ساختمان داده ها و طراحی الگوریتم ها

جلسه اول کلاس حل تمرین

استاد درس: دکتر سجاد شیرعلی شهرضا

تدریسیار حل تمرین: پوریا جمیع

پاییز ۱۴۰۲

مقدمه و برنامه

۱- حل سوال های مختلف برای مباحث تدریس شده

۲- مرور مجدد برخی مباحث مهم

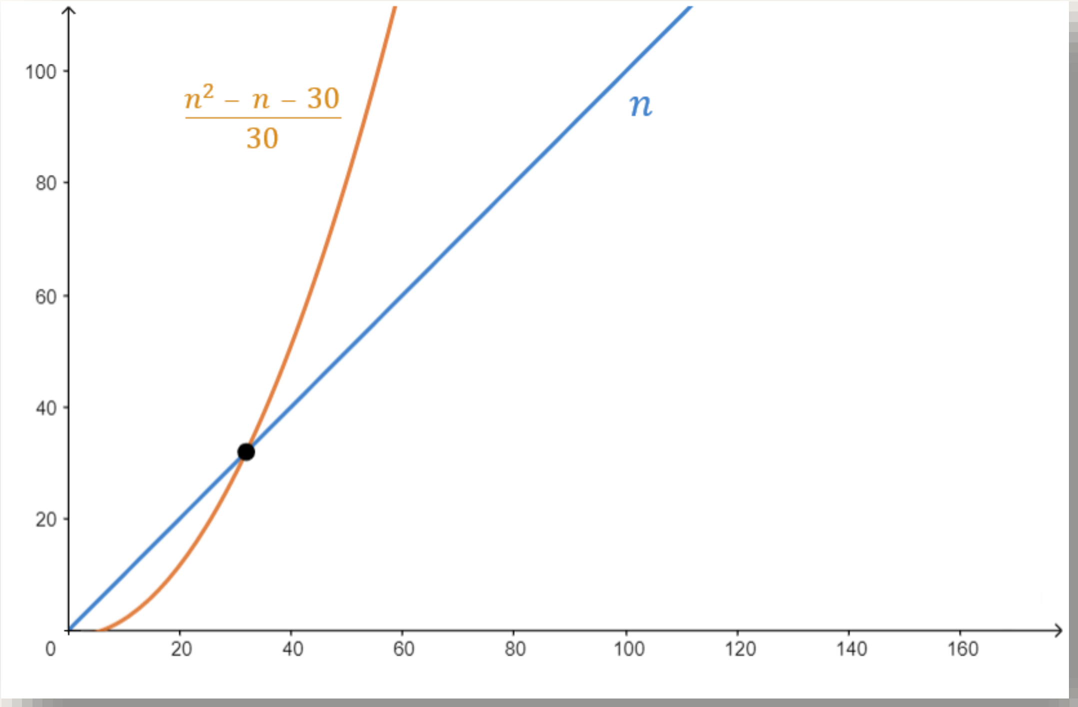
۳- پرسش و پاسخ کلاسی در انتهای هر جلسه

مباحث جلسه اول

- تحلیل زمانی الگوریتم ها
- مرتب سازی درجی و ادغامی
- حل با روش جایگذاری و قضیه اصلی

تحلیل زمانی الگوریتم ها

رشد توابع



در تحلیل توابع، **رشدِ** توابع برای‌مان اهمیت دارد و نه مقدار، به همین خاطر الگوریتمی که نمودار تابع متناظرش آبی باشد اگرچه به ازای مقادیر کوچک کندتر از نارنجی است (زمان بیش‌تری برای اجرا نیاز دارد)، اما برای مقادیر بزرگ‌تر کارآمدتر از نارنجی خواهد بود. در واقع ما باید این دو تابع را به ازای ورودی‌های **بسیار بزرگ** مقایسه کنیم.

ضرایب در تحلیل زمانی

$O(f(n))$	$O(n)$	$O(n^2)$	$O(n^5)$	$O(nlog_2n)$
$g_1(n)$	$3n+2$	$3n^2+4$	$n^5+100n-7$	$2nlog_2n-10$
$g_2(n)$	$\frac{5}{2}n+100$	$\frac{4}{17}n^2-30$	$2n^5+4n^3-100n^2-23$	$4nlog_2n+4n+\sqrt{n}+10$

ضرایب ثابت موجود در تابع‌مان را حذف می‌کنیم.

همچنین فقط بزرگ‌ترین توان را نگه می‌داریم

مثال

```
cnt = 0
for i from 1 to n:
    for j from 1 to n:
        cnt = cnt + i * j
```

به سادگی دیده میشود که عبارت جمع کردن در داخلی ترین حلقه به ازای هر i از ۱ تا n، n بار اجرا میشود پس به ازای هر n، n^2 بار اجرا میشود پس از $O(n^2)$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n \cdot n = n^2$$

مثال

```
cnt = 0
for i from 1 to n:
    for j from 1 to m:
        cnt = cnt + i * j
```

به سادگی دیده میشود که عبارت جمع کردن در داخلی ترین حلقه به ازای هر i از ۱ تا m،
n بار اجرا میشود پس به ازای هر n و m، n^2 بار اجرا میشود پس از $O(mn)$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}1=\sum_{i=1}^{n}m=m\cdot n=mn$$

مثال

```
cnt = 0
for i from 1 to n:
    for j from 1 to m:
        cnt = cnt + 1
    for g from 1 to k:
        cnt = cnt + 1
```

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{m} 1 + \sum_{g=1}^{k} 1\right) = \sum_{i=1}^{n}(m + k) = n \cdot (\mathrm{m} + \mathrm{k})$$

مثال

```
while n > 0:
    n = n / 2
```

در هر مرحله n نصف میشود و تا زمانی که n بیشتر از صفر بماند این کار تکرار میشود.

تعداد دفعه ای که دستور تقسیم به ازای هر n اجرا میشود برابر $\lceil \log_2 n \rceil$ خواهد بود؛ پس از $O(\log_2 n)$ است

مثال

```
// Sum returns the sum 1 + 2 + ... + n, where n >= 1.
func Sum(n int) int {
    if n == 1 {
        return 1
    }
    return n + Sum(n-1)
}
```

فرض کنید T(n) تعداد عملیات های انجام شده به ازای ورودی n برای تابع Sum باشد.

- $T(1) = 1, \ (*)$
- $T(n) = 1 + T(n-1), when\, n > 1. \ (**)$

ادامه مثال

```go
// Sum returns the sum 1 + 2 + ... + n, where n >= 1.
func Sum(n int) int {
    if n == 1 {
        return 1
    }
    return n + Sum(n-1)
}
```

$$T(n) = (**)$$

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

$$1 + (1 + T(n-2)) = 2 + T(n-2) = (**)$$

$$2 + (1 + T(n-3)) = 3 + T(n-3) = \ldots$$

$$k + T(n-k) = \ldots$$

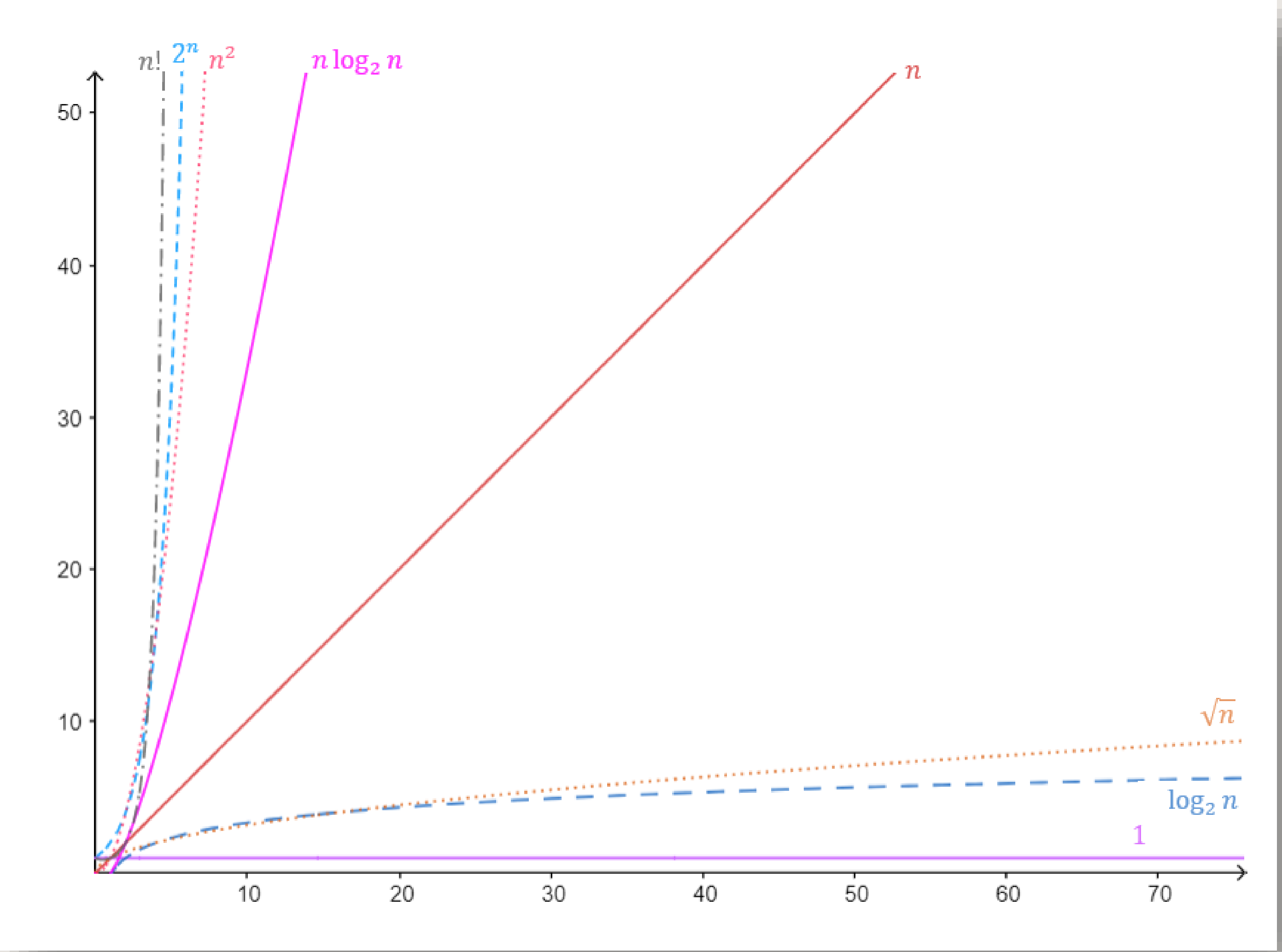
$$n - 1 + T(1) = (*)$$

$$n - 1 + 1 = \Theta(n)$$

منبع مفید

https://www.enjoyalgorithms.com/blog/time-complexity-analysis-of-loop-in-programming

شهود نسبت به میزان رشد توابع معروف



قضیه اصلی

$$T(n) = a \cdot T(\frac{n}{b}) + O(n^d).$$

$$T(n) = \begin{cases} \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

مثال

$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

$$a = 8 \qquad b = 2 \qquad d = 2$$

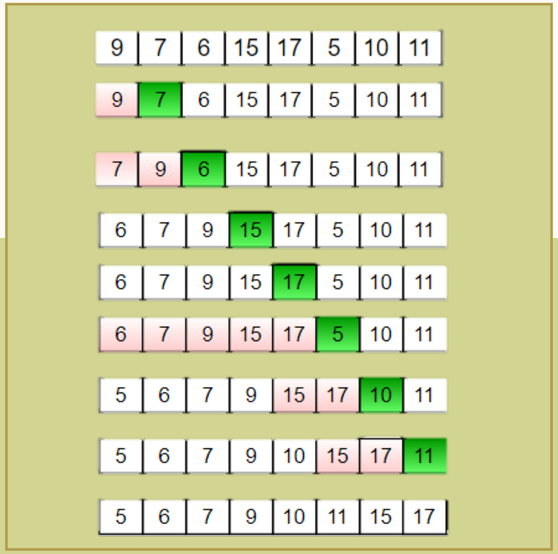
$$a \boxed{?} b^d$$

$$8 > 2^2$$

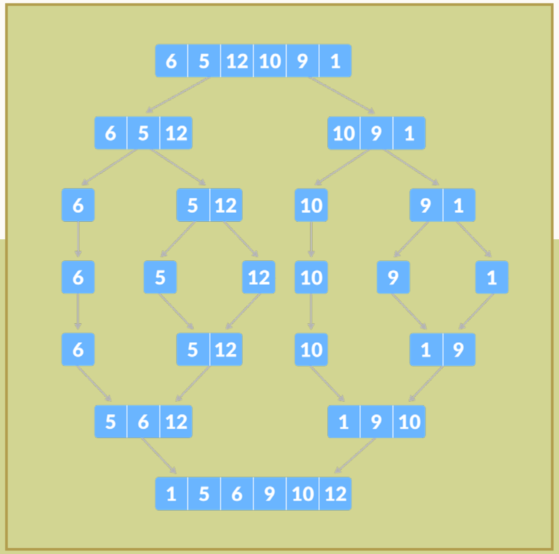
$$\text{Case 3} \rightarrow T(n) = \boldsymbol{\theta}(n^{\log_b a})$$

$$T(n) = \boldsymbol{\theta}(n^{\log_2 8}) = \boldsymbol{\theta}(n^3)$$

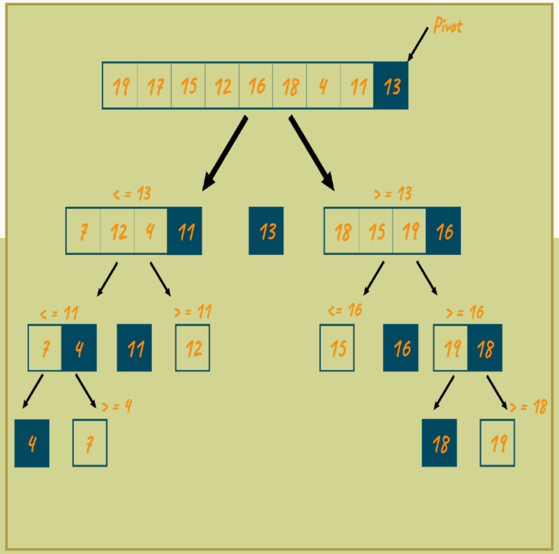
برخی از مرتب سازی ها



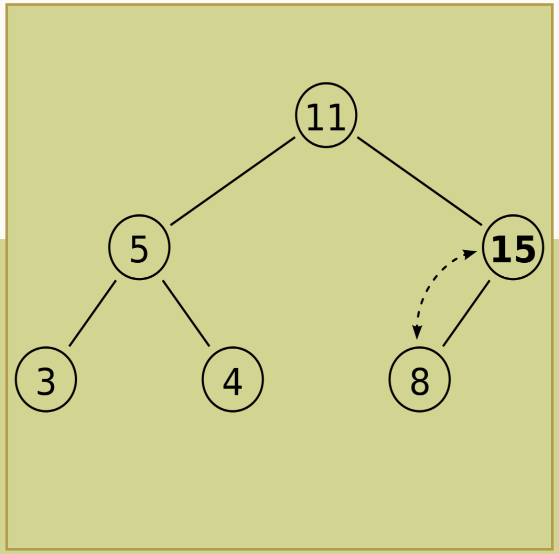
مرتب سازی درجی



مرتب سازی ادغامی



مرتب سازی سریع



مرتب سازی هرمی

مرتب سازی درجی

در این نوع مرتب‌سازی در مرحله‌ی i ام i عنصر اول مرتب شده‌اند حالا برای اینکه a_{i+1} را وارد کنیم تا زمانی که از قبلی‌اش کمتر است باید با قبلی‌اش جابه‌جا شود. هنگامی که از قبلی‌اش بیشتر شود $i+1$ عنصر اول مرتب می‌شوند و این روند را n بار تکرار می‌کنیم

$$< 64,25,12,22,11 >$$
$$< [25], 64,12,22,11 >$$
$$< [12], 25,64,22,11 >$$
$$< 12, [22], 25,64,11 >$$
$$< [11], 12,22,25,64 >$$

مصور سازی
مرتب سازی درجی

visualgo.net

hackerearth.com

مرتب سازی ادغامی

این یکی از الگوریتم‌هایی است که با استفاده از «بازگشت» (recursion) به سادگی پیاده‌سازی می‌شود، چون به جای مسئله اصلی با مسائل فرعی سر و کار داریم.

الگوریتم آن را می‌توان به صورت فرایند ۲ مرحله‌ای زیر توصیف کرد:

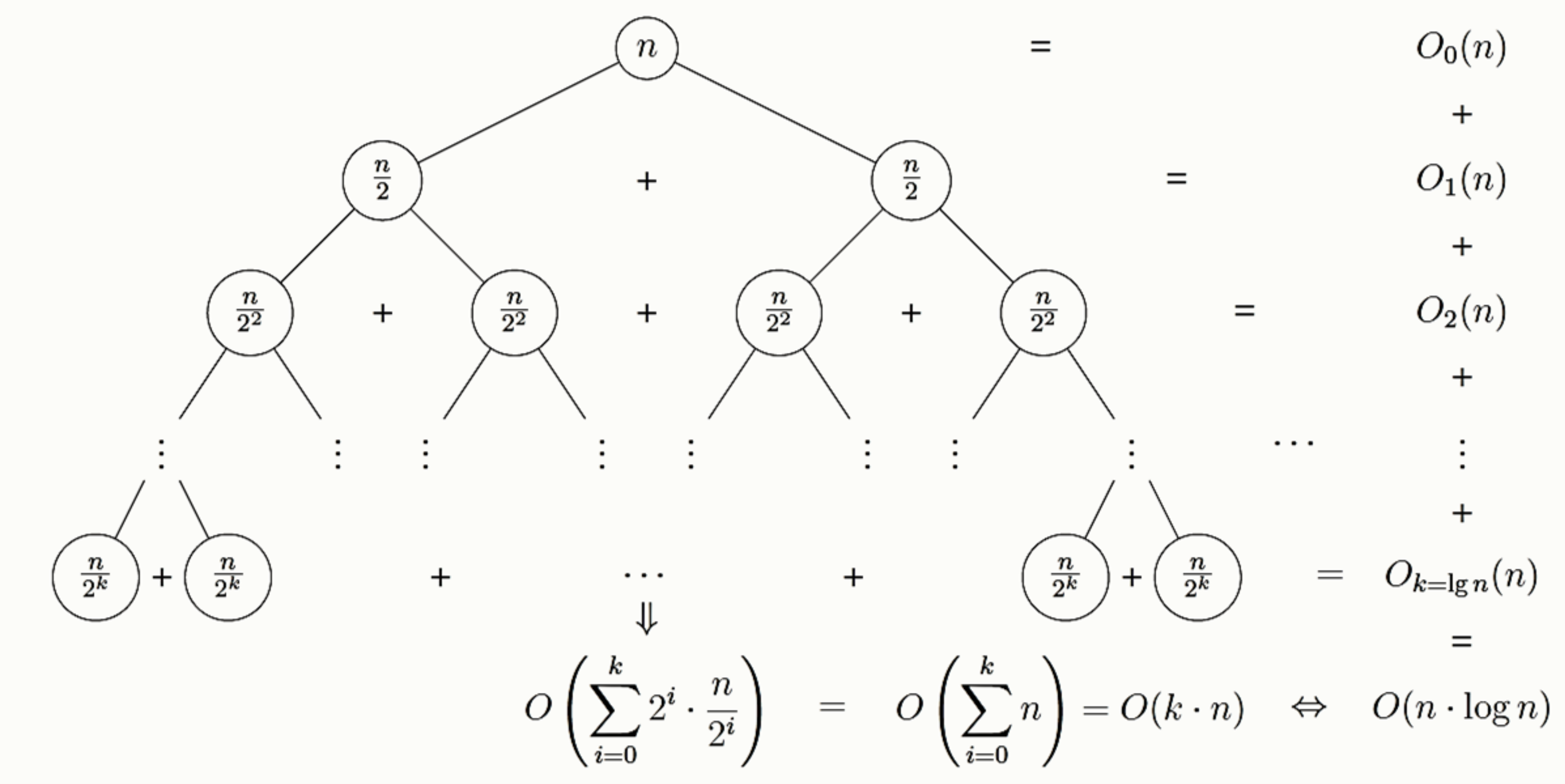
- تقسیم: در این مرحله آرایه ورودی به دو نیمه تقسیم می‌شود. محور تقسیم نقطه میانی آرایه است. این مرحله به صورت بازگشتی روی همه آرایه‌های نیمه انجام می‌یابد تا این که دیگر نیمه آرایه‌ای برای تقسیم وجود نداشته باشد.
- حل: در این مرحله باید آرایه‌های تقسیم‌شده را مرتب‌سازی و ادغام کنیم و این کار از بخش زیرین به سمت بالا برای به دست آوردن آرایه مرتب انجام می‌یابد.

مصور سازی
مرتب سازی ادغامی

hackerearth.com

MERGE SORT RECURSION TREE

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



تمرین بیشتر

پیچیدگی زمانی روابط بازگشتی زیر را یکبار از طریق قضیه اصلی و یکبار از طریق درخت بازگشت محاسبه کنید

$$T(n) = 2T\left(\frac{n}{2}\right) + 1$$

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

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

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

مسائل

1- Any algorithm that sorts by exchanging adjacent elements require $O(N^2)$ on average.

a) True
b) False

Explanation: Each swap removes only one inversion, so $O(N^2)$ swaps are required.

2- For the following question, how will the array elements look like after second pass?
34, 8, 64, 51, 32, 21

a) 8, 21, 32, 34, 51, 64
b) 8, 32, 34, 51, 64, 21
c) 8, 34, 51, 64, 32, 21
✓ d) 8, 34, 64, 51, 32, 21

Explanation: After swapping elements in the second pass, the array will look like, 8, 34, 64, 51, 32, 21.

3- Which of the following examples represent the worst case input for an insertion sort?

a) array in sorted order
b) array sorted in reverse order
c) normal unsorted array
d) large array

Explanation: The worst case input for an insertion sort algorithm will be an array sorted in reverse order and its running time is $O(n^2)$.

4- Merge sort uses which of the following technique to implement sorting?

a) backtracking
b) greedy algorithm
c) divide and conquer
d) dynamic programming

Explanation: Merge sort uses divide and conquer in order to sort a given array. This is because it divides the array into two halves and applies merge sort algorithm to each half individually after which the two sorted halves are merged together.

سوال (نسبتاً) خفن

5- Assume that a merge sort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes?

a) 256
b) 512
c) 1024
d) 2048

Time complexity of merge sort is $\Theta(n\text{Log}n)$

$$c * 64 \log 64 = 30$$
$$c * 64 * 6 = 30$$
$$c = \frac{5}{64}$$

For time 6 minutes

$$\frac{5}{64} * n \log n = 6 * 60$$
$$n \log n = 72 * 64 = 512 * 9$$
$$n = 512$$