طراحی الگوریتم ها

روش تقسیم و حل

استاد درس: مهدی جبل عاملی

روش تقسیم و حل یا تقسیم و غلبه و یا Divide & Conquer

روش تقسیم و حل یا تقسیم و غلبه و یا Divide & Conquer

مراحل این روش بصورت زیر است

۱- شکستن مسئله به مسائلی با اندازه ورودی کوچکتر

۲- حل مسائل با اندازه ورودی کوچکتر اگر لازم شد می توان به مرحله اول بازگشت

۳- ترکیب نتایج

جستجوی دودویی

```
index Binary-Search (l, H) {
    if (l > H)   return 0 ;
    M = ⌊(L + H) / 2⌋
    if  x = A[M]   return M;
    if  x < A[M]
         return  Binary-Search (l, M-1);
    else
         return  Binary-Search (M+1, H);  }
```

جستجوی دودویی

```
index Binary-Search (l, H) {
    if (l > H)   return 0 ;
    M = ⌊(L + H) / 2⌋
    if  x = A[M]   return M;          ← 1
    if  x < A[M]                      ← 1
          return  Binary-Search (l, M-1);          w(n/2)
    else                                                     یا
          return  Binary-Search (M+1, H);   }      w(n/2)
```

$$W(n) = 1 + 1 + W(n/2)$$

$$W(1) = 1 + 1$$

حل رابطه بازگشتی

$$w(n) = w\left(\frac{n}{2}\right) + 2$$

if $n = 2^k \Rightarrow w(2^k) = w(2^k/2) + 2 = w(2^{k-1}) + 2$

با فرض $S(k) = w(2^k)$

$$S(k) = S(k-1) + 2$$

$$S(0) = 2$$

حل رابطه بازگشتی(ادامه...)

$$S(k) = S(k-1) + 2$$

$$S(k) = (S(k-1-1) + 2) + 2 = S(k-2) + 2\times 2 =$$

$$= (S(k-3) + 2) + 2\times 2 = S(k-3) + 3\times 2$$

$$= S(k-4) + 4 \times 2$$

$$\vdots$$

$$= S(k-m) + m \times 2$$

حل رابطه بازگشتی(ادامه...)

$$S(k) = S(k-1) + 2$$

$$S(k) = \big(S(k-1-1) + 2\big) + 2 = S(k-2) + 2\times 2$$

$$= \big(S(k-3) + 2\big) + 2\times 2 = S(k-3) + 3\times 2$$

$$= S(k-4) + 4\times 2$$

$$\vdots$$

$$= S(k-m) + m\times 2$$

$$= S(k-k) + k\times 2$$

تا جایی ادامه می دهیم که $k=m$

$S(0) = 2$

$$= 2 + k\times 2$$

حل رابطه بازگشتی(ادامه...)

$$S(k) = S(k-1) + 2$$

$$S(k) = \big(S(k-1-1) + 2\big) + 2 = S(k-2) + 2 \times 2$$

$$= \big(S(k-3) + 2\big) + 2 \times 2 = S(k-3) + 3 \times 2$$

$$= S(k-4) + 4 + 2$$

$$\vdots$$

$$= S(k-m) + m \times 2$$

تا جایی ادامه می دهیم که $k=m$

$$= S(k-k) + k \times 2$$

$S(0) = 2$

پس خواهیم داشت

$$= 2 + k \times 2 \Rightarrow W(2^k) = 2 + k \times 2 \Rightarrow W(n) = 2 + 2 \times \log_2 n$$

$$\in \Theta(\log n)$$

مرتب سازی ادغامی

15 7 9 3 14 21 34 18

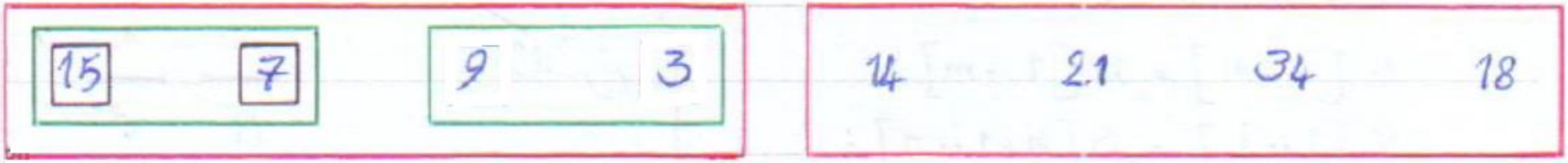
مرتب سازی ادغامی

15	7	9	3

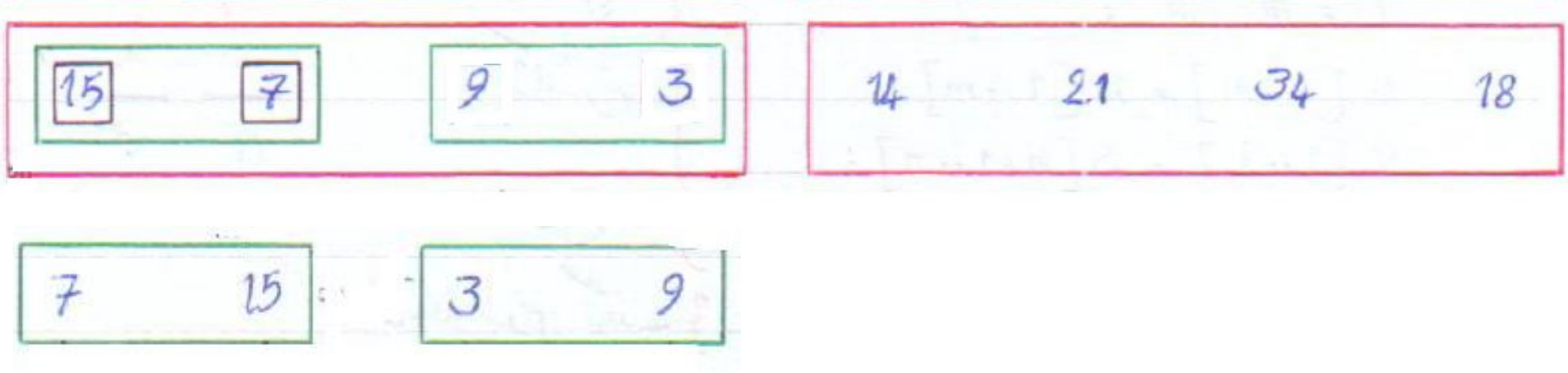
14	21	34	18

مرتب سازی ادغامی

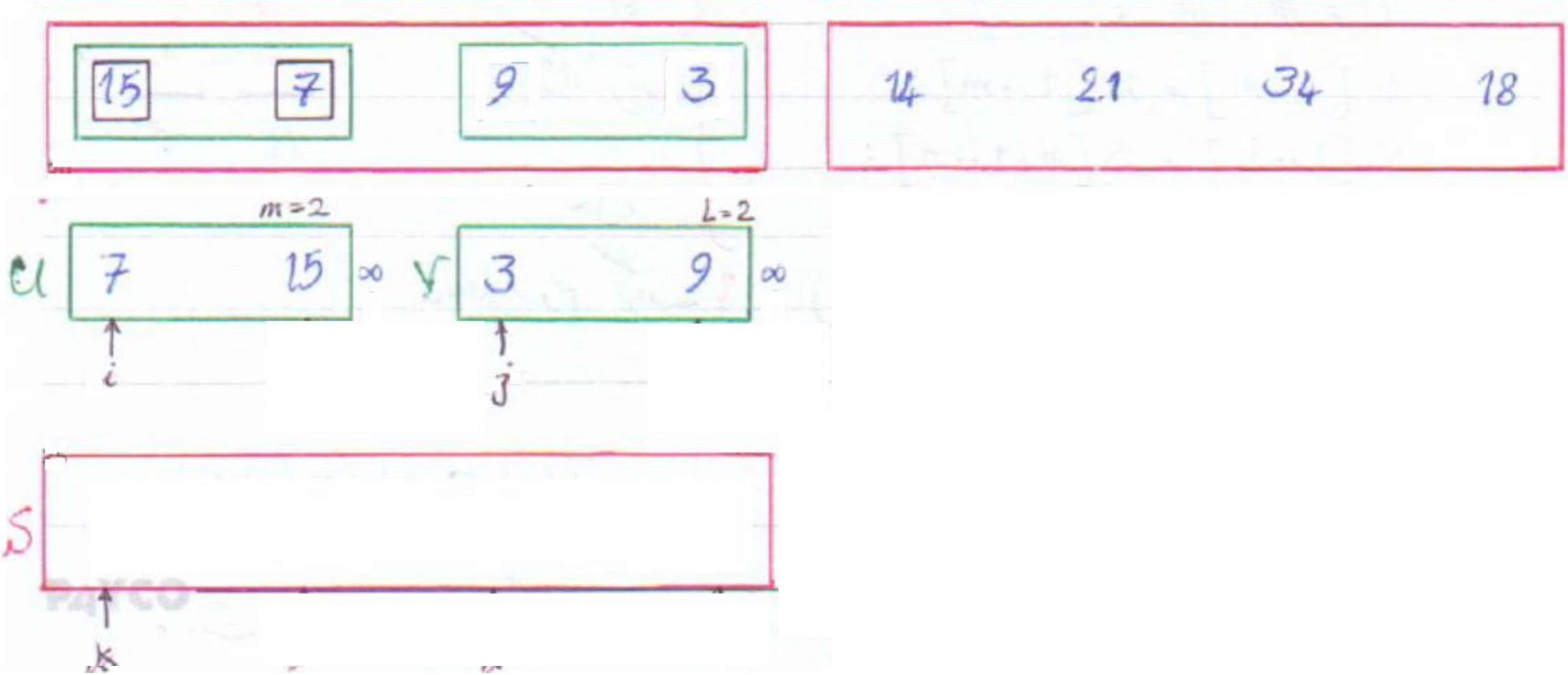
مرتب سازی ادغامی



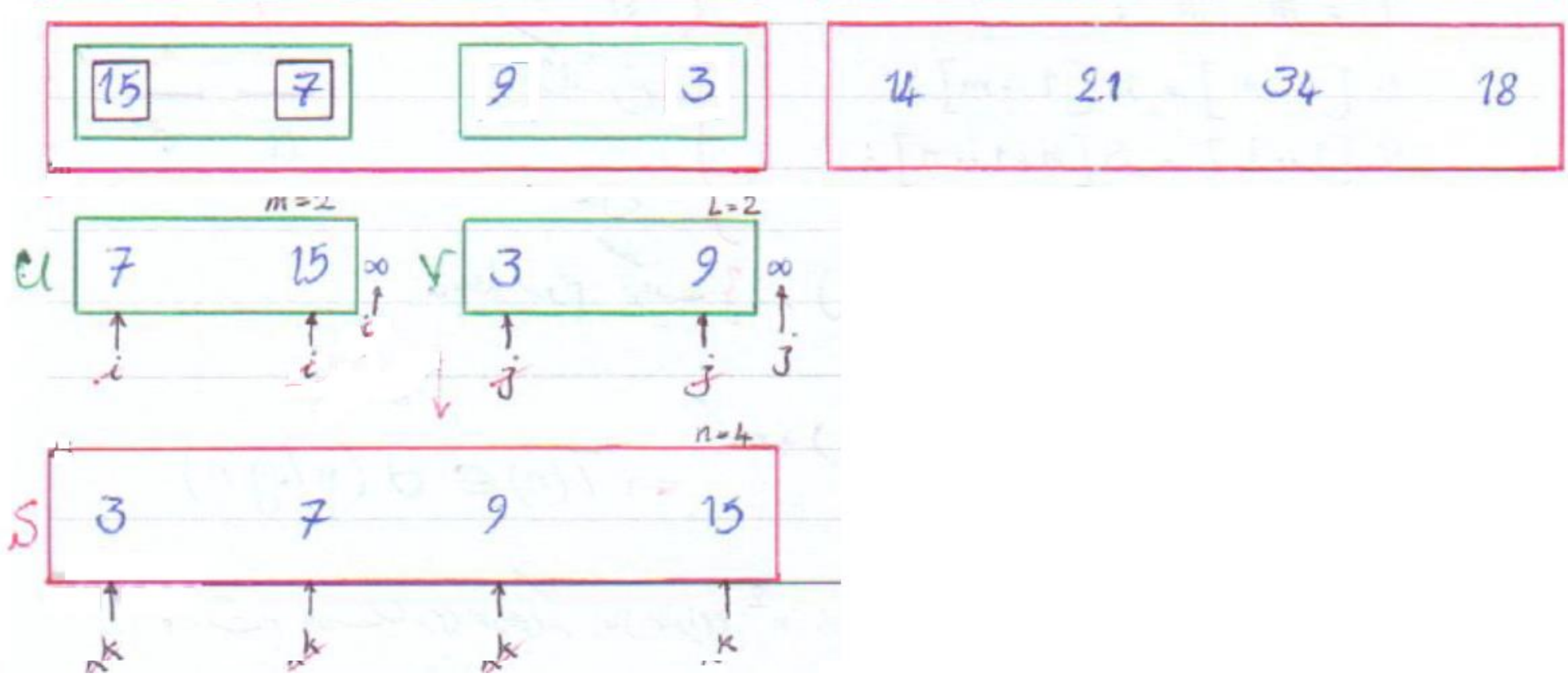
مرتب سازی ادغامی



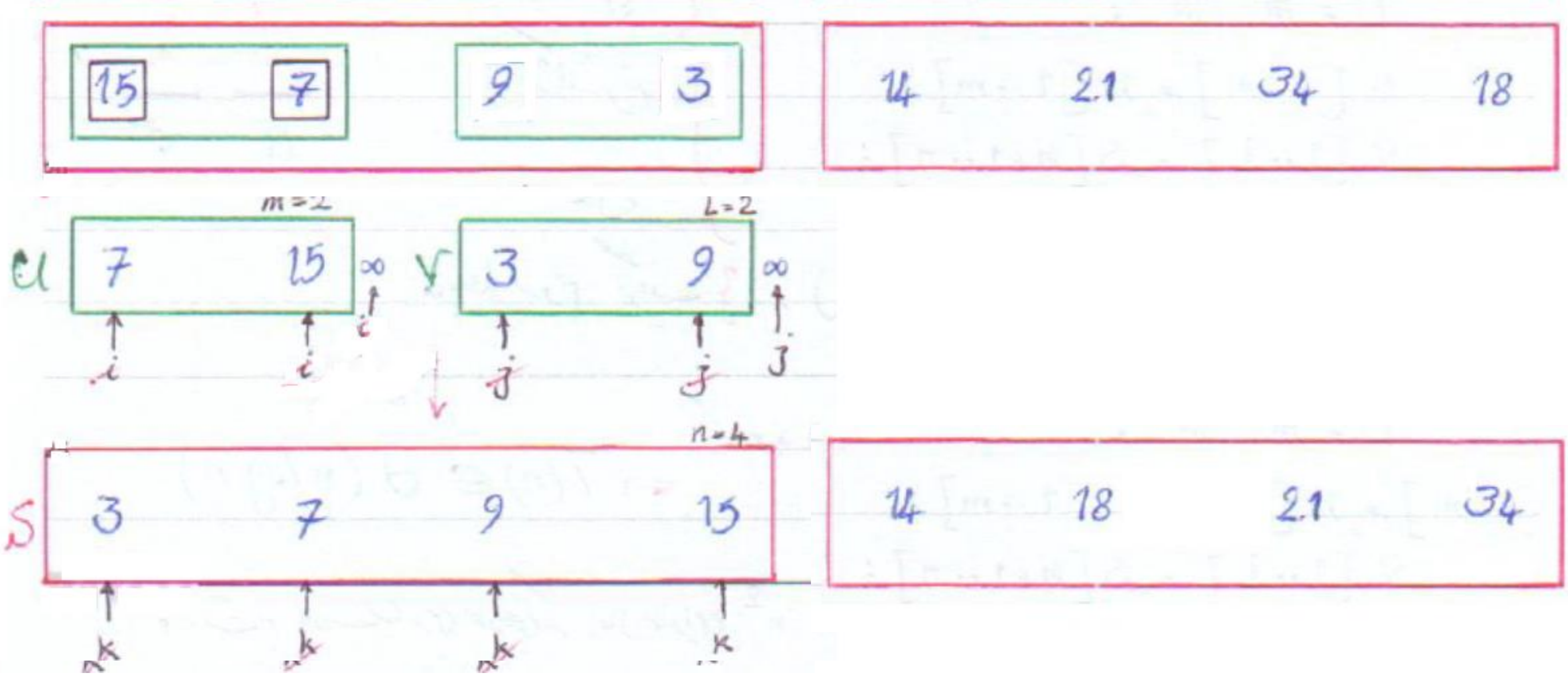
مرتب سازی ادغامی



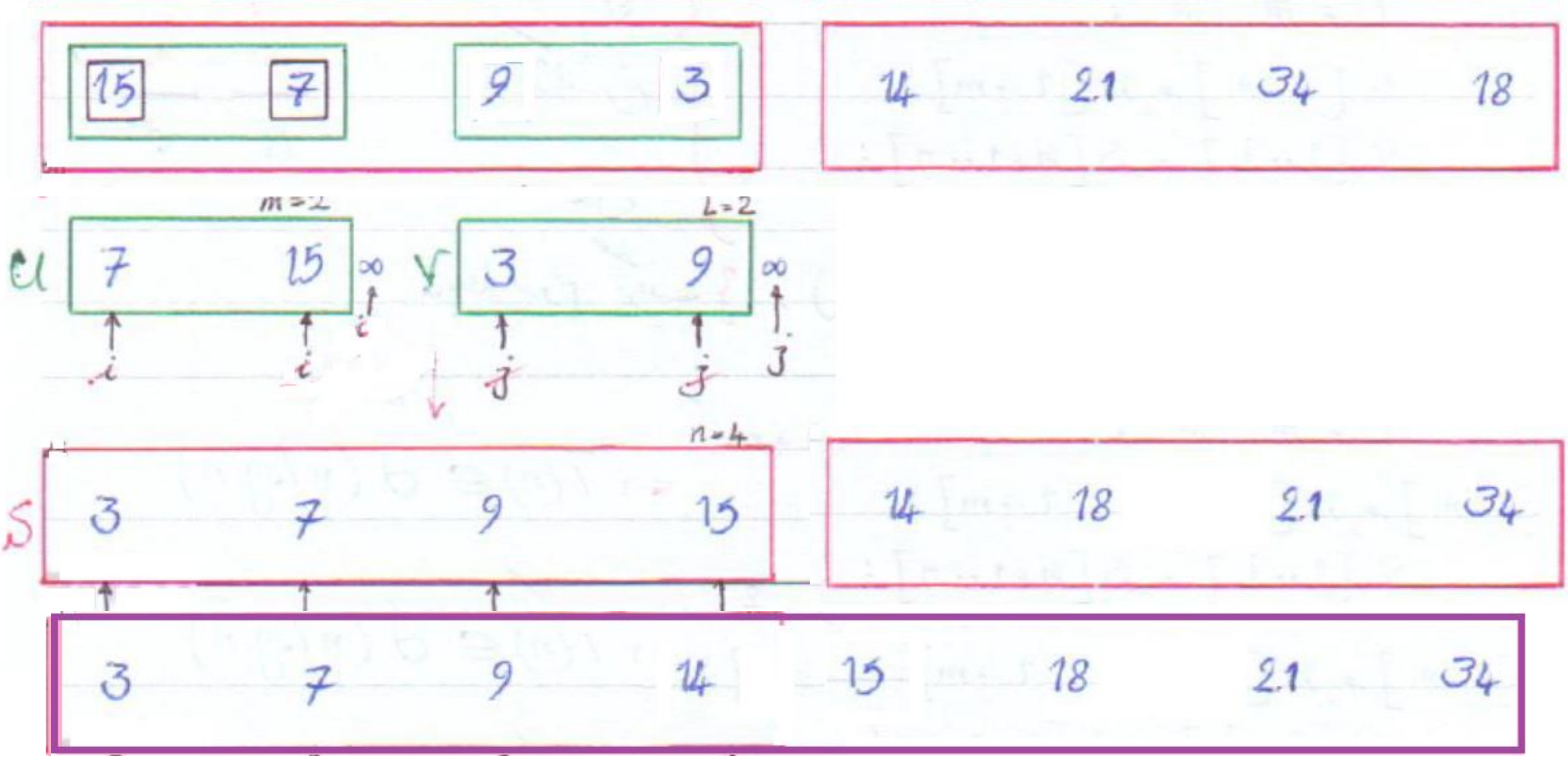
مرتب سازی ادغامی



مرتب سازی ادغامی



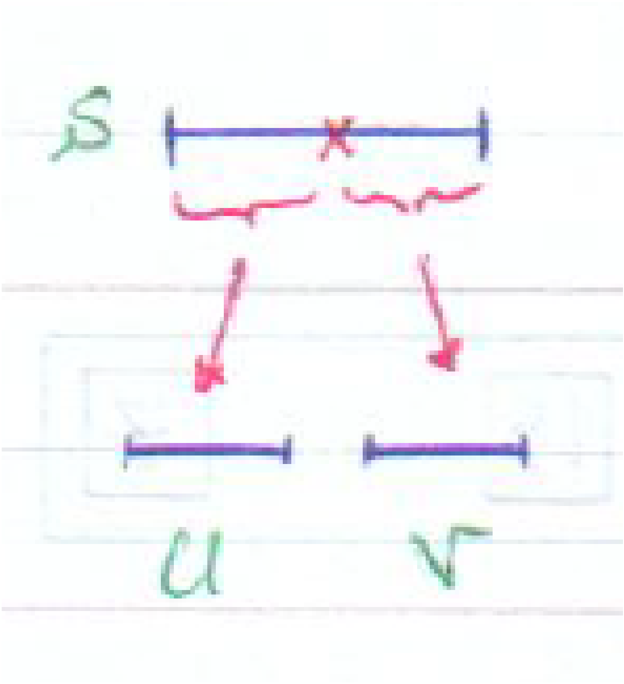
مرتب سازی ادغامی



تابع ترکیب

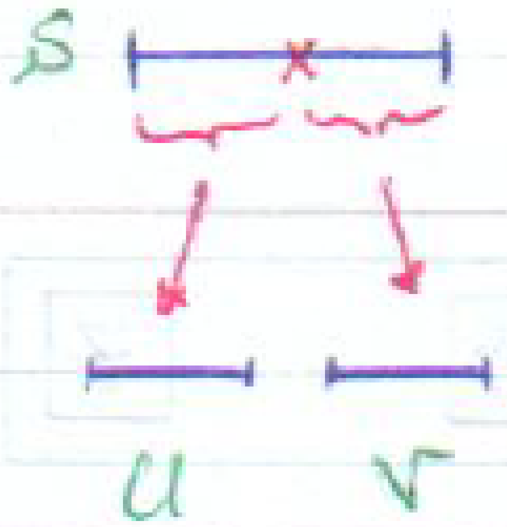
```
Merge (n, m, L, S[1..n], U[1..m], V[1..L]){
    U[m+1] = V[L+1] = ∞ ;
    i = j = 1 ;
    for  k = 1  to  n  do
        if  U[i] < V[j]  {
            S[k] = U[i];
            i++ ; }
        else  {
            S[k] = V[j];
            j++ ;  }}
```

مرتب سازی ادغامی



مرتب سازی ادغامی

```
Merge-Sort (n, S[1..n]) {
    if (n ≤ 1) return;
    m = ⌈n/2⌉ ;
    L = n - m ;
    U[1..m] = S[1..m];
    V[1..L] = S[m+1..n];
    Merge-Sort (m, U);
    Merge-Sort (L, V);
    Merge (n, m, L, S, U, V); }
```



تحلیل زمان اجرای مرتب سازی ادغامی

- زمان اجرای تابع ترکیب = n

تحلیل زمان اجرای مرتب سازی ادغامی

- زمان اجرای تابع ترکیب = **n**

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

$$T(1) = 0$$

زمان اجرای مرتب سازی ادغامی

تحلیل زمان اجرای مرتب سازی ادغامی

- زمان اجرای تابع ترکیب = **n**

زمان اجرای مرتب سازی ادغامی

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

$$T(1) = 0$$

تحلیل زمان اجرای مرتب سازی ادغامی

- زمان اجرای تابع ترکیب = n

زمان اجرای مرتب سازی ادغامی

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

$$T(1) = 0$$

$$\Rightarrow T(n) \in \theta(n \log n)$$

تحلیل زمان اجرای مرتب سازی ادغامی

- زمان اجرای تابع ترکیب = n

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

$$T(1) = 0$$

زمان اجرای مرتب سازی ادغامی

$$\Rightarrow T(n) \in \theta(n \log n)$$

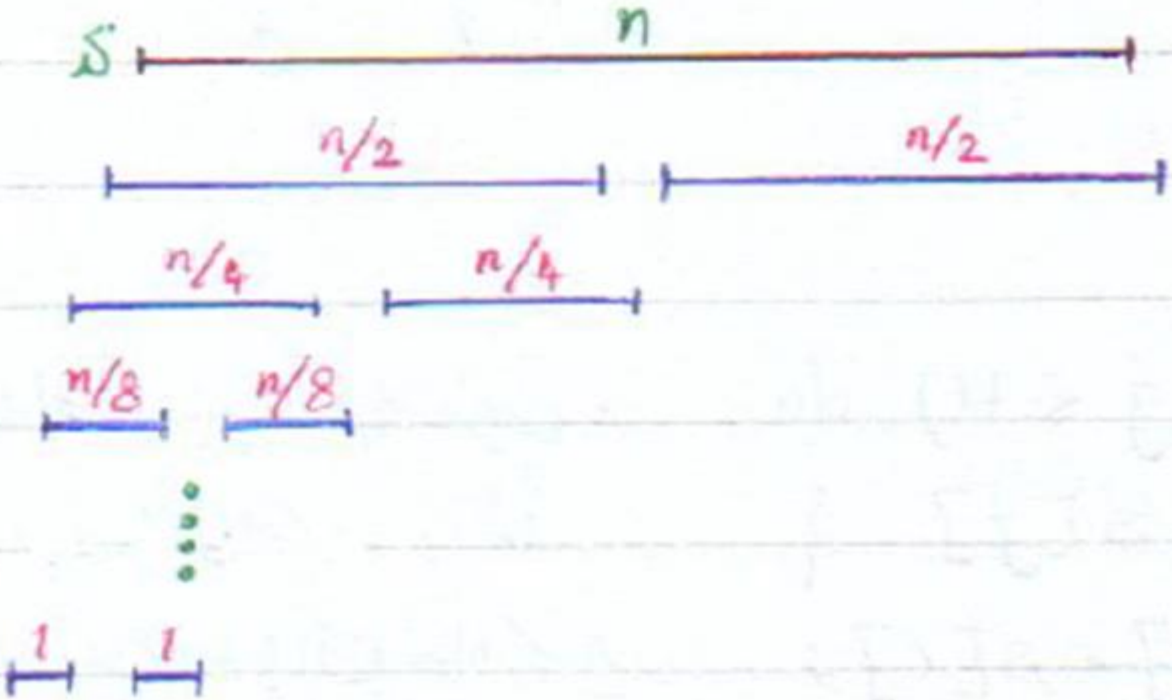
و نسبت به مرتب سازی تعویضی که زمان اجرای n^2 داشت بهتر است

Exchange sort

نکته) ولی مصرف حافظه در الگوریتم فوق چگونه است؟

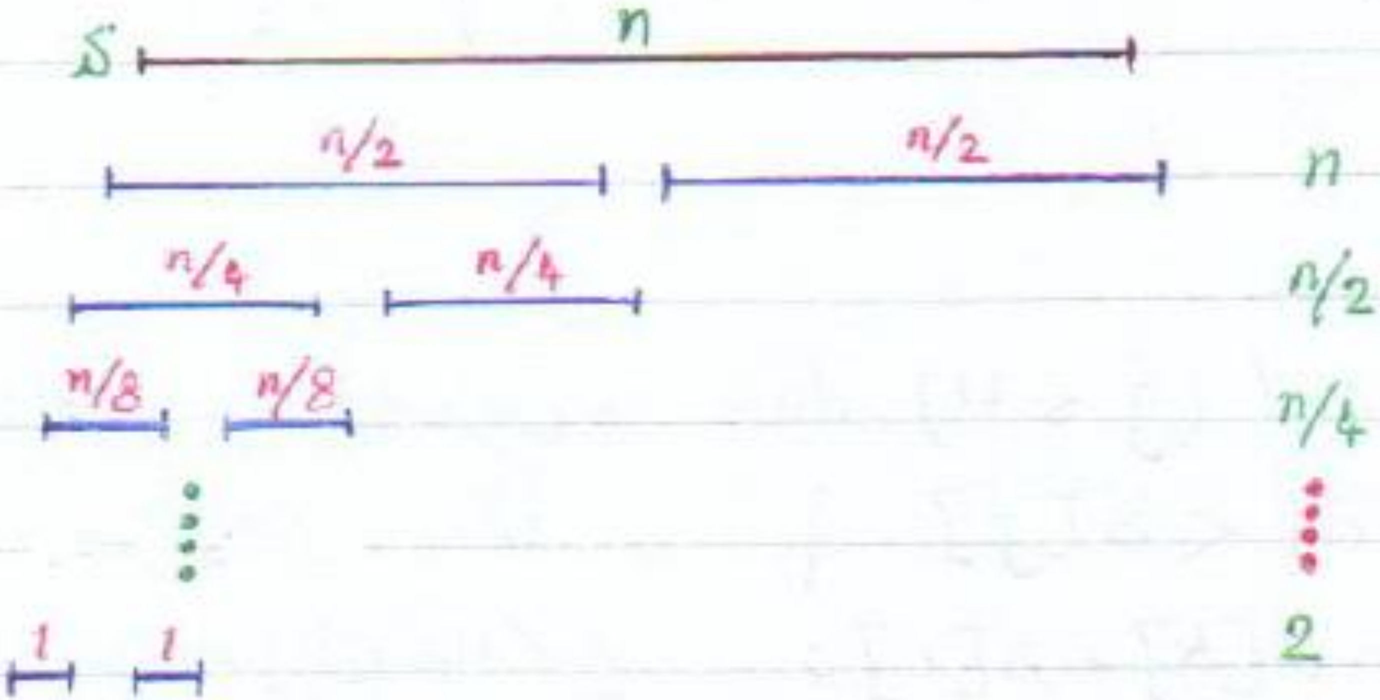
سؤال) وضعیت مصرف حافظه در الگوریتم فوق چگونه است؟

سؤال)



سؤال) ولی مصرف حافظه در الگوریتم فوق چگونه است؟

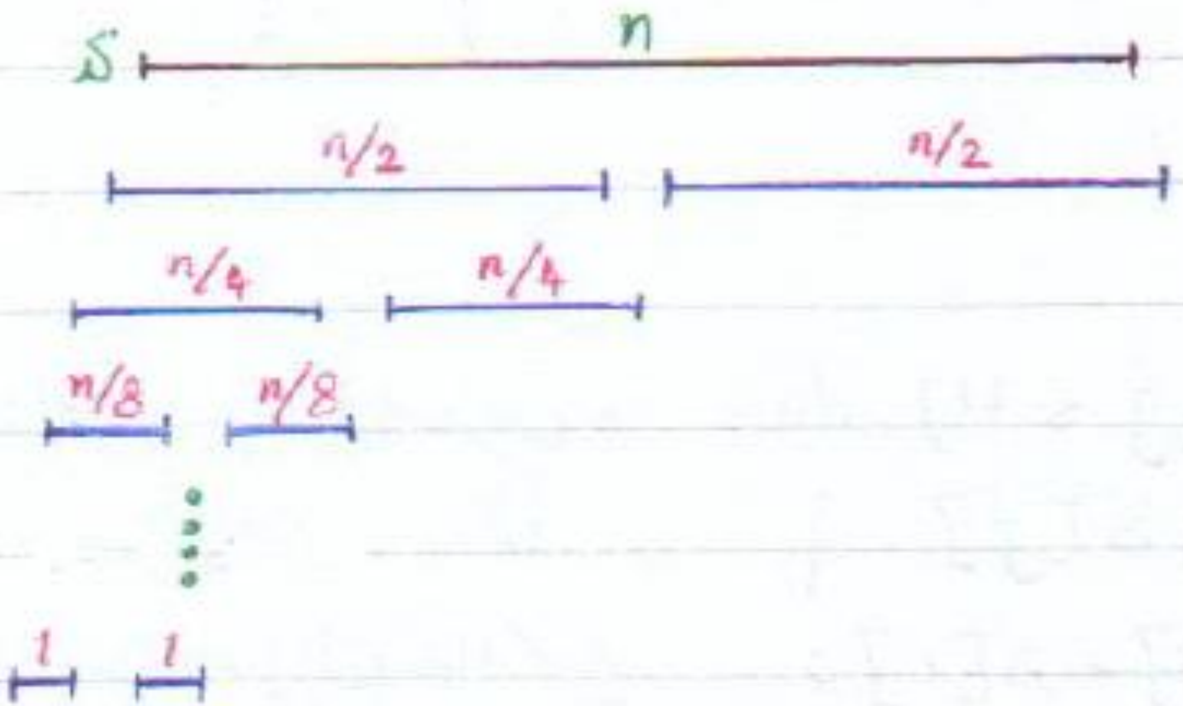
جواب)



$$\sum \simeq 2n$$

نکته) ولی مصرف حافظه در الگوریتم فوق چگونه است؟

نکته) کل



n

$n/2$

$n/4$

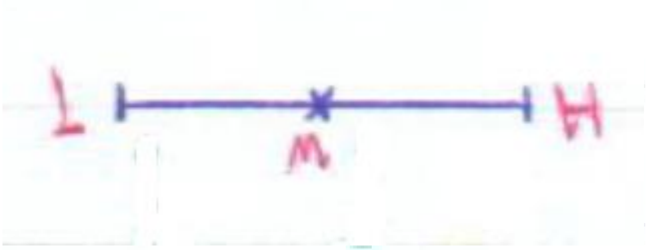
$\vdots$

2

$$\sum \simeq 2n$$

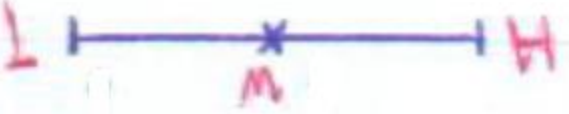
⇐ پس در الگوریتم فوق، حداکثر $2n$ حافظه اضافه نیاز خواهیم داشت تا الگوریتم به پایان برسد

کاهش مصرف حافظه مرتب سازی ادغامی

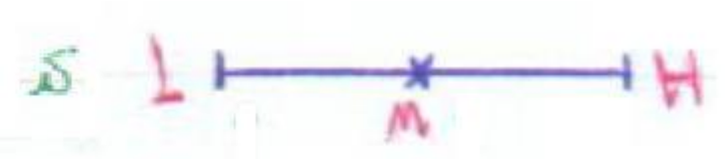


کاهش مصرف حافظه مرتب سازی ادغامی

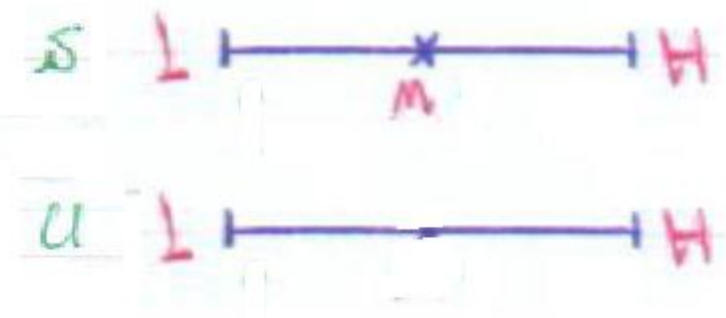
```
Merge-Sort2 (L, H) {
    if (L ≥ H)  return;
    M = ⌊(L+H) / 2⌋ ;
    Merge-Sort2 (L, M);
    Merge-Sort2 (M+1, H);
    Merge2 (L, M, H);
}
```



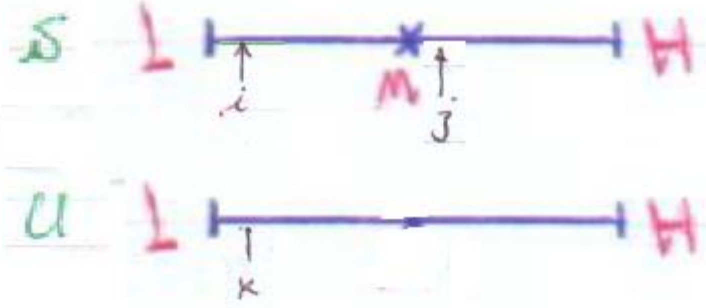
تابع Merge2



تابع Merge2

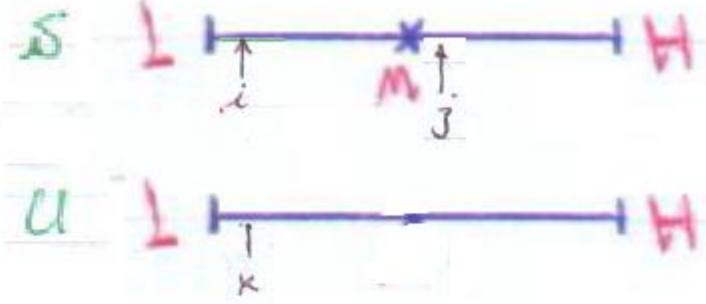


تابع Merge2



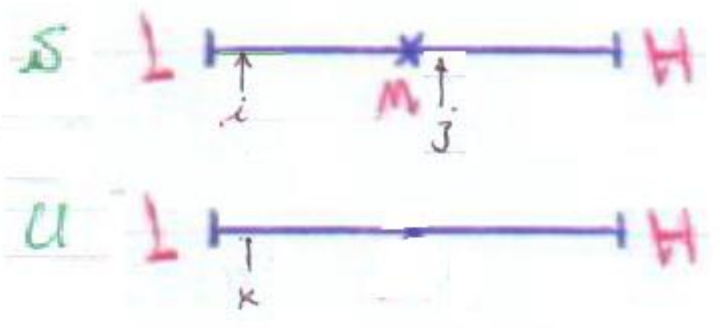
تابع Merge2

```
Merge2 (L, M, H) {
    i = L ;
    j = M + 1 ;
    k = L ;
    while (i ≤ M) and (j ≤ H) do
        if S[i] < S[j] {
            U[k] = S[i] ;
            i++ ;
            k++ ; }
        else {
            U[k] = S[j] ;
            j++ ;
            k++ ; }
```



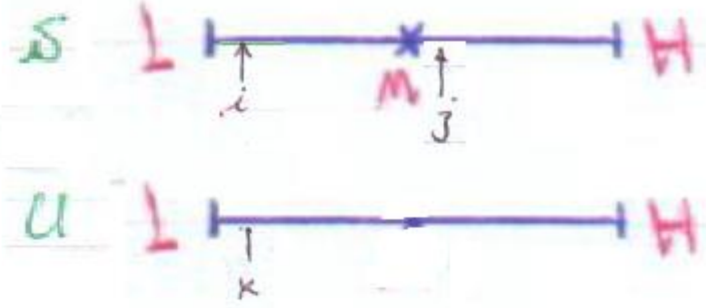
$if\ (i \leq m)$

ادامه تابع Merge2



ادامه تابع Merge2

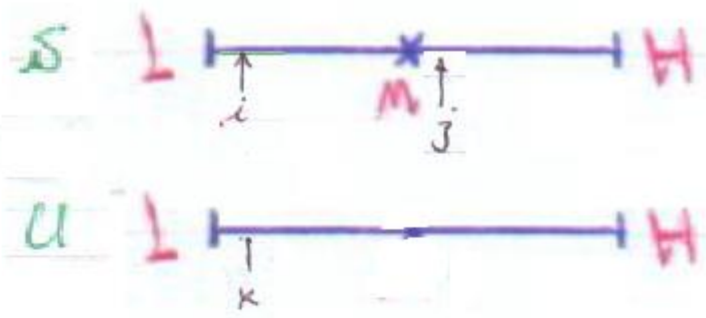
```
if (i ≤ M)
        U[k..H] = S[i..M];
else
        U[k..H] = S[j..H];
```



ادامه تابع Merge2

```
if (i <= M)
        U[K..H] = S[i..M];
else
        U[K..H] = S[j..H];

S[L..H] = U[L..H]
}
```



تحلیل زمان اجرای مرتب سازی ادغامی ۲

تحلیل زمان اجرای مرتب سازی ادغامی ۲

- زمان اجرای تابع Merge2 در بدترین حالت برابر n-1 است

تحلیل زمان اجرای مرتب سازی ادغامی ۲

- زمان اجرای تابع Merge2 در بدترین حالت برابر n-1 است

$$\begin{cases} W(n) = 2W(n/2) + n - 1 \\ W(1) = 0 \end{cases} \longrightarrow W(n) \in \Theta(n \log n)$$

تحلیل زمان اجرای مرتب سازی ادغامی ۲

- زمان اجرای تابع Merge2 در بدترین حالت برابر n-1 است

$$\begin{cases} W(n) = 2W(n/2) + n - 1 \\ W(1) = 0 \end{cases} \longrightarrow W(n) \in \Theta(n \log n)$$

- مصرف حافظه تابع جدید؟

تحلیل زمان اجرای مرتب سازی ادغامی ۲

- زمان اجرای تابع Merge2 در بدترین حالت برابر n-1 است

$$\begin{cases} W(n) = 2W(n/2) + n - 1 \\ W(1) = 0 \end{cases} \longrightarrow W(n) \in \Theta(n \log n)$$

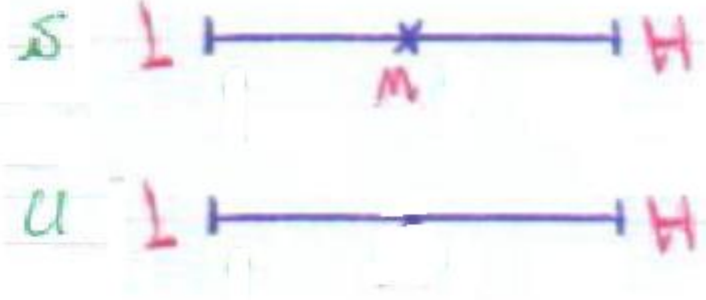
- مصرف حافظه تابع جدید؟

n = میزان حافظه مورد نیاز برای آرایه U در تابع Merge2

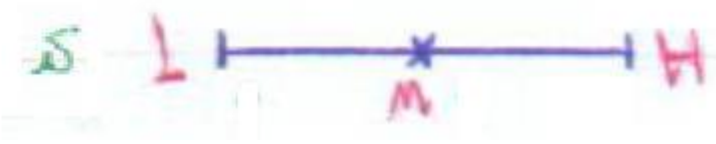
تمرین

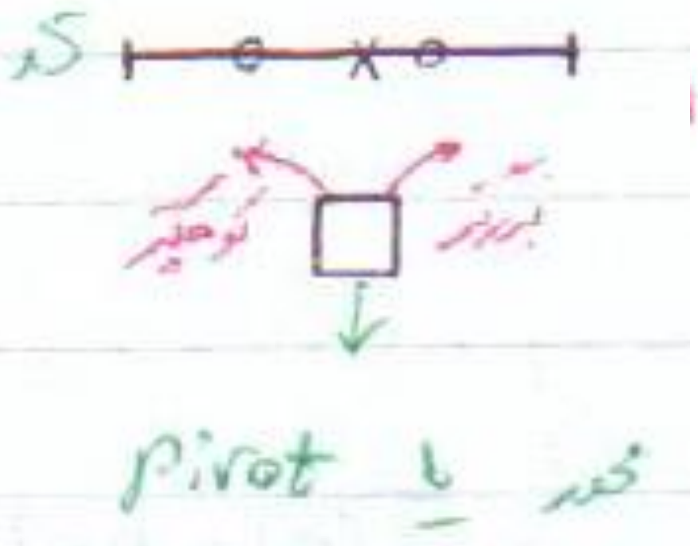
- زمان اجرای تابع MergeSort2 را در بهترین حالت محاسبه کنید.

کاهش بیشتر مصرف حافظه در مرتب سازی



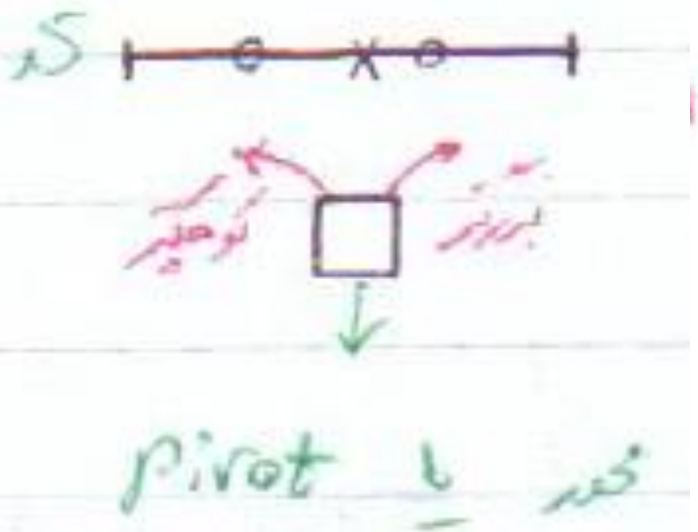
کاهش بیشتر مصرف حافظه در مرتب سازی



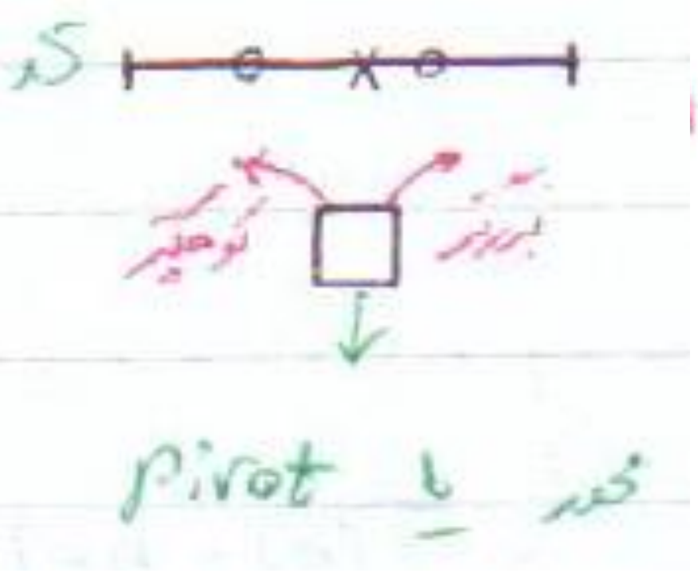


کوچکتر
بزرگتر
محور یا pivot

13 25 16 3 7 15 4

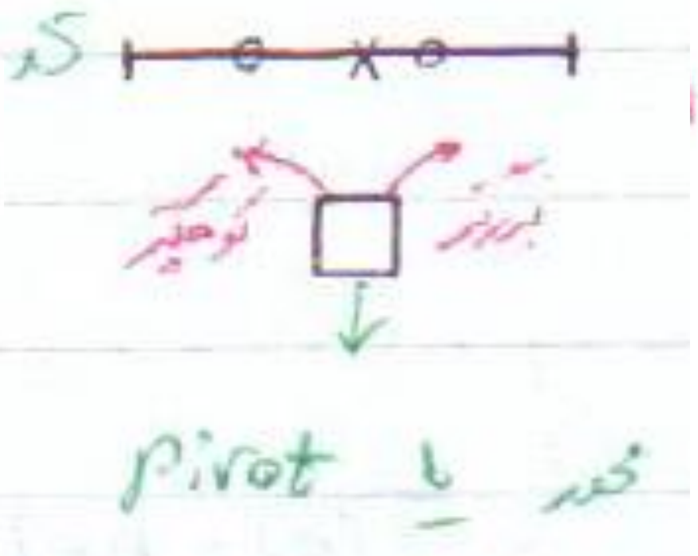


$\boxed{13}$ 25 16 3 7 15 4

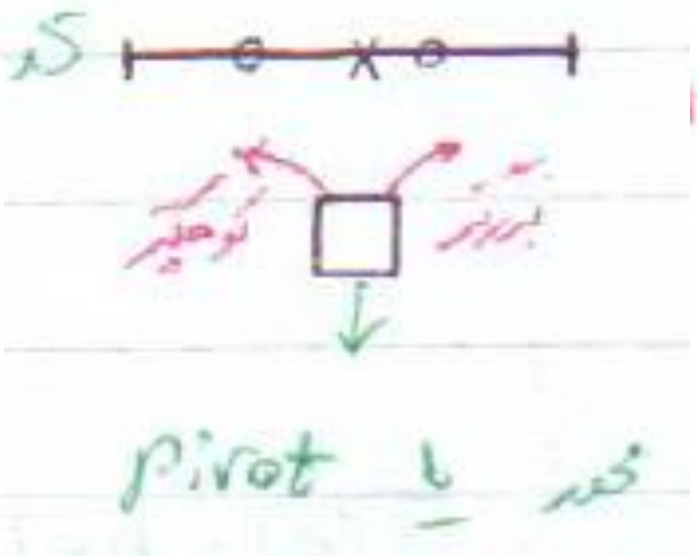


$\boxed{13}$	25	16	3	7	15	4

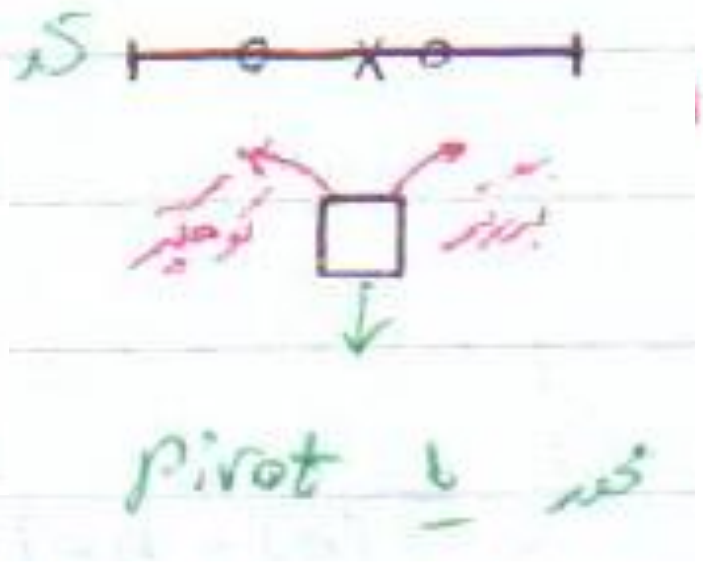
$\boxed{13}$	3	16	25	7	15	4



$\boxed{13}$	25	16	3	7	15	4
$\boxed{13}$	3	16	25	7	15	4
$\boxed{13}$	3	7	25	16	15	4

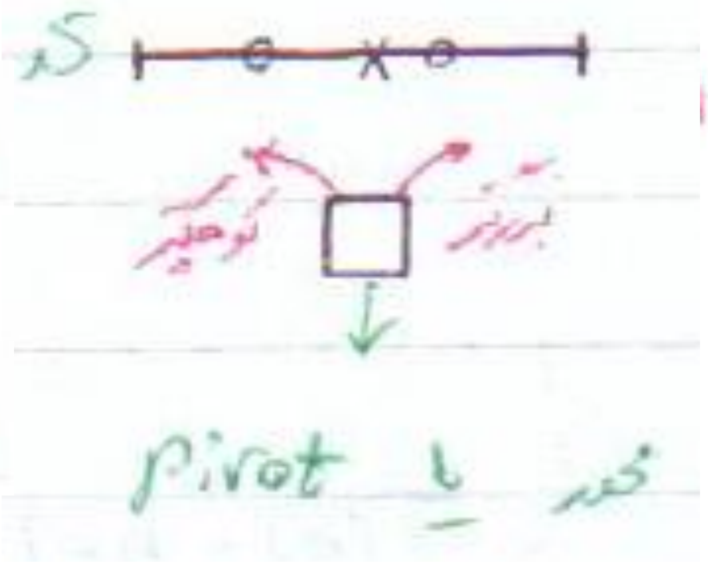


$\boxed{13}$	25	16	3	7	15	4
$\boxed{13}$	3	16	25	7	15	4
$\boxed{13}$	3	7	25	16	15	4



از کجا بدانیم که جای ۷ باید با ۱۶ عوض شود؟

13	25	16	3	7	15	4
13	3	16	25	7	15	4
13	3	7	25	16	15	4

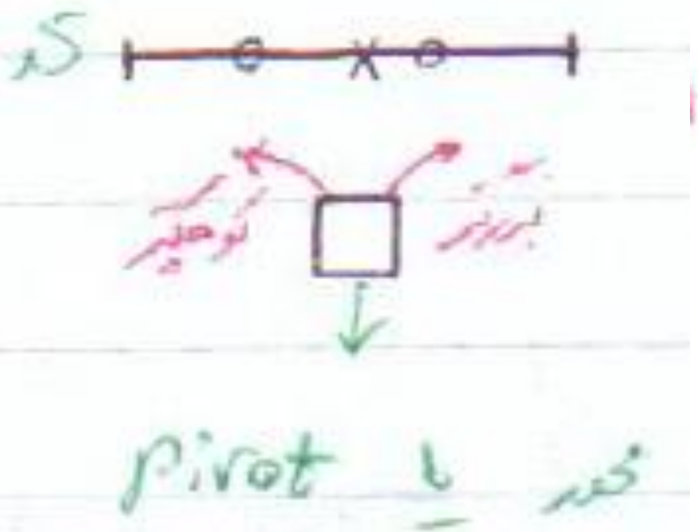


از کجا بدانیم که جای ۷ باید با ۱۶ عوض شود؟

جواب: اندیس j را تعریف می کنیم که آخرین عنصر کوچکتر از محور را مشخص نماید.

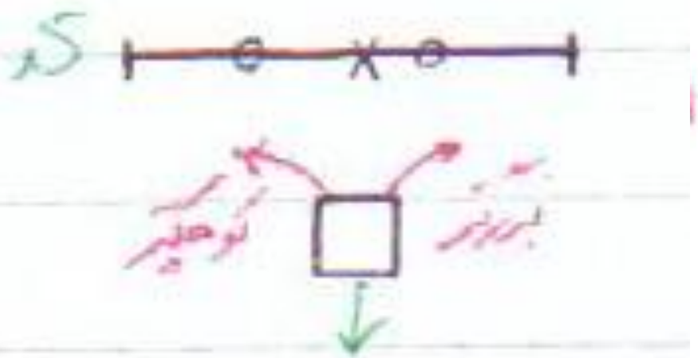
$\boxed{13}$	25	16	3	7	15	4

j



$\boxed{13}$ 25 16 3 7 15 4

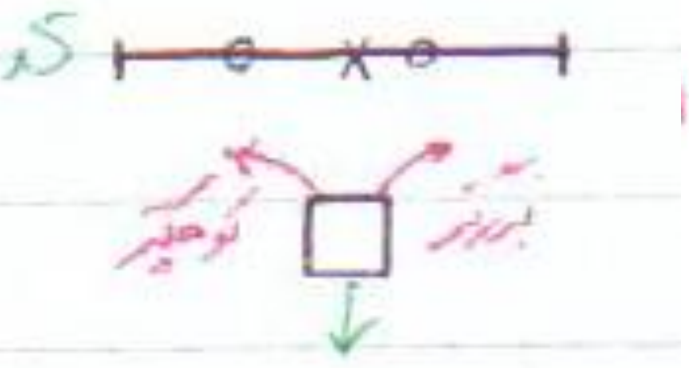
~~j~~ j



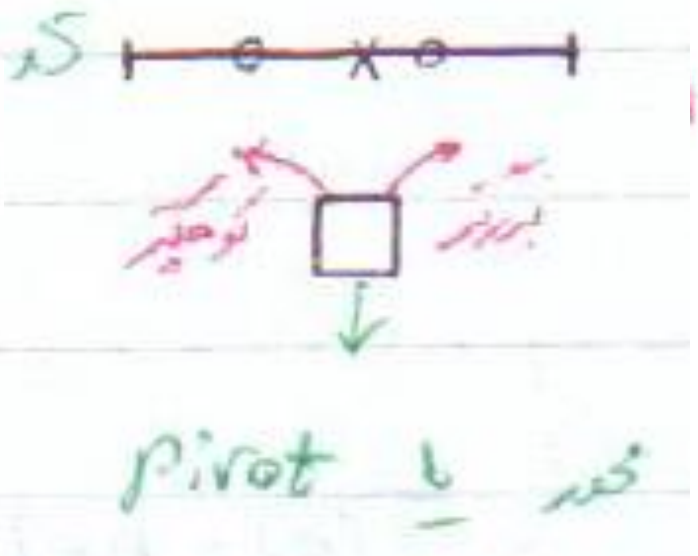
محور یا pivot

$\boxed{13}$	25	16	3	7	15	4

$\boxed{13}$	3	16	25	7	15	4
	j					



$\boxed{13}$	25	16	3	7	15	4
$\boxed{13}$	3	16	25	7	15	4
	~~j~~	j				

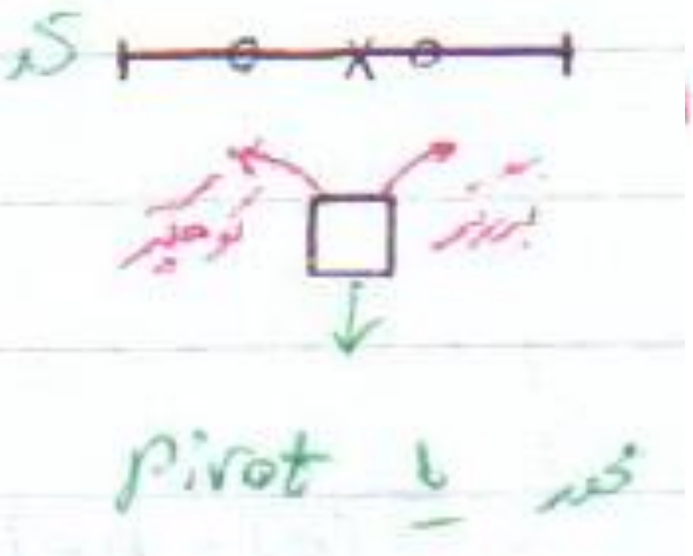


$\boxed{13}$ 25 16 3 7 15 4

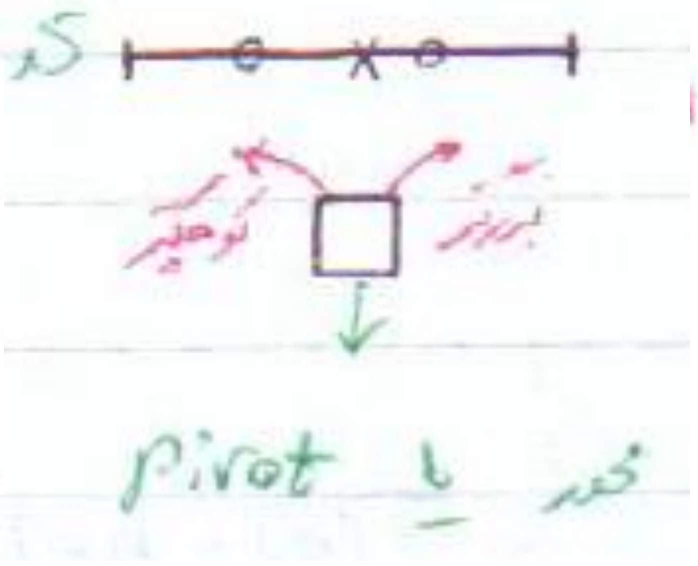
$\boxed{13}$ 3 16 25 7 15 4

$\boxed{13}$ 3 7 25 16 15 4

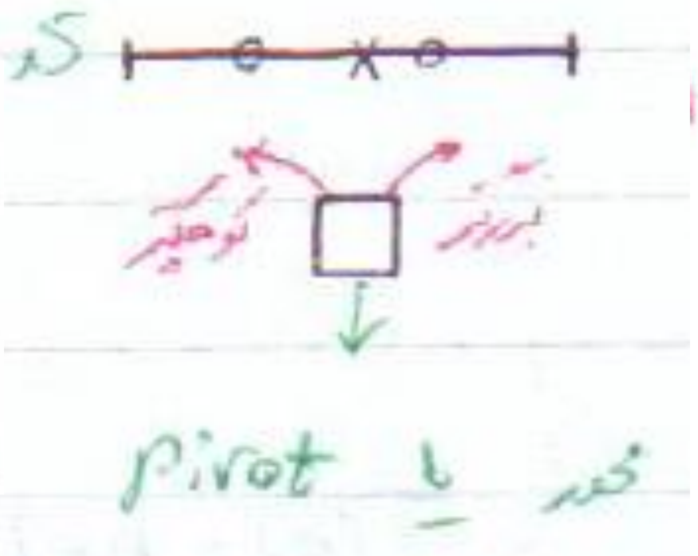
J



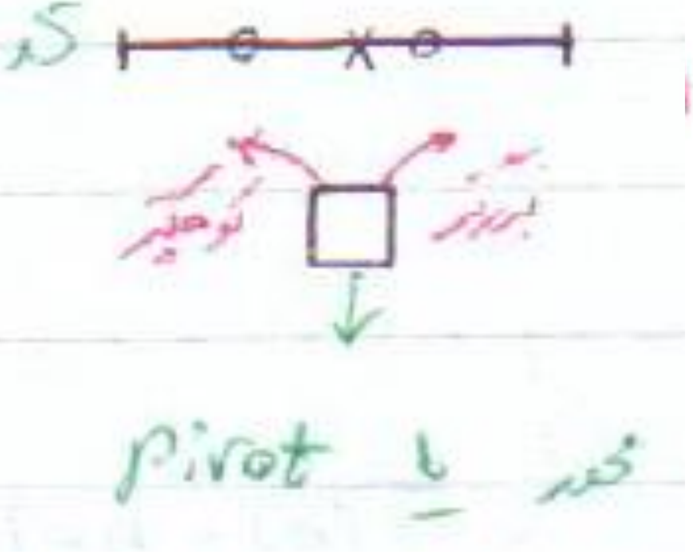
$\boxed{13}$	25	16	3	7	15	4
$\boxed{13}$	3	16	25	7	15	4
$\boxed{13}$	3	7	25	16	15	4



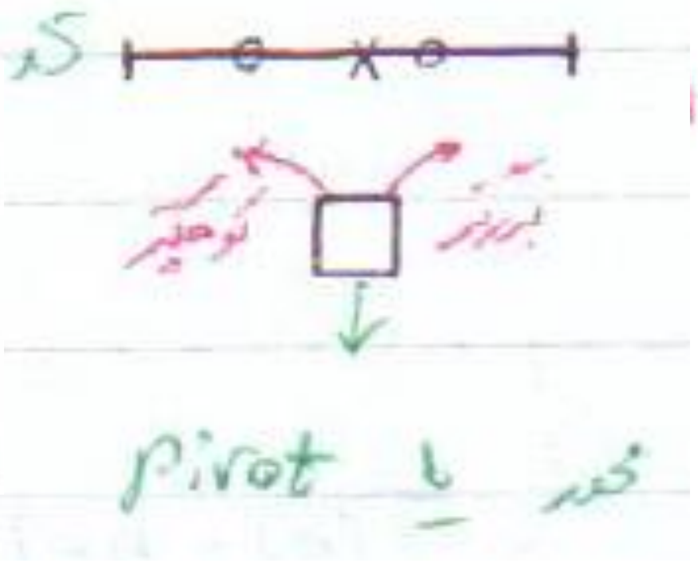
13	25	16	3	7	15	4
13	3	16	25	7	15	4
13	3	7	25	16	15	4
13	3	7	4	16	15	25
			j			



$\boxed{13}$	25	16	3	7	15	4
$\boxed{13}$	3	16	25	7	15	4
$\boxed{13}$	3	7	25	16	15	4
$\boxed{13}$	3	7	4	16	15	25
4	3	7	13	16	15	25
			j			



13	25	16	3	7	15	4
13	3	16	25	7	15	4
13	3	7	25	16	15	4
13	3	7	4	16	15	25
4	3	7	13	16	15	25



این عمل افراز
(Partition)
نام دارد

تابع Partition


```
Index partition (L , H) {

    pivot = S[L] ; J = L ;
    for i = L+1 to H do
        if S[i] < pivot {
            J++ ;
            swap ( S[J] , S[i] ); }
    swap ( S[L] , S[J] );
    return J ;
}
```

مرتب سازی سریع

```
QuickSort (L, H) {

        J = partition (L, H) ;
```

مرتب سازی سریع

```
QuickSort (L , H) {

        J = partition (L , H) ;
        QuickSort (L , J-1) ;
        QuickSort (J+1 , H) ;
```

مرتب سازی سریع

```
QuickSort (L, H) {
    if  L≥H   return;
    J = partition (L, H);
    Quick Sort (L, J-1);
    Quick Sort (J+1, H);
}
```

تحلیل زمان اجرای مرتب سازی سریع

- زمان اجرای تابع Partition برابر n-1 است

تحلیل زمان اجرای مرتب سازی سریع

$$T(n) = n - 1 + ?$$

- زمان اجرای تابع Partition برابر n-1 است

تحلیل زمان اجرای مرتب سازی سریع

- زمان اجرای تابع Partition برابر n-1 است

$$T(n) = n - 1 + ?$$

$$\begin{cases} W(n) = n - 1 + W(n-1) + W(0) \\ W(0) = 0 \end{cases}$$

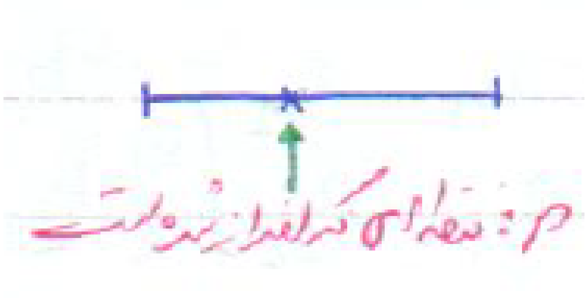
تحلیل زمان اجرای مرتب سازی سریع

- زمان اجرای تابع Partition برابر n-1 است

$$T(n) = n-1 + ?$$

$$\begin{cases} W(n) = n-1 + W(n-1) + W(0) \\ W(0) = 0 \end{cases} \Rightarrow W(n) \in O(n^2)$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط



تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) +$$


```
QuickSort (L , H) {
        if   L >= H   return;
        J = partition (L, H) ;
        Quick Sort (L, J-1);
        Quick Sort (J+1, H);
```

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) + \qquad A(p-1)$$

p: نقطه ای که افراز شده است

```
QuickSort (L , H) {
        if   L >= H   return ;
        J = partition ( L , H ) ;
        Quick Sort ( L , J-1 ) ;
        Quick Sort ( J+1 , H ) ;
```

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) + \quad A(p-1) + A(n-p)$$

p: نقطه ای که افراز شده است

```
QuickSort (L , H) {
        if   L >= H   return;
        J = partition (L, H) ;
        Quick Sort (L, J-1) ;
        Quick Sort (J+1, H) ;
```

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) + \left[A(p-1) + A(n-p)\right] \times \frac{1}{n}$$

p: نقطه‌ای که افراز شده است

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) + \sum_{p=1}^{n} \left[A(p-1) + A(n-p) \right] \times \frac{1}{n}$$

p: نقطه‌ای که افراز شده است

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) + \sum_{p=1}^{n} \left[A(p-1) + A(n-p) \right] \times \frac{1}{n}$$

p: نقطه ای که افراز شده است

$$A(1) = 0$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) + \sum_{p=1}^{n} \left[A(p-1) + A(n-p) \right] \times \frac{1}{n}$$

$$A(n) = n-1 + \frac{1}{n} \left[\sum_{p=1}^{n} A(p-1) + \sum_{p=1}^{n} A(n-p) \right]$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) + \sum_{p=1}^{n} \left[A(p-1) + A(n-p) \right] \times \frac{1}{n}$$

$$A(n) = n-1 + \frac{1}{n} \left[\sum_{p=1}^{n} A(p-1) + \sum_{p=1}^{n} A(n-p) \right]$$

$$A(n) = n-1 + \frac{2}{n} \sum_{p=1}^{n} A(p-1)$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$A(n) = (n-1) + \sum_{p=1}^{n} \left[A(p-1) + A(n-p) \right] \times \frac{1}{n}$$

$$A(n) = n-1 + \frac{1}{n} \left[\sum_{p=1}^{n} A(p-1) + \sum_{p=1}^{n} A(n-p) \right]$$

$$A(n) = n-1 + \frac{2}{n} \sum_{p=1}^{n} A(p-1)$$

$$n\,A(n) = n(n-1) + 2 \sum_{p=1}^{n} A(p-1) \qquad (1)$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$n\,A(n) = n\,(n-1) + 2\sum_{p=1}^{n} A(p-1) \qquad (1)$$

$$(n-1)\,A(n-1) = (n-1)(n-2) + 2\sum_{p=1}^{n-1} A(p-1) \qquad (2)$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$n A(n) = n(n-1) + 2\sum_{p=1}^{n} A(p-1) \qquad (1)$$

$$(n-1) A(n-1) = (n-1)(n-2) + 2\sum_{p=1}^{n-1} A(p-1) \qquad (2)$$

دو رابطه 1 و 2 را از همدیگر کم می‌کنیم

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$nA(n) = n(n-1) + 2\sum_{p=1}^{n} A(p-1) \qquad (1)$$

$$(n-1)A(n-1) = (n-1)(n-2) + 2\sum_{p=1}^{n-1} A(p-1) \qquad (2)$$

دو رابطه 1 و 2 را از همدیگر کم می‌کنیم.

$$nA(n) - (n-1)A(n-1) = 2(n-1) + 2A(n-1)$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$n\,A(n) = n\,(n-1) + 2\sum_{p=1}^{n} A(p-1) \qquad (1)$$

$$(n-1)\,A(n-1) = (n-1)(n-2) + 2\sum_{p=1}^{n-1} A(p-1) \qquad (2)$$

دو رابطه 1 و 2 را از همدیگر کم می‌کنیم.

$$n\,A(n) - (n-1)\,A(n-1) = 2\,(n-1) + 2\,A(n-1)$$

$$n\,A(n) = (n+1)\,A(n-1) + 2\,(n-1)$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$n A(n) = n(n-1) + 2 \sum_{p=1}^{n} A(p-1) \qquad (1)$$

$$(n-1) A(n-1) = (n-1)(n-2) + 2 \sum_{p=1}^{n-1} A(p-1) \qquad (2)$$

دو رابطهٔ 1 و 2 را از همدیگر کم می‌کنیم.

$$n A(n) - (n-1) A(n-1) = 2(n-1) + 2A(n-1)$$

$$n A(n) = (n+1) A(n-1) + 2(n-1)$$

$$\div\, n(n+1) \implies \frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

20

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$n\,A(n) = n\,(n-1) + 2\sum_{p=1}^{n} A(p-1) \qquad (1)$$

$$(n-1)\,A(n-1) = (n-1)(n-2) + 2\sum_{p=1}^{n-1} A(p-1) \qquad (2)$$

دو رابطه 1، 2 را از همدیگر کم می‌کنیم

$$n\,A(n) - (n-1)\,A(n-1) = 2\,(n-1) + 2\,A(n-1)$$

$$n\,A(n) = (n+1)\,A(n-1) + 2\,(n-1)$$

$$S_n = \frac{A(n)}{n+1}$$

$$\div\; n(n+1) \;\Longrightarrow\; \frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2\,(n-1)}{n(n+1)}$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$\begin{cases} S_n = S_{n-1} + \dfrac{2(n-1)}{n(n+1)} \\ S_0 = 0 \end{cases}$$

$$\Rightarrow S_n \simeq 2 \ln n$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$\begin{cases} S_n = S_{n-1} + \dfrac{2(n-1)}{n(n+1)} \\ S_0 = 0 \end{cases}$$

$$\Rightarrow S_n \simeq 2 \ln n$$

$$\frac{A(n)}{n+1} \simeq 2 \ln n$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$\begin{cases} S_n = S_{n-1} + \dfrac{2(n-1)}{n(n+1)} \\ S_0 = 0 \end{cases}$$

$$\Rightarrow S_n \simeq 2 \ln n$$

$$\frac{A(n)}{n+1} \simeq 2 \ln n$$

$$\Rightarrow A(n) \simeq 2(n+1) \ln n$$

تحلیل زمان اجرای مرتب سازی سریع در حالت متوسط

$$\begin{cases} S_n = S_{n-1} + \dfrac{2(n-1)}{n(n+1)} \\ S_0 = 0 \end{cases}$$

$$\Rightarrow S_n \simeq 2 \ln n$$

$$\frac{A(n)}{n+1} \simeq 2 \ln n$$

$$\Rightarrow A(n) \simeq 2(n+1) \ln n$$

$$A(n) \in \Theta(n \log n)$$

تمرین

- فرض کنید در ابتدای تابع Partition دستوری اضافه نماییم که جای عنصر اول و وسط را عوض نماید. n عدد مثال بزنید که اگر به تابع QuickSort داده شود بازهم بدترین حالت رخ دهد.