Lecture 2
Divide & Conquer and Peak Finding

Spring 2023

Zhihua Jiang

Peak Finder

One-dimensional Version

Position 2 is a peak if and only if $b \geq a$ and $b \geq c$. Position 9 is a peak if $i \geq h$.

1	2	3	4	5	6	7	8	9
a	b	c	d	e	f	g	h	i

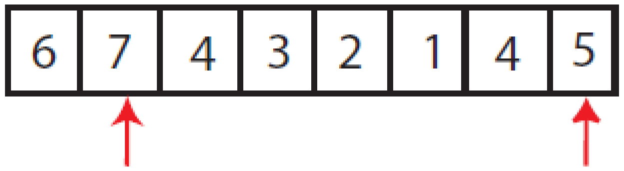
Figure 1: a-i are numbers

Problem: Find a peak if it exists (Does it always exist?)

PS: For the given definition (>=), a peak always exists!
However, if only >, a peak does not always exist. E.g., y(x)=c.

Peak Finder

Straightforward Algorithm



Start from left

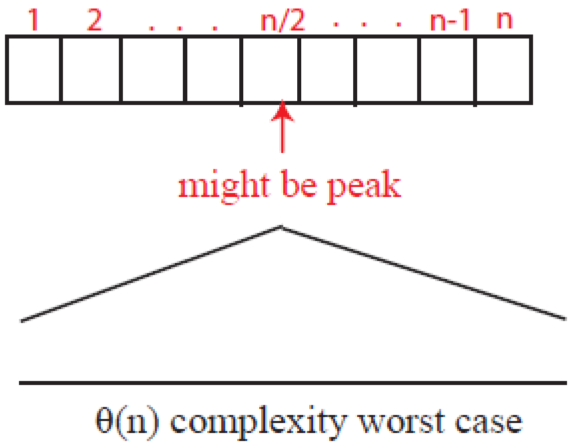
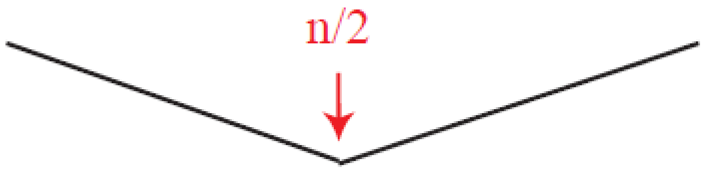


Figure 2: Look at $n/2$ elements on average, could look at n elements in the worst case

Peak Finder

What if we start in the middle? For the configuration below, we would look at $n/2$ elements. Would we have to ever look at more than $n/2$ elements if we start in the middle, and choose a direction based on which neighboring element is larger that the middle element?



Peak Finder

Can we do better?

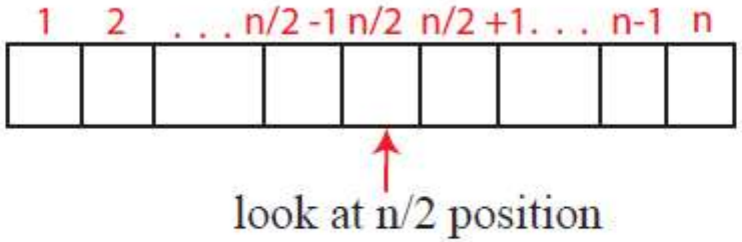


Figure 3: Divide & Conquer

- If $a[n/2] < a[n/2-1]$ then only look at left half $1 \ldots n/2 - 1$ to look for peak
- Else if $a[n/2] < a[n/2+1]$ then only look at right half $n/2+1 \ldots n$ to look for peak
- Else $n/2$ position is a peak: WHY?

$$\boxed{\begin{aligned} a[n/2] &\geq a[n/2-1] \\ a[n/2] &\geq a[n/2+1] \end{aligned}}$$

Peak Finder

What is the complexity?

$$T(n) = T(n/2) + \underbrace{\Theta(1)}_{\text{to compare a[n/2] to neighbors}} = \Theta(1) + \ldots + \Theta(1)\ (\log_2(n)\ times) = \Theta(\log_2(n))$$

In order to sum up the $\Theta(i)$ as we do here, we need to find a constant that works for all.
If $n = 1000000$, $\Theta(n)$ algo needs 13 sec in python. If algo is $\Theta(\log n)$ we only need 0.001 sec.

Peak Finder

Two-dimensional Version

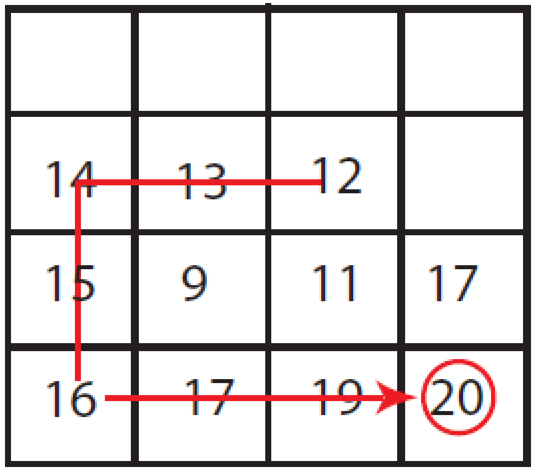


Figure 5: Circled value is peak.

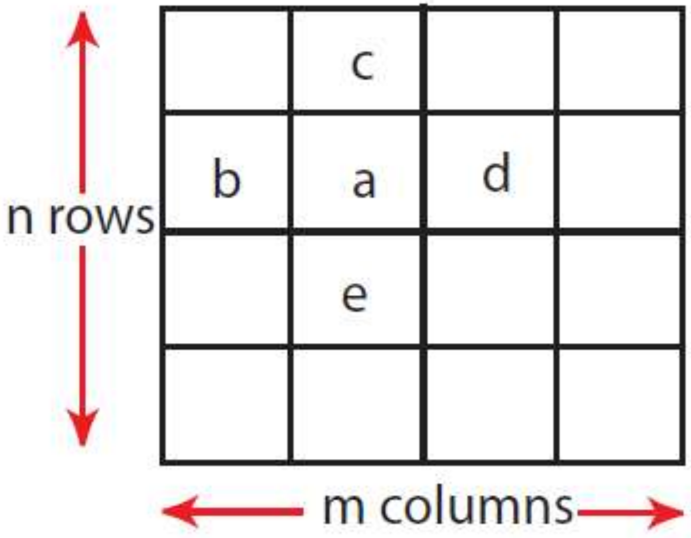
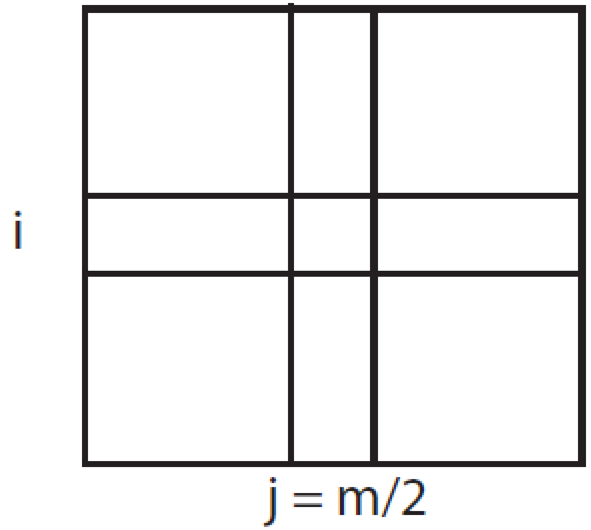


Figure 4: Greedy Ascent Algorithm: $\Theta(nm)$ complexity, $\Theta(n^2)$ algorithm if $m = n$

a is a 2D-peak iff $a \geq b, a \geq d, a \geq c, a \geq e$

Peak Finder

Attempt # 1: Extend 1D Divide and Conquer to 2D



- Pick middle column $j = m/2$.
- Find a 1D-peak at i, j.
- Use (i, j) as a start point on row i to find 1D-peak on row i.

Peak Finder

Attempt #1 fails

<u>Problem</u>: 2D-peak may not exist on row i

		10	
14	13	12	
15	9	11	
16	17	19	20

End up with 14 which is not a 2D-peak.

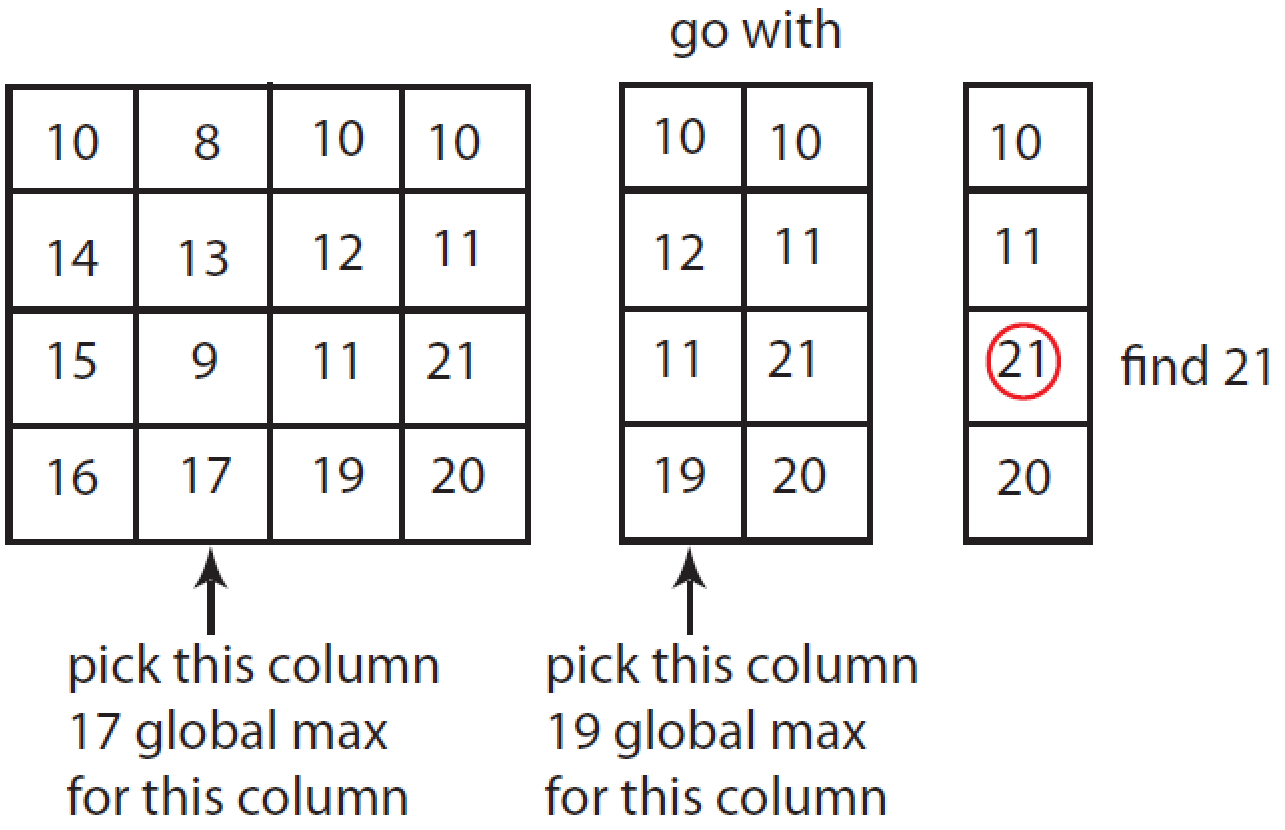
Peak Finder

Attempt # 2

- Pick middle column $j = m/2$
- Find global maximum on column j at (i, j)
- Compare $(i, j-1), (i, j), (i, j+1)$
- Pick left columns of $(i, j-1) > (i, j)$
- Similarly for right
- (i, j) is a 2D-peak if neither condition holds $\leftarrow$ WHY?
- Solve the new problem with half the number of columns.
- When you have a single column, find global maximum and you're done.

Peak Finder

Example of Attempt #2



Peak Finder

Complexity of Attempt #2

If $T(n,m)$ denotes work required to solve problem with n rows and m columns

$$
\begin{aligned}
T(n,m) &= T(n,m/2) + \Theta(n) \text{ (to find global maximum on a column — (n rows))} \\
T(n,m) &= \underbrace{\Theta(n) + \ldots + \Theta(n)}_{\log m} \\
&= \Theta(n \log m) = \Theta(n \log n) \text{ if m = n}
\end{aligned}
$$

Question: What if we replaced global maximum with 1D-peak in Attempt #2? Would that work?

10	8	10	10
14	13	12	11
15	9	11	21
16	17	19	20

- Divide-and-conquer strategy:
  - ***Break*** a problem into subproblems;
  - Recursively ***solve*** subproblems;
  - Appropriately ***combine*** their answers.

divide and conquer

MERGE-SORT $A[1 \,.\,.\, n]$

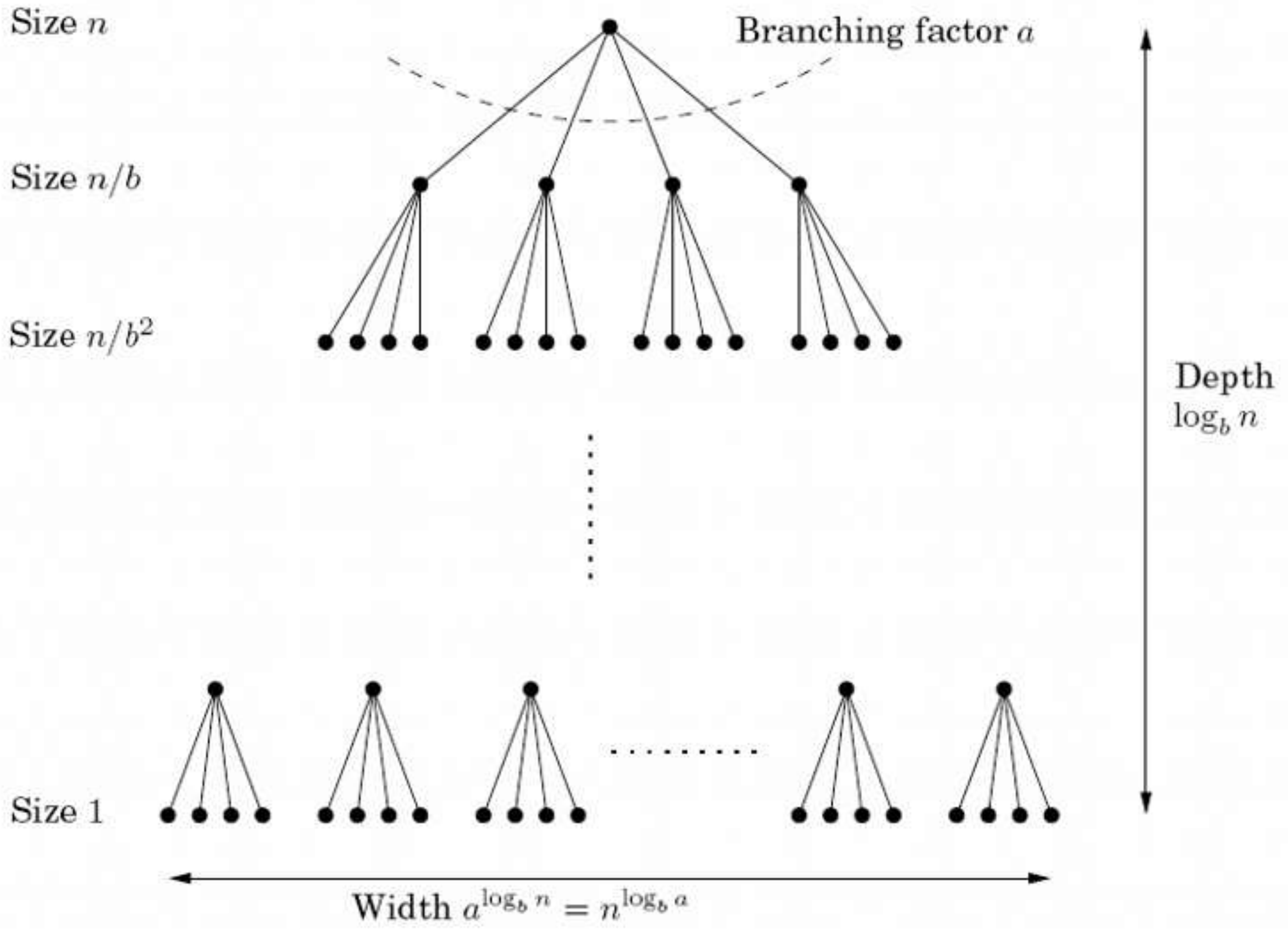
1. If $n = 1$, done (nothing to sort).
2. Otherwise, recursively sort $A[\, 1 \,.\,.\, n/2 \,]$ and $A[\, n/2{+}1 \,.\,.\, n \,]$.
3. "***Merge***" the two sorted sub-arrays.

- Divide-and-conquer algorithms tackle a problem of size n by recursively solving a subproblems of size n/b and then combine these answers in $O(n^d)$ time
  - Often, $a \neq b$
  - $O(n^d)$: polynomial time for all other efforts except for solving subproblems

- ***Master theorem*** If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a>0$, $b>1$, and $d \geq 0$, then

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

Figure 2.3 Each problem of size n is divided into a subproblems of size n/b.



Proof.

- Assume n is a power of b
- The total work done at the kth level

$$a^k \times O(\frac{n}{b^k})^d = O(n^d) \times (\frac{a}{b^d})^k$$

- As k goes from 0 to $\log_b n$, these numbers form a geometric series with ratio a/b^d.

1. *The ratio is less than* 1.

   Then the series is decreasing, and its sum is just given by its first term, $O(n^d)$.

2. *The ratio is greater than* 1.

   The series is increasing and its sum is given by its last term, $O(n^{\log_b a})$:

$$n^d \left(\frac{a}{b^d}\right)^{\log_b n} = n^d \left(\frac{a^{\log_b n}}{(b^{\log_b n})^d}\right) = a^{\log_b n} = a^{(\log_a n)(\log_b a)} = n^{\log_b a}.$$

3. *The ratio is exactly* 1.

   In this case **$O(\log_b n)$** terms of the series are equal to $O(n^d)$.

Solving recurrences

- Master theorem
- Substitution method
- Recursion-tree method

Substitution method

- Guess the form of the solution
- Verify by induction
- Solve for constraints of constants

Ex: T(n)=4T(n/2)+n [T(1)=Θ(1)]

-Guess $T(n)=O(n^3)$

-Assume $T(k) \leq ck^3$ for k<n

$$T(n) = 4T(n/2) + n \leq 4c(n/2)^3 + n = \frac{1}{2}cn^3 + n$$

$$= \underbrace{cn^3}_{desired} - \underbrace{(\frac{1}{2}cn^3 - n)}_{residual} \leq cn^3 \ \text{ if } \frac{1}{2}cn^3 - n \geq 0 \text{ i.e. } c \geq \frac{2}{n^2} \Rightarrow c \geq 2, n \geq 1$$

- -Try $T(n)=O(n^2)$
- -Assume $T(k) \le ck^2$ for $k<n$

$$T(n) = 4T(n/2) + n \le 4c(n/2)^2 + n = cn^2 + n$$

$$= \underbrace{cn^2}_{desired} - \underbrace{(-n)}_{residual} \text{ (fail)}$$

- -Try $T(n)=O(n^2)$
- -Assume $T(k) \le c_1k^2 - c_2k$ for $k<n$

$$T(n) = 4T(n/2) + n \le 4[c_1(n/2)^2 - c_2(n/2)] + n$$

$$= c_1n^2 - 2c_2n + n$$

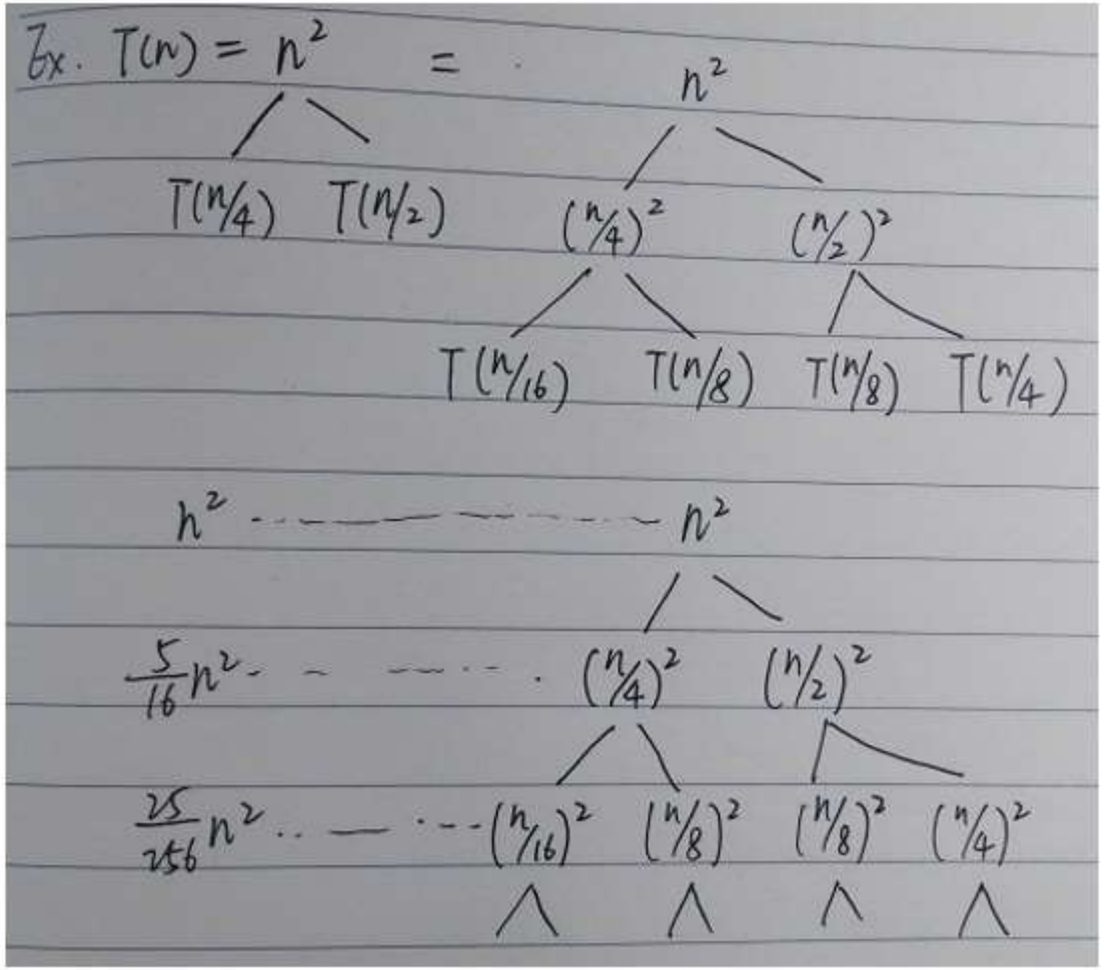
$$= \underbrace{c_1n^2 - c_2n}_{desired} - \underbrace{(c_2n - n)}_{residual} \text{ i.e. } c_2 \ge 1$$

$$T(1) \le c_1 - c_2 \Rightarrow c_1 \ge T(1) + c_2$$

i.e. c_1 is sufficently large w.r.t. c_2

Recursion-tree method

- Ex: $T(n)=T(n/4)+T(n/2)+n^2$

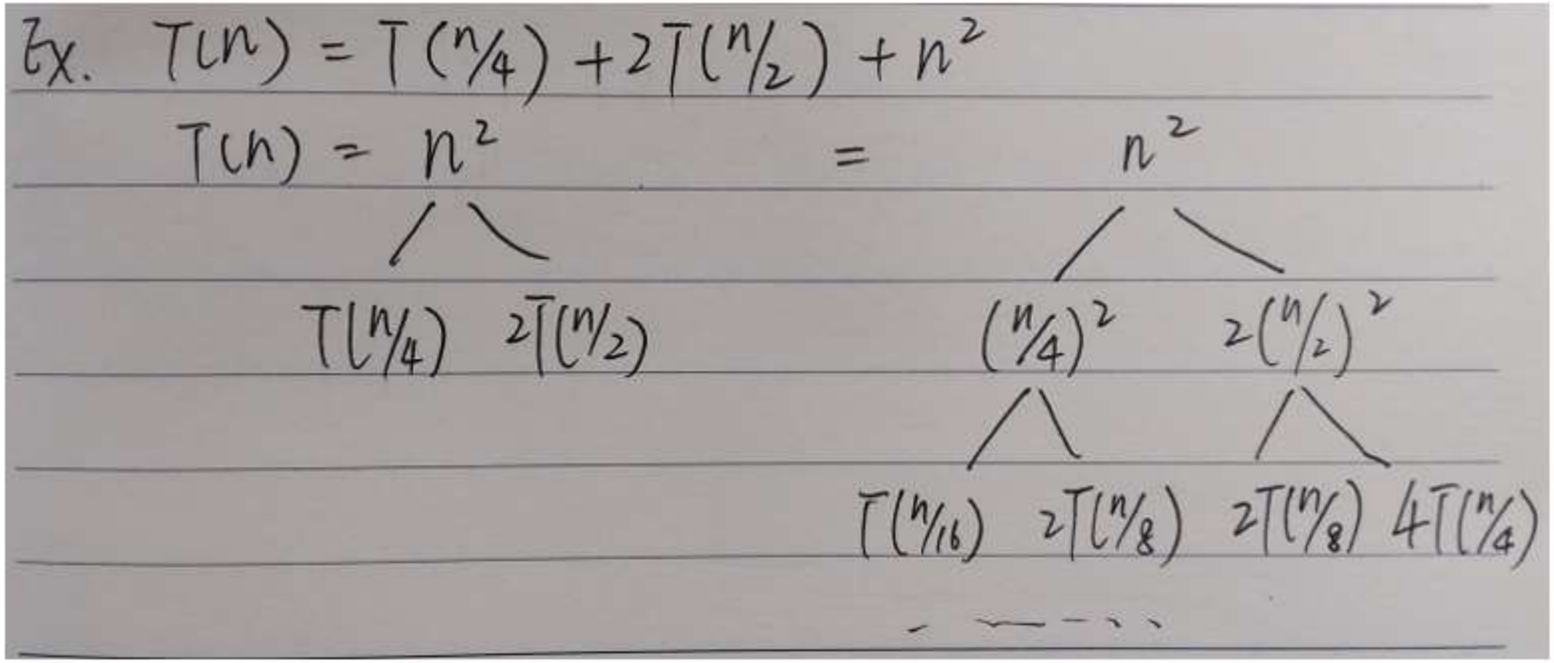


- Total cost (level-by-level)

$$\leq \left(1+\frac{5}{16}+\left(\frac{5}{16}\right)^2+\cdots+\left(\frac{5}{16}\right)^k+\cdots\right)n^2$$

$$< \left(1+\frac{1}{2}+\left(\frac{1}{2}\right)^2+\cdots+\left(\frac{1}{2}\right)^k+\cdots\right)n^2$$

$$= 2n^2 = O(n^2)$$



$$n^2$$

$$\frac{9}{16}n^2 = \left(\frac{n}{4}\right)^2 + 2\left(\frac{n}{2}\right)^2$$

$$\frac{81}{256}n^2 = \left(\frac{n}{16}\right)^2 + 2\left(\frac{n}{8}\right)^2 + 2\left(\frac{n}{8}\right)^2 + 4\left(\frac{n}{4}\right)^2 = \frac{1+2\times4+2\times4+4\times16}{256}n^2$$

Exercise 1

Section 2.2 describes a method for solving recurrence relations which is based on analyzing the recursion tree and deriving a formula for the work done at each level. Another (closely related) method is to expand out the recurrence a few times, until a pattern emerges. For instance, let's start with the familiar $T(n) = 2T(n/2) + O(n)$. Think of $O(n)$ as being $\leq cn$ for some constant c, so: $T(n) \leq 2T(n/2) + cn$. By repeatedly applying this rule, we can bound $T(n)$ in terms of $T(n/2)$, then $T(n/4)$, then $T(n/8)$, and so on, at each step getting closer to the value of $T(\cdot)$ we do know, namely $T(1) = O(1)$.

$$\begin{array}{rcllcl}
T(n) & \leq & 2T(n/2) + cn & & & \\
& \leq & 2[2T(n/4) + cn/2] + cn & = & 4T(n/4) + 2cn \\
& \leq & 4[2T(n/8) + cn/4] + 2cn & = & 8T(n/8) + 3cn \\
& \leq & 8[2T(n/16) + cn/8] + 3cn & = & 16T(n/16) + 4cn \\
& \vdots & & & &
\end{array}$$

A pattern is emerging... the general term is

$$T(n) \le 2^k T(n/2^k) + kcn.$$

Plugging in $k = \log_2 n$, we get $T(n) \le nT(1) + cn\log_2 n = O(n \log n)$.

(a) Do the same thing for the recurrence $T(n) = 3T(n/2) + O(n)$. What is the general kth term in this case? And what value of k should be plugged in to get the answer?

(b) Now try the recurrence $T(n) = T(n-1) + O(1)$, a case which is not covered by the master theorem. Can you solve this too?

a)

$$T(n) \leq 3T\left(\frac{n}{2}\right) + cn \leq \ldots \leq 3^k T\left(\frac{n}{2^k}\right) + cn \sum_{i=0}^{k-1} \left(\frac{3}{2}\right)^i =$$

$$= 3^k T\left(\frac{n}{2^k}\right) + 2cn\left(\left(\frac{3}{2}\right)^k - 1\right)$$

For $k = \log_2 n$, $T(\frac{n}{2^k}) = T(1) = d = O(1)$. Then:

$$T(n) = dn^{log_2 3} + 2cn\left(\frac{n^{log_2 3}}{n} - 1\right) = \Theta(n^{log_2 3})$$

as predicted by the Master theorem.

b) $T(n) \leq T(n-1) + c \leq \ldots \leq T(n-k) + kc$. For $k = n$, $T(n) = T(0) + nc = \Theta(n)$.

$$T(n) \leq 3[3T(n/4) + cn/2] + cn \leq 3^2 T(n/4) + \frac{3}{2}cn + cn$$

$$\leq 3^2[3T(n/8) + cn/4] + \frac{3}{2}cn + cn$$

$$\leq 3^3 T(n/8) + [\left(\frac{3}{2}\right)^2 + \frac{3}{2} + 1]cn$$

Exercise 2

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

a) This is a case of the Master theorem with $a = 5, b = 2, d = 1$. As $a > b^d$, the running time is $O(n^{\log_b a}) = O(n^{\log_2 5}) = O(n^{2.33})$.

b) $T(n) = 2T(n-1) + C$, for some constant C. $T(n)$ can then be expanded to $C\sum_{i=0}^{n-1} 2^i + 2^n T(0) = O(2^n)$.

c) This is a case of the Master theorem with $a = 9, b = 3, d = 2$. As $a = b^d$, the running time is

$O(n^d \log_3 n) = O(n^2 \log_3 n)$

$$T(n) = 2T(n-1) + C = 2[2T(n-2) + C] + C$$

$$= 2^2 T(n-2) + (2+1)C = 2^2[2T(n-3) + C] + (2+1)C$$

$$= 2^3 T(n-3) + (2^2 + 2 + 1)C$$

$$= 2^k T(n-k) + C\sum_{i=0}^{k-1} 2^i$$

Exercise 3

Solve the following recurrence relations and give a Θ bound for each of them.

(a) $T(n) = 2T(n/3) + 1$

(b) $T(n) = 5T(n/4) + n$

(c) $T(n) = 7T(n/7) + n$

(d) $T(n) = 9T(n/3) + n^2$

(e) $T(n) = 8T(n/2) + n^3$

(f) $T(n) = 49T(n/25) + n^{3/2} \log n$

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

(h) $T(n) = T(n-1) + n^c$, where $c \geq 1$ is a constant

(i) $T(n) = T(n-1) + c^n$, where $c > 1$ is some constant

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

(k) $T(n) = T(\sqrt{n}) + 1$

Exercise 3

d) $T(n) = 9T(n/3) + n^2 = \Theta(n^2 \log_3 n)$ by the Master theorem.

e) $T(n) = 8T(n/2) + n^3 = \Theta(n^3 \log_2 n)$ by the Master theorem.

f) $T(n) = 49T(n/25) + n^{3/2}\log n = \Theta(n^{3/2}\log n)$. Apply the same reasoning of the proof of the Master Theorem. The contribution of level i of the recursion is

$$\left(\frac{49}{25^{3/2}}\right)^i n^{3/2} \log\left(\frac{n}{25^{3/2}}\right) = \left(\frac{49}{125}\right)^i O(n^{3/2}\log n)$$

Because the corresponding geometric series is dominated by the contribution of the first level, we obtain $T(n) = O(n^{3/2}\log n)$. But, $T(n)$ is clearly $\Omega(n^{3/2}\log n)$. Hence, $T(n) = \Theta(n^{3/2}\log n)$.

g) $T(n) = T(n-1) + 2 = \Theta(n)$.

$$49^i \times \left(\frac{n}{25^i}\right)^{\frac{3}{2}} \log\frac{n}{25^i} = n^{\frac{3}{2}} \times \left(\frac{49}{125}\right)^i \times (\log n - i\log 25) \le n^{\frac{3}{2}} \times \left(\frac{49}{125}\right)^i \times \log n$$

$$i \le \log_{25} n \Rightarrow i\log 25 \le \log n$$

$$h) T(n) = T(n-1) + n^c = T(n-2) + (n-1)^c + n^c$$

$$= T(n-3) + (n-2)^c + (n-1)^c + n^c = \sum_{i=1}^{n} i^c + T(0) = \Theta(n^{c+1})$$

Exercise

- Solving the following recurrence relations and give a θ bound for each of them

(a) $T(n) = 2T(n/3) + 1$

(b) $T(n) = 5T(n/4) + n$

(c) $T(n) = 7T(n/7) + n$

(i) $T(n) = T(n-1) + c^n$, where $c > 1$ is some constant

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

(k) $T(n) = T(\sqrt{n}) + 1$

a) $T(n) = 2T(n/3) + 1 = \Theta(n^{\log_3 2})$ by the Master theorem.

b) $T(n) = 5T(n/4) + n = \Theta(n^{\log_4 5})$ by the Master theorem.

c) $T(n) = 7T(n/7) + n = \Theta(n \log_7 n)$ by the Master theorem.

$$i) T(n) = T(n-1) + c^n = T(n-2) + c^{n-1} + c^n$$

$$= T(n-3) + c^{n-2} + c^{n-1} + c^n = \sum_{i=1}^{n} c^i + T(0) = \Theta(c^n)$$

$$j) T(n) = 2T(n-1) + 1 = 2[2T(n-2) + 1] + 1$$

$$= 2^2 T(n-2) + (2+1) = 2^2[2T(n-3) + 1] + (2+1)$$

$$= 2^3 T(n-3) + (2^2 + 2 + 1) = 2^k T(n-k) + \sum_{i=0}^{k-1} 2^i = 2^n T(0) + \sum_{i=0}^{n-1} 2^i = \Theta(2^n)$$

$$k) T(n) = [T(\sqrt{\sqrt{n}}) + 1] + 1 = [T(\sqrt{\sqrt{\sqrt{n}}}) + 1] + 2$$

$$= k + T(b) \quad s.t. \quad b^{\overbrace{2*2...*2}^{k}} = n$$

$$\Rightarrow b^{2^k} = n \Rightarrow k = \log\log_b n \Rightarrow T(n) = O(\log\log n)$$