+1].key))
         child++;
      if (rootkey > list[child].key)
      /* compare root and max. root */
         break;
      else {                  /* move to parent */
         list[child / 2] = list[child];
         child *= 2;
      }
   }
   list[child/2] = temp;
}
```

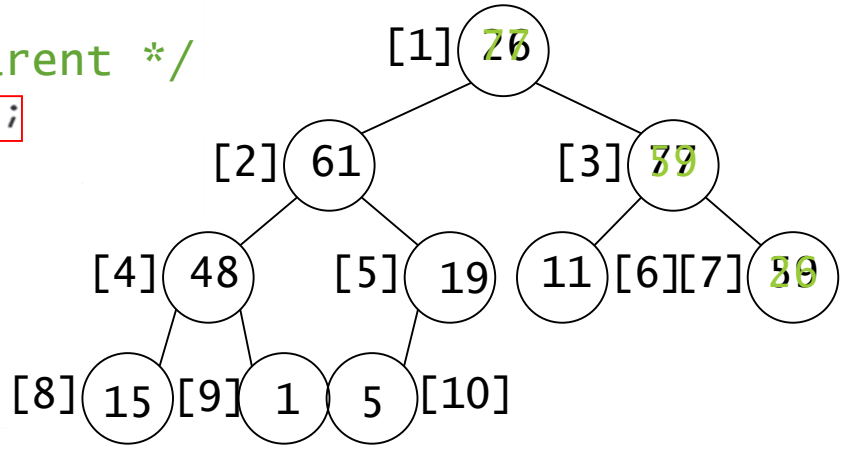
root = 1

n = 10

rootkey = 26

child = 2 4

اگر عمق درخت با ریشه root ، d باشد انگاه حلقه while بار اجرا می شود، dحداکثر است.O(d)بنابراین تابع از



S.Najjar.G@Gmail.com

مرتب سازی درخت نیمه مرتب Heap

- ابتدا با احضار مکرر تابع adjust یک MaxHeap ایجاد می کنیم.
- در ادامه برای مرتب سازی لیست، n-1 گذر بر روی لیست انجام می شود.
- در هر گذر ، اولین رکورد در Heap با آخرین رکورد تعویض می گردد. آنگاه اندازه Heap را کاهش می دهیم و Heap را از نو تنظیم می کنیم.

  - از آنجا که اولین رکورد همیشه شامل بزرگترین کلید است، این رکورد (با بزرگترین کلید) را در محل n ام قرار می دهیم. در گذر دوم رکورد با دومین کلید بزرگ را در موقعیت n-1 قرار می دهیم و در i امین گذر ، رکورد با i امین کلید بزرگ را در موقعیت n - i + 1 قرار می دهیم.

S.Najjar.G@Gmail.com

مرتب سازی درخت نیمه مرتب Heap (ادامه)

```
void heapsort(element list[], int n)
/* perform a heapsort on the array */
{
   int i,j;
   element temp;

   for (i = n/2; i > 0; i--)
      adjust(list,i,n);
   for (i = n-1; i > 0; i--) {
      SWAP(list[1],list[i+1],temp);
      adjust(list,1,i);
   }
}
```

(recreate heap)

(top-down) Sort list

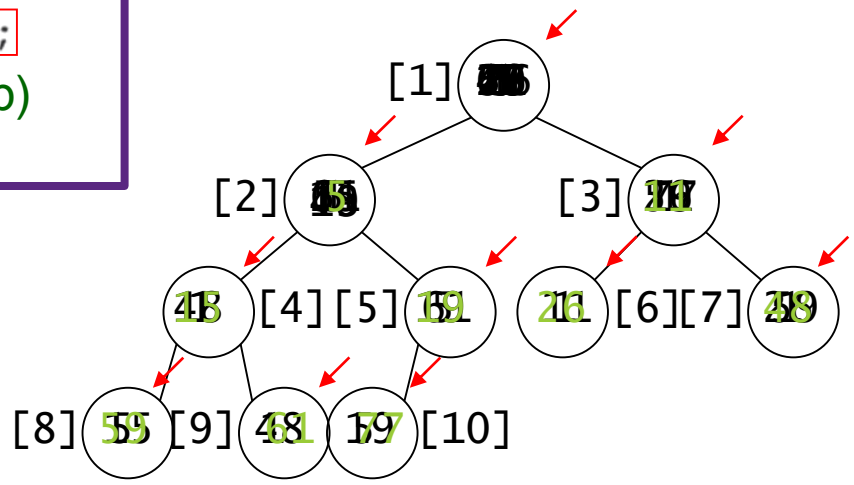
n = 10

j =

(bottom-up) Convert list into a heap

ترتیب صعودی

(max heap)



S.Najjar.G@Gmail.com

مرتب سازی درخت نیمه مرتب Heap (ادامه)

- تحلیل تابع heapsort

فرض کنید $2^{k-1} \le n < 2^k$ $\Longleftarrow$ درخت k سطح دارد.

تعداد گره های سطح i ام کوچکتر یا برابر با 2^{i-1}

- در مرحله ساختن Heap

حلقه for اول برای هر گره ای که (حداقل) یک بچه دارد اجرا می شود.

مجموع زمان مورد نیاز برای حلقه برابر مجموع تعداد گره های هر سطح در حداکثر فاصله ای است که گره می تواند حرکت کند که بیشتر از عبارت زیر نیست

$$\sum_{1 \le i \le k} 2^{i-1}(k-i) = \sum_{1 \le i \le k-1} 2^{k-i-1} i \le n \sum_{1 \le i \le k-1} i/2^i \le 2n = O(n)$$

حداکثر فاصله ای که گره در i سطح می تواند حرکت کند

احد اکثر تعداد گره های سطح

کلیه سطوح

S.Najjar.G@Gmail.com

مرتب سازی درخت نیمه مرتب Heap (ادامه)

- تحلیل تابع heapsort ادامه
- در مرحله مرتب کردن لیست

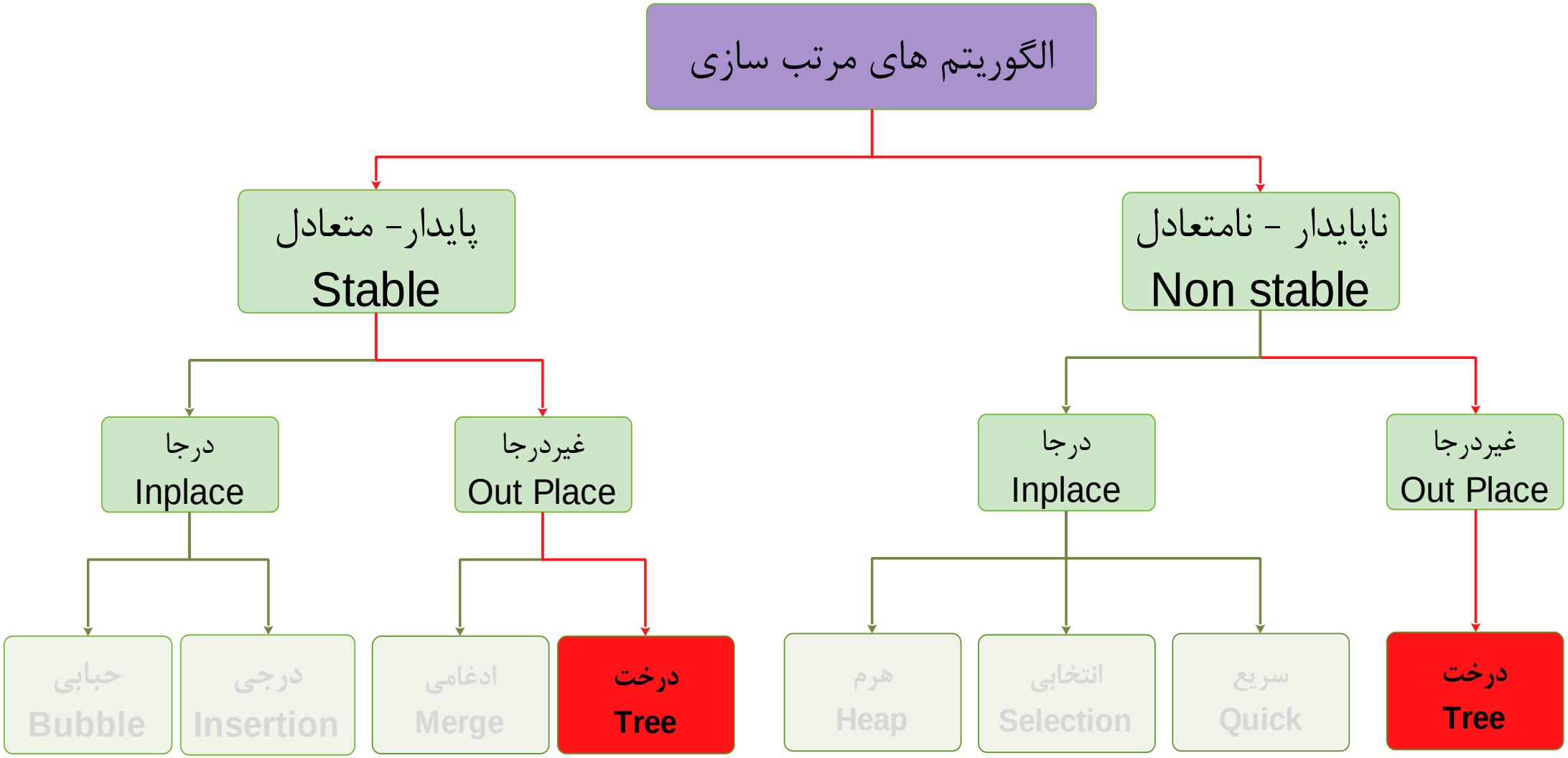
در حلقه for، n-1 احضار adjust با حداکثر عمق $k = \lceil \log_2(n+1) \rceil$

صورت می گیرد. از این رو زمان اجرای حلقه O(n log n) است.

- کل زمان اجرا O(n log n) است.
- صرف نظر از متغیرهای ساده، تنها حافظه اضافی مورد نیاز حافظه مربوط به یک رکورد برای انجام جابجایی در مرحله دوم است

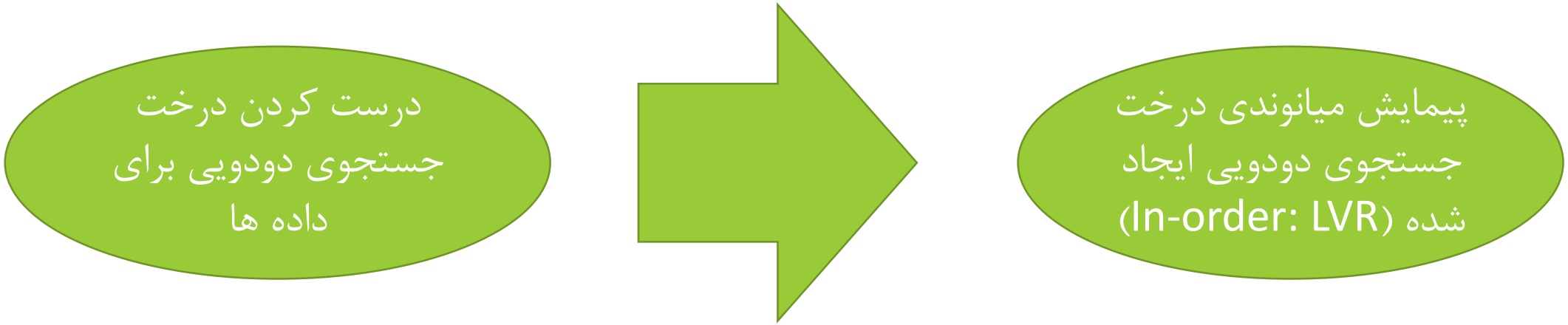
S.Najjar.G@Gmail.com

دسته‌بندی برخی از الگوریتم‌های مرتب‌سازی



S.Najjar.G@Gmail.com

مرتب‌سازی درختی



S.Najjar.G@Gmail.com

مرتب‌سازی درختی (ادامه)

- تحلیل مرتب‌سازی درختی
  - پیچیدگی در این الگوریتم بستگی به ارتفاع درخت (h) و تعداد داده (n) دارد = O(nh)
  - چون کمترین مقدار ارتفاع برابر h=logn و بیشترین مقدار h=n خواهد بود پس:

پیچیدگی	بهترین حالت	حالت متوسط	بدترین حالت
	$O(nlogn)$	$O(nlogn)$	$O(n^2)$

S.Najjar.G@Gmail.com

پروژه‌های پیشنهادی

- پیاده‌سازی الگوریتم مرتب‌سازی Shell
- پیاده‌سازی الگوریتم مرتب‌سازی مبنا (Radix Sort)
- پیاده‌سازی الگوریتم مرتب‌سازی سریع (Quick Sort)
- پیاده‌سازی الگوریتم مرتب‌سازی هرمی (Heap Sort)
- پیاده‌سازی الگوریتم مرتب‌سازی انتخابی (Selection Sort)
- پیاده‌سازی الگوریتم مرتب‌سازی درجی (Insertion Sort)
- پیاده‌سازی الگوریتم مرتب‌سازی ادغامی (Merge Sort)