به نام خدا

جزوه جلسه ششم درس برنامه نویسی با متلب

1- مقدمه ای بر مرتب‌سازی

مرتب‌سازی به معنای آن است که عناصر درون یک آرایه یا بردار را به صورت صعودی یا نزولی کنار هم قرار دهیم. در الگوریتم‌های مرتب سازی یک آرایه (بردار) به عنوان ورودی دریافت می‌شود و سپس با اجرای دستورات مقایسه و جابجایی عناصر آرایه به نحوی در کنار هم قرار می‌گیرند که بصورت صعودی یا نزولی مرتب باشند. در متلب برای مرتب‌سازی عناصر آرایه از دستور sort استفاده می‌شود.

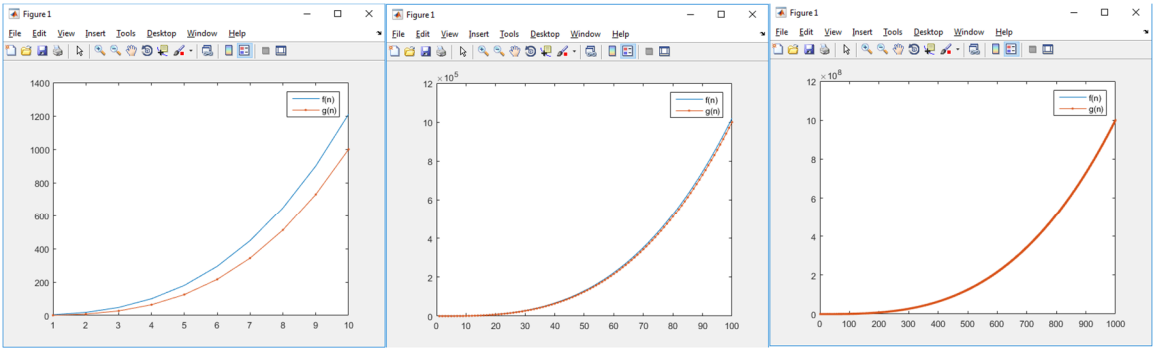
```
>> V = [ 5 9 7 4 1 6 3 8 2 ];
>> sort(V)

ans =

     1     2     3     4     5     6     7     8     9
```

در همه دوره‌های آموزش برنامه‌نویسی مرتب‌سازی به عنوان پایه الگوریتم‌های دیگر تدریس می‌گردد. بیش از 40 روش برای مرتب‌سازی وجود دارد که به لحاظ پیچیدگی، سرعت و حافظه محاسن و معایبی دارند. در این درس سه روش ساده را مورد بررسی قرار می دهیم.

2- نمادگذاری مجانبی O بزرگ (Big O Asymptotic Notation)

هرگاه توابع f(n) و g(n) (در حالت خاص بر روی اعداد طبیعی) مفروض باشند و n به بی نهایت میل کند خواهیم داشت f(n) = O(g(n)) هرگاه اعدادی مانند M و n0 وجود داشته باشند به نحوی که به ازای n>n0 همواره رابطه $|f(n)| \leq M|g(n)| \quad for\ all\ n > n_0$ در این صورت می‌گوییم تابع f(n) از مرتبه O(g(n)) است.

به بیان دیگر نمادگذاری O بزرگ برای بررسی و درک رفتار تابع به ازای n های بزرگ است. به عنوان مثال تابع $f(n) = n^3 + 2n^2 + n$ را در نظر بگیرید این تابع به ازای n های بزرگ رفتاری شبیه تابع $g(n) = n^3$ دارد. و همانطور که در نمودار نشان داده شده با بزرگتر شدن n اثر جمله n^3 بیشتر ظاهر می‌شود و دو تابع شبیه به هم رفتار می‌کنند از این جهت آنرا نمادگذاری مجانبی می‌گویند که تابع g بر تابع f مجانب می‌شود. لذا f(n) از مرتبه $O(n^3)$ است



n = 10 n = 100 n = 1000

از نمادگذاری مجانبی برای مقایسه پیچیدگی محاسباتی الگـــوریتم‌ها استفاده می‌شود. مهمترین نمادگذاری‌ها در جدول زیر نشان داده شده

notation	name
O(1)	constant ثابت
O(log(*n*))	logarithmic لگاریتمی
O(*n*)	linear خطی
O(n^2)	quadratic مربعی
O(n^c)	polynomial چند جمله ای
O(c^n)	exponential نمایی

بطور مثال پیچیدگی تابع $f(n) = n^3 + 3n^2 + \log n$ از نوع چند جمله‌ای است و برابر است با $O(n^3)$ و پیچیدگی تابع $f(n) = n^2 + 3n$ از نوع مربعی است و برابر است با $O(n^2)$ و پیچیدگی تابع $f(n) = 3log(n)$ از نوع لگاریتمی است و برابر است با $O(\log(n))$

3- حالت‌های بهترین، بدترین و متوسط

فرض کنید می‌خواهیم در یک آرایه با n عنصر دنبال عنصر خاصی بگردیم در این صورت در بهترین حالت اولین عنصر آرایه عنصر مطلوب ما خواهد بود (تعداد عملیات مساوی 1) و در بدترین حالت آخرین عنصر آرایه عنصر مطلوب ما خواهد بود (تعداد عملیات مساوی n) اگر هزاران بار این کار را روی هزاران آرایه مختلف انجام دهیم و میانگین بگیریم خواهیم دید که حدودا میانگین آن $\frac{n}{2}$ خواهد بود لذا در الگوریتم‌ها باید به این سه حالت توجه داشت.

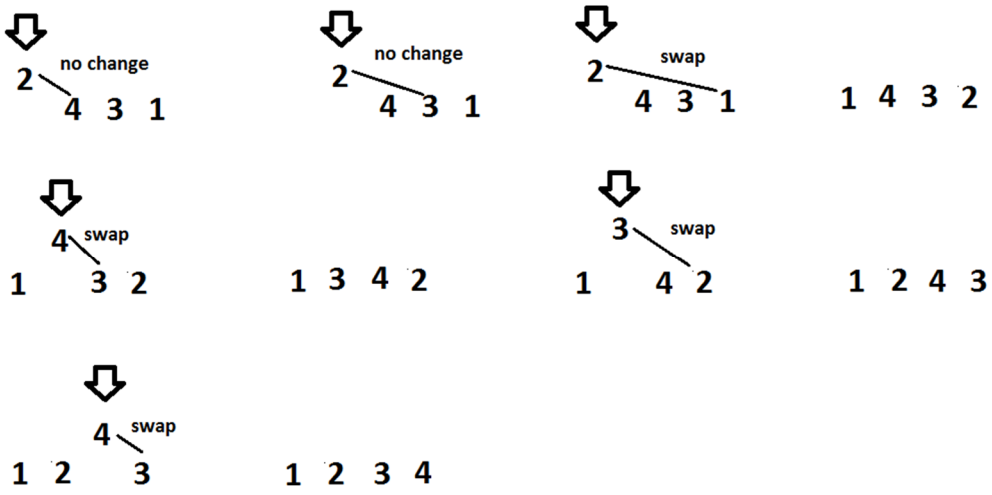
4- الگوریتم مرتب سازی حبابی (Bubble Sort)

دو پیاده‌سازی از الگوریتم مرتب‌سازی حبابی در شکل زیر نشان داده شده است

```
clc
clearvars
A = input('please enter vector A ');
% sort(A)
for k = 1:length(A)
    for h = k+1:length(A)
        if (A(k) > A(h))
            tmp = A(h);
            A(h) = A(k);
            A(k) = tmp;
        end
    end
end
disp(A)
```

```
clc
clearvars
A = input('please enter vector A ');
% sort(A)
for k = 1:length(A)-1
    for h = 1:length(A)-1
        if(A(h) > A(h+1))
            tmp = A(h+1);
            A(h+1) = A(h);
            A(h) = tmp;
        end
    end
end
disp(A)
```

در روش سمت چپ از عنصر اول آرایه تا آخر عناصر با عناصر بعد از خود مقایسه می‌شود و در صورت بزرگتر بودن جای دو عنصر عوض می‌شود یعنی عنصر کوچکتر قبل از عنصر بزرگتر قرار می‌گیرد. پس از اتمام مقایسه هر عنصر با عناصر بعد از خود به سراغ عنصر بعد می رود پیچیدگی محاسباتی الگوریتم مرتب‌سازی حبابی در همه حالات $O(n^2)$ می‌باشد.



نکته : می‌توان k تا یک عنصر مانده به آخر قرار داد.

در الگورتیم سمت راست همین روش تکرار می‌شود با این تفاوت که هر عنصر فقط با عنصر کنار خودش مقایسه می‌شود.

5- الگوریتم مرتب سازی انتخابی (Selection Sort)

الگوریتم مرتب سازی انتخابی به این صورت است که ابتدا مینیمم عنصر شناسایی شده و در ابتدای آرایه جدید قرار می‌گیرد سپس در جایگاه آن عنصر عدد بینهایت را قرار می دهیم تا هرگز از مینیمم کوچکتر نشود. و دوباره مینیمم محاسبه می‌شود. تا آرایه به انتها برسد. برای محاسبه مینیمم می‌توانیم از تابع min استفاده نماییم ولی برای به منظور آموزش محاسبه مینیمم را نیز با الگوریتم پیاده‌سازی می‌کنیم. پیچیدگی محاسباتی این الگوریتم نیز $O(n^2)$ است.

```
clc
clearvars
A = input('please enter vector A ');
B = [];
for h = 1:length(A)
    min = A(1);
    indx = 1;
    for k=2:length(A)
        if A(k)<min
            min = A(k);
            indx = k;
        end
    end
    A(indx) = inf;
    B = [ B min  ];
end
disp(B)
```

6- الگوریتم مرتب سازی درجی (Insertion Sort)

الگوریتم مرتب سازی درجی به این صورت است که متغیر k از 2 تا انتهای آرایه حرکت می‌کند متغیر k در واقع نشان دهنده جای کلید است و خانه k ام آرایه به عنوان کلید فرض می‌شود به ازای هر k متغیر h یک واحد کمتر از k تعریف می‌شود تا زمانی که h بزرگتر از صفر باشد و A(h) از کلید بزرگتر باشد مقدار A(h) را در A(h+1) قرار می‌دهیم و سپس یک واحد از h کم می کنیم این کار سبب می‌شود بین خانه اول تا k کلید که همان مقدار A(k) می‌باشد در

جای خودش قرار گیرد. این حلقه while تا جایی ادامه دارد که یا h برابر با صفر شود یا عنصر h ام دیگر از کلید کوچکتر نیست در این حالت کلـــید در جایگاه h +1 قرار می گیرد به این معنی که اگر اتمـــام حلقه در اثر نادرست شده h>0 باشد یعنی کلید از همه A(h) ها کوچکتر بوده و باید در ابتدا قرار گیرد در ان حالت h=0 است و کلید در مکان h+1 یعنی مکان 1 قرار می‌گیرد زیرا از همه A(h) ها کوچکتر است اگر هم حـــلقه در اثر نادرست شدن A(h)>key خاتمه یابد یعنی A(h)<= key بوده و لذا کلید در جای خودش قرار می‌گیرد (بعد از اولین عنصری که کوچکتر یا مساوی با آن است) این کار برای تمام عناصر بردار انجام می‌شود. پیچیدگی محاسباتی این الگوریتم نیز $O(n^2)$ است.

```
clc
clearvars
A = input('please enter vector A ');

for k = 2:length(A)
    h = k - 1;
    key = A(k);
    while (h > 0)&&(A(h)>key)
        A(h + 1) = A(h);
        h = h - 1;
    end
    A(h + 1) = key;
end

disp(A)
```

مثال شماره 16 : برنامه‌ای بنویسید که دو بردار مرتب از اعداد را دریافت کرده و در قالب یک بردار مرتب ادغام نماید.

```
clc; clearvars;
A = input('please enter sorted vector A ');
B = input('please enter sorted vector B ');
% sort([A B]);
LA = length(A);
LB = length(B);
A(LA + 1) = inf;
B(LB + 1) = inf;
M = [];  h = 1;   k = 1; n = 1;
while (n<=LA+LB)
    if(A(k)<=B(h))
        M(n) = A(k);
        k = k + 1;
    else
        M(n) = B(h);
        h = h + 1;
    end
    n = n + 1;
end
disp(M)
```

مثال شماره 17 : برنامه‌ای بنویسید که یک بردار از اعداد صحیح مثبت و یک عدد (عدد شاخص) را دریافت کرده سپس بردار جدیدی ایجاد کند شامل کلیه اعداد بردار ورودی که از عدد شاخص کوچکتر باشند.

```
clc;
clearvars;
A = input('please enter vector A ');
N = input('please enter N ');
% A(A<N)
B = [];
for k = 1:length(A)
    if(A(k)<N)
        B = [ B A(k)];
    end
end
disp(B)
```

مثال شماره 18 : کلیه مراحل مرتب شدن آرایه A = [3 2 1] به روش مرتب‌سازی درجی را نشان دهید

```
3  3  1
2  3  1
2  3  3
2  2  3
1  2  3
```

مثال شماره 19 : برنامه‌ای بنویسید که یک آرایه از اعداد صحیح مثبت دریافت نموده عناصر تکراری آن آرایه را حذف کند.

```
clc
clearvars
A = input('please enter vector A ');
B = [];
for k = 1:length(A)
    M = A(k);
    for h=k+1:length(A)
        if A(h)==M
           A(h) = -1;
        end
    end
end
for k=1:length(A)
    if A(k)~=-1
        B = [ B A(k) ];
    end
end
disp(B)
```

مثال شماره 20 : برنامه‌ای بنویسید که یک آرایه از اعداد صحیح مثبت را دریافت کرده سپس نشان دهد نسبت به آرایه مرتب شده چند درصد از عناصر آرایه در جای مرتب قرار گرفته‌اند مثلا اگر آرایه ورودی بصورت [5 1 3 2 4] باشد نسبت به مرتب شده آن یعنی [1 2 3 4 5] فقط دو عنصر 2 و 5 در جایگاه مرتب شده قرار دارند لذا 2 تقسیم بر 5 یعنی 40 درصد این آرایه مرتب (صعودی) است.

```
clc
clearvars
A = input('please enter vector A ');
AS = sort(A);
disp(sum([AS==A])/length(A));
```