دانشگاه آزاد اسلامی واحد رودهن

طراحی الگوریتم

استاد: ساسان آزاد

در بحث ساختمان داده ها دیدیم که الگوریتم، روندی است که طی آن عملیات گوناگون بر روی ساختارهای داده مختلف انجام میشد. در آنجا تمرکز بر روی روشهای مختلف به کارگیری حافظه اصلی بوده است. اما در بحث طراحی الگوریتم تمرکز بر روی روشهای مختلف نوشتن یک الگوریتم میباشد، چون به طور معمول برای حل یک مسئله مشخص، بیش از یک روش یا الگوریتم وجود دارد که برخی سریع تر، برخی کندتر، برخی کارآمدتر و برخی ناکارآمد هستند. طبیعی است که هدف از این مباحث و این درس، بررسی و انتخاب الگوریتم هائی است که بیشترین کارآیی یا **Efficiency** را دارند.

در آغاز و برای آشنایی با روش کار، در رابطه با چند روش جستجو یا **Search** و مرتب سازی یا **Sort** الگوریتم هایی را بررسی کرده و ردیابی یا **Trace** می نماییم تا تفاوت ها مشخص شود.

الگوریتم 1-1: جستجوی پی در پی یا ترتیبی یا خطی یا Sequential Search

ساده ترین روش برای جستجو به دنبال یک داده خاص، آن است که تمامی داده ها را به شکلی ترتیبی از ابتدا تا انتها بررسی یا **Scan** نموده تا مورد دلخواه ما یافت شود. در الگوریتم زیر یک آرایه غیر مرتب به نام **K** مفروض است که **N+1** عنصر دارد و هدف، یافتن عنصر **X** می باشد. عنصر **N+1** اُم در این الگوریتم به عنوان عضو نگهبان یا **Sentinel** تلقی می گردد که قبل از انجام عمل جستجو، مقدار **X** در آن قرار می گیرد. در پایان الگوریتم، در صورت موفقیت آمیز بودن جستجو، مکان آن عنصر و در غیر اینصورت عدد صفر به خارج ارسال می گردد:

```
Function Linear_Search (K, N, X)
1. [Initialize Search]
 I ← 1
 K [N+1] ← X
2. [Search the Array]
 Repeat While K [I] ≠ X
          I ← I + 1
3. [Is the search Successful?]
 If I = N+1
Then Write ('Unsuccessful Search')
       Return (0)
Else Write ('Successful Search')
      Return (I)
```

مثال: آرایه K مفروض است: {K={17,9,11,14,5,9 . الگوریتم جستجوی خطی را بر روی آن با فرض X=14 ردیابی نمایید. با توجه به اینکه ۶ عنصر موجود است و در واقع N=6 می باشد، عدد مورد نظر جستجو یعنی ۱۴ را درخانه N+1 اُم یعنی هفتم قرار داده و ردیابی را انجام می دهیم. البته توجه داریم که طبق الگوریتم در ابتدا I=1 می شود.

I (Location)	K[I]
1	K[1] = 17 <> 14 ➔ I=I+1 = 2
2	K[2] = 9 <> 14 ➔ I=I+1 = 3
3	K[3] = 11 <> 14 ➔ I=I+1 = 4
4	K[4] = 14 = 14 ➔ واردحلقه نشده و به مرحله بعد می رویم

از آنجاییکه I مخالف N+1 یعنی ۷ می باشدپس پیغام موفق بودن جستجو داده شده و مقدار I یعنی عدد ۴ به بیرون از الگوریتم ارسال می گردد.

مثال: با فرض X=10 در همان آرایه K فوق الگوریتم را مجدداً ردیابی نمایید.

I (Location)	K[I]
1	K[1] = 17 <> 10 ➔ I=I+1 = 2
2	K[2] = 9 <> 10 ➔ I=I+1 = 3
3	K[3] = 11 <> 10 ➔ I=I+1 = 4
4	K[4] = 14 <> 10 ➔ I=I+1 = 5
5	K[5] = 5 <> 10 ➔ I=I+1 = 6
6	K[6] = 9 <> 10 ➔ I=I+1 = 7
7	K[7] = 10 = 10 ➔ وارد حلقه نشده و به مرحله بعدی می رویم

از آنجاییکه I برابر N+1 یعنی ۷ می باشد، پیغام ناموفق بودن جستجو به نمایش در آمده و مقدار صفر به خارج ارسال می گردد.

تمرین: در آرایه مفروض زیر، الگوریتم جستجوی پی در پی رایک بار با فرض X=20 و یک بار با فرض X=30 ردیابی نمایید:

A = {1, 3, 4, 2, 7, 5, 8, 6, 9, 11, 10, 12, 14, 13, 16, 17, 15, 20, 18, 19}

الگوریتم 2-1: جمع مقادیر موجود در یک آرایه یا Sum

آرایه A شامل N عنصر مفروض است. الگوریتم زیر مجموع این مقادیر را محاسبه می نماید:

```
Function Sum_Values(A,N)
۱ . [Initialization]
 Sum ← 0
2. [Sum the values in the Array]
 Repeat For I = 1 to N
        Sum ← Sum + A [I]
3. [Finished]
 Return (Sum)
```

مثال: آرایه A={2,10,9,11,6} مفروض می باشد. الگوریتم Sum را بر روی آن ردیابی نمایید.

مقداردهی اولیه: Sum = 0

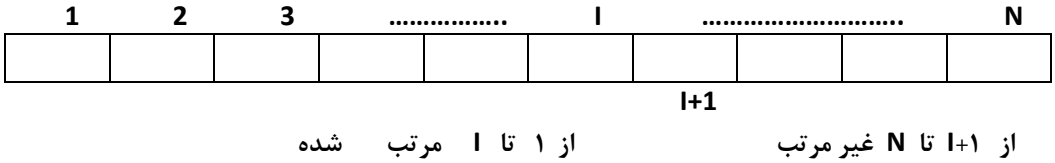
I	A[I]	Sum = Sum + A[I]
1	2	0 + 2 = 2
2	10	2 + 10 = 12
3	9	12 + 9 = 21
4	11	21 + 11 = 32
5	6	32 + 6 = 38

خروجی این الگوریتم ۳۸ می باشد.

هم اکنون شاید این سوال برایتان پیش آمده باشد که با توجه به سادگی الگوریتم ها، ردیابی یا Trace آنها نیز ساده می باشد پس لزوم انجام آن چیست؟ پاسخ آن است که به محض نوشته شدن یک الگوریتم، لازم است بدانیم چگونه آن را تحلیل یا آنالیز نماییم که اولین و ساده ترین نوع تحلیل آزمایش صحت یک الگوریتم است که می تواند با ردیابی و آزمایش با داده های فرضی انجام شود.

الگوریتم 3-1: مرتب سازی تعویضی یا Exchange Sort

در این روش از مرتب سازی که به خصوص بر روی تعداد بالای عناصر به خوبی عمل می نماید، هدف آن است که در هر مرحله، یک داده در محل نهایی خود قرار گیرد و تمام مقادیری که کوچکتر هستند قبل از آن و تمام مقادیری که بزرگتر یا مساوی هستند، بعد از آن قرار گیرند. این به آن معناست که تمام داده هایی که در هر مرحله، قبل از داده ی جاری و مورد بررسی مستقر شده اند، مرتب شده و کوچکتر می باشند و تمام داده هایی که بعد از آن قرار می گیرند، بزرگتر یا مساوی و البته غیر مرتب هستند.



Procedure Exchange_Sort (N,S) تعداد عناصر **N** نام آرایه **S**

1. [Exchange Sort]

Repeat For I = 1 to N-1

Repeat For J = I+1 to N

If S [J] < S [I]

Then S [J] ←→ S [I]

2. [Finished]

Exit

روش عملکرد الگوریتم:

۱. آرایه S از اندیس 1 تا N پیمایش می شود و اولین کوچکترین عنصر، با عضوی که در خانه ی یکم آرایه قرار دارد، جابجا میگردد.

۲. آرایه S از اندیس 2 تا N پیمایش می شود و دومین کوچکترین عنصر، با عضوی که در خانه ی دوم آرایه قرار دارد، جابجا میگردد.

۳. آرایه S از اندیس3 تا N پیمایش می شود و سومین کوچکترین عنصر، با عضوی که در خانه ی سوم آرایه قرار دارد، جابجا میگردد.

.

.

.

N-1. آرایه S از اندیس N-1 تا N پیمایش می شود و N-1 اُمین کوچکترین عنصر، با عضوی که در خانه ی N-1 اُم آرایه قرار دارد، جابجا میگردد.

مثال: الگوریتم مرتب سازی تعویضی یا Exchange Sort را بر روی آرایه S ردیابی نموده و داده ها را مرتب نمایید.

N=5 **S = {10, 6, 12, 9, 6}**

I J S [ ]

---

1 **10, 6, 12, 9, 6**

2 **S [2] < S [1] ➔ جابجایی عناصر دوم و اول انجام می شود ➔ 6, 10, 12, 9, 6**

3 **S [3] < S [1] نیست پس جابجایی نداریم**

4 **S [4] < S [1] نیست پس جابجایی نداریم**

5 **S [5] < S [1] نیست پس جابجایی نداریم**

---

2 **6, 10, 12, 9, 6**

3 **S [3] < S [2] نیست پس جابجایی نداریم**

4 **S [4] < S [2] ➔ جابجایی عناصر چهارم و دوم انجام می شود ➔ 6, 9, 12, 10, 6**

5 **S [5] < S [2] ➔ جابجایی عناصر پنجم و دوم انجام می شود ➔ 6, 6, 12, 10, 9**

---

3 **6, 6, 12, 10, 9**

4 **S [4] < S [3] ➔ جابجایی عناصر چهارم و سوم انجام می شود ➔ 6, 6, 10, 12, 9**

5 **S [5] < S [3] ➔ جابجایی عناصر پنجم و سوم انجام می شود ➔ 6, 6, 9, 12, 10**

---

4 **6, 6, 9, 12, 10**

5 **S [5] < S [4] ➔ جابجایی عناصر پنجم و چهارم انجام می شود ➔ 6, 6, 9, 10, 12**

همانگونه که ملاحظه می شود عناصر مرتب گردیدند.

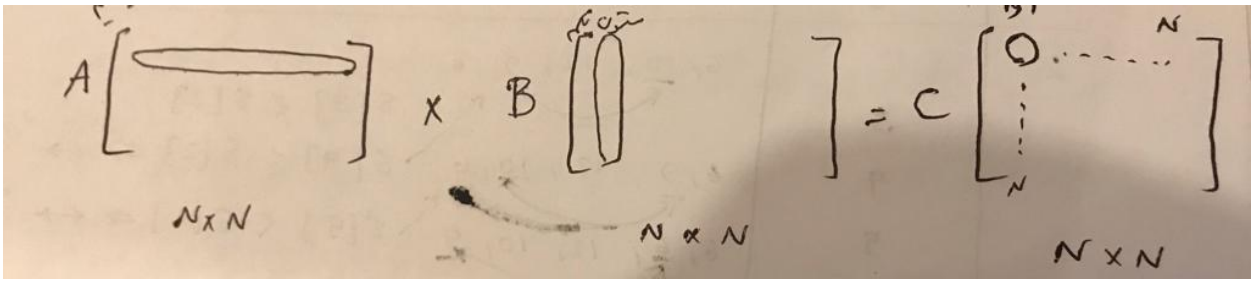
تمرین: الگوریتم مرتب سازی تعویضی را بر روی آرایه مقابل ردیابی نمایید. **S = {20, 12, 24, 18, 12, 10, 6, 12, 9, 6}**

الگوریتم 4-1: ضرب دو ماتریس NxN

یادآوری: حاصلضرب دو ماتریس NxN، یک ماتریس NxN دیگر می باشد که هر یک از عناصر ماتریس حاصلضرب به شکل زیر محاسبه می شود:

$$Cij = \sum_{k=1}^{n} Aik * Bkj$$

مجموع حاصلضرب سطر I اُم از ماتریس A در ستون J اُم از ماتریس B



Procedure Matrix_Multiplication (A, B, C, N)

1. [Multiply Matrices A and B and Store the Result in Matrix C]

Repeat For I = 1 to N

Repeat For J = 1 to N

Sum ← 0

Repeat For K = 1 to N

Sum ← Sum + A [I, K] * B [K, J]

C [I, J] ← Sum

2. [Finished]

Exit

مثال: دو ماتریس A و B به شرح زیر مفروض می باشند، با ردیابی الگوریتم ضرب دو ماتریس، حاصلضرب را بدست آورید:

$$A = \begin{bmatrix} 2 & 1 & -2 \\ 3 & 0 & 1 \\ 2 & 2 & 5 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & -1 & 0 \\ 4 & 2 & 2 \\ 1 & 3 & -3 \end{bmatrix} \qquad C = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

I	J	K	Sum (C [I, J])	
1				
	1		Sum = 0 (C [1, 1])	
		1	Sum = Sum + (A [1, 1] * B [1, 1]) = 0 + (2 * 1) = 2	
		2	Sum = Sum + (A [1, 2] * B [2, 1]) = 2 + (1 * 4) = 6	
		3	Sum = Sum + (A [1, 3] * B [3, 1]) = 6 + (-2 * 1) = 4	===> C [1, 1] = 4
	2		Sum = 0 (C [1, 2])	
		1	Sum = Sum + (A [1, 1] * B [1, 2]) = 0 + (2 * -1) = -2	
		2	Sum = Sum + (A [1, 2] * B [2, 2]) = -2 + (1 * 2) = 0	
		3	Sum = Sum + (A [1, 3] * B [3, 2]) = 0 + (-2 * 3) = -6	===> C [1, 2] = -6
	3		Sum = 0 (C [1, 3])	
		1	Sum = Sum + (A [1, 1] * B [1, 3]) = 0 + (2 * 0) = 0	
		2	Sum = Sum + (A [1, 2] * B [2, 3]) = 0 + (1 * 2) = 2	
		3	Sum = Sum + (A [1, 3] * B [3, 3]) = 2 + (- 2 * -3) = 8	===> C [1, 3] = 8
2				
	1		Sum = 0 (C [2, 1])	
		1	Sum = Sum + (A [2, 1] * B [1, 1]) = 0 + (3 * 1) = 3	
		2	Sum = Sum + (A [2, 2] * B [2, 1]) = 3 + (0 * 4) = 3	
		3	Sum = Sum + (A [2, 3] * B [3, 1]) = 3 + (1 * 1) = 4	===> C [2, 1] = 4
	2		Sum = 0 (C [2, 2])	
		1	Sum = Sum + (A [2, 1] * B [1, 2]) = 0 + (3 * -1) = -3	
		2	Sum = Sum + (A [2, 2] * B [2, 2]) = -3 + (0 * 2) = -3	
		3	Sum = Sum + (A [2, 3] * B [3, 2]) = -3 + (1 * 3) = 0	===> C [2, 2] = 0
	3		Sum = 0 (C [2, 3])	
		1	Sum = Sum + (A [2, 1] * B [1, 3]) = 0 + (3 * 0) = 0	
		2	Sum = Sum + (A [2, 2] * B [2, 3]) = 0 + (0 * 2) = 0	
		3	Sum = Sum + (A [2, 3] * B [3, 3]) = 0 + (1 * -3) = -3	===> C [2, 3] = -3
3				
	1		Sum = 0 (C [3, 1])	
		1	Sum = Sum + (A [3, 1] * B [1, 1]) = 0 + (2 * 1) = 2	
		2	Sum = Sum + (A [3, 2] * B [2, 1]) = 2 + (2 * 4) = 10	
		3	Sum = Sum + (A [3, 3] * B [3, 1]) = 10 + (5 * 1) = 15	===> C [3, 1] = 15
	2		Sum = 0 (C [3, 2])	
		1	Sum = Sum + (A [3, 1] * B [1, 2]) = 0 + (2 * -1) = -2	

	2	Sum = Sum + (A [3, 2] * B [2, 2]) = -2 + (2 * 2) = 2
	3	Sum = Sum + (A [3, 3] * B [3, 2]) = 2 + (5 * 3) = 17 ===> C [3, 2] = 17
3		Sum = 0 (C [3, 3])
	1	Sum = Sum + (A [3, 1] * B [1, 3]) = 0 + (2 * 0) = 0
	2	Sum = Sum + (A [3, 2] * B [2, 3]) = 0 + (2 * 2) = 4
	3	Sum = Sum + (A [3, 3] * B [3, 3]) = 4 + (5 * -3) = -11 ===> C [3, 3] = -11

$$C = \begin{bmatrix} 4 & -6 & 8 \\ 4 & 0 & -3 \\ 15 & 17 & -11 \end{bmatrix}$$

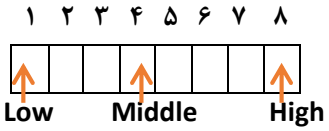
تعداد مراحل فوق با توجه به وجود داشتن ۳ حلقه تودرتو با N بار تکراردر الگوریتم، برابر با NxNxN یا N^3 می باشد. از آنجا که N=3در این مثال می باشد، ۲۷ حالت را تجربه نمودیم.

تمرین: دو ماتریس A و B به شرح زیر مفروض می باشند، با ردیابی الگوریتم ضرب دو ماتریس، حاصلضرب را بدست آورید:

$$A = \begin{bmatrix} 1 & -2 & 2 \\ 0 & 1 & 3 \\ 2 & 5 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & -1 \\ 2 & 4 & 2 \\ -3 & 1 & 3 \end{bmatrix} \qquad C = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

الگوریتم ۵-۱: الگوریتم جستجوی دودویی یا Binary Search

قبل از هر چیز در خصوص این الگوریتم باید اشاره داشت که شرط اولیه جهت اجرا، داشتن یک آرایه مرتب یا یک مجموعه مرتب یا Sort شده از اطلاعات است. ایده کلی این روش، تقسیم آرایه یا مجموعه اطلاعاتی به دو قسمت و مقایسه عنصری که به دنبال آن هستیم با عنصر میانی مجموعه است. با فرض اینکه ترتیب صعودی است، پس از مقایسه عنصر مورد نظر X با عنصر میانی، چنانچه برابر باشد پیغـام موفقیت آمیز بودن جستجو داده شده و محل عضو مـورد نظر به عنوان خروجی به بیـرون بازگردانده می شود. چنانچه عضو مورد نظر کوچکتر از عضو میانی باشد، جستجو در نیمه سمت چپ و چنانچه عضو مورد نظر بزرگتر از عضو میانی باشدجستجو در نیمه سمت راست ادامه می یابد. در انتها و در صورت موفقیت آمیز نبودن جستجو، عدد صفر به بیرون باز گردانده می شود.



$Middle = \left\lfloor \frac{Low+High}{2} \right\rfloor$ **High=8 Low=1** $Middle=\left\lfloor \frac{1+8}{2} \right\rfloor$ =۴ (حد پایین یا جزء صحیح)

در شکل فوق با فرض داشتن ۸ عنصر که به شکل صعودی مرتب شده اند، در مرحله اول عضو میانی از فرمول فوق محاسبه و برابر با ۴ می گردد. در ادامه جستجو به همین طریق ادامه یافته و مثلاً اگر عنصر از عضو میانی کوچکتر بود طبق فرمول فوق عضو میانی جدید، عضو دوم خواهد شد و الی آخر.

Function Binary_Search (A, N, X) **A=**نام آرایه **N=**تعداد عناصر **X=** عضوی که به دنبال آن هستیم

1. [Initialize]
Low=1 **(Low=**مقدار پایین**)**
High=N **(High=**مقدار بالا**)**
2. [Perform Binary Search]
Repeat thru Step 4 while Low ≤ High
3. [Obtain Middle]
Middle ← $\left\lfloor \frac{Low+High}{2} \right\rfloor$
4. [Compare]
If X < A [Middle]
Then High ← Middle – 1
Else If X > A [Middle]
Then Low ← Middle + 1
Else Write ('Successful Search')
Return (Middle)
5. [Unsuccessful Search]
Write ('Unsuccessful Search')
Return (0)

مثال: آرایه مرتب A مفروض است. با ردیابی نمودن الگوریتم جستجوی دودویی، به دنبال X=25 می گردیم:

A= {2, 7, 9, 9, 13, 18, 25, 36, 44, 48, 55, 69} X=25 N=12

Low	High	Middle	A [Middle]		
1	12	$\left\lfloor \frac{1+12}{2} \right\rfloor$=6	A [6] =18	X=25 > 18 →	Low ← Middle + 1 = 7
7	12	$\left\lfloor \frac{7+12}{2} \right\rfloor$=9	A [9] =44	X=25 < 44 →	High ← Middle – 1 = 8
7	8	$\left\lfloor \frac{7+8}{2} \right\rfloor$=7	A [7] =25	X=25 = 25 →	Successful Search, Return (7)

تعداد مقایسات این روش: ۳ تعداد مقایسات روش جستجوی ترتیبی: ۷ (چون ۲۵ در خانه هفتم قرار دارد)

تمرین: آرایه مرتب A مفروض است. با ردیابی نمودن الگوریتم جستجوی دودویی، X= 69 را بیابید.

A= {2, 7, 9, 9, 13, 18, 25, 36, 44, 48, 55, 69} X=۶۹ N=12

مثال: آرایه مرتب A مفروض است. با ردیابی نمودن الگوریتم جستجوی دودویی، به دنبال X=5 می گردیم:

A= {2, 7, 9, 9, 13, 18, 25, 36, 44, 48, 55, 69} X=5 N=12

Low	High	Middle	A[Middle]		
1	12	$\left\lfloor \frac{1+12}{2} \right\rfloor$=6	A [6] =18	X=5 < 18 →	High ← Middle - 1 = 5
1	5	$\left\lfloor \frac{1+5}{2} \right\rfloor$=3	A [3] =9	X=5 < 9 →	High ← Middle - 1 = 2
1	2	$\left\lfloor \frac{1+2}{2} \right\rfloor$=1	A [1] =2	X=5 > 2 →	Low ← Middle + 1 = 2
2	2	$\left\lfloor \frac{2+2}{2} \right\rfloor$=2	A [2] =7	X=5 < 7 →	High ← Middle - 1 = 1
2	1	Low ≤ High نیست → Unsuccessful Search, Return (0)			

تعداد مقایسات این روش: ۴ تعداد مقایسات روش جستجوی ترتیبی: ۱۳ (چون ۵ وجود ندارد و ما خود آن را در خانه سیزدهم قرار داده ایم)

تمرین: آرایه مرتب A مفروض است. با ردیابی نمودن الگوریتم جستجوی دودویی، X= ۱۰۰ را بیابید.

A= {2, 7, 9, 9, 13, 18, 25, 36, 44, 48, 55, 69} X=۱۰۰ N=12

با بررسی دو مثال فوق یعنی حالتی که عضو مورد نظر یافت می شود و حالتی که یافت نمی شود می توان دریافت که در هر حالت، جستجوی دودویی نسبت به جستجوی خطی سریع تر است.

سؤال: حداکثر تعداد مقایسات یک جستجوی دودویی چقدر است؟
پاسخ: زمانی که عضو مورد نظر وجود نداشته باشد (چه زمانی که از همه بزرگتر و یا کوچکتر است).

مثال: اگر یک آرایه با ۴ عنصر داشته باشیم و عضوی که به دنبال آن هستیم وجود نداشته باشد، حداکثر مقایسات چقدر است؟
A= {2, 5, 8, 10} N=4 X=20

Low	High	Middle	A [Middle]	
1	4	$\left\lfloor\frac{1+4}{2}\right\rfloor$=2	A [2] =5	X=20 > 5 → Low ← Middle + 1 = 3
3	4	$\left\lfloor\frac{3+4}{2}\right\rfloor$=3	A [3] =8	X=20 > 8 → Low ← Middle + 1 = 4
4	4	$\left\lfloor\frac{4+4}{2}\right\rfloor$=4	A [4] =10	X=20 > 10 → Low ← Middle + 1 = 5
5	4		Low ≤ High نیست → Unsuccessful Search, Return (0)	

حداکثر تعداد مقایسات برابر با عدد ۳ می باشد.

هم اکنون با روشی شبه استقرایی، می توان گفت:

N	حداکثر تعداد مقایسات	
۴ ($۲^۲$)	(۱+۲=۳)	**حداکثر تعداد مقایسات در آرایه ای با ۴ عنصر برابر با ۳ (۲ + ۱) می باشد.**
۸ ($۲^۳$)	(۱+۳=۴)	**حداکثر تعداد مقایسات در آرایه ای با ۸ عنصر برابر با ۴ (۳ + ۱) می باشد.**
۱۶ ($۲^۴$)	(۱+۴=۵)	**حداکثر تعداد مقایسات در آرایه ای با ۱۶ عنصر برابر با ۵ (۴ + ۱) می باشد.**
۳۲ ($۲^۵$)	(۱+۵=۶)	**حداکثر تعداد مقایسات در آرایه ای با ۳۲ عنصر برابر با ۶ (۵ + ۱) می باشد.**
۶۴ ($۲^۶$)	(۱+۶=۷)	**حداکثر تعداد مقایسات در آرایه ای با ۶۴ عنصر برابر با ۷ (۶ + ۱) می باشد.**

حداکثر تعداد مقایسات در آرایه ای با 2^N عنصر برابر با ($\log_2 N$ + ۱) می باشد.

نکته: در جدول فوق، تمام اعدادی که معرف تعداد عناصر آرایه هستند، به صورت توانی از ۲ انتخاب و نوشته شده اند. چنانچه تعداد عناصر موجود در مسئله ای توانی از ۲ نبود (مثلاً ۱۰ عنصر داریم که بین ۸ و ۱۶ در جدول فوق می باشد)، برای تخمین حداکثر تعداد مقایسات عدد کمتر را در نظر میگیریم (تعداد مقایسات برای ۸ عنصر برابر ۴ و برای ۱۶ عنصر برابر ۵ است و لذا در زمان داشتن ۱۰ عنصر عدد کمتر یعنی ۴ را انتخاب می نماییم).

تمرین: فرض کنید آرایه ای با ۱۶ میلیارد عنصر (16×10^9) موجود است. در سناریوی جستجوی ناموفق، حداکثر تعداد مقایسات را برای الف) جستجوی ترتیبی و ب) جستجوی دودویی بدست آورید.

الگوریتم ۶-۱: الگوریتم جمله n اُم فیبوناچی Fibonacci (روش بازگشتی یا Recursive – بالا به پایین یا جزء به کل)

در ریاضیات، سری فیبوناچی به دنباله ای از اعداد گفته می شود که به صورت زیر تعریف می گردد:

$$F(n) = \begin{cases} 0 & \text{If } n = 0 \\ 1 & \text{If } n = 1 \\ F(n-1) + F(n-2) & \text{Otherwise} \end{cases}$$

در واقع غیر از دو عدد اول این سری، اعداد بعدی از جمع دو عدد قبلی خود بدست می آیند. می توان برای سری یا دنباله فیبوناچی ضابطه یا معیار زیر را نوشت:

N=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	….
F(N)=	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	….

به عنوان مثال در اینجا عضو ششم به نام F_6 برابر با ۸ است.

Function Fib (N) با فرض آنکه **N** عدد صحیح است

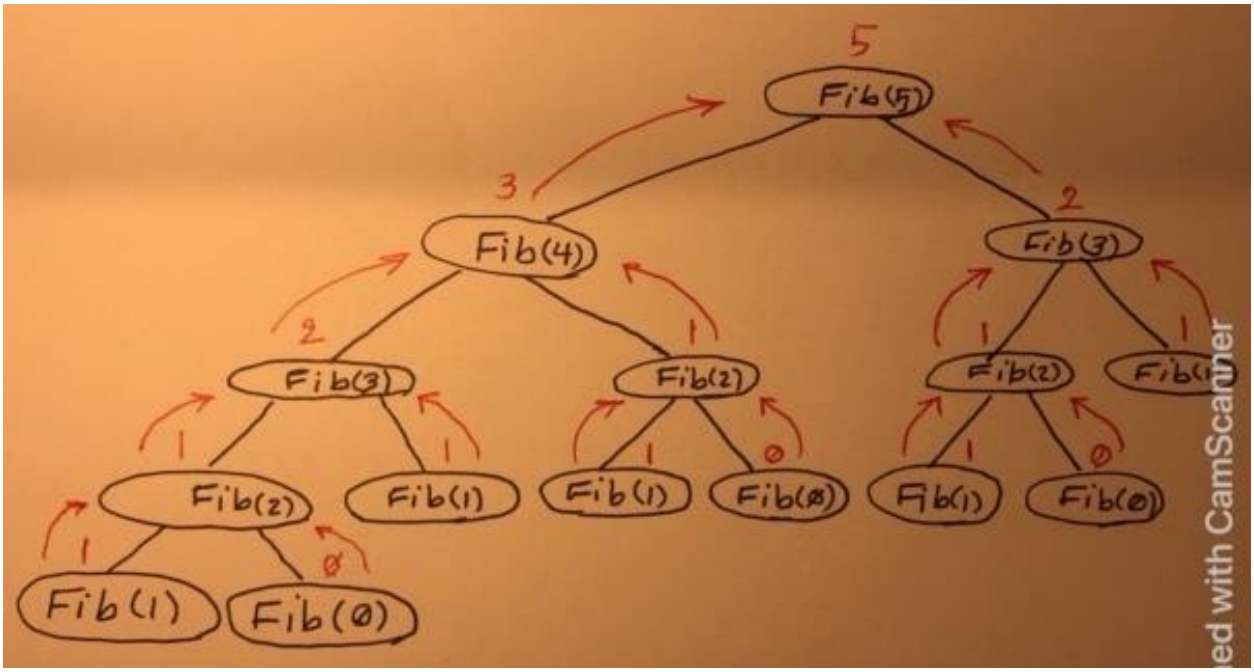
1. [Calculate Fibonacci in Recursive Form]

If N ≤ 1

Then Return (N)

Else Return (Fib (N-1) + Fib (N-2))

مثال: الگوریتم Fib (N) را با فرض N=5 ردیابی نمایید: **در واقع داریم Fib(5) = Fib(4) + Fib(3)**



خروجی این الگوریتم برابر با ۵ خواهد بود. به عبارت دیگر، $F_5 = 5$ خواهد شد.

تمرین: الگوریتم Fib (N) را با فرض N=10 ردیابی نمایید.

همانگونه که در ردیابی الگوریتم Fib(N) (شکل فوق) ملاحظه می گردد، جملات تکراری در این محاسبه درختی زیاد می باشند:

جمله	تعداد دفعات محاسبه	
۵	۱	
۴	۱	
۳	۲	پس در مجموع برای محاسبه جمله پنجم یا Fib(5)، ۱۵ محاسبه انجام شده است
۲	۳	(1+ 1+ 2+ 3+ 5+ 3) = 15
۱	۵	
۰	۳	

با یک بررسی اجمالی می توان دریافت که تعداد فراخوانی ها و در واقع تعداد محاسبات جملات برای F(N) ، در قیاس با مقدار N، مقرون به صرفه نیست و دلیل آن تکرارهای فراوان در محاسبهٔ جملات میانی است. به عنوان مثال در محاسبه Fib(5)، محاسبه F(1) پنج بار تکرار شده.

سؤال: برای محاسبه جمله ششم، چند محاسبه لازم است؟

پاسخ: **Fib(6) = Fib(5) + Fib(4)**

جمعاً برابر با ۲۵ محاسبه ↑ ۱ محاسبه ↑ ۱۵ محاسبه ↑ ۹ محاسبه

سؤال: برای محاسبه جمله هفتم، چند محاسبه لازم است؟

پاسخ: **Fib(7) = Fib(6) + Fib(5)**

جمعاً برابر با ۴۱ محاسبه ↑ ۱ محاسبه ↑ 25 محاسبه ↑ 15 محاسبه

و لذا به همین ترتیب می توان برای تعداد محاسبات مورد نیاز جملهٔ N اُم از رابطه زیر استفاده نمود:

$$T(N) = T(N-1) + T(N-2) + 1$$

جدول ۶-۱:

جمله مورد نظر N	T(N) تعداد محاسبات مورد نیاز
۵	۱۵
۶	۲۵
۷	۴۱
۸	۶۷
۹	۱۰۹
۱۰	۱۷۷

الگوریتم ۷-۱: الگوریتم جمله n اُم فیبوناچی Fibonacci (روش غیر بازگشتی – تکرار شونده – بهبود یافته – پایین به بالا)

در این الگوریتم، از آرایه F برای ذخیره نتایج میانی استفاده می شود و F[I] برابر با مقدار جمله I اُم می باشد:

```
Function Fib2 (N, F)
1. [Initialization]
   F [0] = 0            طبق تعریف تابع
2. [Calculate Fibonacci]
   If N > 0
   Then F [1] ← 1       طبق تعریف تابع
        Repeat For I = 2 to N
             F [I] ← F [I-1] + F [I-2]
3. [Finished]
   Return (F [I])
```

البته در نهایت I = N می گردد //// Return (F [N]) هم صحیح می باشد

مثال: الگوریتم Fib2 را با فرض N=10 ردیابی نمایید:

جدول ۷-۱:

I	F
	F[0] = 0 , F[1] = 1 طبق تعریف تابع
2	F[2] = F[1] + F[0] = 1 + 0 = 1
3	F[3] = F[2] + F[1] = 1 + 1 = 2
4	F[4] = F[3] + F[2] = 2 + 1 = 3
5	F[5] = F[4] + F[3] = 3 + 2 = 5
6	F[6] = F[5] + F[4] = 5 + 3 = 8
7	F[7] = F[6] + F[5] = 8 + 5 = 13
8	F[8] = F[7] + F[6] = 13 + 8 = 21
9	F[9] = F[8] + F[7] = 21 + 13 = 34
10	F[10] = F[9] + F[8] = 34 + 21 = 55

خروجی این مثال برابر با ۵۵ است.

تعداد محاسبات با این روش برابر با ۱۱ می باشد (۲ + ۹) که در واقع برابر با N+1 است. (۱ + ۱۰) T(N) = N + 1

تمرین: الگوریتم Fib2 را با فرض N = 20 ردیابی نموده و مقدار جمله بیستم را بدست آورید.

مقایسه کارایی دو الگوریتم ۶-۱ و ۷-۱ :

همانگونه که قبلاً اشاره شد، T(N) عبارت از تعداد محاسبات مورد نیاز برای بدست آوردن مقدار جملهٔ N اُم است. با یک بازنگری به جدول ۶-۱ در می یابیم که هر بار N به اندازه ۲ واحد افزایش می یابد، تعداد جملات محاسبه شده بیـــش از ۲ برابر افزایش می یابد.

N	T (N)	
5	15	
7	41	بیش از ۲ برابر قبلی یعنی ۱۵ است
9	109	بیش از ۲ برابر قبلی یعنی ۴۱ است

قبلاً رابطه زیر را داشتیم:

T (N) = T (N-1) + T (N-2) +1

T (0) = T (1) = 1 / T (5) = 15 / T (6) = 25 / T (7) = 41 / T (8) = 67 .....

اما هم اکنون می توان طبق تحلیلی که در بالا انجام شد گفت:

T (N) > 2 x T (N-2) برای **N ≥ 2**

T (N) > 2 x 2 x T (N-4)

T (N) > 2 x 2 x 2 x T (N-6)

...

...

...

T (N) > 2 x 2 x 2 x 2 x ..... x 2 x T (0)

↑----------↑

N/2 مرتبه عدد ۲ در خود ضرب شده

بنابراین می توان گفت تعداد محاسبات در روش بازگشتی فیبوناچی از عبارت روبرو بدست می آید $T(N) > 2^{N/2}$

و نیز قبلاً دیده شد که تعداد محاسبات در روش غیر بازگشتی فیبوناچی از عبارت روبرو بدست می آید **T (N) = N + 1**

اکنون با یک مثال دو الگوریتم را با یکدیگر مقایسه نموده و تعداد محاسبات را در هر یک بررسی می نماییم (فرض بر آن است که زمان تقریبی انجام هر محاسبه ۱ نانو ثانیه یا به عبارت دیگر $۱۰^{-۹}$ ثانیه می باشد):

N	روش غیر بازگشتی $T(N) = N + 1$	روش بازگشتی $T(N) > 2^{N/2}$	زمان اجرای روش غیر بازگشتی (۱-۷)	زمان اجرای روش بازگشتی (۱-۶)
۴۰	۴۱	بیش از 1,048,576	41 x 1 Nano second = 41 Ns	1048 μ s (10^{-6})
۶۰	۶۱	بیش از 1.1×10^{9}	61 Ns	1 s
۸۰	۸۱	بیش از 1.1×10^{12}	81 Ns	18 Minute
۱۰۰	۱۰۱	بیش از 1.1×10^{15}	101 Ns	13 Days
۱۲۰	۱۲۱	بیش از 1.2×10^{18}	121 Ns	36 Years
۱۶۰	۱۶۱	بیش از 1.2×10^{24}	161 Ns	3.8×10^{7} Years
۲۰۰	۲۰۱	بیش از 1.3×10^{30}	201 Ns	4×10^{13} Years

تحلیل پیچیدگی زمانی الگوریتم ها

هدف از تحلیل پیچیدگی زمانی، تخمین زمان اجرای الگوریتم و محاسبهٔ کارایی بر حسب زمان می باشد و این تحلیل، با تعیین تعداد دفعاتی که عملیات اصلی یک الگوریتم (مثل عمل جمع، عمل ضرب، عمل مقایسه، حلقه و ...) انجام می شود، صورت میپذیرد. در واقع باید دید عمل اصلی الگوریتم چیست تا با بدست آوردن تقریبی زمان اجرای آن، زمان تقریبی کل الگوریتم بدست آید. همانطور که قبلاً اشاره شد، برای تحلیل پیچیدگی یک الگوریتم، تابعی به نام T(N) در نظر می گیریم. در این راستا، برای تحلیل پیچیدگی زمانی در بهتریـــن حالت یا Best Case تابع B(N)، برای تحلیل پیچیدگی زمانی در بدتریـــن حالت یا Worst Case تابع W(N)، و برای تحلیل پیچیدگی زمانی در حالت میانگین یا Average تابع A(N) را در نظر می گیریم که در مسائلی که می توان آنها را مطرح کرد و می تواند وجود داشته باشد مورد استفاده واقع می شوند.

هم اکنون به تحلیل پیچیدگی الگوریتم های ارائه شده قبلی (۱-۱ الی ۷-۱) می پردازیم:

الگوریتم 1-1: جستجوی ترتیبی یا Sequential Search

در این الگوریتم جستجو از ابتدا تا انتها انجام می شود (N+1) مرتبه

```
Function Linear_Search (K, N, X)
1. [Initialize Search]
  I ← 1
  K [N+1] ← X
2. [Search The Array]
  Repeat While K [I] ≠ X
        I ← I + 1
3. [Is the search Successful?]
   If I = N+1
  Then Write ('Unsuccessful Search')
        Return (0)
  Else Write ('Successful Search')
       Return (I)
```

عملیات اصلی: عمل مقایسه

بهترین حالت: زمانی که عنصر مورد نظر در خانه اول باشد (Best Case) B (N) = 1

بدترین حالت: زمانی که عنصر مورد نظر در آرایه نباشد (Worst Case) W (N) = N+1

چون در الگوریتم ابتدا خودمان عنصر مورد نظر را در انتها قرار دادیم

الگوریتم 2-1: جمع مقادیر موجود در یک آرایه یا Sum

```
Function Sum_Values (A, N)
۱ . [Initialization]
  Sum ← 0
2. [Sum the values in the Array]
  Repeat For I = 1 to N                    تعداد انجام تکرار N
        Sum ← Sum + A [I]                  عملیات اصلی: عمل جمع
3. [Finished]                              T (N) = N
  Return (Sum)
```

الگوریتم 3-1: مرتب سازی تعویضی یا Exchange Sort

```
Procedure Exchange_Sort (N, S)
1. [Exchange Sort]
 Repeat For I = 1 to N-1
        Repeat For J = I+1 to N
              If S [J] < S [I]             عملیات اصلی: عمل مقایسه
              Then S [J] ←→ S [I]
2. [Finished]
 Exit
```

در این الگوریتم دو حلقه For تو در تو داریم.

در حلقه بیرونی از ۱ تا N – 1 تکراردایریم که در مجموع N – 1 بار می شود پس $T(N) = N - 1$

در حلقه درونی تعداد دفعات تکرار متغیر بوده که از I + 1 آغاز و تا N ادامه می یابد:

I	J	تعداد مقایسه
۱	2..N	بار اول N-1 بار تکرارانجام می شود بنابراین N-1 مقایسه داریم و لذا $T(N) = N - 1$
۲	3..N	بار دوم N-2 بار تکرارانجام می شود بنابراین N-2 مقایسه داریم و لذا $T(N) = N - 2$
۳	4..N	بار سوم N-3 بار تکرارانجام می شود بنابراین N-3 مقایسه داریم و لذا $T(N) = N - 3$
…….	……..	…….
N – 1	N..N	بار N اُم 1 بار تکرارانجام می شود بنابراین 1 مقایسه داریم و لذا $T(N) = 1$

$$T(N) = (N-1) + (N-2) + (N-3) + \ldots\ldots\ldots + 3 + 2 + 1 = (N-1) \times N / 2$$

بنابراین تعداد محاسبات کل الگوریتم برابر است با: $T(N) = (N-1) \times N / 2$

N=5 S = {10, 6, 12, 9, 6}

I	J	S[ ]
1		10, 6, 12, 9, 6
	2	S [2] < S [1] ➔ جابجایی عناصر دوم و اول انجام می شود ➔ 6, 10, 12, 9, 6
	3	S [3] < S [1] نیست پس جابجایی نداریم
	4	S [4] < S [1] نیست پس جابجایی نداریم
	5	S [5] < S [1] نیست پس جابجایی نداریم
2		6, 10, 12, 9, 6
	3	S [3] < S [2] نیست پس جابجایی نداریم
	4	S [4] < S [2] ➔ جابجایی عناصر چهارم و دوم انجام می شود ➔ 6, 9, 12, 10, 6
	5	S [5] < S [2] ➔ جابجایی عناصر پنجم و دوم انجام می شود ➔ 6, 6, 12, 10, 9
3		6, 6, 12, 10, 9
	4	S [4] < S [3] ➔ جابجایی عناصر چهارم و سوم انجام می شود ➔ 6, 6, 10, 12, 9
	5	S [5] < S [3] ➔ جابجایی عناصر پنجم و سوم انجام می شود ➔ 6, 6, 9, 12, 10
4		6, 6, 9, 12, 10
	5	S [5] < S [4] ➔ جابجایی عناصر پنجم و چهارم انجام می شود ➔ 6, 6, 9, 10, 12

الگوریتم 4-1: ضرب دو ماتریس NxN

```
Procedure Matrix_Multiplication (A, B, C, N)
1. [Multiply Matrices A and B and Store the Result in Matrix C]
   Repeat For I = 1 To N
         Repeat For J = 1 To N
               Sum ← 0
               Repeat For K = 1 To N
                     Sum ← Sum + A [I, K] * B [K, J]      عملیات اصلی: عمل ضرب سطرها و ستون ها
               C [I, J] ← Sum
2. [Finished]
   Exit
```

تعداد سطر ها و ستون ها برابر با N می باشد. سه حلقه تو در تو نیز داریم که هر یک N مرتبه تکرار می شوند:

$N \times N \times N = N^3$ ➔ $\boxed{T(N) = N^3}$

الگوریتم ۵-۱: الگوریتم جستجوی دودویی یا Binary Search

Function Binary_Search (A, N, X) **تعداد اعضا برابر با N**

1. [Initialize]

Low=1 **(مقدار پایین=Low)**

High=N **(مقدار بالا=High)**

2. [Perform Binary Search]

Repeat thru Step 4 while Low ≤ High

3. [Obtain Middle]

$$\text{Middle} \leftarrow \left\lfloor \frac{Low+High}{2} \right\rfloor$$

4. [Compare]

If X < A [Middle]

Then High ← Middle – 1 **عملیات اصلی: مقایسه**

Else If X > A [Middle]

Then Low ← Middle + 1

Else Write ('Successful Search')

Return (Middle)

5. [Unsuccessful Search]

Write ('Unsuccessful Search')

Return (0)

تعداد مقایسه ها در این الگوریتم اولاً وابسته به اعداد ذخیره شده در آرایه و ثانیاً وابسته به عضوی است که به دنبال آن می گردیم می باشد.

بهترین حالت یا Best Case: عضو مورد نظر در وسط باشد. $\boxed{B(N) = 1}$

بدترین حالت یا Worst Case: عضو مورد نظر وجود نداشته باشد و از همه بزرگتر یا کوچکتر باشد. $\boxed{W(N) = 1 + \log_2 N}$

الگوریتم ۶-۱: الگوریتم جمله n اُم فیبوناچی Fibonacci **(روش بازگشتی یا Recursive – بالا به پایین یا جزء به کل)**

Function Fib (N) **با فرض آنکه N عدد صحیح است**

1. [Calculate Fibonacci in Recursive Form]

If N ≤ 1

Then Return (N)

Else Return (Fib (N-1) + Fib (N-2))

عملیات اصلی: محاسبه یک جمله که بستگی به بزرگی N دارد.

در این الگوریتم دیدیم که هر گاه عدد مورد نظر ۲ واحد افزایش می یابد، تعداد محاسبات بیش از ۲ برابر بود: N T (N)

5 15

بیش از ۲ برابر قبلی یعنی ۱۵ است 7 41

بیش از ۲ برابر قبلی یعنی ۴۱ است 9 109

و لذا داشتیم: $T(N) > 2^{N/2}$

الگوریتم ۷-۱: الگوریتم جمله n اُم فیبوناچی Fibonacci (روش غیر بازگشتی – تکرار شونده – بهبود یافته – پایین به بالا)

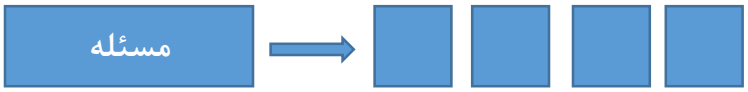
```
Function Fib2 (N, F)
1. [Initialization]
   F [0] = 0                طبق تعریف تابع
2. [Calculate Fibonacci]
   If N > 0
   Then F [1] ← 1           طبق تعریف تابع
        Repeat For I = 2 To N
             F [I] ← F [I-1] + F [I-2]        عملیات اصلی: عمل جمع
3. [Finished]
   Return (F [I])           البته در نهایت I = N می گردد //// Return (F [N]) هم صحیح می باشد
```

$T(N) = N + 1$

فصل دوم: روش تقسیم و حل یا تقسیم و غلبه (**Divide and Conquer**) در حل مسائل

روش تقسیم و غلبه، روشی بالا به پایین یا **Top – Down** است که فی الواقع در آن، یک مسئله به دو یا چند زیر مسئلهٔ کوچکتر تقسیم می شود و هر یک از آنها به طور جداگانه حل می گردند و در نهایت، نتایج حاصل به طریقی در یکدیگر ادغام شده تا جواب مسئلهٔ اصلی به دست آید:

مسائل کوچکتر ← حل ساده تر



در واقع برخی از این موارد را قبلاً بررسی کرده ایم که عبارت از روش اول محاسبه فیبوناچی (روش بازگشتی) بوده است. حال به بررسی مواردی دیگر خواهیم پرداخت.

الگوریتم ۱-۲: الگوریتم مرتب سازی ادغامی یا Merge Sort:

در این مسئله، در واقع یک مجموعه اطلاعات نامرتب داریم که در آرایه ای به نام **S** ذخیره شده اند و تعداد آنها برابر با **N** است. روش انجام کار چنین است که اطلاعات موجود در آرایه **S** قرار را نصف نموده و به دو زیر آرایه **U** و **V** تقسیم می نماییم. البته در نظر داریم که در زمان تقسیم آرایه به دو قسمت و نصف کردن آن، در صورت فرد بودن تعداد عناصر، حد پایین را در نظر بگیریم:

$\left\lfloor \frac{N}{2} \right\rfloor$

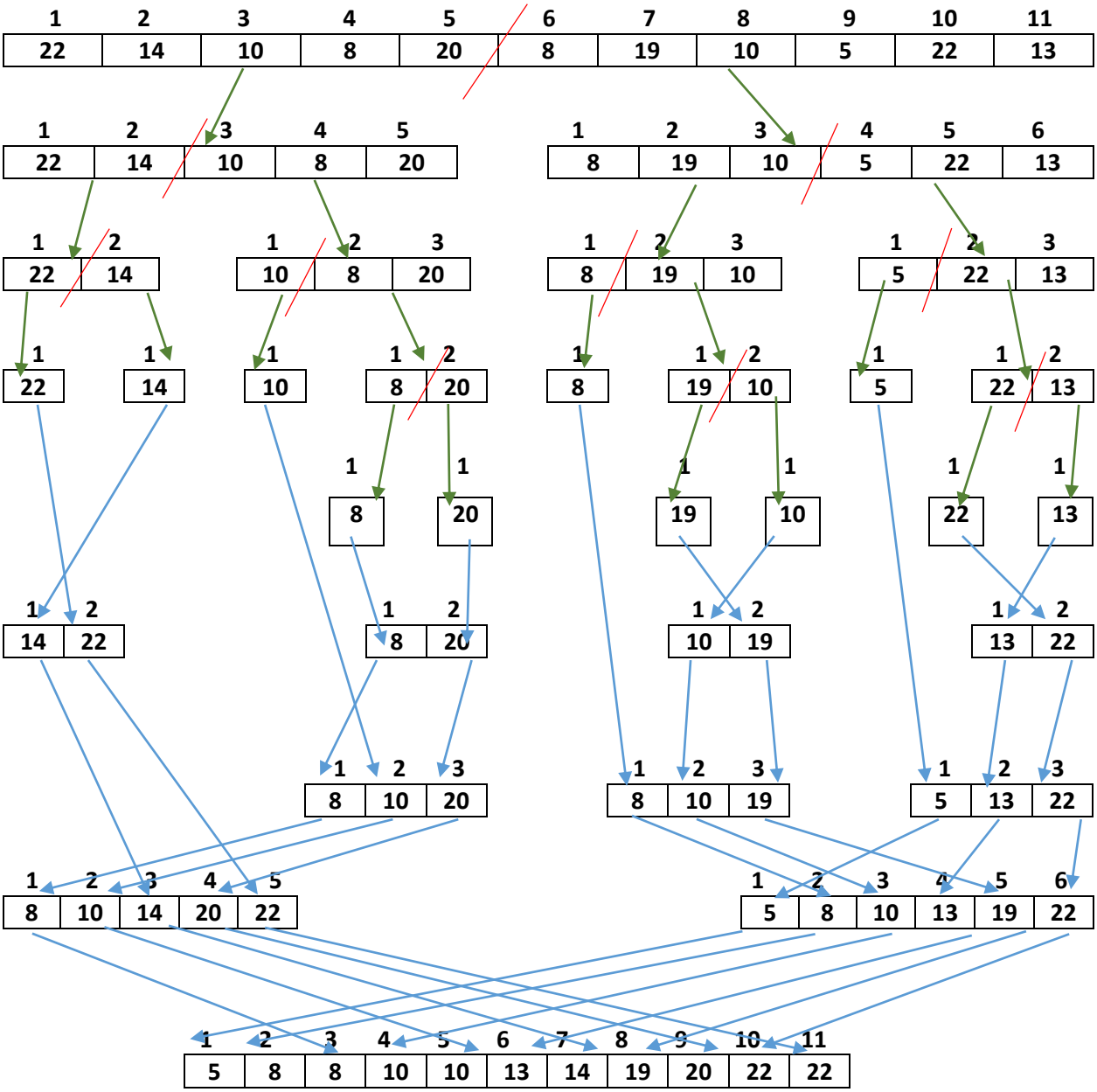
سپس این تقسیمات را به همین ترتیب آنقدر ادامه می دهیم تا به آرایه هایی با طول ۱ برسیم و آرایه ای که دارای ۱ عنصر باشد، ذاتاً مرتب است.

الی آخر تا به آرایه های با طول ۱

تا به اینجای کار، عمل تقسیم نمودن یا **Divide** انجام شد و به اتمام رسید.

حال با آرایه های تک عضوی و ذاتاً مرتب، عمل ادغام (غلبه یا **Conquer**) و مرتب سازی را انجام می دهیم تا در نهایت آرایه مرتب شود.

مثال: آرایه زیر مفروض است، آن را با روش **Merge Sort** مرتب می نماییم: (۱۱ عنصر داریم که حد پایین ۱۱/۲ برابر ۵ است)



نکته: در صورت برابر بودن دو مقدار در آرایه های **U** و **V**، اولویت با آرایه **V** (سمت راستی) می باشد که علت آن را در ادامه و در الگوریتم می بینیم:

الگوریتم **Merge_Sort** برای مرتب سازی یک آرایه یا مجموعه از اطلاعات غیر مرتب که دارای دو بخش تقسیم و ادغام می باشد.

```
Procedure Merge_Sort (N, S)                    بخش تقسیم:  آرایه اصلی S   تعداد عناصر N
1. [Splitting into two halves]
   If N > 1
   Then H ← ⌊N/2⌋                              در مثال بررسی شده برابر است با ۵
        M ← N – H                              در مثال بررسی شده برابر است با ۶ = ۵ – ۱۱
2. [Copying Form S to U and V]
        Repeat For I = 1 To H                  کپی بخش اول آرایه اصلی درون آرایه U   (Copy S [1 .. H] to U)
              U [I] ← S [I]
        Repeat For J = M to N                  کپی بخش دوم آرایه اصلی درون آرایه V   (Copy S [H+1 .. N] to V)
              K ← 1
              V [K] ← S [J]
              K ← K + 1
3. [Merge Sorting Array U]
        Call Merge_Sort (H, U)
4. [Merge Sorting Array V]
        Call Merge_Sort (M, V)
5. [Merging the Sorted Arrays U and V in S]
        Call Merge (H, M, U, V, S)
6. [Finished]
   Exit
```

الگوریتم **Merge**، برای ادغام دو آرایه یا مجموعه از اطلاعات مرتب **U** و **V**:

```
Procedure Merge (H, M, U, V, S)    S آرایه حاصل ,V آرایه دوم ,U آرایه اول ,M طول آرایه دوم ,H طول آرایه اول
1. [Initialization]
   I ← J ← K ← 1
2. [Sorting]
   Repeat While I ≤ H and J ≤ M        این حلقه تا زمانی اجرا می شود که یکی از دو آرایه مرتب شده
                                       U یا V
         If U [I] < V [J]              به انتها برسد و تمام شود، سپس در مرحلهٔ بعد (۳) ، عناصر باقی
         Then S [K] ← U [I]            مانده در آرایه ای که هنوز به اتمام نرسیده، در آرایه حاصل کپی شود.
              I ← I + 1
         Else S [K] ← V [J]
              J ← J + 1
         K ← K + 1
3. [Copying the rest of one remaining sorted array]
   If I > H
   Then For Index = J To M             کپی باقیمانده V در S
              S [K] ← V [Index]          (Copy V [J .. M] to S [K .. H+M])
              K ← K + 1
   Else For Index = I To H             کپی باقیمانده U در S
              S [K] ← U [Index]          (Copy U [I .. H] to S [K .. H+M])
              K ← K + 1
4. [Finished]
   Exit
```

مثال: دو آرایه مرتب U و V به شرح زیر مفروض هستند. با ردیابی الگوریتم Merge، یک آرایه مرتب شده بدست آورید:

U=	7	12	19	22	35	H= 5	/	V=	5	9	12	20	38	40	M= 6

I	J	K	S[ ]
1	1	1	I ≤ H (1 ≤ 5) , J ≤ M (1 ≤ 6)
			U[I] NOT < V[J] == > S[K] ← 5
	2	2	I ≤ H (1 ≤ 5) , J ≤ M (2 ≤ 6)
			U[I] < V[J] == > S[K] ← 5, 7
2		3	I ≤ H (2 ≤ 5) , J ≤ M (2 ≤ 6)
			U[I] NOT < V[J] == > S[K] ← 5, 7, 9
	3	4	I ≤ H (2 ≤ 5) , J ≤ M (3 ≤ 6)
			U[I] NOT < V[J] == > S[K] ← 5, 7, 9, 12 From Array V
	4	5	I ≤ H (2 ≤ 5) , J ≤ M (4 ≤ 6)
			U[I] < V[J] == > S[K] ← 5, 7, 9, 12, 12 From Array U
3		6	I ≤ H (3 ≤ 5) , J ≤ M (4 ≤ 6)
			U[I] < V[J] == > S[K] ← 5, 7, 9, 12, 12, 19
4		7	I ≤ H (4 ≤ 5) , J ≤ M (4 ≤ 6)
			U[I] NOT < V[J] == > S[K] ← 5, 7, 9, 12, 12, 19, 20
	5	8	I ≤ H (4 ≤ 5) , J ≤ M (5 ≤ 6)
			U[I] < V[J] == > S[K] ← 5, 7, 9, 12, 12, 19, 20, 22
5		9	I ≤ H (5 = 5) , J ≤ M (5 ≤ 6)
			U[I] < V[J] == > S[K] ← 5, 7, 9, 12, 12, 19, 20, 22, 35
6		10	I = 6 > H = 5 == > Copy From V to S
			J = 5 (S[K] ← V[5]) == > S[K] ← 5, 7, 9, 12, 12, 19, 20, 22, 35, 38 (Copy 38 From V)
	6	11	J = 6 (S[K] ← V[6]) == > S[K] ← 5, 7, 9, 12, 12, 19, 20, 22, 35, 38, 40 (Copy 40 From V)

تمرین: دو آرایه مرتب U و V به شرح زیر مفروض هستند. با ردیابی الگوریتم Merge، یک آرایه مرتب شده بدست آورید:

U = {8, 10, 14, 20, 22} H = 5 / V = {5, 8, 10, 13, 19, 22} M = 6

الگوریتم ۲-۲: الگوریتم مرتب سازی سریع یا Quick Sort یا Partition Exchange Sort:

در پیاده سازی این الگوریتم، در راستای استراتژی تقسیم و حل، از دو بخش مجزا بهره گیری شده است. بخش اول عمل تقسیم بندی یا Partition می باشد (قسمت Divide) و بخش دوم (قسمت Conquer) به شکل بازگشتی، عمل مرتب سازی یا Quick Sort را انجام می دهد.

در بخش تقسیم بندی (Partition)، ابتدا یک عنصر محوری یا Pivot که معمولاً عضو اول است (اما می تواند عنصر انتهایی یا میانی هم باشد)، انتخاب می گردد و سپس تقسیم بندی آرایه انجام می شود. با این عمل (تقسیم بندی) در واقع چینش یا نحوهٔ قرار گرفتن عناصر آرایه به شکلی تغییر می یابد که تمامی عناصر کوچکتر یا مساوی عنصر محوری در سمت چپ آن، و تمام عناصر بزرگتر در سمت راست آن قرار خواهند گرفت که موسوم به زیرآرایه های چپ و راست بوده و الزاماً خود این زیرآرایه ها در مراحل اولیه مرتب نیستند، اما عنصر محوری درانتهای هر دور از محاسبات دقیقاً در جای صحیح خود در آرایه مرتب مستقر می شود و دیگر جابجا نخواهد شد و در واقع محل نهایی آن معلوم خواهد شد:

بزرگتر از محور	محور	کوچکتر از محور

حال کافی است این روند را برای دو زیر آرایه چپ و راست آنقدر تکرار نماییم تا کل عناصر مرتب شوند:

```
Procedure Quick_Sort (S, Low, High)          نام آرایه = S , اندیس پایان = High ,  اندیس آغاز = Low
If High > Low
Then Pivot ← Partition (S, Low, High)        در اینجا، محل نهایی همان عضو محور انتخابی اولیه است Pivot
     Call Quick _Sort (S, Low, (Pivot – 1))
     Call Quick_Sort (S, (Pivot + 1), High)
Exit
```

```
Function Partition (S, Low, High)
Pivot_Value ← S [Low]                        مقدار عضو انتخابی محور
J ← Low
Repeat For I = (Low + 1) to High
        If S [I] < Pivot_Value
        Then J ← J + 1
             S [I] ←→ S [J]
Pivot_Point ← J                              محل نهایی عضو محور
S [Low] ←→ S [Pivot_Point]
Return (Pivot_Point)
```

مثال: آرایه زیر را به روش **Quick Sort** مرتب سازی نمایید:

	1	2	3	4	5	6	7	8
S =	15	22	13	27	12	10	20	25

Low = 1 , High = 8 , High > Low

الگوریتم **Partition: Pivot_Value = S[Low] = 15 , J = Low = 1**

J	I	Pivot_Value	آرایه S[ ]	Pivot_Piont
		15	15, 22, 13, 27, 12, 10, 20, 25	
1				
	2	S[2]=22 NOT < 15		
	3	S[3]=13 < 15		
2			15, 13, 22, 27, 12, 10, 20, 25 (عناصر دوم و سوم با هم جابجا)(I ,J)	
	4	S[4]=27 Not < 15		
	5	S[5]=12 < 15		
3			15, 13, 12, 27, 22, 10, 20, 25 (عناصر سوم و پنجم با هم جابجا)	
	6	S[6]=10 < 15		
4			15, 13, 12, 10, 22, 27, 20, 25 (عناصر چهارم و ششم با هم جابجا)	
	7	S[7]=20 NOT < 15		
	8	S[8]=25 Not < 15		
		تثبیت ۱۵ در محل ۴	10, 13, 12, 15, 22, 27, 20, 25 (S[Low] ←→ S[Pivot_Point])	4 (J = 4)

Quick_Sort : Pivot = 4

Low = 1 , High = Pivot – 1 = 3 , Quick_Sort (S, 1, 3) , High > Low

الگوریتم Partition: Pivot_Value = S[Low] = 10 , J = Low = 1

J	I	Pivot_Value	S[ ] آرایه	Pivot_Piont
		10	10, 13, 12, 15, 22, 27, 20, 25	
1				
	2	S[2]=13 NOT < 10		
	3	S[3]=12 NOT < 10		
		تثبیت ۱۰ در محل ۱	10, 13, 12, 15, 22, 27, 20, 25 (S[Low] ←→ S[Pivot_Point]) درواقع بدون جابجایی	1 (J = 1)

Quick_Sort : Pivot = 1

Low = 1, High = Pivot – 1 = 0 , Quick_Sort (S, 1, 0) , High NOT > Low == >

(S, (Pivot + 1), High) عملیات در زیر آرایه راست **== >**

Low = Pivot + 1 = 2 , High = 3 , Quick_Sort (S, 2, 3) , High > Low

الگوریتم Partition: Pivot_Value = S[Low] = 13 , J = Low = 2

J	I	Pivot_Value	S[ ] آرایه	Pivot_Piont
		13	10, 13, 12, 15, 22, 27, 20, 25	
2				
	3	S[3]=12 < 13		
3			10, 13, 12, 15, 22, 27, 20, 25 (جابجایی ۳و ۳ در واقع بدون جابجایی)	
		تثبیت ۱۳ در محل ۳	10, 12, 13, 15, 22, 27, 20, 25 (S[Low] ←→ S[Pivot_Point])	3 (J = 3)

Quick_Sort : Pivot = 3

Low = 2, High = Pivot – 1 = 2 , Quick_Sort (S, 2, 2) , High NOT > Low == >

(S, (Pivot + 1), High) عملیات در زیر آرایه راست == > (دیگر ۱۲ خود به خود در جای خود باقی می ماند)

Low = Pivot + 1 = 5 , High = 8 , Quick_Sort (S, 5, 8) , High > Low

الگوریتم Partition: Pivot_Value = S[Low] = 22 , J = Low = 5

J	I	Pivot_Value	آرایه S[ ]	Pivot_Piont
		22	10, 12, 13, 15, 22, 27, 20, 25	
5				
	6	S[6]=27 NOT < 22		
	7	S[7]=20 < 22		
6			10, 12, 13, 15, 22, 20, 27, 25 (جابجایی عناصر ششم و هفتم)	
	8	S[8]=25 NOT < 22		
		تثبیت ۲۲ در محل ۶	10, 12, 13, 15, 20, 22, 27, 25 (S[Low] ←→ S[Pivot_Point])	6 (J = 6)

Quick_Sort : Pivot = 6

Low = 5, High = Pivot – 1 = 5 , Quick_Sort (S, 5, 5) , High NOT > Low == >

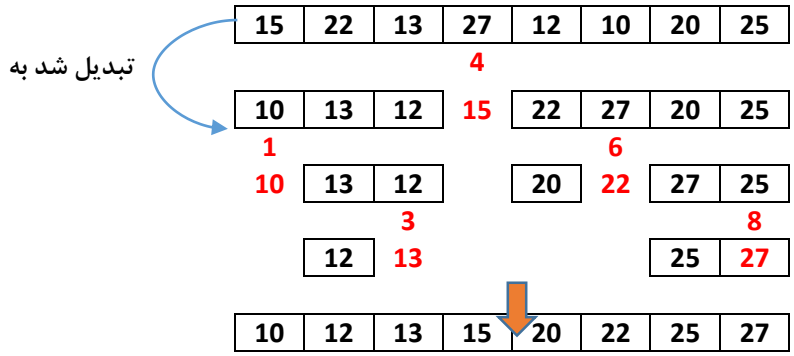
(S, (Pivot + 1), High) عملیات در زیر آرایه راست == > (دیگر ۲۰ خود به خود در جای خود باقی می ماند)

Low = Pivot + 1 = 7 , High = 8 , Quick_Sort (S, 7, 8) , High > Low

الگوریتم پارتیشن: Pivot_Value = S[Low] = 27 , J = Low = 7

J	I	Pivot_Value	آرایه S[ ]	Pivot_Piont
		۲۷	10, 12, 13, 15, 20, 22, 27, 25	
7				
	8	S[8]=25 < 27		
8			S[I] ←→ S[J] (جابجایی ۸ و ۸ در واقع بدون جابجایی)	
		تثبیت ۲۷ در محل ۸	10, 12, 13, 15, 20, 22, 25, 27 (S[Low] ←→ S[Pivot_Point])	8 (J = 8)

خروجی عدد ۸ خواهد بود و چون High>Low نیست،عملیات به اتمام رسیده و عناصر مرتب شده اند.



تمرین: آرایه زیر را به روش **Quick_Sort** مرتب سازی نمایید: (ردیابی الگوریتم)

	1	2	3	4	5	6	7
S=	19	10	8	10	13	5	11

Low = 1 , High = 7

فصل سوم: برنامه سازی پویا Dynamic Programming

روش برنامه سازی پویا معمولاً برای الگوریتم هایی به کار می رود که به دنبال حل مسئله ای به صورت بهینه هستند.

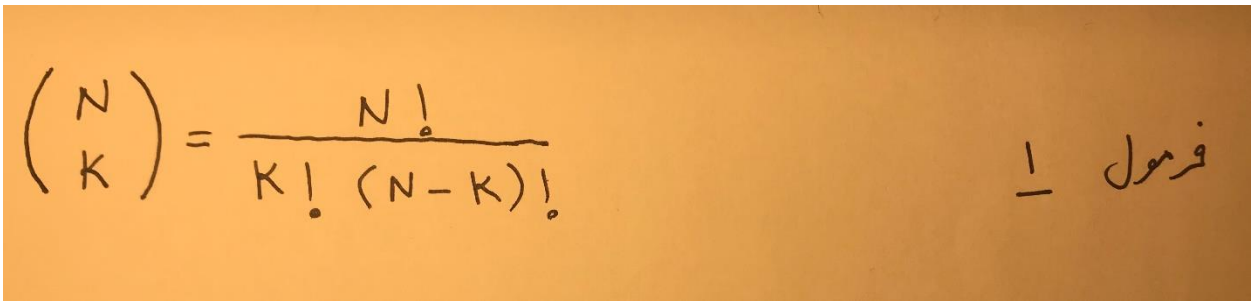
در روش بررسی شده ی قبلی یعنی روش تقسیم و حل یا Divide and Conquer، دیدیم که ممکن است بعضی از زیر مسائل کوچکتر، با هم برابر و یکسان باشند (به عنوان مثال الگوریتم بازگشتی حل مسئله فیبوناچی ۶-۱) که حلِ این زیر مسائلِ با هم برابر، به طور تکراری، چندین بار انجام و در واقع حل می شدند و این، بعضاً مطلوب نبود.

ایده ای که در برنامه سازی پویا نهفته است، جلوگیری از این محاسبات تکراری است و روشی که معمولاً به کار گرفته می شود، استفاده از یک جدول برای ذخیره نتایج حاصل از حل زیر مسائل است. در این صورت، اگر الگوریتم به زیر مسئله ای برخورد کرد که پیش از این حل شده است، به جای حل مجدد آن، نتیجه محاسبات قبلی را از جدول برداشته و کار را با حل زیر مسئله بعدی دنبال می کند.

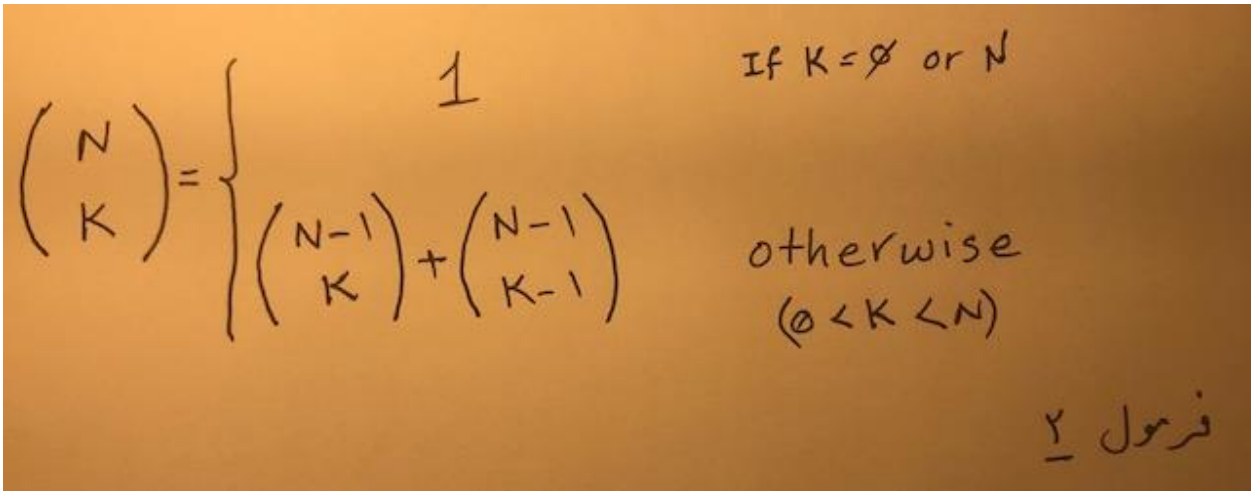
روش برنامه سازی پویا که یک روش پایین به بالا یا Bottom Up می باشد،و آن به این معنی است که از حل مسائل کوچکتر، به حل مسائل بزرگتر می رسیم (به عنوان مثال الگوریتم تکرار شونده حل مسئله فیبوناچی ۷-۱).

هم اکنون در رابطه با مفهوم روش برنامه سازی پویا، الگوریتم یافتن تعداد Combinations یا ترکیب های K عضو از یک مجموعه N عضوی را با استفاده از دو روش تقسیم و حل (بالا به پایین یا Top Down) و برنامه سازی پویا (پایین به بالا یا Bottom Up) بررسی می نماییم.

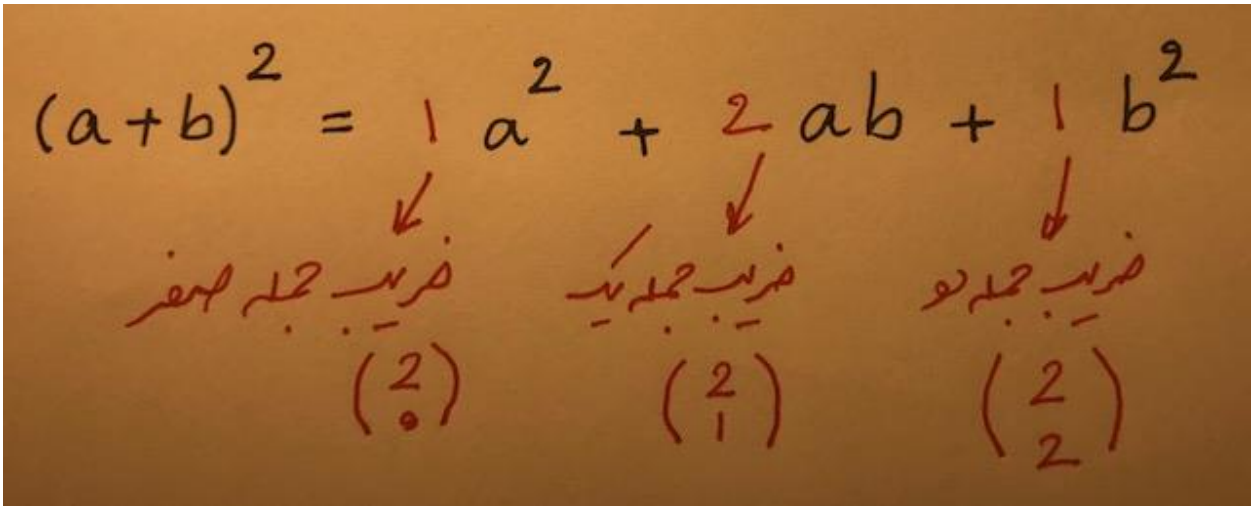
طبق تعاریف موجود در ریاضیات، برای یافتن تعداد ترکیب های K عضو از یک مجموعه N عضوی، فرمول زیر را داریم که دارای تعداد زیادی عمل ضرب است:

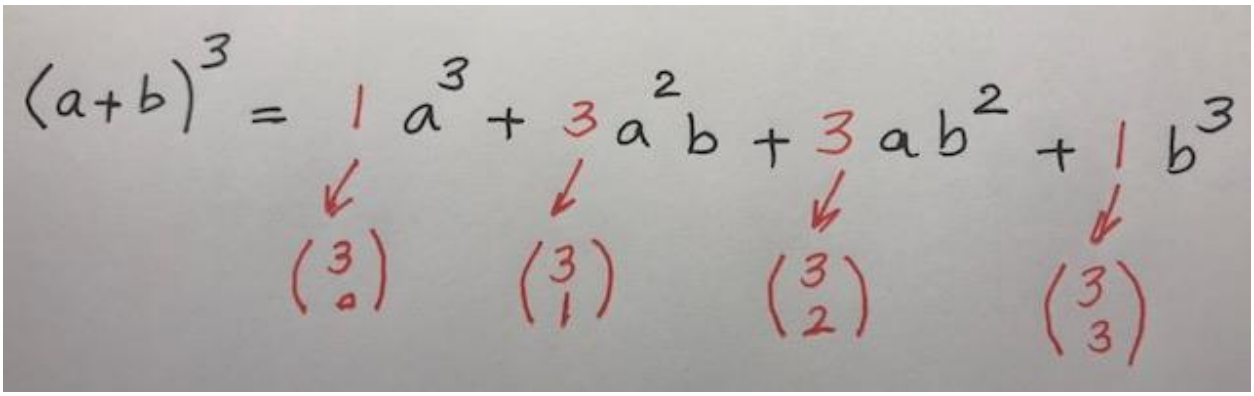


به منظور پیشگیری از انجام عمل های ضرب زیاد و نیز برای بدست آوردن رابطه ای بهینه، می توان از فرمول بازگشتی زیر که معادل فرمول فوق است، استفاده نمود:



ضمناً یادآوری می گردد که در ریاضیات، به هر یک از اعداد صحیح که به صورت ضریب در بسط دو جمله ای ظاهر می شوند، ضریب دو جمله ای یا Binomial Coefficient می گویند که آنها نیز از همان فرمول یافتن تعداد ترکیب های K عضو از یک مجموعه N عضوی بدست می آیند:





الگوریتم ۱-۳: الگوریتم محاسبه تعداد ترکیب های K عضو از یک مجموعه N عضوی (ضرایب دو جمله ای) با روش تقسیم و حل:

Function Bin1 (N, K)

If K = 0 or K = N **با توجه به فرمول ۲**

Then Return (1)

Else Return (Bin1 (N-1, K) + Bin1 (N-1, K-1))

الگوریتم ۲-۳: الگوریتم محاسبه تعداد ترکیب های K عضو از یک مجموعه N عضوی (ضرایب بسط دو جمله ای) با روش برنامه سازی پویا. نام جدول مورد نیاز برای ذخیره نتایج حاصل از حل زیر مسائل، C می باشد که اندازه آن (N + 1) x (N + 1) است:

```
Function Bin2 (N, K)
Repeat For I_N = 0 To N
        C [I_N, 0] ← 1
        C [I_N, I_N] ← 1
Repeat For I_N = 2 To N
        Repeat For K = 1 To I_N – 1
                C [I_N, K] ← C [I_N-1, K] + C [I_N-1, K-1]
Return(C [I_N, K])
```

مثال: با ردیابی الگوریتم فوق (۲-۳)، تعداد ترکیب های ۵ عضو از یک مجموعه ۸ عضوی را بدست آورید. $\binom{8}{5}$
در عبارت فوق، N = 8 و K = 5 بوده و جدول حاصل نیز، 9 x 9 می باشد (۰..8 x ۰..8)

C K →

I_N ↓	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1		1						
3	1			1					
4	1				1				
5	1					1			
6	1						1		
7	1							1	
8	1								1

پس از اجرای حلقه اول، جدول به شکل فوق در می آید.

حال در ردیابی حلقه دوم داریم:

I_N	K	C[I_N , K]
2		
	1	C[I_N , K] = C[2 , 1] = C[I_N-1 , K] + C[I_N-1 , K-1] = C[1 , 1] + C[1 , 0] = 1 + 1 = 2
3		
	1	C[I_N , K] = C[3 , 1] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[2 , 1] + C[2 , 0] = 2 + 1 = 3
	2	C[I_N , K] = C[3 , 2] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[2 , 2] + C[2 , 1] = 1 + 2 = 3
4		
	1	C[I_N , K] = C[4 , 1] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[3 , 1] + C[3 , 0] = 3 + 1 = 4
	2	C[I_N , K] = C[4 , 2] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[3 , 2] + C[3 , 1] = 3 + 3 = 6
	3	C[I_N , K] = C[4 , 3] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[3 , 3] + C[3 , 2] = 1 + 3 = 4
5		
	1	C[I_N , K] = C[5 , 1] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[4 , 1] + C[4 , 0] = 4 + 1 = 5
	2	C[I_N , K] = C[5 , 2] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[4 , 2] + C[4 , 1] = 6 + 4 = 10
	3	C[I_N , K] = C[5 , 3] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[4 , 3] + C[4 , 2] = 4 + 6 = 10
	4	C[I_N , K] = C[5 , 4] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[4 , 4] + C[4 , 3] = 1 + 4 = 5
6		
	1	C[I_N , K] = C[6 , 1] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[5 , 1] + C[5 , 0] = 5 + 1 = 6
	2	C[I_N , K] = C[6 , 2] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[5 , 2] + C[5 , 1] = 10 + 5 = 15
	3	C[I_N , K] = C[6 , 3] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[5 , 3] + C[5 , 2] = 10 + 10 = 20
	4	C[I_N , K] = C[6 , 4] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[5 , 4] + C[5 , 3] = 5 + 10 = 15
	5	C[I_N , K] = C[6 , 5] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[5 , 5] + C[5 , 4] = 1 + 5 = 6
7		
	1	C[I_N , K] = C[7 , 1] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[6 , 1] + C[6 , 0] = 6 + 1 = 7
	2	C[I_N , K] = C[7 , 2] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[6 , 2] + C[6 , 1] = 15 + 6 = 21
	3	C[I_N , K] = C[7 , 3] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[6 , 3] + C[6 , 2] = 20 + 15 = 35
	4	C[I_N , K] = C[7 , 4] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[6 , 4] + C[6 , 3] = 15 + 20 = 35
	5	C[I_N , K] = C[7 , 5] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[6 , 5] + C[6 , 4] = 6 + 15 = 21
	6	C[I_N , K] = C[7 , 6] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[6 , 6] + C[6 , 5] = 1 + 6 = 7
8		
	1	C[I_N , K] = C[8 , 1] = C[I-1 , K] + C[I_N -1 , K-1] = C[7 , 1] + C[7 , 0] = 7 + 1 = 8
	2	C[I_N , K] = C[8 , 2] = C[I-1 , K] + C[I_N -1 , K-1] = C[7 , 2] + C[7 , 1] = 21 + 7 = 28
	3	C[I_N , K] = C[8 , 3] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[7 , 3] + C[7 , 2] = 35 + 21 = 56

	4	C[I_N , K] = C[8 , 4] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[7 , 4] + C[7 , 3] = 35 + 35 = 70
	5	C[I_N , K]= C[8 , 5]= C[I_N -1 , K] + C[I_N -1 , K-1]= C[7 , 5] + C[7 , 4]= 21 + 35 = 56 حاصل ترکیب ۵ از ۸
	6	C[I_N , K] = C[8 , 6] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[7 , 6] + C[7 , 5] = 7 + 21 = 28
	7	C[I_N , K] = C[8 , 7] = C[I_N -1 , K] + C[I_N -1 , K-1] = C[7 , 7] + C[7 , 6] = 1 + 7 = 8

حال با توجه به نتایج حاصل از حلقه تو در تو دوم و محاسبات فوق داریم:

C K ⟶

I_N ↓	0	1	2	3	4	5	۶	۷	۸
0	1								
1	1	1							
ضرایب جمله ۲ ⟶ 2	1	۲	1						
ضرایب جمله ۳ ⟶ 3	1	۳	۳	1					
ضرایب جمله ۴ ⟶ 4	1	۴	۶	۴	1				
ضرایب جمله ۵ ⟶ 5	1	۵	۱۰	۱۰	۵	1			
ضرایب جمله ۶ ⟶ 6	1	۶	۱۵	۲۰	۱۵	۶	1		
ضرایب جمله ۷ ⟶ 7	1	۷	۲۱	۳۵	۳۵	۲۱	۷	1	
ضرایب جمله ۸ ⟶ 8	1	۸	۲۸	۵۶	۷۰	۵۶	۲۸	۸	1

جدول فوق به جدول خیام – پاسکال (مثلث خیام) معروف است و ضرایب تمام دو جمله ای های $(A + B)^N$ از روی هر سطر آن پیدا می گردد:

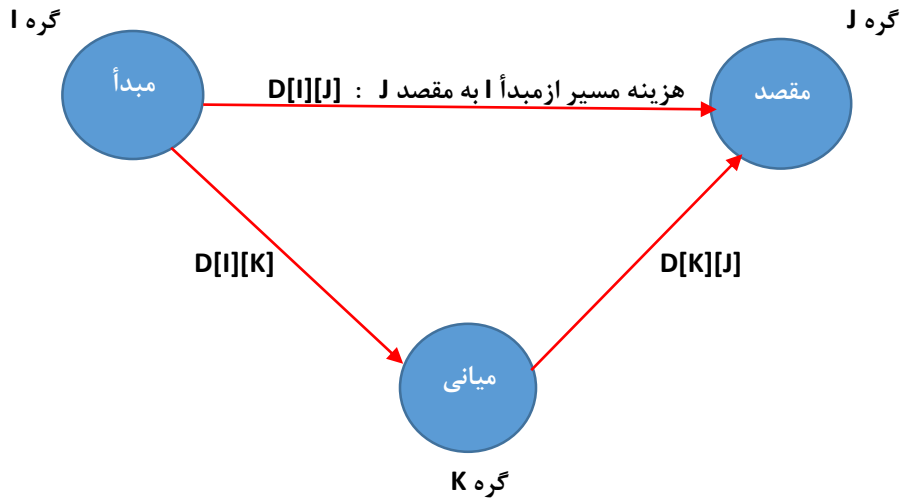
$$(A + B)^2 = 1A^2 + 2AB + 1B^2$$

$$(A + B)^4 = 1A^4 + 4A^3B + 6A^2B^2 + 4AB^3 + 1B^4$$

تمرین: با ردیابی الگوریتم (۲-۳)، تعداد ترکیب های ۸ عضو از یک مجموعه ۱۰ عضوی را بدست آورید. $\binom{10}{8}$

الگوریتم ۳-۳: کوتاه ترین مسیر از هر رأس به هر رأس دیگر در یک گراف (الگوریتم فلوید – وارشال Floyd – Warshall)

هدف این الگوریتم آن است که بررسی نماید آیا به غیر از مسیر مستقیم موجود یا غیر موجود میان دو گره یا رأس در یک گراف، گره یا رأس میانی دیگری وجود دارد که مسیر را کوتاه تر کرده و یا به عبارت دیگر هزینه مسیر میان مبدأ تا مقصد را کمتر نماید یا خیر:



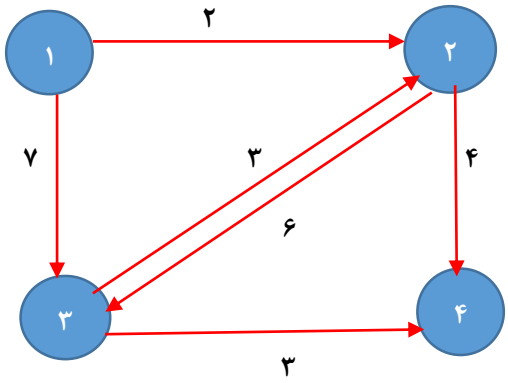
چنانچه در گراف فوق داشته باشیم: D[I][J] > D[I][K] + D[K][J]، در اینصورت گره K هزینه مسیر را کمتر کرده است. الگوریتم فلوید در واقع این موضوع را بررسی می کند که آیا به ازای تمامی گره ها یا رئوس مبدأ، مقصد و میانی، مسیر کوتاه تری یافت می شود یا خیر.

ورودی این الگوریتم، ماتریس مجاورت مربوط به یک گراف جهت دار و وزن دار است و خروجی این الگوریتم، ماتریس کوتاه ترین مسیرها از هر رأس به هر رأس دیگر است. لازم به یادآوری است که ماتریس مجاورت یا همسایگی برای هر گراف جهت دار و وزن دار، عبارت از یک ماتریس N x N است که N تعداد رئوس یا گره ها می باشد و چنانچه بین دو گره ارتباط مستقیم وجود داشته باشد، W[I][J] برابر با مقدار وزن آن و چنانچه ارتباط مستقیمی وجود نداشته باشد، W[I][J] برابر با بی نهایت (∞) خواهد بود.

ضمناً تعریف عمومی گراف عبارت بود از تعدادی عضو و ارتباط های میان آن اعضا در هر سیستم فرضی، و در یک گراف وزن یا Weight عملاً می تواند معرف هر چیزی مرتبط با آن باشد (مثلاً در یک گراف مربوط به راه های موجود درمیان چند شهر مختلف، معرف پمپ های بنزین موجود، یا استراحت گاه ها، یا تعمیرگاه ها و ..... باشد اما در کل آن گراف، مربوط به یک موضوع یکسان است) که به صورت یک عدد بر روی هر لبه یا ارتباط یا Edge نوشته می شود.

به عنوان مثال، ماتریس مجاورت یا همسایگی یا **Weight** برای گراف شکل ۱ به شرح زیر می باشد:

شکل ۱:



ماتریس مجاورت: (I سطر و J ستون) J I

$$W = \begin{bmatrix} 0 & 2 & 7 & \infty \\ \infty & 0 & 6 & 4 \\ \infty & 3 & 0 & 3 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

در این الگوریتم، **D** ماتریس کوتاه ترین مسیر و در واقع هدف می باشد، **W** ماتریس مجاورت و **N** تعداد گره ها می باشد.

```
Procedure Floyd (N, W, D)
1. [Initialize]
   D ← W          در واقع هر آنچه در ماتریس مجاورت وجود دارد در ماتریس کوتاه ترین مسیر کپی می گردد
2. [Finding Shortest Paths]
   Repeat For K = 1 To N
         Repeat For I = 1 To N
               Repeat For J = 1 To N
                     If D [I] [K] + D [K] [J] < D [I] [J]
                     Then D [I] [J] ← D [I] [K] + D [K] [J]
3. [Finished]
   Exit
```

تخمین پیچیدگی الگوریتم:

عمل اصلی: مقایسه

۳ حلقه تو در تو که هر کدام N مرتبه انجام می شوند پس: $T(N) = N \times N \times N = N^3$

مثال: الگوریتم کوتاه ترین مسیر فلوید را بر روی گراف شکل ۱ اعمال نموده و ماتریس کوتاه ترین مسیر را بیابید.

$$W = \begin{bmatrix} 0 & 2 & 7 & \infty \\ \infty & 0 & 6 & 4 \\ \infty & 3 & 0 & 3 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$ ماتریس مجاورت

K میانی	I مبدأ	J مقصد	D[I][J]	D[I][K]	D[K][J]	IS D[I][K] + D[K][J] < D[I][J]	Matrix
1							
	1						
		1	0	0	0	NO	
		2	2	0	2	NO	
		3	7	0	7	NO	
		4	∞	0	∞	NO	
	2						
		1	∞	∞	0	NO	
		2	0	∞	2	NO	
		3	6	∞	7	NO	
		4	4	∞	∞	NO	
	3						
		1	∞	∞	0	NO	
		2	3	∞	2	NO	
		3	0	∞	7	NO	

		4	3	∞	∞	NO	
	4						
		1	∞	∞	0	NO	
		2	∞	∞	2	NO	
		3	∞	∞	7	NO	
		4	0	∞	∞	NO	
2							
	1						
		1	0	2	∞	NO	
		2	2	2	0	NO	
		3	7	2	6	NO	
		4	∞	2	4	YES	D[I][J]=D[1][4]=6 (∞→6)
	2						
		1	∞	0	∞	NO	
		2	0	0	0	NO	
		3	6	0	6	NO	
		4	4	0	4	NO	
	3						
		1	∞	3	∞	NO	
		2	3	3	0	NO	
		3	0	3	6	NO	
		4	3	3	4	NO	
	4						
		1	∞	∞	∞	NO	
		2	∞	∞	0	NO	
		3	∞	∞	6	NO	
		4	0	∞	4	NO	
3							
	1						
		1	0	7	∞	NO	

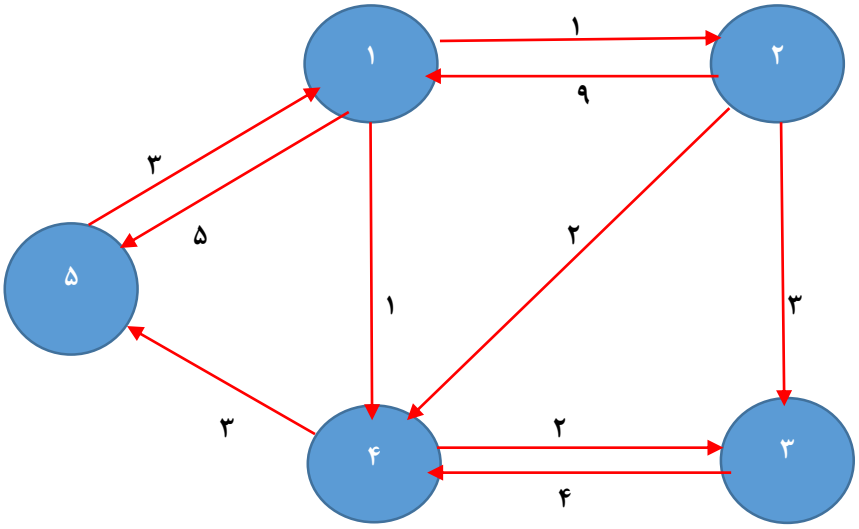
		2	2	7	3	NO	
		3	7	7	0	NO	
		4	6	7	3	NO	قبلاً شده (∞→6) D[1][4]=6
	2						
		1	∞	6	∞	NO	
		2	0	6	3	NO	
		3	6	6	0	NO	
		4	4	6	3	NO	
	3						
		1	∞	0	∞	NO	
		2	3	0	3	NO	
		3	0	0	0	NO	
		4	3	0	3	NO	
	4						
		1	∞	∞	∞	NO	
		2	∞	∞	3	NO	
		3	∞	∞	0	NO	
		4	0	∞	3	NO	
4							
	1						
		1	0	∞	∞	NO	
		2	2	∞	∞	NO	
		3	7	∞	∞	NO	
		4	6	∞	0	NO	قبلاً شده (∞→6) D[1][4]=6
	2						
		1	∞	4	∞	NO	
		2	0	4	∞	NO	
		3	6	4	∞	NO	
		4	4	4	0	NO	
	3						

		1	∞	3	∞	NO	
		2	3	3	∞	NO	
		3	0	3	∞	NO	
		4	3	3	0	NO	
	4						
		1	∞	0	∞	NO	
		2	∞	0	∞	NO	
		3	∞	0	∞	NO	
		4	0	0	0	NO	

بدین ترتیب کلیه رئوس مبدأ، مقصد و میانی، بررسی شد و تنها یک مسیر کوتاه تر میانی یافت گردید:

$$D = \begin{bmatrix} 0 & 2 & 7 & \infty \rightarrow 6 \\ \infty & 0 & 6 & 4 \\ \infty & 3 & 0 & 3 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

تمرین: الگوریتم کوتاه ترین مسیر فلوید را بر روی گراف زیر اعمال نموده و ماتریس کوتاه ترین مسیر را بیابید.



$$W = \begin{bmatrix} 0 & 1 & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$