Analysis Of Algorithms

Fri 01/25

Problem of sorting

Input: Sequence of numbers $\langle a_1, a_2, \ldots, a_n \rangle$

Output: A permutation $\langle a'_1, a'_2, \ldots a'_n \rangle$ rearrangement of three numbers such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Insertion sort:

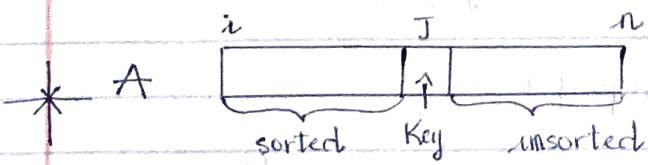
★ Index starts with 1 ★

A.length

```
Insertion_Sort (A, n)                    Execution time    # of time
1. for j = 2 to n                        c1                n
2.     key = A[j]                        c2                n-1
3.     // insert A[j] into sorted A[1..j-1]   c3 = 0
4.     i = j-1                           c4                n-1
5.     while i > 0 and A[i] > key        c5                Σ_{j=2}^{n} t_j
6.     {  A[i+1] = A[i]                  c6
7.        i = i-1                        c7
       }
8.     A[i+1] = key                      c8                n-1
```

Example:

①	②	③	④	⑤	⑥
8	②	4	9	3	6
2	8	④	9	3	6
2	4	8	⑨	3	6



a. $A[i \ldots j-1]$: sorted

b. $A[j]$: key

c. $A[j+1 \ldots n]$: unsorted

✶ Loop invariants: At the start of each iteration of for loop $A[1..j-1]$ consists the elements originally in $A[1..j-1]$, but in sorted order

Initialization:

show loop invariant holds before first iteration

Maintenance:

loop invariant holds ~~from~~ for every ~~information~~ iteration

Termination:

examine what happens when the loop ends

✶ Running time:

Assume:

c_i: i^{th} line takes c_i

t_j: # of times that while loop test is executed for that value of j

Mon 01/25 line 5: # of execution $\sum_{j=2}^{n} t_j$

line 6: $\sum_{j=2}^{n} (t_j - 1)$

line 7: same as line 6 $\sum_{j=2}^{n} (t_j - 1)$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Best case: $A[i] > key$ always false

$t_j = 1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} 1 + c_6 \cdot 0 + c_7 \cdot 0 + c_8(n-1)$$

$$\sum_{j=2}^{n} 1 = \underbrace{1 + 1 + \dots + 1}_{n-1} = n - 1$$

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$T(n) = an + b$ (a, b: constants)

↳ linear function

Worst case: reversed order

$t_j = j$ ($A[i] > key$ is always true, depends on i: $j-1 \sim 0$)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} j + c_6 \sum_{j=2}^{n} (j-1) + c_7 \sum_{j=2}^{n} (j-1) + c_8(n-1)$$

$$\sum_{j=2}^{n} j = 2 + 3 + \dots + n = \sum_{j=1}^{n} j - 1$$

$$= \frac{n(n+1)}{2} - 1 = \frac{1}{2}(n^2 + n - 2)$$

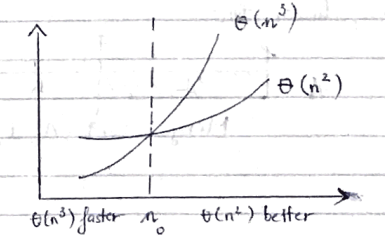
$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2} = \frac{1}{2}(n^2 - n)$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \times \frac{1}{2}(n^2 + n - 2) + c_6 \times \frac{1}{2}(n^2 - n) + c_7 \times \frac{1}{2}(n^2 - n) + c_8(n-1)$$

$$= \frac{1}{2}(c_5 + c_6 + c_7)n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$T(n) = an^2 + bn + c$ (a, b, c: constants)

✶

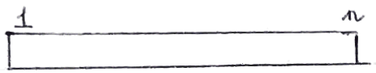


① asymptotically: $\Theta(n^3)$ algorithm is slower ($n \to \infty$)

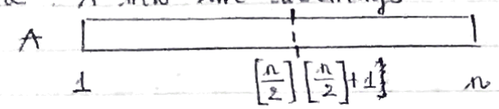
② before n_0: $\Theta(n^3)$ algorithm is faster than $\Theta(n^2)$

Ved 01/30

Merge Sort



- Divide : A into two subarrays



- Conquer : base case : if $n = 1$, done

general case { recursively sort $A[1 \ldots \lfloor \frac{n}{2} \rfloor]$
recursively sort $A[\lfloor \frac{n}{2} \rfloor + 1 \ldots n]$

- Combine : Merge $A[1 \ldots \lfloor \frac{n}{2} \rfloor]$ and $A[\lfloor \frac{n}{2} \rfloor + 1 \ldots n]$

Example :

2	7	13	20	∞

L

1	9	11	12	∞

R

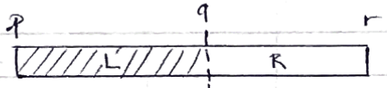
Compare 1 & 2 → 1
Compare 2 & 9 → 2
Compare 7 & 9 → 7

A

1	2	7	9	11	12	13	20	

$\theta(n)$ order of growth, linear time

Algorithm : $A[p \ldots r]$

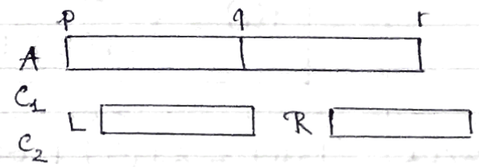


$T(n) \Leftarrow$ Merge - Sort (A, p, r)

if $(p < r)$
  $q = \lfloor \frac{p+r}{2} \rfloor$
$T(\frac{n}{2})$ Merge - Sort (A, p, q)
$T(\frac{n}{2})$ Merge - Sort $(A, q+1, r)$
Merge (A, p, q, r)

Merge :

+ Input : A, p, q, r $(p \leq q < r)$
  $A[p \ldots q]$ sorted subarray,
  $A[q+1 \ldots r]$ sorted subarray
+ Output : Two sub-arrays are merged into a single sorted subarray $A[p \ldots r]$



MERGE (A, p, q, r)

$\theta(1)$	1. $n_1 = q - p + 1$	C_1	1
	2. $n_2 = r - q$	C_2	1
	3. Let $L[1 \ldots n_1 + 1]$ and $R[1 \ldots n_2 + 1]$ be new arrays	C_3	1
$\theta(n)$	4. for $i = 1$ to n_1	C_4	$n_1 + 1$
	5. $L[i] = A[p + i - 1]$	C_5	n_1
	6. for $j = 1$ to n_2	C_6	$n_2 + 1$
	7. $R[j] = A[q + j]$	C_7	n_2
$\theta(1)$	8. $L[n_1 + 1] = \infty$	C_8	1
	9. $R[n_2 + 1] = \infty$	C_9	1
$\theta(1)$	10. $i = 1$	C_{10}	1
	11. $j = 1$	C_{11}	1
$\theta(n)$	12. for $k = p$ to r	C_{12}	$r - p + 2$
	13. if $L[i] \leq R[j]$	C_{13}	$r - p + 1$
	14. $A[k] = L[i]$	C_{14}	$r - p + 1$
	15. $i = i + 1$	C_{15}	$r - p + 1$
	16. else $A[k] = R[j]$	C_{16}	$r - p + 1$
	17. $j = j + 1$	C_{17}	$r - p + 1$

$T(n)$ is used to represent the running time for merge

$T(n)_{\text{Merge}} = \theta(1) + \theta(n) + \theta(1) + \theta(1) + \theta(n)$
$= \theta(n)$

$\theta(n)$

$T(n)_{\text{merge-sort}} = D(n) + \underbrace{\text{Time for recursion}}_{T(\frac{n}{2}) + T(\frac{n}{2})} + C(n)$
($D(n)$ ↑ Divide; $C(n)$ ↑ Combine)
$= \theta(1) + T(\frac{n}{2}) + T(\frac{n}{2}) + \theta(n)$
$=$

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

$\boxed{T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)}$ → Running time for Merge-Sort

Recurrence: an equation or inequality that describes a function of its value on smaller inputs

$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + \theta(n) & \text{if } n > 1 \end{cases}$$

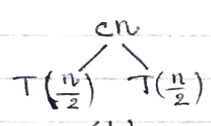
Fri 02/01

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

$= 2T\left(\frac{n}{2}\right) + cn$ (c: positive constant)

$T(n)$

(a)

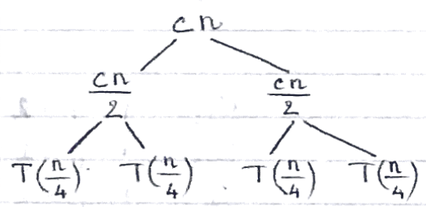


(b)

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

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

$T\left(\frac{n}{4}\right)$ $T\left(\frac{n}{4}\right)$

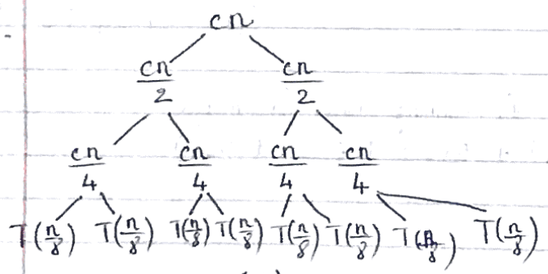


(c)

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + c\frac{n}{4}$

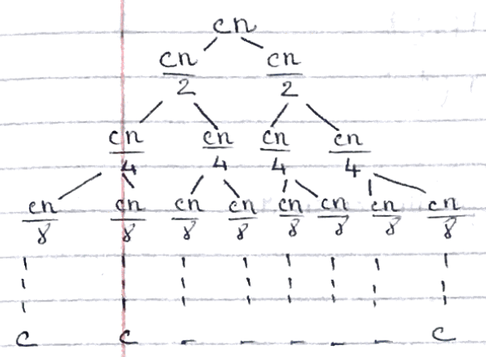
$T\left(\frac{n}{4}\right)$ $\frac{cn}{4}$

$T\left(\frac{n}{8}\right)$ $T\left(\frac{n}{8}\right)$



(d)

⇒ Continue expand if the leaves until the problem size get down to 1



Level	Each node in terms of bcd
... cn	# 1	$\frac{cn}{2^0}$
... cn	# 2	$\frac{cn}{2^1}$
... cn	# 3	$\frac{cn}{2^2}$
... cn	# 4	$\frac{cn}{2^3}$
... cn	# i	$\frac{cn}{2^{i-1}}$

(e)

$\frac{cn}{2^{i-1}} = c$

$n = 2^{i-1}$

$\log_2 n = \log_2(2^{i-1})$

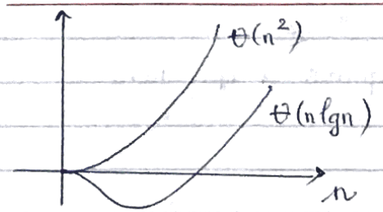
$\log_2 n = i - 1 \Rightarrow i = \log_2 n + 1$

$i = \lg n + 1$

$T(n) = cn(\lg n + 1)$

$T(n) = cn\lg n + cn = \theta(n\lg n)$

Insertion sort: $\theta(n^2)$ VS. Merge Sort $\theta(n\lg n)$



Asymptotically:
Merge sort is faster than Insertion sort

Algorithm Unknown (n) # times of execution time

1	for i = 1 to n	c_1	n + 1
2	print "..."	c_2	n
3.	Alg_unknown($\frac{n}{2}$)	$T(\frac{n}{2})$	1
4.	Alg_unknown($\frac{n}{2}$)	$T(\frac{n}{2})$	1

$T(n) = c_1(n+1) + c_2 n + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right)$

$T(n) = 2T\left(\frac{n}{2}\right) + (c_1 + c_2)n + c_1$

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

Mon 02/04 a/ What if we change line 1 to "for i = 1 to n^2"

$T(n) = c_1(n^2+1) + c_2 n^2 + 2T(\frac{n}{2})$

$= (c_1 + c_2)n^2 + c_1 + 2T(\frac{n}{2})$

$= 2T(\frac{n}{2}) + \theta(n^2)$

$= 2T(\frac{n}{2}) + cn^2$ (c: positive constant)

$= \theta(n^2)$

* Change line 1 to for i = 1 to 20

$T(n) = 21c_1 + 20c_2 + 2T(\frac{n}{2})$

$= 2T(\frac{n}{2}) + c$ (c: positive constant)

$= \theta(\lg n)$

* Summations:

- A loop of size n, the i^{th} step ($i \leq n$) requires 1 operation:
  The total number of operations:
  $\sum_{i=1}^{n} 1 = n$

  for i = 1 to n
    print " "

- A loop of size n, i^{th} step requires i operation
  The total # of operations:
  $\sum_{i=1}^{n} i = 1 + 2 + \dots + n$
  $= \frac{n(n+1)}{2}$

- A loop of size n, the i^{th} step requires 2^i operations
  The total # of operations:
  $\sum_{i=1}^{n} 2^i = 2^1 + 2^2 + \dots + 2^n$
  $= 2^{n+1} - 2$

$\sum_{i=0}^{n} 2^i = 2^0 + 2^1 + \dots + 2^n$
$= 2^{n+1} - 1$

① $F(n) = \sum_{i=0}^{n} 2^i = 1 + 2 + 4 + \dots + 2^{n-1} + 2^n$

② $2F(n) = 2 + 4 + 8 + \dots + 2^n + 2^{n+1}$

② - ① $F(n) = 2^{n+1} - 1$

More generally: if $a_n = c \cdot a_{n-1}$ where $c \neq 1$, a constant

$a_1 + a_2 + \dots + a_n$
$= a_1\left(\frac{c^n - 1}{c - 1}\right)$

if $0 < c < 1$, $\sum_{i=1}^{\infty} a_i = \frac{a_i}{1-c}$

Chapter 3:

O-notation (big-Oh notation)

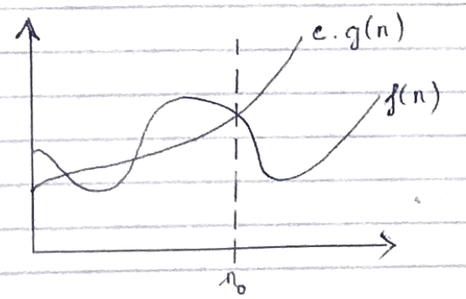
Formal definition:

$O(g(n)) = \{ f(n)$: there are constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0 \}$

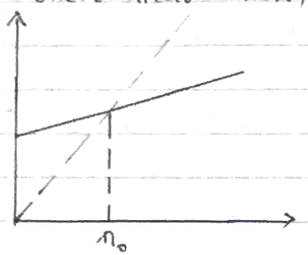
This means: $f(n)$ is bounded by $0 \sim c \cdot g(n)$
$g(n)$ is an asymtotically upper bound for $f(n)$

$f(n) \in O(g(n))$

$\boxed{f(n) = O(g(n))}$



Ex1 : $T(n) = 10000 + 10n$, try to find $g(n)$ such that $T(n) \in O(g(n))$



$T(n) = 10000 + 10n$

Slope : $20 = c$ Let $c = 20$

$f(n) \leq c.g(n)$

$c = 20$

$10000 + 10n \leq 20n$

$10000 \leq 10n$

$\underbrace{1000}_{n_0} \leq n$

$\therefore$ $c = 20$, $n_0 = 1000$

$n \geq n_0 = 1000$

$10000 + 10n \leq 20n$ $(c = 20, n_0 = 1000)$

$g(n) = n$

$T(n) = 10000 + 10n \in O(g(n)) = O(n)$

$10000 + 10n \in O(n)$

$10000 + 10n = O(n)$

Ex2 : $2n^2 = O(n^3)$ true or false ?

$2n^2 \leq cn^3$

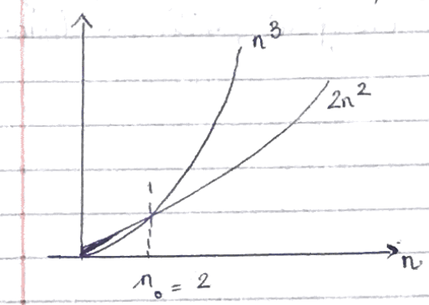
Let $c = 1$

$2n^2 \leq n^3$

$\underbrace{2}_{n_0} \leq n$

$c = 1$, $n_0 = 2$ $2n^2 \in O(n^3)$

$2n^2 = O(n^3)$



Ex3 : $1000n^2 + 1000n = O(n^2)$

$1000n^2 + 1000n \leq cn^2$

$1000n^2 + 1000n \leq 2000n^2$

$1000n \leq 1000n^2$

$n \leq n^2$

$\underbrace{1}_{n_0} \leq n$

$n_0 = 1$

When $c = 2000$, $n_0 = 1$ for any $n \geq n_0$ $(n_0 = 1)$:

$1000n^2 + 1000n \leq 2000n^2$

$1000n^2 + 1000n = O(n^2)$

Ex4 : $n^{1.99} = O(n^2)$

Let $c = 1$

$n^{1.99} \leq n^2$

$n_0 = 1$ (any positive n_0)

Ex5 : $n = O(n^3)$

Let $c = 1$

$n \leq n^3$

$1 \leq n^2$

$n_0 = 1$

Algorithm $T(n)$

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

can we say that this algorithm is slow (True)

* O-notation merely claims that some constant multiple of $g(n)$ is an asymptotical upper bound on $f(n)$, with no claim how tight an upper bound it is

Ex6 : $\frac{n^2}{\lg\lg\lg n} = O(n^2)$

Let $c = 1$

$\frac{n^2}{\lg\lg\lg n} \leq n^2$

$1 \leq \lg\lg\lg n$

$16 \leq n$ $(2^0 \leq n)$

$c = 1, n_0 = 16$

02/08

* Logarithms:

1. $\log_b(ac) = \log_b a + \log_b c$
2. $\log_b(\frac{a}{c}) = \log_b a - \log_b c$
3. $\log_b a^c = c \log_b a$
4. $\log_b a = \frac{\log_c a}{\log_c b}$
5. $b^{\log_c a} = a^{\log_c b}$

$\lg 2^n = n \lg 2 = n$

$2^{\lg n} = n^{\lg 2} = n^1 = n$

$2^{2\lg n} = 2^{\lg n^2} = n^2$

$\lg \frac{1}{n} = -\lg n$

Ex7 : claim:

if $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n^1 + a_0$

then $f(n) = O(n^k)$

Proof:

Choose $n_0 = 1$ and

$c = |a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|$

need to show $\forall n \geq 1 \quad f(n) \leq c \cdot n^k$

$f(n) \leq |a_k| n^k + |a_{k-1}| n^{k-1} + \dots + |a_1| n + |a_0|$

$\leq |a_k| n^k + |a_{k-1}| n^k + \dots + |a_1| n^k + |a_0| n^k$

$= n^k (|a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|)$

$= c \cdot n^k$

$O(n^2)$ vs. $O(cn^2)$

Ex8 : 2^{10n} is not $O(2^n)$

Proof by contradiction:

if $2^{10n} = O(2^n)$

$2^{10n} \leq c \cdot 2^n$ (c: positive const)

$2^{9n} \leq c$

When $n \to \infty$, it's not possible to find such c

$(a^m)^n = a^{mn}$

$a^m \cdot a^n = a^{m+n}$

$a^m / a^n = a^{m-n}$

Ex9 : $n^2 \in O(n)$ $(n^2 = O(n))$

Proof: Let $c = n$ then

$n^2 \leq cn = n \cdot n = n^2$

Problem: c has to be a constant

Ex10 : $e^{3n} = O(e^n)$ because const factor doesn't matter.

WRONG: e^{3n} is larger than e^n by a factor of e^n

Ex11 : Claim $10^n \in O(2^n)$ true or false

False. 10^n is larger than 2^n by a factor of 5^n

Function	Common name
$O(1)$	constant
$\subset O(\log n)$	logarithmic
$\subset O(\log^2 n)$	log - squared
$\subset O(\sqrt{n})$	root n
$\subset O(n)$	linear
$\subset O(n \log n)$	
$\subset O(n^2)$	quadratic
$\subset O(n^3)$	cubic
$\subset O(n^4)$	
⋮	
$\subset O(2^n)$	exponential
$\subset O(e^n)$	exponential
$\subset O(n!)$	
$\subset O(n^n)$	

useless

* Polynomials grow more slowly than exponentials

Logarithms grow more slowly than polynomials.

Ω - notation

$\Omega(g(n)) = \{ f(n)$ there exists positive constants c and n_0, such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0 \}$

$g(n)$ is asymptotic lower bound for $f(n)$

$f(n) = \Omega(g(n))$

$g(n) = O(f(n))$

Ω is reverse of big O if $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$

① $2n = \Omega(n)$; $n = O(2n)$

$2n \geq c \cdot n$

② $n^2 = \Omega(n)$; $n = O(n^2)$

③ $n^2 = \Omega(3n^2 + n\log n)$; $3n^2 + n\lg n = O(n^2)$

④ $n^2 = \Omega(n^2)$

$\sqrt{n} = \Omega(\lg n)$

Let $c = 1$

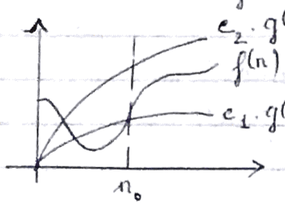
$\sqrt{n} \geq \lg n$

$n_0 = 16$

Mon 02/11

$\Theta(g(n)) = \{ f(n)$: there exists positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0 \}$

meaning: $O(g(n))$ is the set of all functions are in both $O(g(n))$ and $\Omega(g(n))$



$g(n)$ is an asymptotically tight bound for f

c_1 and c_2 are different constants

Example: $\frac{n^2}{2} - 2n = \Theta(n^2)$

$c_1 \cdot n^2 \leq \frac{n^2}{2} - 2n \leq c_2 \cdot n^2$ for all $n \geq n_0$

$c_1 \leq \frac{1}{2} - \frac{2}{n} \leq c_2$

$\frac{1}{2} - \frac{2}{n} \leq c_2$

$(c_2 \geq \frac{1}{2}) \Rightarrow c_2 = \frac{1}{2}$ when $n_0 \geq 1$

$c_1 \leq \frac{1}{2} - \frac{2}{n}$

$(c_1 \leq \frac{1}{4}) \Rightarrow c_1 = \frac{1}{4}$ when $n \geq 8$

$c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$, $n_0 = 8$

★ Theorem: $f(n) = \Theta(g(n))$ if only if (iff) $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Symetric property of Θ

If $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$

① $n^3 = \Theta(3n^3 - n^2) \Rightarrow 3n^3 - n^2 = \Theta(n^3)$

② $n^8 \neq \Theta(n) \rightarrow n \neq \Theta(n^3)$

Example: * If an algorithm is $O(n^8)$, can we say that "it is slow"?

⇒ No, we can't. $O(n^8)$ is not tight bound

* If an algorithm is $\Theta(n^8)$, can we say that "it is slow"?

⇒ Yes, we can. $\Theta(n^8)$ is a tight bound

$< O(n^8)$, $\Omega(n^8)$

Not every function has something in Θ

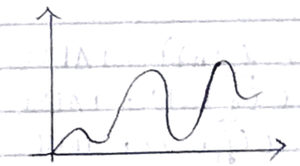
$f(n) = n(1 + \sin n)$

$-1 \leq \sin 1 \leq 1$

$f(n) = O(n)$

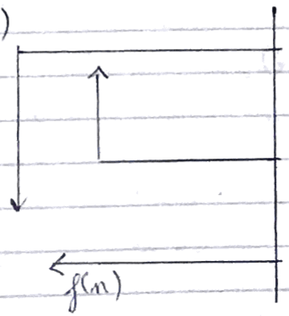
$= \Omega(0)$

not in $\Theta(n)$ or $\Theta(0)$



★ * Arrow diagram

$f(n) = O(g_3(n))$



① Draw a vertical line

② List all $g(n)$ in the order of growth, most fastest on the top, least fastest are on the bottom

$g_1(n) < g_2(n) < g_3(n)$ → $f_3(n) = O(g_3(n))$

③ If O-notation, draw a ↓ for $f(n)$

If Ω-notation, draw a ↑ for $f(n)$

$f_2(n) = \Omega(g_2(n))$

If Θ-notation, draw a ← for $f(n)$

$f_1(n) = \Theta(g_1(n))$

Ex: We have 3 functions

$(\lg n)^2$, n^{-2} , n^2

$\lg^2 n$, $\frac{1}{n^2}$, n^2

and 4 notations:

a/ $f_1(n) = \Omega(\lg^2 n)$

b/ $f_2(n) = O(n^2 - \frac{1}{n}) = n^2$

c/ $f_3(n) = \Omega(\frac{1}{n^2})$

d/ $f_4(n) = \Theta(n^{-2})$

For each statement below, state if it is True or False otherwise (always True)

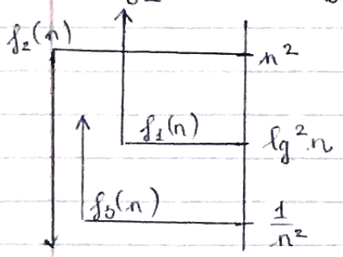
Solutions:

a/ $f_1(n) \in \Omega(f_3(n))$ FALSE

b/ $f_2(n) = \Omega(f_3(n))$: FALSE

c/ $f_1(n) = O(f_2(n))$: FALSE

d/ $f_4(n) = O(f_1(n))$: TRUE



O - notation and ω - notation (slides)

Fri, 02/15

Maximum - Subarray Problem:

Input: An array $A[1 \ldots n]$ of numbers

assume: some of numbers are negative

Output: Indices i and j such that $A[i \ldots j]$ has the greatest sum of any nonempty contiguous subarray of A, along with the sum of the values in $A[i \ldots j]$

Solutions: ① Write - force solution

$\Theta(n^2)$

② Divide and Conquer

Divide: A | A_1 | A_2 |

Conquer: 1. Subarray in A_1 | | A_1 |

m_1

2. Subarray in A_2 A_2

m_2

3. | A_1 | A_2 |

m_3

Find - Maximum - Subarray (A, low, high)

```
base case divide
1.  if high == low
2.      return (low, high, A[low])      // base case: only one
3.  else mid = [(low + high)/2]
m1  4.      (left-low, left-high, left-sum) = FIND-MAXIMUM-SUBARRAY(A, low, mid
m2  5.      (right-low, right-high, right-sum) = FIND-MAXIMUM-SUBARRAY(A, mid+1, high
m3  6.      (cross-low, cross-high, cross-sum) = FIND-MAX-CROSSING-SUBARRAY
                                                (A, low, mid, high)
    7.      if left-sum ≥ right-sum and left-sum ≥ cross-sum
    8.          return (left-low, left-high, left-sum)
    9.      else if right-sum ≥ left-sum and right-sum ≥ cross-sum
    10.         return (right-low, right-high, right-sum)
    11.     else return (cross-low, cross-high, cross-sum)
```

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

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

$= \Theta(n \lg n)$

Find-Max-Crossing-Subarray (A, low, mid, high)

```
1.  left-sum = -∞
2.  sum = 0
3.  for i = mid downto low
4.      sum = sum + A[i]
5.      if sum > left-sum
6.          left-sum = sum
7.          max-left = i
8.  right-sum = -∞
9.  sum = 0
10. for j = mid + 1 to high
11.     sum = sum + A[j]
12      if sum > right-sum
13.         right-sum = sum
14.         max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

Matrix Multiplication

Input: Two $n \times n$ (square) matrices.

$A = (a_{ij})$

$B = (b_{ij})$

Output: $n \times n$ matrix $C = (c_{ij})$

where $C = A \cdot B$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

for $i, j = 1, 2, \ldots, n$

example: $n = 2$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Partition A, B, C into $n/2 \times n/2$ submatrices

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$C = A \cdot B \Rightarrow \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$C_{11} = A_{11} B_{11} + A_{12} B_{21}$
$C_{12} = A_{11} B_{12} + A_{12} B_{22}$
$C_{21} = A_{21} B_{11} + A_{22} B_{21}$
$C_{22} = A_{21} B_{12} + A_{22} B_{22}$

two $\frac{n}{2} \times \frac{n}{2}$ multiplications

add $\frac{n^2}{4}$ products Running Time $= \theta(n^{\lg n})$

* Rec - Mat - Mult (A, B, n)

$\theta(1)$ 1. let C be a new $n \times n$ matrix
$\theta(1)$ 2. if $n == 1$
$\theta(1)$ 3. $\quad c_{11} = a_{11} \cdot b_{11}$
$\theta(1)$ 4. else partition A, B and C into $n/2 \times n/2$ submatrices
5. $\quad C_{11} =$ Rec - Mat - Mult $(A_{11}, B_{11}, n/2)$ + Rec - Mat - Mult $(A_{12}, B_{21}, n/2)$
6. $\quad C_{12} =$ $(A_{11}, B_{12}, n/2)$ + $(A_{12}, B_{22}, n/2)$
7. $\quad C_{21} =$ $(A_{21}, B_{11}, n/2)$ + $(A_{22}, B_{21}, n/2)$
8. $\quad C_{22} =$ $(A_{21}, B_{12}, n/2)$ + $(A_{22}, B_{22}, n/2)$
9. return C

$T(n) = 8T(\frac{n}{2}) + \underbrace{\theta(n^2)}_{\text{sub-matrices addition}} + \theta(n)$

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

$= \theta(n^3)$

Tue 02/19

Substitution Method

1. Guess a solution
2. Use induction to find constants and show that solution works

Ex1: $T(n) = 2T(\frac{n}{2}) + \theta(n)$
① $T(n) = \theta(n \lg n)$
② Name the constants
Show the O and Ω

✓ Upperbound:
$T(n) = 2T(\frac{n}{2}) + O(n)$
$\leq 2T(n/2) + cn$ (c: positive constant)
Guess: $T(n) = O(n \lg n)$
$\leq d \cdot n \lg n$ (d: positive constant)
(assume the hypothesis holds for all $m < n$,
m can be $n/2$, $n/4$, $n/3$, ... so we prove our guess is true for n)
$T(\frac{n}{2}) \leq d \cdot \frac{n}{2} \lg \frac{n}{2}$

Substitution: $T(n) \leq 2T(\frac{n}{2}) + cn$
$\leq 2(d \cdot \frac{n}{2} \lg \frac{n}{2}) + cn$
$= dn(\lg n - \lg 2) + cn$
$= dn \lg n - dn + cn \leq dn \lg n$ (Guess) if $(-dn + cn \leq 0)$, $c \leq d$
Therefore: $T(n) = O(n \lg n)$

✓ Lowerbound:
$T(n) \geq 2T(\frac{n}{2}) + cn$ (c: positive constant)
Guess: $T(n) = \Omega(n \lg n)$
$\geq d \cdot n \lg n$ (d: positive constant)
$T(\frac{n}{2}) \geq d \cdot \frac{n}{2} \lg \frac{n}{2}$
Substitution: $T(n) \geq 2(d \frac{n}{2} \lg \frac{n}{2}) + cn$
$= \dots$
$= dn \lg n - dn + cn$
$\geq dn \lg n$ if $c \geq d$

Therefore $T(n) = \Omega(n \lg n)$

$\Rightarrow T(n) = \Theta(n \lg n)$

Ex2. $T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$

✓ Upper bound:

$T(n) \leqslant 8T(\frac{n}{2}) + cn^2$ (c: positive constant)

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

$\leqslant dn^3$ (d: positive constant)

$T(\frac{n}{2}) \leqslant d(\frac{n}{2})^3$

Substitution:

$T(n) \leqslant 8T(\frac{n}{2}) + cn^2$

$= 8(d(\frac{n}{2})^3) + cn^2$

$= dn^3 + cn^2$

$\leqslant dn^3$ if $\underbrace{cn^2 \leqslant 0}_{\text{impossible}}$

Subtract off a lower-order term:

$cn^2 - d'n^2 \leqslant 0$

Wed 02/20

Make a new Guess:

$T(n) = O(n^3) - d'n^2$

$\leqslant dn^3 - d'n^2$ (d': positive constant)

$T(\frac{n}{2}) \leqslant d(\frac{n}{2})^3 - d'(\frac{n}{2})^2$

Substitution: $T(n) \leqslant 8T(\frac{n}{2}) + cn^2$

$\leqslant 8(d(\frac{n}{2})^3 - d'(\frac{n}{2})^2) + cn^2$

$= dn^3 - 2d'n^2 + cn^2$

$= dn^3 - d'n^2 - d'n^2 + cn^2$

$\leqslant \underbrace{dn^3 - d'n^2}_{\text{new guess}}$ if $(-d'n^2 + cn^2) \leqslant 0$

$c \leqslant d'$

but not the dd one

$\therefore T(n) = O(n^3)$

✓ Lower Bound: $T(n) \geqslant 8T(\frac{n}{2}) + cn^2$ (c: some positive constant

Guess: $T(n) \geqslant dn^3$ (d: positive constant)

$T(\frac{n}{2}) \geqslant d(\frac{n}{2})^3$

Substitution: $T(n) \geqslant 8T(\frac{n}{2}) + cn^2$

$\geqslant 8(d(\frac{n}{2})^3) + cn^2$

$= dn^3 + cn^2$

$\geqslant dn^3$ if $cn^2 \geqslant 0$

$\therefore T(n) = \Omega(n^3)$

Therefore: $T(n) = \Theta(n^3)$

Ex3: $T(n) = 4T(\frac{n}{4}) + xn$

$= 4T(\frac{n}{4}) + \theta(n)$

$= 4T(\frac{n}{4}) + cn$ (c: positive constant)

✓ Upper Bound: $T(n) = O(n)$

$\leq dn$ (d: positive constant)

$T(\frac{n}{4}) \leq d(\frac{n}{4})$

Substitution: $T(n) \leq 4(d.\frac{n}{4}) + cn$

$= dn + cn$

$\leq dn$ if $cn \leq 0$ is not possible

new guess $T(n) \leq dn - d'$

$T(\frac{n}{4}) \leq d(\frac{n}{4}) - d'$

$T(n) \leq 4(d(\frac{n}{4}) - d') + cn$

$= dn - 4d' + cn$

$= dn - d' - 3d' + cn$

$\leq dn - d'$ if $(-3d' + cn) \leq 0$

$cn \leq 3d'$ impossible

$T(n) = O(n \lg n)$

Guess again: $T(n) \leq dn\lg n$ (d: positive constant)

$T(n) \leq 4(d\frac{n}{4}\lg\frac{n}{4}) + cn$

$= dn(\lg\frac{n}{4}) + cn$

$= dn(\lg n - \lg 4) + cn$

$= dn\lg n - 2dn + cn$

$\leq dn\lg n$ if $(-2dn + cn \leq 0)$

$\therefore T(n) = O(n\lg n)$ $\rightarrow c \leq 2d$

✓ Lower Bound:

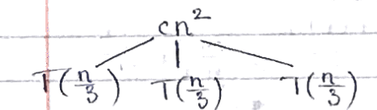
✳ Recursion Tree:

Mystery (n)	Cost	# of times of execution
1. if ($n \leq 1$)	C_1	1
2. return 1	C_2	1
3. for i = 1 to 3	C_3	4
4. for j = 1 to n^2	C_4	$3(n^2+1)$
5. print " * "	C_5	$3n^2$
6. Mystery ($\frac{n}{3}$)	$T(\frac{n}{3})$	1
7. Mystery ($\frac{n}{3}$)	$T(\frac{n}{3})$	1
8. Mystery ($\frac{n}{3}$)	$T(\frac{n}{3})$	1

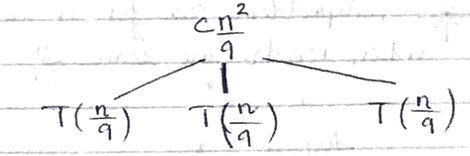
$T(n) = C_1 + C_2 + 4C_3 + 3C_4(n^2+1) + 3C_5n^2 + 3T(\frac{n}{3})$

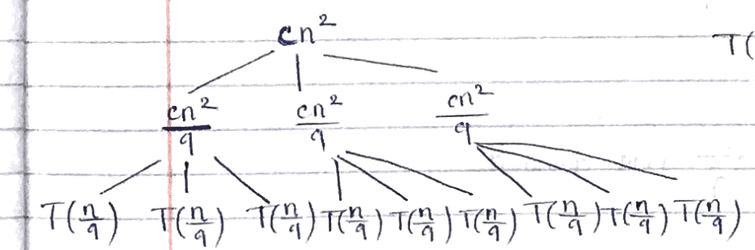
$= 3T(\frac{n}{3}) + \theta(n^2)$

$= 3T(\frac{n}{3}) + cn^2$ (c: positive)



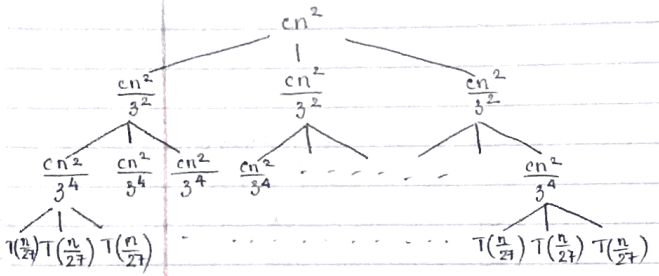
$T(\frac{n}{3}) = 3T(\frac{n}{9}) + c(\frac{n}{3})^2$

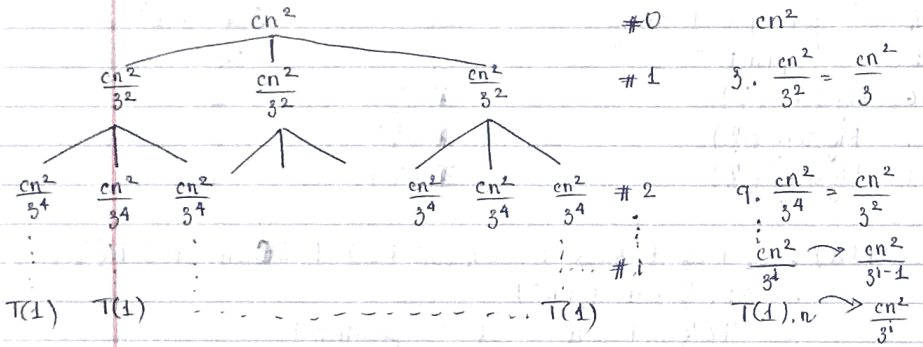




Fri, 02/22 $T(\frac{n}{9}) = 3T(\frac{n}{9\times3}) + c(\frac{n}{9})^2$

$= 3T(\frac{n}{27}) + c\frac{n^2}{9^2}$





$$T(n) = cn^2 + \frac{cn^2}{3} + \frac{cn^2}{3^2} + \dots + \underbrace{\frac{cn^2}{3^i} + T(1) \cdot n}_{\rightarrow \frac{cn^2}{3^{i-1}} + \frac{cn^2}{3^i}}$$

✶ Subproblem size for a node at depth i in $\frac{n}{3^i}$

when reach the leave $\frac{n}{3^i} = 1$

$n = 3^i$

$\log_3 n = i$

cost for each level:

level 0: cn^2 ← root dominate

1: $cn^2/3$

2: $cn^2/3^2$

⋮

$i-1$: $cn^2/3^{i-1}$

i: $cn^2/3^i$

$\left.\vphantom{\frac{1}{1}}\right\} \frac{c}{2} n^2$

$$T(n) = \sum_{i=0}^{\log_3 n} \frac{cn^2}{3^i}$$

$$\leqslant \sum_{i=0}^{\infty} \frac{cn^2}{3^i} = cn^2 \sum_{i=0}^{\infty} \frac{1}{3^i} = cn^2 \frac{1}{1-\frac{1}{3}} = \frac{3}{2} cn^2$$

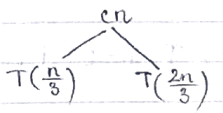
A.6

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

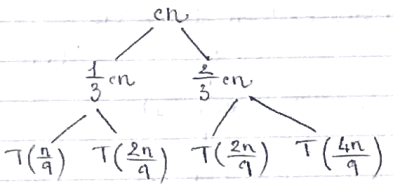
$$3T\left(\frac{n}{3}\right) + cn^2 = \frac{3}{2} cn^2 = \Theta(n^2)$$

Example 2: $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n)$

$= T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$ (c: positive constant)

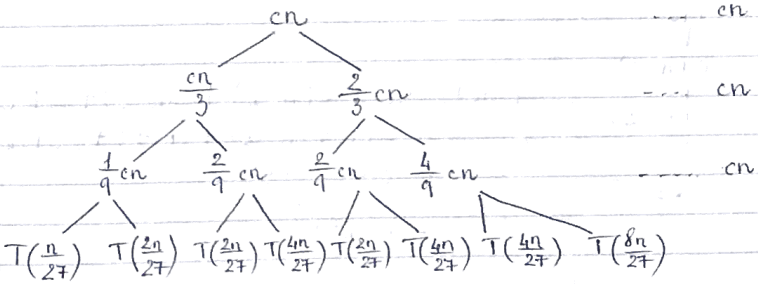


(a)



$$T\left(\frac{n}{3}\right) = T\left(\frac{n}{9}\right) + T\left(\frac{2}{3} \times \frac{n}{3}\right) + c\left(\frac{n}{3}\right)$$

$$T\left(\frac{2n}{3}\right) = T\left(\frac{2n}{9}\right) + T\left(\frac{4n}{9}\right) + \frac{2}{3} cn$$

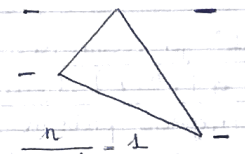


Leftmost branch peters out after $\log_3 n$ levels

$\frac{n}{3^i} = 1$

$i = \log_3 n$

Rightmost branch peters out after $\log_{\frac{3}{2}} n$ level



$\Rightarrow n = \left(\frac{3}{2}\right)^i$

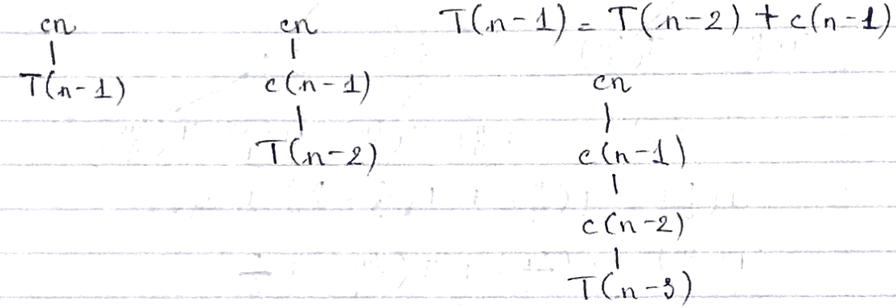
$\log_3 n \cdot cn \leq T(n) \leq \log_{\frac{3}{2}} n \cdot cn$

$c \cdot n\log_3 n \leq T(n) \leq c \cdot n\log_{\frac{3}{2}} n$

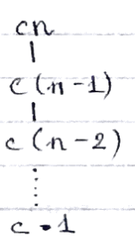
$\Omega(n\lg n) \leq T(n) \leq O(n\lg n)$

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

Example 3. $T(n) = T(n-1) + cn$



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



$T(n) = cn + c(n-1) + c(n-2) + \dots + c$

$= c(n + (n-1) + (n-2) + \dots + 1)$

$= c \cdot \frac{n(n+1)}{2}$

$= \Theta(n^2)$

Mon 02/25

Method 3: Master Method

A "cook book" for recurrences in form:

$\boxed{T(n) = aT(n/b) + f(n)}$

n : original problem size

divide into a subproblems, each subproblem n/b

make a recursive calls

$T(n/b)$: running time to solve each subproblem

$f(n)$: running time to divide & combine

- Strassen's algorithm:

$T(n) = 7T(n/2) + \Theta(n^2)$

$a = 7$, $b = 2$

- Simple D&C matrix

$T(n) = 8T(n/2) + \Theta(n^2)$

$a = 8$, $b = 2$

} $f(n) = \Theta(n^2)$

$a \geq 1$ (# of subproblems or # of recursive calls)

$b > 1$ ($< n$)

$f(n)$: asymptotically positive function ($f(n) > 0$)

$T(n) = aT(n/b) + f(n)$

Compare $n^{\log_b a}$ vs. $f(n)$

Case 1: $f(n)$ is polynomially smaller than $n^{\log_b a}$

$\boxed{T(n) = \Theta(n^{\log_b a})}$

$f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$

Case 2: $f(n) = \Theta(n^{\log_b a})$

$f(n)$ is polynomially equal to $n^{\log_b a}$

$\boxed{T(n) = \Theta(n^{\log_b a} \cdot \lg n)}$

Case 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$

$f(n)$ is polynomially greater than $n^{\log_b a}$

$\boxed{T(n) = \Theta(f(n))}$

Master Method Examples:

Merge Sort,

eg 1: $T(n) = 2T(n/2) + \Theta(n)$

$a = 2$, $b = 2$

$\log_b a = \log_2 2 = 1$

$n^{\log_b a}$ vs. $f(n)$

n^1 vs. n

Case 2 in Master Method
$T(n) = \Theta(n \lg n)$

e.g.2 : $T(n) = 7T(n/2) + \Theta(n^2)$

$a = 7, \; b = 2$

$\log_b a = \log_2 7$

$n^{\log_2 7}$ vs. n^2

Case 1 in Master Method
$T(n) = \Theta(n^{\log_2 7})$

e.g.3 : $T(n) = 8T(n/2) + \Theta(n^2)$

$a = 8, \; b = 2$

$\log_b a = \log_2 8$

$n^{\log_2 8}$ vs. n^2

Case 1 in Master Method
$T(n) = \Theta(n^{\log_2 8}) = \Theta(n^3)$

e.g.4 : $T(n) = 3T(n/3) + 2^{\log_4 n}$

$a = 3, \; b = 3$

$\log_b a = \log_3 3 = 1$

n^1 vs. $2^{\log_4 n} \rightarrow n^{\log_4 2} = n^{1/2}$

Case 1 in Master Method
$T(n) = \Theta(n)$

e.g.5 : $T(n) = 7T(n/3) + n^2$

$a = 7, \; b = 3$

$\log_b a = \log_3 7$

$n^{\log_3 7}$ vs. n^2

Case 3 in Master Method
$T(n) = \Theta(n^2)$

e.g.6 : $T(n) = 0.5T(n/2) + 1/n$?

$a = 0.5, \; b = 2$ $\quad a = 0.5 < 1$

$\log_b a = \log_2 (0.5) = -1$ Can't use Master Method

n^{-1} vs $1/n$

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

If $n \rightarrow \infty$, $f(n) \rightarrow 0$

Can't use MM because $f(n)$ has to be greater than 0

e.g.7 : $T(n) = 64T(n/8) - n^2$

$f(n) = -n^2$ always negative

Master Method does n't apply

e.g.8 : $T(n) = 3T(n/3) + cn^2$

Compare n^1 vs. cn^2

Case 3 : $T(n) = \Theta(cn^2)$

e.g.9 : $T(n) = 4T(n/2) + n^2$

Case 2 : $T(n) = \Theta(n^2 \lg n)$

e.g.10 : $T(n) = 27T(n/3) + \Theta(n^3/\lg n)$

$a = 27, \; b = 3$

$\log_b a = \log_3 27 = 3$

Compare n^3 vs. $n^3/\lg n$

$f(n) \neq O(n^{\log_b a - \varepsilon})$ with $\varepsilon > 0$

Master Method can not apply

non-polynomical difference between $f(n)$ and $n^{\log_b a}$

e.g 11 : $T(n) = \lg n \cdot T(\frac{n}{2}) + n^3$

$a = \lg n$ not a constant

Can't use Master Method

e.g.12 : $T(n) = 2T(\frac{n}{2}) + n \lg n$

$a = 2, \; b = 2$

Compare n^1 vs $n \lg n$

no polynomial difference between n and $n \lg n$

Can't use MM, although $n \lg n$ asymptotically grow faster than n

e.g 13 : $T(n) = 16T(\frac{n}{4}) + n$

$a = 16, \; b = 4 \rightarrow \log_b a = \log_4 16 = 2$

Compare n^2 vs n

Case 1 in MM

$T(n) = \Theta(n^2)$

Wed 02/27

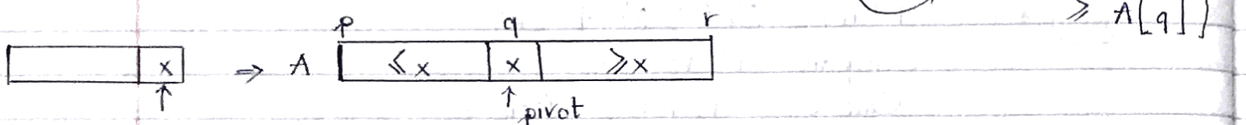
Quick Sort

- Quick Sort is 2-3 times faster than Merge Sort and Heap Sort
- Use Divide and Conquer

✳ QS to sort subarray $A[p \dots r]$

Divide: partition $A[p \dots r]$ into two subarrays

$A[p \dots q-1]$ (each element is $\leq A[q]$), $A[q+1 \dots r]$ (each element is $\geq A[q]$)



(Subarray can be empty)

Conquer: Call Quicksort for $A[p \dots q-1]$, $A[q+1 \dots r]$ (recursively)

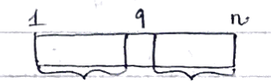
Combine: no work is needed

✳ QUICK_SORT (A, p, r)

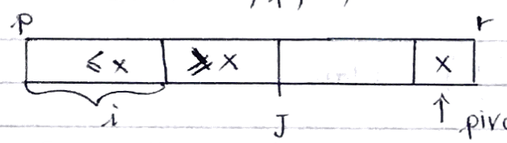
q_0 = PARTITION (A, p, r) $\quad \Theta(n)$

QUICK_SORT (A, p, q-1);

QUICK_SORT (A, q+1, r);



✳ PARTITION (A, p, r)



① $A[p \dots i]$: elements $\leq x$

② $A[i+1 \dots j-1]$: elements $\geq x$

③ $A[j \dots r-1]$: unchecked elements

④ $A[r]$: pivot

Running time of Quick Sort $\quad T(n) = \Theta(n) + T(^{q-1}_{q}) + T(n-q)$ $\quad$ (q : index of pivot)

Fri 03/01

Example 1: 6 10 13 5 8 23 12 [11]

≤ 11 [11] ≥ 11

Initially:

① Empty

② Empty

③ Except of 11

6 | 10 13 5 8 23 12 [11]

① ③

6 10 | 13 5 8 23 12 [11]

① i ③

6 10 | 13 | 5 8 23 12 [11]

① i ② J ③

6 10 5 | 13 | 8 23 12 [11]

① ② ③

6 10 5 8 | 13 | 23 12 [11]

① ② ③

6 10 5 8 | 13 23 | 12 [11]

① ② ③

6 10 5 8 | 13 23 | 12 [11] $\quad q$

if q, swap q with first element in ②

→ 6 10 5 8 9 | 13 23 | [11]

① ② ③ swap

6 10 5 8 9 11 13 23

Example 2: 5 5 5 5 [5] [5]

① ②

② Empty

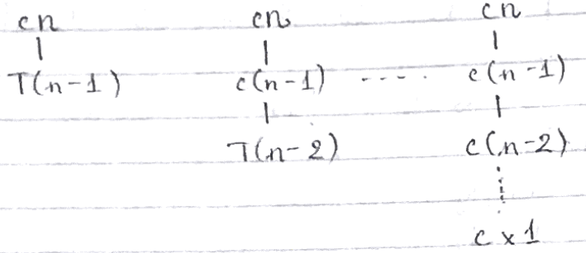
$T(n) = \Theta(n) + T(n-1) + T(0)$

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

$= T(n-1) + cn$ $\quad$ (c: positive constant)

$= \Theta(n^2)$

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



Example 3. Analyze running time when PARTITION always produce a 9-to-1 proportional split

We are lucky (unbalanced) $T(n) = T(\frac{9n}{10}) + T(\frac{n}{10}) + \Theta(n) = \Theta(n \lg n)$

$T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + \Theta(n)$
$= \Theta(n \lg n)$

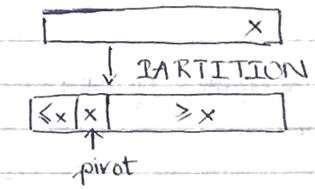
Wed
03/20

Chapter 5

✶ Concept 1. Sample space
S = "all possible outcomes"
in algorithm analysis, S is finite
each outcome $i \in S$ has a probability
$\Pr(i) \geq 0$

$\sum_{i \in S} \Pr(i) = 1$

Ex: Choosing a random pivot in Quicksort
$S = \{1, 2, \ldots, n\}$ (randomly pick a pivot)



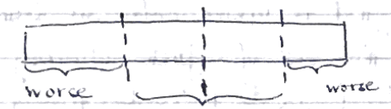
$\Pr(i) = \frac{1}{n}$

✶ Concept 2. Event
An event is subset of S, $A \subseteq S$
The probability of A is $\sum_{i \in A} \Pr(i)$ (i: outcome in A)

Event: sum of two dice is 8

Eg: Consider the event
"the chosen pivot gives a 25-to-75 split or better"
What is the probability of this event?

A. $1/n$
B. $1/4$
(C) $1/2$
D. $3/4$



✶ Concept 3: Random variables
X is real-valued function whose domain is the sample space

eg 1: Let X be the sum of two dice
Now X is a function that maps each outcome in the sample space to a number
$X(1,1) = 2$ $X(6,5) = 11$ $X: 2 \sim 12$

e.g.2 : Let Y be the size subarray passed to the first recursive call in QUICKSORT

Now Y is function that maps each outcome in S to a number

$Y(\text{pivot} = 1) = 0$

$Y(\text{pivot} = 2) = 1$

$\vdots$

$Y(\text{pivot} = 3) = n-1$

Concept 4: Expectation

Let X be a random variable is S expectation (or expected value) of X

$E[X]$ of X:

average value of X, naturally weighted by the probability of various possible outcomes

$$E[X] = \sum_{i \in S} X[i] \Pr(i)$$

i: possible outcome in S

Ex1: What is the expectation of the value of rolling one single dice?

$$E[X] = 1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6}$$
$$= 3.5$$

Ex2: What is the expectation of the size of the subarray passed to the first recursive call in Quicksort?

Let Y be the size of first subarray

$$E[Y] = 1 \times \frac{1}{n} + 2 \times \frac{1}{n} + 3 \times \frac{1}{n} + \dots + (n-1) \times \frac{1}{n} + n \times \frac{1}{n}$$
$$= \sum_{i=0}^{n-1} i \times \frac{1}{n} = (0 + 1 + \dots + n-1) \times \frac{1}{n} = \frac{n-1}{2}$$

Concept 5: Linearity of Expectation

Let $X_1, X_2, \dots X_n$ be random variables in S. Then

$$E[X_1 + X_2 + \dots + X_n] = E[X_1] + E[X_2] + \dots + E[X_n]$$

$$E\left(\sum_{J=1}^{n} X_J\right) = \sum_{J=1}^{n} E[X_J]$$

Ex: What is the expectation of X (the sum of two dice)?

$$E[X] = 2 \times \frac{1}{36} + 3 \times \frac{2}{36} + 4 \times \frac{3}{36} + \dots + 12 \times \frac{1}{36}$$

$E[X_1] = 3.5$

$E[X_2] = 3.5$

$E[X_1 + X_2] = E[X_1] + E[X_2] = 2E[X_i]$

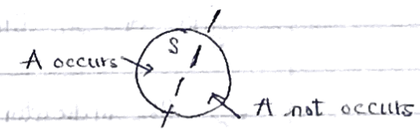
Fri 03/22

Indicator Random Variables

(Indicator) is a random variable that maps every outcome to either 0 or 1

Let A be an event in S, indicator random variable

$$I(A) = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{if } A \text{ does not occur } \bar{A} \end{cases}$$



e.x: A: two dice have the same value

X_A: is a random variable of

$X_A = 0$, if two dices have different values

$X_A = 1$, if two dices have the same value

$$X_A = I\{A\} = \begin{cases} 1 \\ 0 \end{cases}$$

Lemma: For an event A, Let $X_A = I\{A\}$

Then $E[X_A] = \Pr\{A\}$

Proof: $E[X_A] = E[I\{A\}] = 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{A\} = \Pr\{A\}$

Ex1. (simple example)

Determine the expected number of heads when flip a fair coin one time

$S = \{H, T\}$

$\Pr\{H\} = \Pr\{T\} = 1/2$

H: event that head will show up in one flip

$X_H = I\{H\} = \begin{cases} 1 & \text{Heads up} \\ 0 & \text{Tail up} \end{cases}$

$E[X_H] = \Pr\{H\} = \frac{1}{2}$

$= 1 \times \frac{1}{2} + 0 \times \frac{1}{2}$

$n = 2$ Expectation of heads when flip twice

$S = \{HH, HT, TH, TT\}$

$E[X_H] = 2 \times \frac{1}{4} + 1 \times \frac{1}{4} + 1 \times \frac{1}{4} + 0 \times \frac{1}{4} = 1$

$n = 3$ $S = \{HHH, HHT, HTH, \ldots\}$

Determine the expected number of heads when flip a fair coin n times (or # of heads in n flips)

⇒ Use indicator random variable:

For $i = 1, 2, \ldots, n$

define $X_i = I\{\text{the } i^{th} \text{ flip results in event } H\}$

$= \begin{cases} 1 \\ 0 \end{cases}$

$X = \sum_{i=1}^{n} X_i$ ← total # of H in n flips

$E[X] = E\left[\sum_{i=1}^{n} X_i\right]$

$= E[X_1] + E[X_2] + \ldots + E[X_n]$

$= \frac{1}{2} + \frac{1}{2} + \ldots + \frac{1}{2}$ ← Lemma

$= \frac{n}{2}$

✶ Probabilistic Analysis:

- When to use: have knowledge, make assumptions about input distribution
- Not applicable: most cases
  impose distribution
  ⇒ randomized algorithm
- Average running ⇒ Expected running time

✶ Random Number Generator:

RANDOM(a, b)

- returns one integer in $[a, b]$, with each number being equally likely

RANDOM(2, 6)

either i: 2, 3, 4, 5, 6

$\Pr(i) = \frac{1}{5}$

e.g.: Randomly permutating array

Input: array A contains elements 1 ~ n

Output: random permutation of array

$\frac{1}{n!}$ (equally likely)

Uniform random permutation: if the probability to get each permutation is the same

Mon
03/25

QuickSort Analysis

Worst-case partitioning

Unbalanced partitioning arises in each recursive call to avoid worst case.

=> Avoid unbalanced partitioning

Best way is Randomization

① Choose pivot randomly from range A[low..high]

② Initially permute the array

```
RANDOMIZED-PARTITION (A, p, r)
    i = Random (p, r)
    exchange A[r] with A[i]
    return PARTITION (A, p, r)
```

$p, p+1, p+2, \ldots, r$

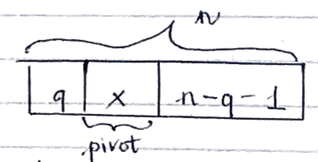
$$E[x] = \frac{1}{r-p+1}\left[p+(p+1)+(p+2)+\ldots+r\right]$$

```
RANDOMIZED-QUICKSORT (A, p, r)
  if p < r
      q = R-P (A, p, r)
      R-Q (A, p, q-1)
      R-Q (A, q+1, p)
```

1/ QUICK-SORT

Worst-case analysis

q: size of one subarray



$$T(n) = \max_{0 \le q \le n-1}\left(T(q)+T(n-q-1)\right) + \Theta(n)$$

2/ QUICK-SORT

* Average-case Analysis

=> We call the running time as expected running time because we use randomization

* Expected running time:

=> the dominate case is partitioning

=> partition is called at most n times

=> work in partitioning is some constant plus.

sub-routine

The comparision done in line 4 in PARTITION in for loop

Let X = the total number of comparisions performed in all calls to PARTITION

∴ total work done over the entire program execution $O(n+X)$

PARTITION => 1-2 $\Theta(1)$

7-8 $\Theta(1)$

$O(n+X)$

Rename elements in A as $z_1, z_2, \ldots z_n$ with z_i the i^{th} smallest element in the array

Define the set $Z_{ij} = \{z_i, z_{i+1}, z_{i+2}, \ldots, z_j\}$ where $i < j$

Q. When does the algorithm compare z_i and z_j?

8, 1, 6, 4, 0, 3, 9, 5

1	5	9

Do we compare (1, 9) -> No, because they are in different subarray

(5, 6) -> Once, cuz 5 is a pivot

(8, 9) -> compare

1	5	6	9	8

1/ We may compare them if they are in the same subarray but not always

2/ Z_i and Z_J will be compared if one of them is a pivot

3/ We will definitely not compare them if they are in different subarray

2 cases:

① If pivot is between Z_i and Z_J, they end up in 2 subarrays
Never compare them

② If Z_i and Z_J in the same subarray and if one of them is picked as a pivot, we will compare them.

Let $X_{ij} = I\{ Z_i \text{ is compared to } Z_J \}$

$$= \begin{cases} 1 & \text{case ②} \\ 0 & \text{case ①} \end{cases}$$

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

↳ total no. of comparisions.

Wed
03/27

Total no. of Comparisons:

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

↘ linearity of expectation

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{ Z_i \text{ is compared to } Z_J \}$$

$$= \Pr\{ Z_i \text{ is (first one) picked as pivot or } Z_J \text{ ——— // ——— } \}$$

$$= 2 \Pr\{ \text{any given element is the first one chosen as pivot} \}$$

$$= 2 \cdot \frac{1}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Let $k = j - 1$

$$E[X] = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

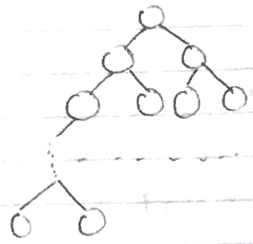
$$= \sum_{i=1}^{n-1} O(\lg n)$$

$$= O(n \lg n)$$

Expected running time for average case for QuickSort is $O(n \lg n)$

Fri 03/29

Heap with n nodes, how many nodes have the height of 0, 1, 2?



Level	# of nodes
0	2^0
1	2^1
2	2^2
i	2^i
Last level: ℓ	2^ℓ

$\ell = \lfloor \log_2 n \rfloor$

Sum of # nodes $= 2^0 + 2^1 + 2^2 + \dots + 2^\ell$

$= \frac{2^{\ell+1} - 1}{2-1} = 2^{\ell+1} - 1 \approx n$

$\Rightarrow 2^{\ell+1} = n + 1$

$2^\ell \approx \frac{n}{2}$ $\Rightarrow$ node with height of 0

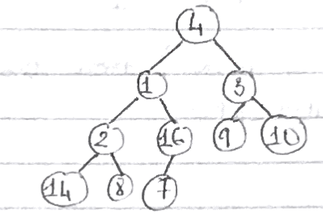
$\frac{n}{4}$: height of 1

$\frac{n}{8}$: height of 2

$\underbrace{\frac{n}{2} + \frac{n}{4} + \frac{n}{8} = \frac{7}{8} n \approx 87\%}_{\text{90\% of total nodes}}$

at the bottom of the tree

Build a max heap for the following unsorted Array 4, 1, 3, 2, 16, 9, 10, 14, 8, 7



Swap (2) with (14)

Max-heapify (3)

Swap (3) with (10)

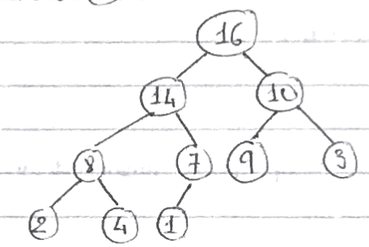
Swap (16) with (1)

Swap (1) with (7)

Swap (4) with (16)

Swap (4) with (14)

Swap (4) with (8)



$\Rightarrow$ 16 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 1

Mon 04/01 Analysis of building ~~maps/heap~~ max_heap (tight)

# of nodes of height h is $\left\lceil \frac{n}{2^{h+1}} \right\rceil$

1/ There $\frac{n}{2}$ nodes (half of nodes) have $h = 0$ (leaves)
no need to call MAX_HEAPIFY

2/ $\frac{n}{4}$ nodes have $h = 1$
of ~~nodes~~ times to call MAX_HEAPIFY at most once.

3/ $\frac{n}{8}$ nodes have $h = 2$
# of times to call MAX_HEAPIFY at most twice

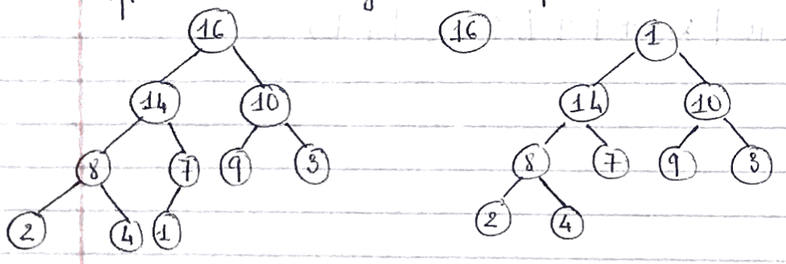
When we go up to the root,
∴ the running time should be less than $O(n \lg n)$

$$T(n) \leq \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{n}{2^{h+1}} \cdot O(h)$$

$$\Rightarrow O(n)$$

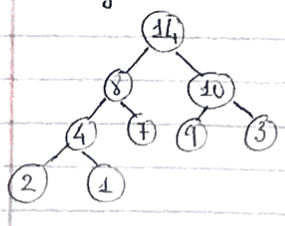
We can build a max_heap from unsorted in linear times

✳ Extract Max for max_heap



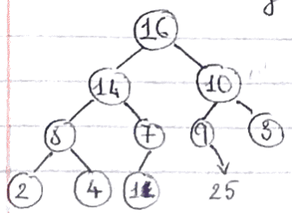
$T(n) = O(\lg n)$

⇒ Why use the last element on the new root ?



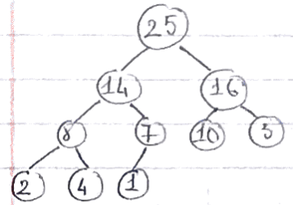
Return Max or Min
return A[1]

Increase key value

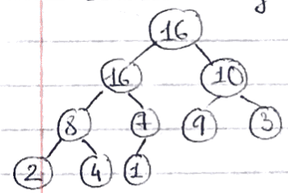


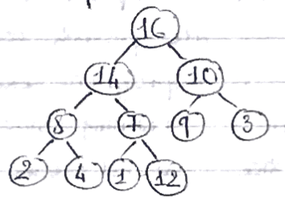
Increase (9) to (25)

$O(\lg n)$



Insert key (12) into the heap





Why insert, new key as last element ?

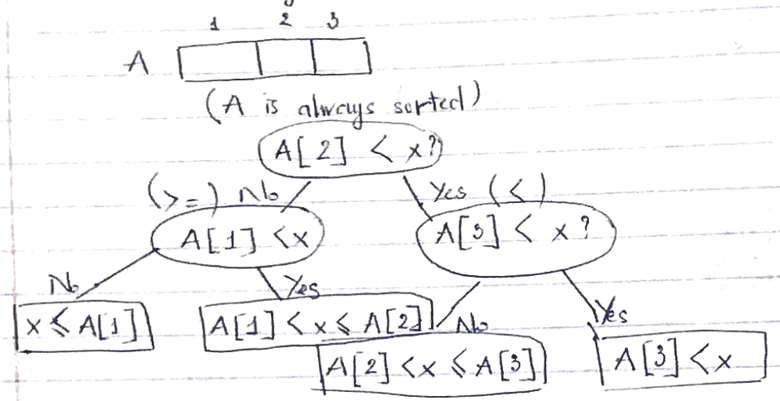
Fri 04/05

Chapter 8 :

Decision tree

eg : Binary search, $n = 3$

Search key x



+ Internal nodes : Comparisons
+ Leaves : all possible results
+ root - leaf : program execution
+ path length : running time
+ height of tree : worst - running time

Mon 04/08

Theorem: Search Lower bound:

n (pre-processed) items, finding a given item among them in comparision model requires $\Omega(\lg n)$ in worst case

Proof: decision tree is binary

# of leaves must be $\geq n$

Height of decision tree $\geq \lg n$

Theorem: Sort n elements in comparison model, requires $\Omega(n \lg n)$ in worst case

Proof: # of leaves $\geq$ possible answers $= n!$

height of decision tree $\geq \lg(n!)$

$$\lg(n!) = \lg n + \lg(n-1) + \dots + \lg 2 + \lg 1$$

$$= \sum_{i=1}^{n} \lg i$$

$$\geq \sum_{i=\frac{n}{2}}^{n} \lg i \geq \sum_{i=\frac{n}{2}}^{n} \lg\left(\frac{n}{2}\right)$$

$$= \sum_{i=\frac{n}{2}}^{n} \left(\lg n - 1\right) = \frac{n}{2}\lg n - \frac{n}{2}$$

$$= \Omega(n \lg n)$$

$n! > \left(\frac{n}{e}\right)^n$

* Heapsort and Mergesort are asymptotically optimal comparison sorts.

Counting Sort

$\langle 2, 5, 3, 0, 2, 3, 0, 3 \rangle$

Intuition:

2s: 2
5s: 1
3s: 3
0: 2
1: 0
4: 0

$\langle 0, 0, 2, 2, 3, 3, 3, 5 \rangle$

Fri 04/12

Input A: 2 | 5 | 3 | ☆ | 2 | 3 | ♡ | 3
[1]

Output B: ☆ | ♡ | 2 | 2 | 3 | 3 | 3 | 5

Auxiliary C: 2 | 0 | 2 | 3 | 0 | 1
[0] [1] [2] [3] [4] [5]

Stable Sorting Algorithm:
Numbers with the same value appear in the output array in the same order as they do in the input array.

Radix Sort

Ex:

3	2	9
4	5	7
6	5	7
8	3	9
4	3	6
7	2	0
3	5	5

Sort least significant digit

7	2	0
3	5	5
4	3	6
4	5	7
6	5	7
3	2	9
8	3	9

Middle digit

7	2	0
3	2	9
4	3	6
8	3	9
3	5	5
4	5	7
6	5	7

Most significant digit

3	2	9
3	5	5
4	3	6
4	5	7
6	5	7
7	2	0
8	3	9

Analysis: $\theta(n+k')$ per digit (digit range $s[0..k']$)
- d digits
- total running time for radix sort (d - digit number)

$\theta(d(n+k'))$

Let's have n integers, each integer is in b bits
each integer ranges: $0 \sim 2^b - 1$
break each integer into r-bit digits
each digit ranges: $0 \sim 2^r - 1$
Total # of digits: $d = \frac{b}{r}$

$$T(n) = \theta\left(\frac{b}{r}(n+k')\right)$$
$$= \theta\left(\frac{b}{r}(n+2^r)\right)$$

To minimize $T(n)$: $\frac{b}{r}$ wants r to be bigger
2^r ——//—— smaller

Let $n = 2^r$ can make $T(n)$ minimum ($r = \lg n$)

$$T(n) = \theta\left(\frac{b}{r}(n+2^r)\right)$$
$$= \theta\left(\frac{b}{r}(n+n)\right)$$
$$= \theta\left(\frac{bn}{\lg n}\right)$$

$\theta(n)$ if $b = O(n \lg n) \Rightarrow T(n) = \theta(n)$
$r = \lg n$

Because the value of each integer $0 \sim 2^b - 1$
$n^c - 1$ (constant $c > 0$)
$n^c - 1 = 2^b - 1$
$\Rightarrow n^c = 2^b$
$\Rightarrow b = c \lg n$

Total time: $\theta\left(\frac{bn}{\lg n}\right) = \theta(cn) = \theta(n)$

32 bits long number, set them into 8-bit
4 rounds of counting sort

Wed 04/17

Bucket Sort

Input: array A with n elements $[0,1)$

Assume input is generated a random process that distributes elements uniformly over $[0,1)$

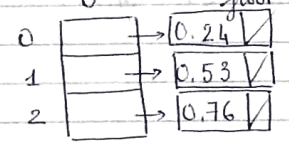
Idea:

- Divide $[0,1)$ into n equal-sized buckets
- Distribute n input values into n buckets
  (use a function of key values to index into an array)
- Sort each bucket (using a linked list)
- Go through the buckets in order

e.g. 1. Input: Array A with 3 elements (3 buckets)

Auxiliary array B[0..2] of link-list

A = <0.76, 0.53, 0.24>



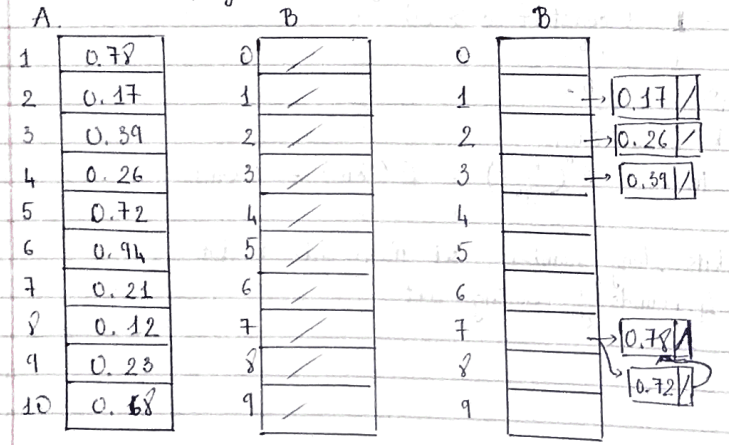
floor (0.76 × 3) = floor (2.28) = 2

floor (0.53 × 3) = floor (1.09) = 1

floor (0.24 × 3) = floor (0.72) = 0

e.g. Input, A with 10 elements (10 buckets)

B[0..9] of linked list



floor (0.78 × 10) = floor (7.8) = 7

(0.17 × 10) = 1

(0.39 × 10) = 3

(0.26 × 10) = 2

(0.72 × 10) = 7

(0.94 × 10) = 9

(0.21 × 10) = 2

(0.12 × 10) = 1

(0.23 × 10) = 2

(0.68 × 10) = 6

Wed, 04/24

Hash Table

Ex: Insert key 7, 18, 11, 5, 26 in the order given into a hash table of length 5, $h(k) = k^3 \bmod m$ ($m = 5$)

a/ Chaining:

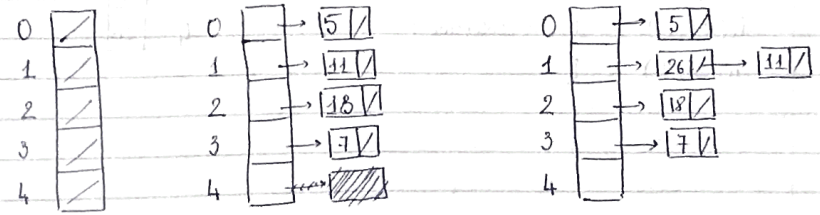
$h(7) = 7^3 \bmod 5 = 3 \Rightarrow \#3$

$h(18) = 18^3 \bmod 5 = 2 \Rightarrow \#2$

$h(11) = 11^3 \bmod 5 = 1 \Rightarrow \#1$

$h(5) = 5^3 \bmod 5 = 0 \Rightarrow \#0$

$h(26) = 26^3 \bmod 5 = 1 \Rightarrow \#1$



b/ Open-addressing, linear probing

$h(k,i) = (h'(k) + i) \bmod m$

$h'(k) = k^3$, $i = 0, 1, \ldots, m-1$

$h(7,0) = 7^3 \bmod 5 = 3$

$h(18,0) = 2$

$h(11,0) = 1$

$h(5,0) = 0$

$h(26,0) = 1$ collision

$h(26,1) = (26^3 + 1) \bmod 5 = 2$ collision

$h(26,2) = (26^3 + 2) \bmod 5 = 3$ collision

$h(26,3) = (26^3 + 3) \bmod 5 = 4$

0	5
1	11
2	18
3	7
4	26

c/ Open addressing, quadratic probing:

$h(k,i) = (h'(k) + c1\,i + c2\,i^2) \bmod m$

$h'(k) = k^3$, $c1 = 1$, $c2 = 2$

$h(7,0) = 3$

$h(18,0) = 2$

$h(11,0) = 1$

$h(5,0) = 0$

$h(26,0) = 1$ (collision)

$h(26,1) = 4$

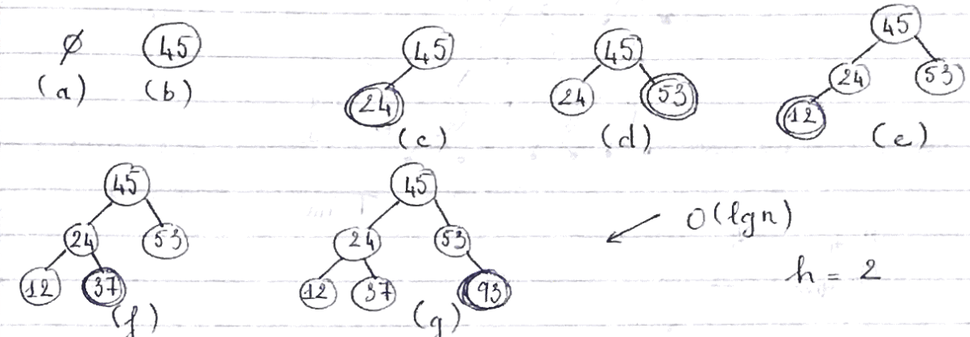
0	5
1	11
2	18
3	7
4	26

Fri,

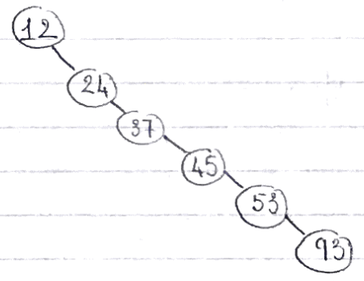
04/26

BTS, insertion

Insert <45, 24, 53, 12, 37, 93>



Insert <12, 24, 37, 45, 53, 93>



⇒ Conclusion. BTS can be used unbalanced, O(n)

Red-Black Tree (RB-Tree)

① BTS with extra data (color) for each node, satisfying the following properties

① Every node has a color either RED ◎ or BLACK ○

② Color Root as BLACK

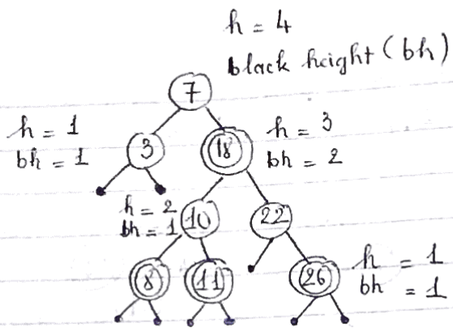
③ Every ~~leaf node~~ leaf node is T.nil, and it is Black

④ Every red node has a black parent and black children

i.e. no 2 red nodes connected

◎—◎ X

⑤ All simple paths from a node x to a descendant leaf of x have the same numbers of black nodes



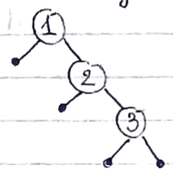
$h_{tree} = 4$

$bh_{tree} = 2$

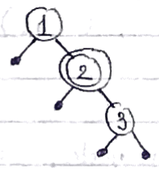
Height of a node: # of edges in a longest path to a leaf

Black height of a node: $bh(x)$

Example 1: a chain of length of 3

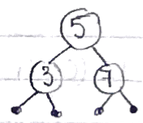


Color the node(s), make it R-B tree

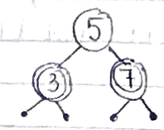


Can't make it a R-B tree

Example 2:



✓ R-B tree



✓ R-B tree

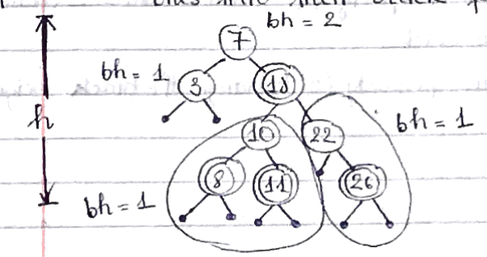
The red-black tree properties force the tree has small height of $lg(n)$ (n: number of keys)

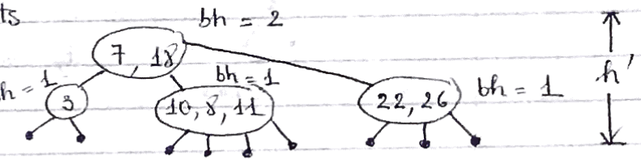
Theorem: a red-black tree with n keys has a height

$$h \leqslant 2 \lg(n+1)$$

Mon, 04/29

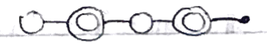
Intitution: black-height doesn't care about the red-nodes, merge red ones into their black parents





2-3-4 tree (each node has 2, 3, 4 children)

① for any path in Red-Black tree

# of red ones ? 

at most, half of the nodes are red ones.

$$h' \geqslant \frac{h}{2}$$

② # of leaves nodes remain the same after merge

n: # of keys, then the # of leave nodes: $n+1$

③ # of leave nodes in 2-3-4 tree (in terms of h')

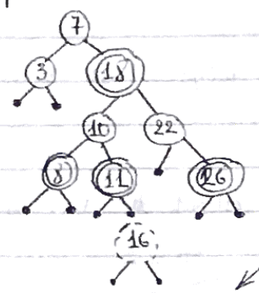
$\underbrace{2^{h'}}_{\text{branch 2 way}} \leqslant$ # of leave nodes $\leqslant \underbrace{4^{h'}}_{\text{branch 4 way}}$

$$n+1 \geqslant 2^{h'} \Rightarrow \frac{h}{2} \leqslant h' \leqslant \log_2(n+1)$$

$$\frac{h}{2} \leqslant \lg(n+1)$$

$$h \leqslant 2\lg(n+1)$$

* Operations on red-black tree:



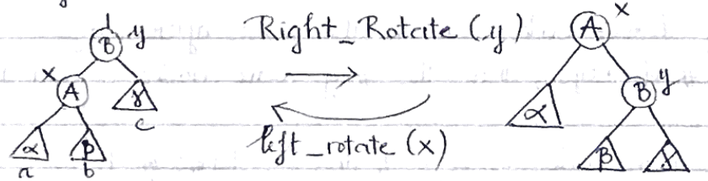
Insert (16) to the tree
① insert it using BTS insertion
② How to color?
a/ color to red?
violate property ④ - can't have 2 connected red
b/ color to black?
violate property ⑤ - change the black heights

Choose a

* Overall strategy:
  + Still use BTS insertion
  + Color new node to be RED
  + Recolor if needed
  + Restructure the tree via "~~violations~~ rotations"

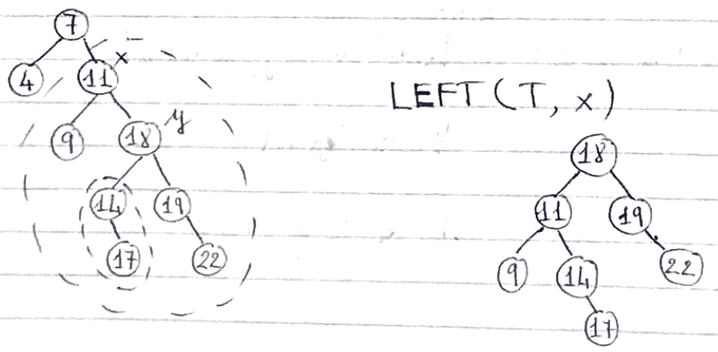
* Rotations:

Right rotation



$\forall a \in \alpha, b \in \beta, c \in \gamma$

$\Rightarrow a \leq A \leq b \leq B \leq c$



Wed, 05/01

```
RB_INSERT (T, z)
1. TREE_INSERT' ( T , z )                    O(lgn)
2. z.color = RED                             O(1)
3. RB_INSERT_FIXUP (T, z)                    O(lgn)
```

① ③ ⑤ can't be violated
② if root is red
④

~~Recursion~~ = ~~Procedure~~ a

RB-Insert: $O(\lg n) + O(\lg n) = O(\lg n)$

```
RB_INSERT_FIXUP (T, z)
While ( z.p.color == RED) and ( z ≠ T.root)
    if { z.p == z.p.p.left      // z' parent is a left child
        y = z.p.p.right         // y is an cuunt/uncle of z
        if y.color == RED
            <case 1>            // recolor
        else {
            if z == z.p.right   // z is a right child
            {<case 2>}          // rotation and recolor (do 1st)
            <case 3>            // recolor, rotation
        }
    else // z.p == z.p.p.right
        (same as if clause, with "right" and "left" exchange)
        y = z.p.p.left
        if y.color == RED
            <case 1'>
        else if z == z.p.left
            {<case 2'>}
            <case 3'>
T.root.color = BLACK
```

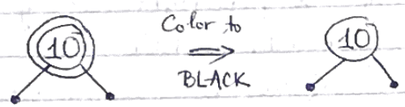
Case 1: z.p.color = BLACK
or 1' y.color = BLACK
O(1) z.p.p.color = RED
z = z.p.p

Case 2: z = z.p
O(1) or 2' LEFT-ROTATE (T, z)
Case 3: z.p.color = BLACK
or 3' z.p.p.color = RED
O(1) RIGHT-ROTATE (T, z.p.p)

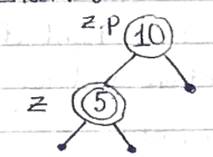
Ex: Insert < 10, 5, 25, 15, 20 >
leaf: RED (O) BLACK O

∅

(1) Insert 10

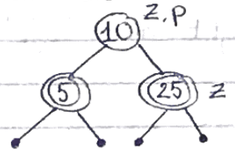


(2) Insert 5



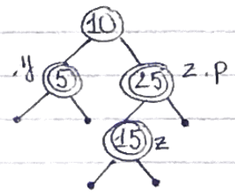
z.p is not RED
no fix-up

(3) Insert 25

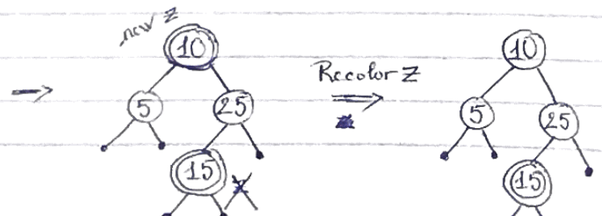


z.p is not RED
no fix-up

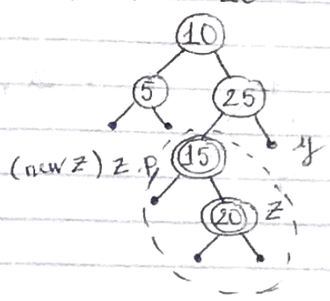
(4) Insert 15



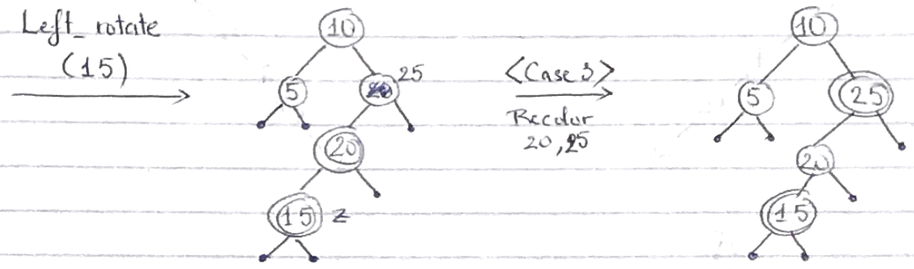
z.p is a right child
y is RED
Case 1'

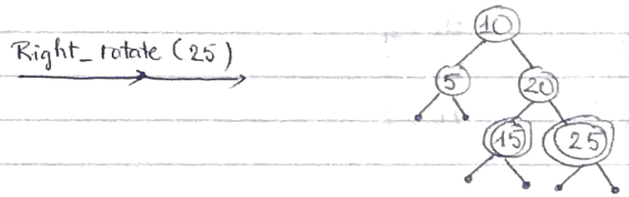


(5) Insert 20



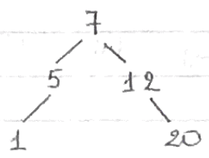
z is a right child
z' parent is a left child
y is BLACK
<Case 2> → Left_rotate (15)
<Case 3>



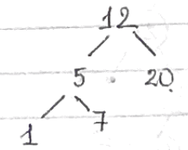


Hw 9 Solutions

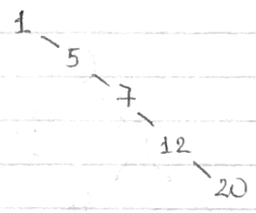
1/ Min: 2

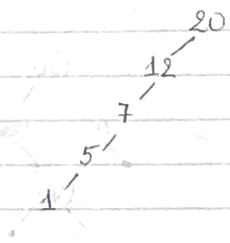


7, 5, 12, 1, 20

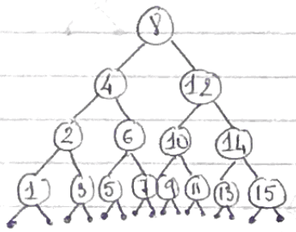


12, 5, 20, 1, 7





2/



bh = 4

bh = 3

bh = 2