نمونه سوالات مربوط به بخش ۸.۷

۸.۷.۱ دو آرایه ی مرتب شده ی به طول های m و n داده شده اند. می خواهیم میانه ی ام m+n عدد را پیدا کنیم.
الف) روشی با زمان(O(logn * logm برای این کار ارائه دهید.
ب) روشی با زمان (O(logm + logn برای این کار ارائه دهید.

راه حل:

الف) عضو i ام از آرایه اول و عضو j ام از آرایه دوم را به صورت مخواهیم که i+j = (m+n)/2 باشد. با دو جستجوی دودویی تو در تو این دو عضو را پیدا می کنیم. binary search اول روی اعضای آرایه ی اول به جستجوی i می پردازد (با زمان (O(nlogn) و به ازای هر عضو با زمان (O(logm دنبال عضو j هستیم. خاصتی که این دو عضو دارند این است که عدد j از آرایه ی دوم نزدیک ترین عدد به عضو i از آرایه ی اول می باشد. (در نتجه در زمان logm بدست می آید) حال اگر i+j از m+n)/2) کمتر بود عضو i را به جلو حرکت می دهیم و اگر بشتر بود عضو i به عقب حرکت می کند.

ب) مجدد دنبال همان اعضا هستیم، اما نیازی نیست به ازای هر عدد i که از آرایه ی اول پیدا می کنیم زمان logm مصرف کنیم و دو اندیس i و j را همزمان حرکت می دهیم تا هر کدام در مجموع به ترتیب زمان های logn و logm طول بکشند. نکته ی این قسمت این است که اگر i+j کمتر از m+n)/2) باشد هیچ 'i کوچکتری از i هم نمتواند 'j معادلی پیدا کند که 'i'+j برابر m+n)/2) باشد. همنطور برای j هم چنین چیزی داریم. در نتجه یکی درمیان یکی از آنها را رو به جلو یا عقب در آرایه ی خود حرکت می دهیم تا به جایی برسند که دیگر حرکت نکنیم (یعنی نزدیک ترین اعداد به هم باشند) و شرط مورد نظر نیز برقرار باشد.

۸.۷.۲ مساله زیر آرایه بیشینه (maximum-subarray problem) ، در این مسأله هدف پیدا کردن بزرگترین زیر مجموعه پیوسته از اعداد یک آرایه است، که بیشترین مقدار ممکن را داشته باشد. فرض بر این است که تعدادی از اعداد منفی میباشند، زیرا درصورت مثبت بودن اعداد مسأله بدیهی است.

راه حل:

مراحل مختلف این ر وش را به شکل زیر دسته بندی میکنیم:

- زیرمسأله ها: زیرآرایه بیشینه از [A[low...highکه در فراخوانی اول low = ۱ و high = n است.
- تقسیم: محاسبه نقطه میانی که آن را mid مینامیم و تقسیم کردن زیر آرایه به دو زیرآرایه با اندازههای تقریباً یکسان.
- حل: یافتن زیرآرایهای بیشینه از [A[low...midو [A[mid+1...high
- ترکیب: یافتن زیرآرایهای بیشینه که از نقطه میانی میگذرد و انتخاب بهترین جواب از میان این ۳ راه حل.

برای ترکیب از تابع زیر استفاده میکنیم :

FIND-MAX-CROSSING-SUBARRAY$(A, low, mid, high)$

$left\text{-}sum = -\infty$
$sum = 0$
for $i = mid$ **downto** low
  $sum = sum + A[i]$
  if $sum > left\text{-}sum$

```
            left-sum = sum
            max-left = i
right-sum = −∞
sum = 0
for j = mid + 1 to high
      sum = sum + A[j]
      if sum > right-sum
            right-sum = sum
            max-right = j
return (max-left, max-right, left-sum + right-sum)
```

این تابع در حلقه اول که mid - low +1 بار اجرا می شود، زیرآرایهای بشنه که شامل عنصر مانی و عناصری که در سمت چپ آن قرار دارد مسابد.

حلقه دوم هم که high - (mid+1) +1 بار اجرا مىشود، زیرآرایه ای بشنه که شامل اولىن عنصر بعد از عنصر مانی و عناصری که در سمت راست آن قرار دارد می یابد. پس مرتبه اجرای این الگور یتمΘ(n) است که n = high - low +1.

ر وند حل و تقسم برای مسأله زیرآرایه بشنه توسط تابع زیر انجام مىشود:

Algorithm 2 FIND MAXIMUM SUBARRAY

```
function FMS(A, low, high)
    if high == low then
        return (low, high, A[low])
    else
        mid ← ⌊(low+high)/2⌋
        (low_l, high_l, sum_l) ← FMS(A, low, mid)
        (low_r, high_r, sum_r) ← FMS(A, mid + 1, high)
        (low_c, high_c, sum_c) ← FMCS(A, low, mid, high)
        if sum_l ≥ sum_r and sum_l ≥ sum_c then
            return (low_l, high_l, sum_l)
        else
            if sum_r ≥ sum_l and sum_r ≥ sum_c then
                return (low_r, high_r, sum_r)
            else
                return (low_c, high_c, sum_c)
```

برای سادگی تحلىل زمان اجرای الگور یتم فرض مىکنىم n توانی از ۲ باشد. زمان اجرا را هنگامی که اندازه ور ودی مسأله n است را با n(T) نشان مىدهىم. برای حالت پایه یعنی وقتی که n = ۱ است، خط دوم اجرا خواهد شد که زمان ثابتی طول مىکشد پس:

$$T(۱) = \Theta(۱)$$

حالت بازگشتی زمانی رخ خواهد داد که n > ۱ باشد، خطوط ۱و۳ زمان ثابتی طول مىکشند. همچنىن زمان اجرای خطوط چهارم و پنجم 2(T(n/2)) مىباشد. همچنىن مىدانىم زمان اجرای FMCS از مرتبه Θ(n)است، پس:

$$T(n) = \Theta(۱) + ۲T(n/۲) + \Theta(n) + \Theta(۱) = ۲T(n/۲) + \Theta(n)$$

در نتیجه رابطه کلی n(T) به شکل زیر خواهد بود:

$$T(n) = \Theta(۱) : n = ۱$$

$$T(n) = ۲T(n/۲) + \Theta(n) : n > ۱$$

که همان رابطه بازگشتی MergeSort مباشد، پس:

$$T(n) = \Theta(n \log n)$$

۸.۷.۳ . n نقطه در فضای سه بعدی داریم. الگوریتمی با بهترین زمان ارائه دهد که نزدیک ترین جفت نقاط را پیدا کند . اگر فضا چهار بعدی باشد چه تغىیری در روش ایجاد می شود و آیا زمان اجرای آن تفاوتی با حالت قبل دارد؟

راه حل:

به روش تقسیم و غلبه حل می کنم و هربار صفحه ای انتخاب می کنم z=k (که k وسط zهای نقاط در این مرحله باشد) و دو طرف صفحه را جداگانه حل می کنم و d را کمىنه ی جواب های دو قسمت درنظر مگىریم. حال باید از هر دو قسمت نقاطی را بررسی کنم که در فاصله ی d از این صفحه هستند. به طور مشابه حالت دو بعدی، برای هر نقطه تعداد متنهایی نقطه وجود دارند که باید بررسی شوند. (زیرا در قسمتی که این نقطه است فاصله ی حداقل d دارند و نقاطی که درقسمت دیگر نزدیک تر از d هستند از هم حداقل d فاصله دارند) برای این که هزینه مرتب سازی نداشته باشم (یا مجبور نباشم همه نقاط را بررسی کنم) مانند حالت دو بعدی که نقاط را روی خط تصویر می‌کردیم، با O(n) نقاط را روی صفحه انتخاب شده تصویر می کنم. تا اینجا هزینه $T(n) = 2(T(n/2)) + O(n)$ شده، درنتجه T(n, 2) که حل مساله در دو بعد است را برای آنها انجام داد. در واقع:

$$T(n, 3) = 2(T(n/2, 3)) + T(n, 2) + O(n)$$

که مىدانىم $T(n, 2) = O(n\log n)$ است.

برای محاسبه ی T(n, d) از استقرا استفاده می کنم و فرض می کنم $T(n, d-1) = n(\log n)^{d-2}$ باشد. با استفاده از قضىه اصلی داریم:

$$T(n, d) = 2T(n/2, d) + n(\log n)^{d-2} + O(n) = n(\log n)^{d-1}$$

تفاوت سه بعدی و چهاربعدی بودن مسئله هم در زمان اجرا مشخص است.

۸.۷.۴ دو عدد n بىتی A و B داریم. به دلىل بزرگ بودن آنها جمع این دو عدد در O(1) توسط پردازنده امکان پذیر نىست و اعمال جمع و تفریق این دو عدد O(n) زمان زم دارد. ابتدا محاسبه کنىد که ضرب این دو عدد چه مقدار زمان نىاز دارد و سپس الگوریتمی با مرتبه ی زمانی $O(n^{1.5})$ برای ضرب این دو عدد ارائه دهىد.

راه حل:

ضرب دو عدد در حالت عادی $O(n^2)$ است. هر کدام از بىت های عدد B را در عدد A ضرب می کنم و این n عدد بدست آمده را (که عدد iام i تا شىفت داده شده) با هم جمع می کنم که n^2 مىشود.

هرکدام از دو عدد را از وسط به دو قسمت مساوی کم ارزش و پر ارزش تقسم مکنم، در این صورت:

A = A1* a+ A0

B = B1* b+ B0

که a و b برابر ۲ به توان len(A0) هستند. (وظفه ی شفت دادن بت های A1 و B1 را دارند تا ارزش انها حفظ شود) سپس حاصل A*B را منویسم:

A*B = (A1*a+ A0) * (B1*a+B0) = A1* B1* a^2 + (A1*B0 + A0*B1)*a + A0*B0

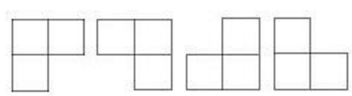
که ۴ ضرب اعداد n/2 بتی دارد و زمان الگوریتم T(n)= 4T(n/2) است و همان n^2 مشود. ولی متوان همن عبارت را با ۳ ضرب حساب کرد که زمان مورد نظر سوال بدست باید. دو ضرب اول و آخر انجام مشوند و برای محاسبه A1*B0 + A0*B1 به صورت زیر عمل مکنم:

(A1*B0 + A0*B1) = (A1+A0)(B1+B0) - A1*B1 - A0*B0

که A1*B1 و A0*B0 یکبار حساب شده‌اند و نیاز نست دوباره محاسبه شوند. در نتجه در کل سه ضرب انجام مشود و تعداد کمی هم جمع داریم که O(n) هستند. زمان الگوریتم از رابطه T(n) = 3T(n/2) + cn بدست می‌آید که با توجه به قضه اصلی محاسبه شده و$O(n^{\log_2 3})$

می باشد.

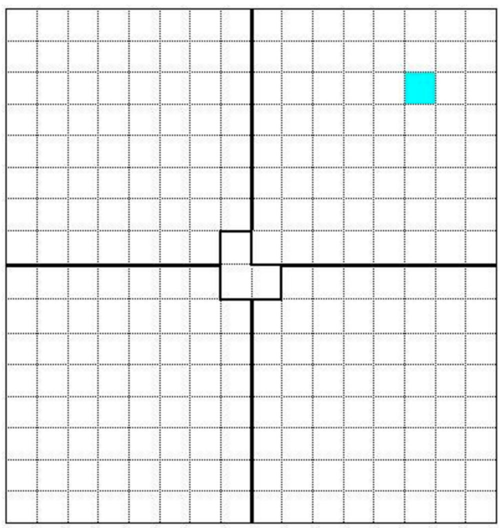
۸.۷.۵ یکی از مسائل جالب طراحی الگوریتم مسأله‌ی کاشکاری یا فرش کردن زمن با موزایک است فرض کنند قطعه زمن مربعی شکل با ابعادی از توان عدد دو داریم. مثلا با ابعاد 16 مترهدف فرش کردن این قطعه زمن با استفاده از موزایک‌هایی با شکل‌های زیر است:



به قسمی که یکی از خانه‌های زمن شطرنجی شده‌ی فوق پوشده نشود. می‌توان فرض کرد از این خانه برای احداث باغچه یا حوضچه‌ی کوچکی استفاده خواهد شد. توجه داشته باشد که اندازه‌ی اضلاع مربع‌های کوچک موزایک‌ها نز همانند صفحه‌ی شطرنجی فوق یک متر است. در ضمن حق شکستن این موزایک‌ها به تکه‌های کوچکتر را نداریم.

راه حل:

با توجه به اینکه ابعاد زمین توانی از عدد دو است، روش تقسیم و حل را برای پوشاندن این قطعه زمین امتحان می‌کنم. بر اساس تعریف روش تقسیم و حل، باید بتوان مسأله را به زیرمسائلی از نوع خود مسأله تقسیم کرد و با ادغام نتایج حاصل از آنها، به نتیجه‌ی اصلی دست پیدا کرد. چون ابعاد این قطعه زمین توانی از عدد دو است، پس می‌توان آن را به دو قسمت تقسیم کرد. با تقسیم طول و عرض زمین به دو قسمت، چهار قطعه زمین کوچکتر به دست می‌آید، اما همه‌ی این چهار قطعه زمین شرایط مسأله‌ی اصلی را دارا نستند. در صورت مسأله عنوان شده است که باید از پوشاندن یکی از خانه‌های قطعه زمین صرف نظر کرد. از چهار قطعه زمین کوچکتر تنها یکی از آنها این شرط را برآورده می‌کند و بقیه‌ی قطعه زمن‌ها باید به طور کامل پوشانده شوند. این نقص را می‌توان با به کار بردن یک موزاییک حل کرد:



این موزاییک در مرکز قطعه زمین به قسمی قرار داده شده است که هر کدام از سه قسمت مربعی شکل آن، داخل یکی از قطعه زمین‌های کوچکتری قرار گرفته است که شرط مسأله را برآورده نمی‌کردند. با این کار، داخل این قطعه زمن‌ها نیز خانه‌ای وجود دارد که نباید پوشانده شود. چرا که از قبل توسط موزاییکی پوشیده شده است. به این ترتیب همه‌ی چهار قطعه زمین کوچکتر خانه‌ای دارند که نیاز به پوشش ندارد. در نتیجه می‌توان بر روی حل مسأله در قطعه زمین‌های کوچکتر تمرکز کرد

۸.۷.۶ دو ماتریس A و B داریم. می دانیم ضرب آنها در حالت عادی زمان $O(n^3)$ نیاز دارد. الگوریتمی ارائه دهد که ضرب این دو ماتریس را با زمان $O(n^{\log_2 7})$ ام دهد.

راه حل:

هر کدام از دو ماتریس را به ۴ قسمت n/2 در n/2 تقسیم می کنم. در این صورت ضرب دو ماتریس به شکل زیر می شود:

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}}_{C}$$

که برای محاسبه ی حاصل ضرب نیاز به محاسبه ی ۸ ضرب است و جمع هایی که در مجموع اندازه ی جمع دو ماتریس n*n است. طبق قضیه ی اصلی اردر این روش $T(n) = 8T(n/2) + O(n^2)$ است ($O(n^2)$ برای جمع زدن) که برابر است با $O(n^3)$

در نتیجه اینجا هم نیاز است تا تعداد ضرب ها را کاهش دهیم. (از زمان خواسته شده هم پیداست که باید به ۷ ضرب با اندازه ی n/2*n/2 برسیم!(حاصل های r1 تا r7 را)که هر کدام یک ضرب دارند) به این صورت تعریف می کنیم:

$r1 = a(f-h)$

$r2 = (a+b)h$

$r3 = (c+d)e$

$r4 = d(g-e)$

$r5 = (a+d)(e+h)$

$r6 = (b-d)(g+h)$

$r7 = (a-c)(e+f)$

در نتیجه داریم :

$c11 = ae + bg = r5 + r4 - r2 + r6$

$c12 = ce + dg = r3 + r4$

$c21 = af + bh = r1 + r2$

$c22 = cf + dh = r1 + r5 - r3 - r7$

زمان هم با قضیه ی اصلی از رابطه ی $T(n) = 7T(n/2) + O(n^2)$ بدست می آید.

۸.۷.۷ آرایه ای به طول n داریم. با زمان O(n) عددی را پیدا کنند که بیش از n/2 بار در آرایه تکرار شده است.

راه حل:

هربار با پیدا کردن یک pivot به اندازه ی میانگین اعداد آرایه ی آرایه را به دو قسمت تقسیم می کنیم. باO(n) روشی داریم که اعداد بزرگتر از pivot انتخاب شده بعد از آن و اعداد کوچکتر یا مساوی قبل از آن قرار بگیرند. مث در مرحله ی اول واضح است مسئله را باید در قسمت

بزرگتر ادامه دهیم. در نهایت به دنبال حالتی هستیم که آرایه به طول k به دو زیرمسئله به اندازه های 0 و k تقسیم شود (که یعنی همه ی اعداد برابرند) و n=<k/2 باشد. تحلیل زمان هم مشابه مسئله ی quicksort انجام می شود.

۸.۷.۸ فرض کنید یک درخت جستجوی دودویی روی اعداد $a_1, a_2, \ldots, a_n$ داده شده. الگوریتمی طراحی کنید که با استفاده از این درخت، دنباله ی مرتب شدهی این اعداد را به عنوان خروجی تولید کند. نشان دهید زمان اجرای الگوریتم شما O(n) است. توجه کنید که درخت ورودی لزوما متوازن نیست. (راهنمایی: درخت چپ و راست را بهشکل بازگشتی پردازش کنید و خطی بودن زمان اجرا را با استقرا نشان دهید. اگر غیربازگشتی ترجیح مدهید، طراحی الگوریتم خطی غیربازگشتی نیز ساده است.)

راه حل:

مدانم که با پیمایش DFS روی BST به آرایه مرتب مرسم، برای این کار نیز درخت را به دو زیر درخت چپ و راست تقسیم مکنم و مساله را روی هرکدام از زیر درخت ها حل مکنم.

```
inorder (root)

{

        if (root == null)
                print root

        inorder (root -> left)

        print root

        inorder (root -> right)

}
```

چون در این الگوریتم هر گره دقیقا یکبار دیده مشود هزینه الگوریتم O(n) مشود.
