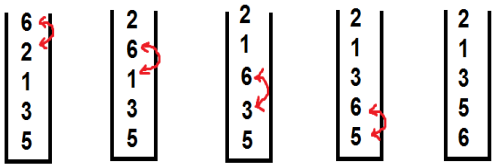
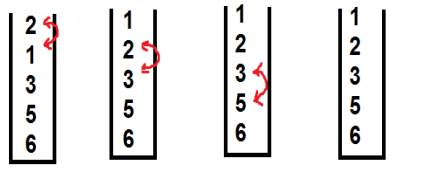
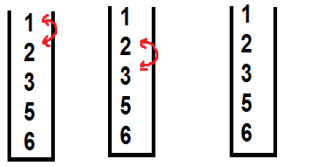
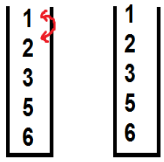
5.5-مرتب سازی حبابی (Bubble Sort)

مرتب سازی حبابی یک الگوریتم مرتب سازی است که لیست را پیمایش میکند و اگر عنصری از عنصر بعد از خود بزرگتر بود جای آن دو را با هم عوض میکند و در پیمایش بعد یکی از تعداد عناصرارایه که بررسی میشوند کم میکند. به ازای هر عنصر بجز عنصر اخر،یک پیمایش صورت میگیرد(زیرا آخرین عنصر خود به خود مرتب میشود) که در آن ممکن است جابجایی انجام شود یا نشود.در این الگوریتم آرایه را به صورت عمودی تصور میکنیم و این الگوریتم از آن رو حبابی نامیده می‌شود که هر عنصر با عنصر زیری خود سنجیده شده و درصورتی که از آن بزرگ تر باشد جای خود را به آن می‌دهد و این کار ادامه می یابد تا کوچک‌ترین عنصر به بالای آرایه برسد و سایر عناصر نیز به ترتیب در جای خود قرار گیرند (این عمل همانند حرکت حباب به بالای مایع است. به این الگوریتم ، مرتب سازی سنگی نیز گفته میشود.چون عناصر بزرگتر به سمت پایین آرایه حرکت میکنند)









این الگوریتم از آن رو که برای کار با عناصر آن‌ها را با یکدیگر مقایسه میکند، یک مرتب سازی بر مبنای مقایسه است.

همچنین چون میزان استفاده از حافظه ی اضافی در آن O(1) است پس مرتب سازی حبابی درجا نیز است.

با فرض داشتن n عضو، در بدترین حالت $\frac{n(n-1)}{2}$ عمل لازم خواهد بود.

یک آرایه با مقادیر "6 ,2 ,1 , 3, 5" را در نظر بگیرید.آن را با استفاده از مرتب سازی حبابی مرتب می کنیم.

(6 , 2 , 1 , 3 , 5) ➔ (2 , 6 , 1, 3 ,5)

(2 , 6 , 1 , 3 , 5) ➔ (2 , 1 , 6 , 3 , 5)

(2 , 1 , 6 , 3 , 5) ➔ (2 , 1 , 3 , 6 , 5)

(2 , 1 , 3 , 6 , 5) ➔ (2 , 1 , 3 , 5 , 6)

(2 , 1 , 3 , 5 , 6) ➔ (1 , 2 , 3 , 5 , 6)

حالا آرایه مرتب شده است.همانطور که در شکل مشاهده میشود بعد از دومین بررسی در دور دوم جابجایی صورت نگرفته است.پس عملا عملیات مرتب سازی در آن مرحله پایان یافته است ولی طبق الگوریتم مقایسه کردن عناصر ادامه پیدا کرده است که بیهوده است.پس الگوریتم قابل بهینه سازی است.

```
void BubbleSort(int temp[], int len)
{
     int i, j, item;
     for(i=0;i<len-1;i++)
     {
          for(j=0;j<len-i-2;j++)
         {
               if(temp[j]>=temp[j+1])
               {
                    item=temp[j];
                    temp[j]=temp[j+1];
                    temp[j+1]=item;
               }
        }
     }
}
```

با حذف مساوی مشخص شده از کد، الگوریتم stable **میشود .پس مرتب سازی حبابی** stable **است.**

همانطور که مشاهده میشود هزینه ی کد در بدترین حالت بهترین حالت و حالت متوسط برابر($O(n^2)$ است. با تغییراتی جزیی در کد الگوریتم میتوان هزینه ی آن را در بهترین حالت به O(n) کاهش داد. اگر از مرحله ای به بعد هیچ جابجایی صورت نگرفت میتوان نتیجه گرفت که تمام عناصر مرتب شدند پس میتوان الگوریتم را متوقف کرد.

```
void BubbleSort(int temp[], int len)
{
     int i, j, item;
     bool stop=false;
     while (!stop && i<len-1)
     {
          stop=true;
          for(j=0;j<len-i-2;j++)
         {
               if(temp[j]>temp[j+1])
               {
                    item=temp[j];
                    temp[j]=temp[j+1];
                    temp[j+1]=item;
                    stop=false;
               }
        }
       i++;
     }
}
```

5.6-مرتب سازی درجی (Insertion Sort)

یک الگوریتم مفید برای مرتب کردن تعداد عنصرهای کم است که مبنای آن مقایسه است.ایده ی این مرتب سازی به این صورت است که در هر مرحله عنصر k اُم به k-1 عنصر مرتب شده ی قبل از خود insert میشود. Insert شدن به این معناست که عنصر را با هرکدام از عناصر مقایسه میکند و زمانی که جای مناسب آن را پیدا کرد آن را به عناصر وارد میکند. مزیت مرتب‌سازی درجی این است

که برای تشخیص مکان درست عنصر $(k + 1)$ام، فقط عناصر مورد نیاز را بررسی می‌کند .

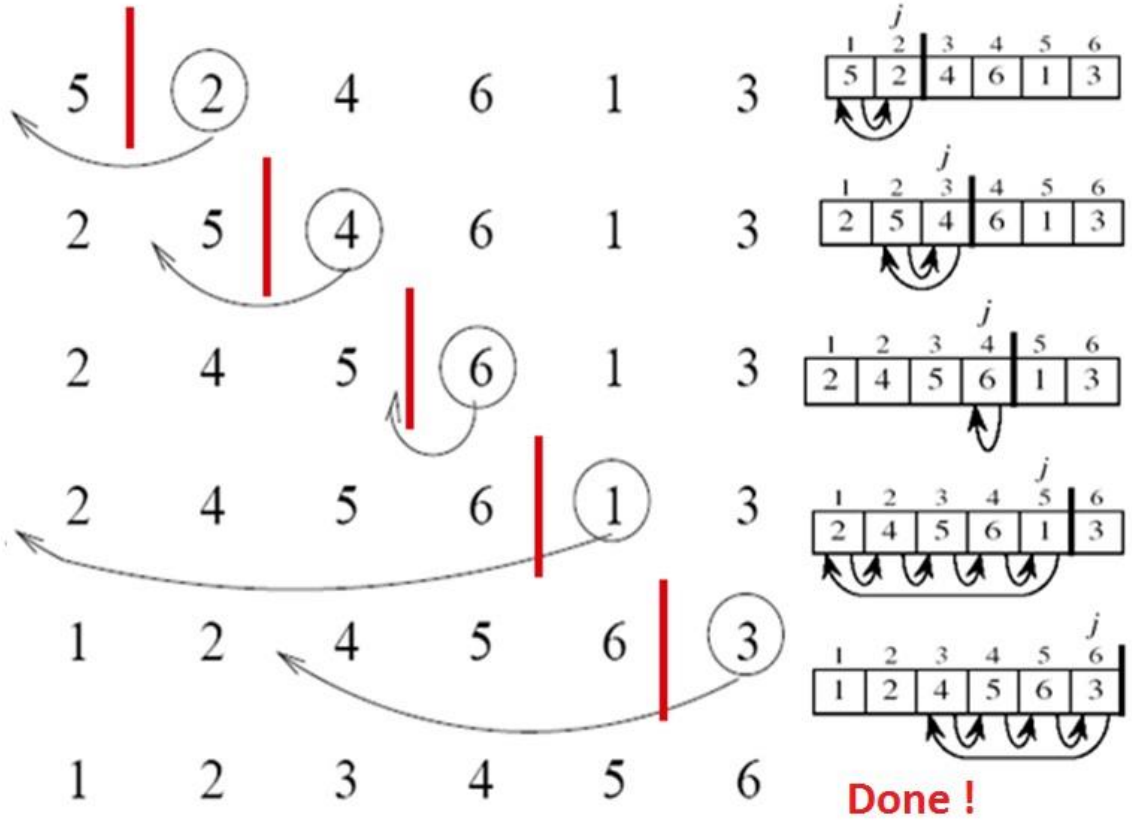
شبه کد ما یک آرایه A[1...n] به عنوان پارامتر می‌گیرد. A یک رشته به طول n است که باید مرتب شود. در کد، تعداد n عنصر در A با Length[A] مشخص می شود. عددهای ورودی در محل مرتب می‌شوند.(Sorted in place) در واقع عددها درون آرایه A مجدداً چیده می‌شوند، با حداکثر یک عدد از آنها که بیرون آرایه ذخیره می‌شود. (حداکثر یک حافظه اضافی) پس این مرتب سازی یک مرتب سازی درجا است.وقتی که insertion-sortتمام می‌شود آرایه ورودی A دارای رشته خروجی مرتب شده است.

```
Ins_sort(A[0,…,n-1])
{
        for (k=0 ;k<Length[A]-1 ;k++)
        {
                key=A[k];
                j=k;
                while (j>0 & A[j] >=key)
                {
                        A[j+1] = A[j];
                        j--;
                }
                A[j+1]=key;
        }
}
```

اگر علامت مساوی مسخص شده را از کد حذف کنیم ، الگوریتم stable میشود. پس مرتب سازی درجی stable است.

بهترین حالت زمانی است که آرایه از قبل مرتب شده باشد. در این حالت زمان اجرای الگوریتم از O(n)است: در هر مرحله اولین عنصر باقیمانده از لیست اولیه فقط با آخرین عنصر لیست مرتب شده مقایسه می‌شود. این حالت بدترین حالت برای مرتب‌ساز سریع (غیرتصادفی و با پیاده‌سازی ضعیف) است که زمان $O(n^2)$ صرف می‌کند.

بدترین حالت این الگوریتم، زمانی است که آرایه به صورت معکوس مرتب شده باشد. در این حالت هر اجرای حلقه داخلی، مجبور است که تمام بخش مرتب شده را بررسی کرده و انتقال دهد. در این زمان اجرای الگوریتم، مثل حالت متوسط، دارای زمان اجرای $O(n^2)$ است که باعث می‌شود استفاده از این الگوریتم برای مرتب‌سازی تعداد داده‌های زیاد، غیرعملی شود. گرچه حلقه داخلی مرتب‌ساز درجی، خیلی سریع است و این الگوریتم را به یکی از سریع‌ترین الگوریتم‌های مرتب‌سازی، برای تعداد داده‌های کم (معمولاً کمتر از ۱۰)، تبدیل می‌کند. در بدترین حالت و در حالت متوسط هزینه اجرای این تابع درجه 2 می شود و برابر با $O(n^2)$ است.



5.7-مرتب سازی انتخابی (Selection Sort)

مرتب‌سازی انتخابی یکی از انواع الگوریتم مرتب‌سازی می‌باشد که جزو دسته ی الگوریتمهای مرتب‌سازی مبتنی بر مقایسه‌است. این الگوریتم دارای پیچیدگی زمانی از درجه ی $O(n^2)$ است که به همین دلیل اعمال آن روی مجموعه ی بزرگی از اعداد کارا به نظرنمی رسدو به طور عمومی ضعیفتر از نوع مشابهش که مرتب‌ساز درجی است عمل می‌کند. این مرتب‌سازی به دلیل سادگی اش قابل توجه‌است. این الگوریتم اینگونه عمل می‌کند: ابتدا کوچکترین عنصر مجموعه اعداد را یافته با اولین عدد جابجا می‌کنیم. سپس دومین عنصر کوچکتر را یافته با دومین عدد جابجا می‌کنیم و این روند را برای $n-1$ عدد اول تکرار می‌کنیم. پس درکل $n-1$ بار عملیات *minimum* گیری انجام میشود. (عنصر n ام خود به خود در جای مناسب قرار میگیرد.)در حقیقت در هر مرحله ما لیست خود را به دو بخش تقسیم می‌کنیم. زیرلیست اول که قبلاً مرتب کرده‌ایم و سایر اعضای لیست که هنوز مرتب نشده‌است.

برای مثال آرایه ی روبرو را با این الگوریتم مرتب کنیم.

2	8	4	1	7

در مرحله ی اول کل لیست بررسی شده و کوچکترین عنصر با عنصر اول لیست جا بجا میشود.

1	8	4	**2**	7

در مرحله ی بعد پیمایش از عنصر دوم لیست شروع شده و کوچکترین عنصر با عنصر دوم لیست جابجا میشود.پیمایش از عنصر دوم شروع میشود زیرا در مرحله ی قبل عنصر اول به عنوان کوچکترین عنصر انتخاب شده بود.

1	**2**	4	**8**	7

همین روند ادامه پیدا میکند تا تمامی عناصر در جای درستی قرار گیرند.

1	2	4	**7**	**8**

به این ترتیب آرایه مرتب میشود.

شبه کد زیر کارکرد این الگوریتم را نشان میدهد.

```
Void selection_sort (A[0,…,n-1])
{
        for (k=0;k<n-2;k++)
        {
                min=A[k];
                minidix=k;
                for (j=k+1;j<n-1;j++)
                {
                        if (A[j]<=min)
                        {
                                minidx=j;
                                min=A[j];
                        }
                }
                Swap(A[minidx],A[k]);
        }
}
```

هزینه ی این الگوریتم از رابطه ی روبرو محاسبه میشود:

$$T(n) = (n - 1) + (n - 2) + (n - 3) + ... + 2 + 1 = n (n - 1) / 2$$

که از مرتبه $\Theta (n^2)$ **است.**

پیچیدگی زمانی اجرای این الگوریتم بر اساس محاسبات فوق در بدترین حالت(n^2) Θاست. با توجه به قطعه کد نوشته شده، ترتیب عناصر تغییری در عملکرد آن اینجا نمی‌کند. یعنی این الگوریتم برای داده‌های کاملا مرتب، نامرتب تصادفی و مرتب معکوس به یک ترتیب عمل کرده و تمام مقایسه‌های محاسبه شده در رابطه فوق را انجام می‌دهد. بنابراین پیچیدگی این الگوریتم در بهترین حالت و حالت منوسط نیز (n^2) Θاست.

مرتب‌سازی انتخابی یک روش مرتب‌سازی درجا است. یعنی عملیات مرتب‌سازی به در داخل خود لیست و بدون نیاز به حافظه کمکی بزرگ انجام می‌گیرد.

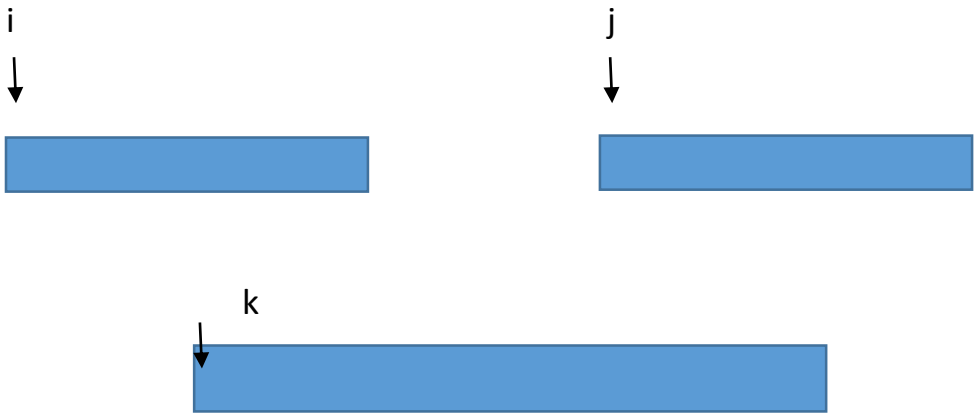
در پیاده‌سازی مرتب‌سازی انتخابی به روش فوق، اگر دو عنصر با مقدار کمینه داشته باشیم، دومی انتخاب شده و به ابتدای لیست منتقل می‌شود. در نتیجه ترتیب آنها به هم می‌خورد. بنابراین این پیاده‌سازی روش پایدار نیست. در روش پایدار ترتیب عناصر با مقدار یکسان تغییر نمی‌کند. اما اگر در مقایسه عناصر آرایه به جای < از <= استفاده کنید، مرتب‌سازی پایدار خواهد شد.

5.8-مرتب سازی ادغامی (Merge Sort)

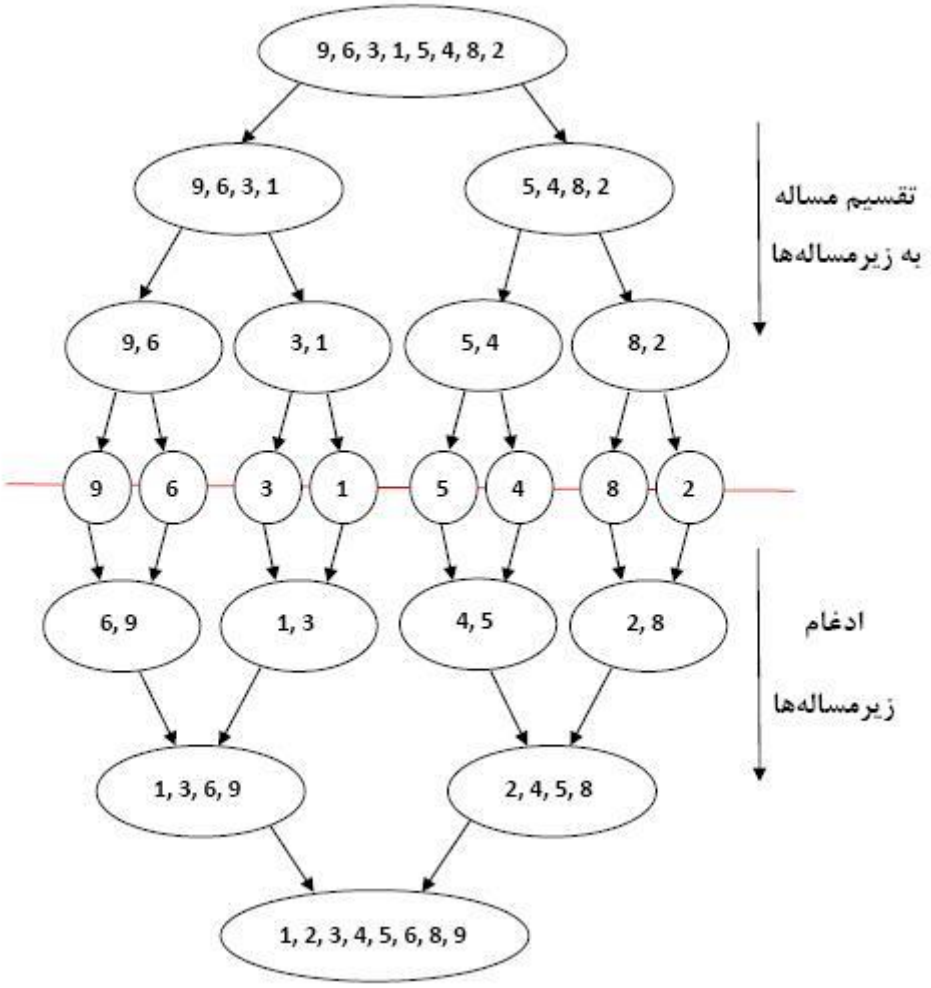
روش مرتب‌سازی ادغامی (Merge Sort) یک روش مرتب‌سازی مبتنی بر مقایسه است که از الگوریتمی بازگشتی پیروی میکند .

این مرتب سازی به این صورت عمل میکند که آرایه ی اولیه را به زیر آرایه هایی با اندازه ی برابر تقسیم میکند و هرکدام از این زیر آرایه ها را به صورت جداگانه مرتب میکند و در آخر دو زیر آرایه را با هم ادغام میکند و همین عملیات را تکرار میکند تا تمامی عناصر درجای خود قرار گیرند.

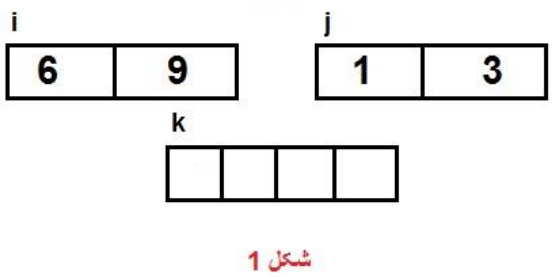
ادغام دو زیر آرایه به این صورت است که آرایه ای کمکی به اندازه ی مجموع تعداد اعضای 2 زیر آرایه میگیریم (پس الگوریتم درجا نیست).اشاره گر i به ابتدای زیر آرایه ی اول و اشاره گر j به ابتدای زیر ارایه ی دوم و اشاره گر k به ابتدای ارایه ی کمکی اشاره میکند.



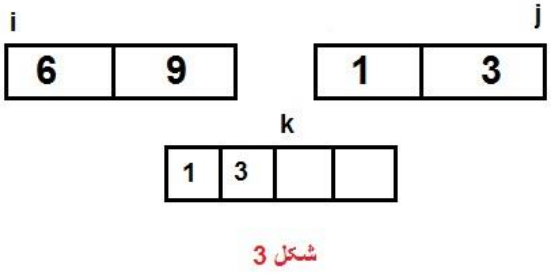
در هر مرحله عددی که اشاره گر i و اشاره گر j به آن ها اشاره میکنند با یکدیگر مقایسه میشوند و هر کدام کوچکتر بود (برای چینش صعودی) را در خانه ای از آرایه ی کمکی که k به آن اشاره میکند قرار میدهیم و اشاره گر k و همچنین اشاره گری که خانه ی آن به آرایه ی کمکی اضافه شده است را یک خانه به جلو میبریم .این کار را تا زمانی ادامه میدهیم یکی از اشاره گر های i یا j به پایان آرایه ی خود برسند. در آن زمان کل عناصر باقی مانده از آرایه ی دیگر را در آرایه ی کمکی ، به همان ترتیب قرار میدهیم .به مثال روبرو دقت کنید:



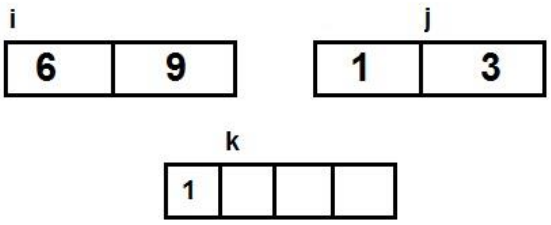
ادغام کردن دو زیر آرایه ی (9 و 6) و (3 و 1) به صورت زیر انجام میگیرد:



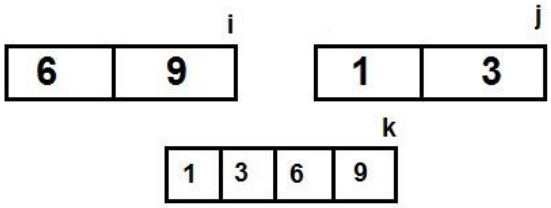
شکل 1



شکل 3



شکل 2



شکل 4

ادغام بقیه ی زیر آرایه ها نیز به همین صورت است.

شبه کد این مرتب سازی به صورت زیر است:

```
merge_sort(int A[], int l , int r )
{
        if (l ≥ r )
                return;
        m=(l+r)/2;
        merge_sort(A,l,m);
        merge_sort(A,m+1,r);
        merge(A,l,m,r);
}
```

```
merge(int A[] , int l , int m , int r )
{
        n=r-l+1;
        allocate B with size n;
        i=l;
        j=m+1;
        for(k=0 ; k<n-1 ; k++)
        {
                if (i>m)
                        copy right part;
                else if (j> r)
                        copy left part;
                else if (A[i] <= A[j])
                        B[k++]=A[i++];
                else if (A[i] > A[j] )
                        B[k++]=A[j++];
        }
}
```

اگر در این الگوریتم درحالت تساوی ،همواره اولویت به اندیس i داده شود، مرتب سازی پایدار میشود.

هزینه ی الگوریتم مرتب سازی ادغامی را به کمک رابطه بازگشتی زیر بیان می کنیم:

$$T(n) = \begin{cases} \theta(1), & if\ n = 1 \\ 2\,T\left(\frac{n}{2}\right) + \theta(n), & if\ n > 1 \end{cases}$$

که در نهایت به $T(n) = \theta\ (n \lg n)$ منجر خواهد شد.

پیچیدگی زمانی اجرای الگوریتم در تمامی حالات (Θ(n log n) است. چرا که این الگوریتم تحت هر شرایطی آرایه را به دو قسمت کرده و مرتب‌سازی را انجام می‌دهد.

پیچیدگی حافظه مصرفی بستگی به روش پیاده‌سازی مرحله ادغام دارد، که تا (O(n افزایش می‌یابد. پیاده‌سازی درجای این الگوریتم حافظه مصرفی مرتبه (1) Θ دارد. اما اجرای آن در بدترین حالت زمانبر است.((Θ(n^2))

الگوربتم مرتب‌سازی ادغامی با پیاده‌سازی فوق یک روش پایدار است. چنین الگوریتمی ترتیب عناصر با مقدار یکسان را پس از مرتب‌سازی حفظ می‌کند.