



توجه

- در هر سوال، پس از طراحی الگوریتم و توضیح آن، شبهه کد (pseudocode) مربوطه را نوشته و تحلیل زمان اجرا و فضای حافظه‌ی آن را بدست آورید.
- توصیه می‌شود قبل از خواندن پاسخ‌ها، سعی کنید سوال را خودتان حل نمایید.

۱. (۲۰ نمره) فرض کنید در الگوریتم quicksort، هر بار آرایه به نسبت α و $1 - \alpha$ تقسیم می‌شود که α یک ثابت است و $0 < \alpha \leq \frac{1}{2}$. نشان دهید که حداقل عمق یک برگ در درخت بازگشتی تقریباً $-\log n / \log \alpha$ و حداکثر عمق تقریباً $-\log n / \log(1 - \alpha)$ است. (نگران خطای گرد کردن اعداد صحیح در محاسبات نباشید).

پاسخ: (۱۰ نمره اثبات مقدار کمینه و ۱۰ نمره اثبات مقدار بیشینه) اگر درخت اجرای بازگشتی را رسم کنید، کمترین عمق متعلق به قسمتی است که هر بار بخش کوچکتر آرایه به آن اختصاص پیدا می‌کند (به عبارتی، بخشی که ضریب α دارد). هربار الگوریتم، اندازه‌ی آرایه را از n به αn کاهش می‌دهد. پس از i مرحله (در عمق i)، اندازه به $\alpha^i n$ کاهش می‌یابد. هنگامی که برگ می‌رسیم که آنقدر آرایه شکسته شده باشد که طول آن یک باشد. به عبارتی، $\alpha^m n = 1$. این عبارت را می‌توانیم به صورت $\alpha^m = \frac{1}{n}$ نیز بنویسیم. با حل این رابطه (با گرفتن لگاریتم از طرفین) خواهیم داشت:

$$m \log \alpha = -\log n \Rightarrow m = -\log n / \log \alpha$$

به طور مشابه، بیشترین عمق، مربوط به بخشی است که هربار بخش بزرگتر آرایه به آن اختصاص پیدا می‌کند (به عبارتی، بخشی که ضریب $1 - \alpha$ دارد). در عمق M هنگامی که آرایه‌ای با یک عنصر می‌رسیم که داشته باشیم $(1 - \alpha)^M n = 1$. با حل این عبارت می‌بینیم که حداکثر عمق $M = -\log n / \log(1 - \alpha)$ خواهد بود.

دقت کنید که این مقادیر تقریبی است. چون در هر مرحله، آرایه دقیقاً به α و $1 - \alpha$ تقسیم نمی‌گردد.

۲. (۲۵ نمره) یک آرایه مرتب شده به طول n در حافظه خوانده شده که به اندازه k بار به راست شیفت چرخشی داده شده است. مثلاً آرایه‌ی $[۱, ۳, ۵, ۹, ۱۲]$ اگر به اندازه $k=2$ شیفت چرخشی به راست داده شود، به آرایه $[۹, ۱۲, ۱, ۳, ۵]$ تبدیل می‌گردد. الگوریتمی بیابید که میزان k (یا تعداد شیفت‌ها) را در کمترین زمان پیدا کند.

پاسخ: (۱۰ نمره توضیح الگوریتم و حالت‌بندی درست برای رابطه بازگشتی، ۱۰ نمره شبهه کد، ۵ نمره تحلیل زمان و حافظه؛ راه‌حل خطی، فقط ۳ نمره دارد). این مسئله، معادل یافتن اندیس عنصر کمینه است (با فرض اینکه آرایه از اندیس صفر شروع شود). راه‌حل بدیهی، بررسی همه‌ی عناصر آرایه است که پیچیدگی زمان اجرای $O(n)$ دارد. الگوریتم بهتر، استفاده از روشی مشابه جستجوی دودویی است که پیچیدگی زمان اجرای $O(\log n)$ دارد. جزئیات این روش بدین صورت است. کمترین عنصر آرایه، تنها عنصری است که عنصر قبلی آرایه از آن بزرگ‌تر است. اگر عنصری قبل آن نبود، آرایه شیفت نخورده یا تعداد شیفت‌ها ضریبی از n است. در ابتدا عنصر وسط با دو عنصر کناری ($mid + 1$ و $mid - 1$) مقایسه می‌شوند:

- اگر عنصر وسط کوچکتر از عنصر $mid - 1$ باشد، عنصر وسط عنصر کمینه است.
- اگر عنصر وسط بزرگتر از عنصر $mid + 1$ باشد، عنصر $mid + 1$ عنصر کمینه است.

در غیر این صورت، عنصر کمینه یا در نیمه‌ی راست یا در نیمه چپ قرار دارد:

- اگر عنصر وسط از آخرین عنصر کوچک‌تر باشد، عنصر کمینه در نیمه چپ قرار دارد.
- در غیر این صورت، در نیمه‌ی راست قرار دارد.

با این روش، آرایه هربار به نصف شکسته شده تا کمترین عنصر پیدا شود. شبه کد مربوطه به صورت زیر می‌باشد:

```
FindRotationCount(A, left, right) {
    if left = right
        return left
    mid = (left + right) / 2
    if mid > 0 AND A[mid] < A[mid - 1]
        return mid

    if mid < right AND A[mid] > A[mid + 1]
        return mid + 1
    // Right array is sorted, search in the left array
    if A[mid] <= A[right]
        return FindRotationCount(A, left, mid-1)

    // Left array is sorted, search in the right array
    return FindRotationCount(A, mid+1, right)
}
```

زمان اجرا $O(\log n)$ و میزان حافظه مصرفی $O(n)$ و حافظه کمکی $O(1)$ می‌باشد.

۳. (۴۰ نمره) فرض کنید یک فایل متنی به صورت یک رشته با n حرف به شما داده شده است که تمام کلمات آن به هم چسبیده‌اند. به عبارتی فاصله‌ای بین کلمات وجود ندارد. شما می‌خواهید طوری فاصله بین کلمات درج کنید که متن حاصل با معنی و خوانا شود. به عنوان مثال، اگر در ورودی عبارت «منمشتعلعشقعلیمچهکنم» را دریافت کردید، رشته‌ی فاصله‌گذاری شده به صورت «من مشتعل عشق علیم چه کنم» خواهد بود. برای کمک به شما، تابعی در اختیارتان قرار گرفته به نام $f(w)$ که به ازای هر کلمه (مجموعه‌ای از حروف) نشان‌دهنده‌ی میزان با معنی بودن آن است. هدف شما یافتن الگوریتم با زمان اجرای $O(n^2)$ است که مجموع $f(w)$ را روی همه کلمات جدا شده متن به حداکثر برساند. نیازی به چاپ رشته‌ی فاصله‌گذاری شده نمی‌باشد.

پاسخ: (۱۰ نمره توضیح الگوریتم، ۱۰ نمره رابطه بازگشتی صحیح، ۱۵ نمره شبهه‌کد، ۵ نمره تحلیل حافظه) فرض کنید پاسخ بهینه $X_1X_2X_3...X_k$ باشد که متن را به k کلمه‌ی $X_1, X_2, ..., X_k$ شکسته است. بر همین اساس، $X_1X_2X_3...X_{k-1}$ پاسخ بهینه است اگر کلمه‌ی X_k را از متن حذف کنیم، زیرا اگر پاسخ بهینه نباشد، می‌توان پاسخ دیگری پیدا کرد تا بر آن اساس پاسخی بهتر از $X_1X_2X_3...X_k$ بدست آورد. حال فرض کنید $Opt(i)$ برابر بیشینه‌ی مجموع $f(w)$ روی i حرف اول است. با توجه به ساختار بهینه‌ی زیرمسئله‌ها می‌توان رابطه‌ی بازگشتی زیر را نوشت:

$$Opt(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max_{1 \leq j \leq i} \{Opt(j-1) + f(j \dots i)\} & \text{otherwise} \end{cases}$$

در این رابطه، $f(j \dots i)$ مقدار تابع را برای حروف اندیس j الی i محاسبه می‌کند. پاسخ مطلوب مسئله، $Opt(n)$ می‌باشد که به کمک برنامه‌ریزی پویا می‌توان آن را محاسبه کرد. به دلیل وجود دو حلقه‌ی تو در تو، زمان اجرا $O(n^2)$ خواهد بود. پیچیدگی حافظه، $O(n)$ می‌باشد.

```
WordSegmentation(X) {  
    // Array indices are zero based.  
    opt[0] = 0  
    n = X.length  
    for i = 1 to n {  
        opt[i] = -∞  
        for j = 1 to i {  
            opt[i] = max(opt[i], opt[j-1] + f(X[j-1] ... i-1))  
        }  
    }  
}
```

```

    }
    return opt[n]
}

```

۴. (۲۵ نمره) آرایه‌ی $A = \{a_1, a_2, \dots, a_n\}$ و عدد S به شما داده شده است. الگوریتمی به کمک برنامه‌ریزی پویا با زمان اجرای غیر نمایی (شبهه چندجمله‌ای) طراحی کنید که مشخص کند آیا می‌توان زیرمجموعه‌ای از A را پیدا کرد که مجموع آن‌ها برابر S شود.

پاسخ: (۱۰ نمره توضیح الگوریتم و رابطه بازگشتی صحیح، ۱۰ نمره شبهه کد، ۵ نمره تحلیل زمان و حافظه) این سوال مشابه مسئله کوله‌پشتی صفر-یک می‌باشد، با این تفاوت که ارزش اشیا برابر است و می‌خواهیم بدانیم آیا می‌توانیم کوله‌پشتی را به طور کامل پر کنیم یا خیر.

فرض کنید $S(i, j)$ نشان‌دهنده‌ی این است که آیا می‌توان زیرمجموعه‌ای از $\{a_1, a_2, \dots, a_i\}$ پیدا کرد که مجموعش j شود. $0 \leq j \leq S$. $S(i, j)$ می‌تواند $true$ یا $false$ باشد. بر این اساس، می‌توان رابطه بازگشتی زیر را نوشت. دقت کنید OR همان یای منطقی است.

$$S(i, j) = \begin{cases} true & \text{if } i = j = 0 \\ false & \text{if } i = 0, j \neq 0 \\ S(i-1, j) & \text{if } j < a_i \\ S(i-1, j) \text{ OR } S(i-1, j-a_i) & \text{otherwise} \end{cases}$$

شبهه کد برنامه‌ریزی پویا برای رابطه‌ی بازگشتی به این صورت می‌باشد:

```

SubsetSum(A, s) {
    // Assume array indices are zero-based
    n = A.length
    S[0,0] = true
    for j = 1 to n {
        S[0,j] = false
    }
    for i = 1 to n {
        for j = 0 to s {
            S[i,j] = S[i-1,j]
            if j >= A[i-1]
                S[i,j] |= S[i-1, j-A[i-1]]
        }
    }
    return S[n, s]
}

```

پیچیدگی زمان اجرای الگوریتم، $O(s \times n)$ است که شبیه چندجمله‌ای است. پیچیدگی حافظه در این پیاده‌سازی، $O(s \times n)$ است که قابل کاهش به $O(s)$ می‌باشد.