« به نام خدا »

مریم اسعدی‌فر - تکلیف سری اول الگوریتم

۱-

الف) c^n, n^k : $(c, k > 1)$

$\lim_{n\to\infty} \frac{c^n}{n^k} = \infty \Rightarrow n^k = O(c^n)$

$\Rightarrow n^k = o(c^n)$ (چون اکیداً بزرگتر است $c^n > n^k$)

$\Rightarrow c^n = \Omega(n^k)$

$\Rightarrow c^n = \omega(n^k)$

$\Rightarrow c^n \neq \Theta(n^k)$

ب) $\sqrt{n}, n^{\sin(n)}$ ← قابل مقایسه نیستن ... 😬.

چون $\sin(n)$ در بینهایت حد نداره.

$\lim_{n\to\infty} \frac{\sqrt{n}}{n^{\sin(n)}} = ?$ یعنی IDK

پ) $2^n, 2^{n/2}$:

$\lim_{n\to\infty} \frac{2^n}{2^{n/2}} = \infty \rightarrow 2^{n/2} = O(2^n)$

$2^{n/2} = o(2^n)$

$2^n = \Omega(2^{n/2})$

$2^n = \omega(2^{n/2})$

$2^2 \neq \Theta(2^{n/2})$

ت) $n^{\log c}, c^{\log n}$

$\lim_{n\to\infty} \frac{n^{\log c}}{c^{\log n}} = 1 \rightarrow n^{\log c} = O(c^{\log n})$

$n^{\log c} \neq o(c^{\log n})$

$n^{\log c} = \Omega(c^{\log n})$

$n^{\log c} \neq \omega(c^{\log n})$

$n^{\log c} = \Theta(c^{\log n})$

ث) $\log(n!), \log(n^n)$

$\lim_{n\to\infty} \frac{\log(n!)}{\log(n^n)} = 1 \rightarrow \log(n!) = \Theta, \Omega, O(\log(n^n))$

$\log(n!) \neq \omega, o(\log(n^n))$

ج) $\log(n!), (\log n)!$:

$\lim_{n\to\infty} \frac{(\log(n))!}{\log(n!)} = \infty \rightarrow \log(n!) = o, O((\log(n))!)$

$(\log(n))! = \omega, \Omega(\log(n!))$

$(\log(n))! \neq \Theta(\log(n!))$

A	B	O	o	Ω	ω	Θ خلاصه:
n^k	c^n	yes	yes	no	no	no
$\sqrt{n}$	$n^{\sin n}$	no	no	no	no	no
2^n	$2^{n/2}$	no	no	yes	yes	no
$n^{\log c}$	$c^{\log n}$	yes	no	yes	no	yes
$\log(n!)$	$\log(n^n)$	yes	no	yes	no	yes
$\log(n!)$	$(\log n)!$	yes	yes	yes	yes	no

۲-

I. $f(n) = O(g(n)) \Rightarrow g(n) = O(f(n))$ false ☒

ex: $f(n) = n^2$, $g(n) = 2^n$

II. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$ false ☒

ex: $f(n) = n$, $g(n) = n^2$

III. $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$ True ☑

$f(n) = O(g(n)) \rightarrow \exists c, n_0 > 0, \forall n > n_0 ; f(n) \leq c g(n)$

$\rightarrow g(n) \geq \frac{1}{c} f(n) \rightarrow g(n) = \Omega(f(n))$

IV. $f(n) = \Theta(f(n/2))$ false ☒

ex: $f(n) = 2^{2n}$, $f(n/2) = 2^n$

۳-

```
Void Sort(int *array, int size){
    for(int i=0; i<size; i++)
        while(array[i] != i){
            swap(array[i], array[array[i]]);
        }
}
```

ورودی: یک آرایه از جایگشت اعداد صفر تا $(n-1)$.

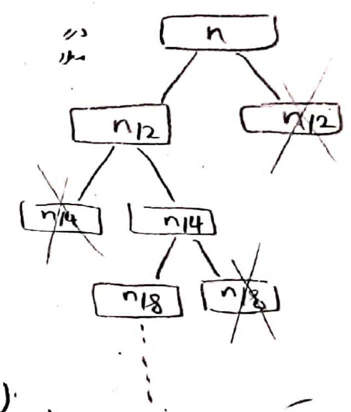
خروجی: آرایه مرتب شده.

این مرتب سازی خیلی ساده است، درواقع با بردن هر داده را در خانه مربوط به همان اندیس میگذارد، پس کافیست یک بار از اول تا آخر آرایه را بررسی کند یعنی از $O(n)$ است.

۴- در یک آرایه‌ی مرتب شده به طول n، عنصری که با اندیس خانه برابر است را بیابید. (اندیس از ۱ شروع می‌شود.)
(الگوریتم شما Order آن، log n باشد.)

```
int find(int a[], int start, int end, int s)
{
    if (a[start-1+size/2] <= start-1+size/2)
    {
        if (a[start-1+size/2] == start-1+size/2)
            return start-1+size/2;
        else
            find(a, size/2+1, end, size/2);
    }
    else
        find(a, start, size/2, size/2);
}
```

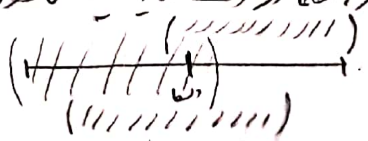
اردر این الگوریتم هم log n است زیرا



یعنی درخت تقسیم به صورت مورب است و عمق درخت log n می‌شود (بین زمان Devide و حل همان log n است) و مرج (merge ≡ Combine) کردن خاصی نداریم (O(1)).

۵- یافتن عدد غالب در یک دنباله‌ی سورت شده با O(log n)

اگر عنصری غالب باشد، حتماً در وسط الایه یافت می‌شود؛



پس کافیست چک کنم آیا عنصر وسط غالب هست یا نه؟!

با یک الگوریتم راحت! فقط اولین ظهور عنصر وسط آرایه را با binary search پیدا کرده و چک می‌کنم آیا در $\frac{n}{2}$ بعد از اولین ظهور هم آن عنصر وسط یافت می‌شود یا نه؟

اگر شد عنصر غالب است، در غیر این صورت نیست!

اردر تابع هم فقط مربوط به binary search است که O(log n)

```
bool isMajority(int a[], int n){
    int i = binarysearch(a, n, a[n/2]);
    // در دستور فوق تابع سرچ دودویی، اولین ظهور عنصر a[n/2]
    // که همان عنصر وسط است را پیدا و اندیس آن را در i قرار می‌دهد
    if (i == -1)
        return false;
    if (((i+n/2) <= (n-1)) && a[i+n/2]==a[n/2])
        return true;
    else
        return false;
}
```

۶- یافتن عنصر مینیمال در آرایه با اردر log n:

این الگوریتم بر مبنای سرچ باینری است! اول عنصر وسط را با عناصر مجاورش (در صورت وجود) مقایسه می‌کنم و ...

```
int localMin(int a[], int low, int high, int n)
{
    int med = low + (high-low)/2;
    if ((mid==0 || a[mid-1] > a[mid]) &&
        (mid==(n-1) || a[mid+1] > a[mid]))
        return mid;
    else if (mid > 0 && a[mid-1] < a[mid])
        return localMin(a, low, mid-1, n);
    return localMin(a, mid+1, high, n);
};
```

اردر زمانی هم که log n می‌شود. چون هربار تصمیم می‌گیرد، نیمه‌ی اول مسئله را حل کند یا دوم و اینقدر این کار را ادامه می‌دهد تا به نتیجه برسد. [مشابه binary search است دیگر، لذا همان O(log n) را نیز دارد.]