BASIRA lab

Analysis of Algorithms

BLG 335E

Recitation 1 (10.10.19)

Kübra Cengiz Toyan
kcengiz@itu.edu.tr

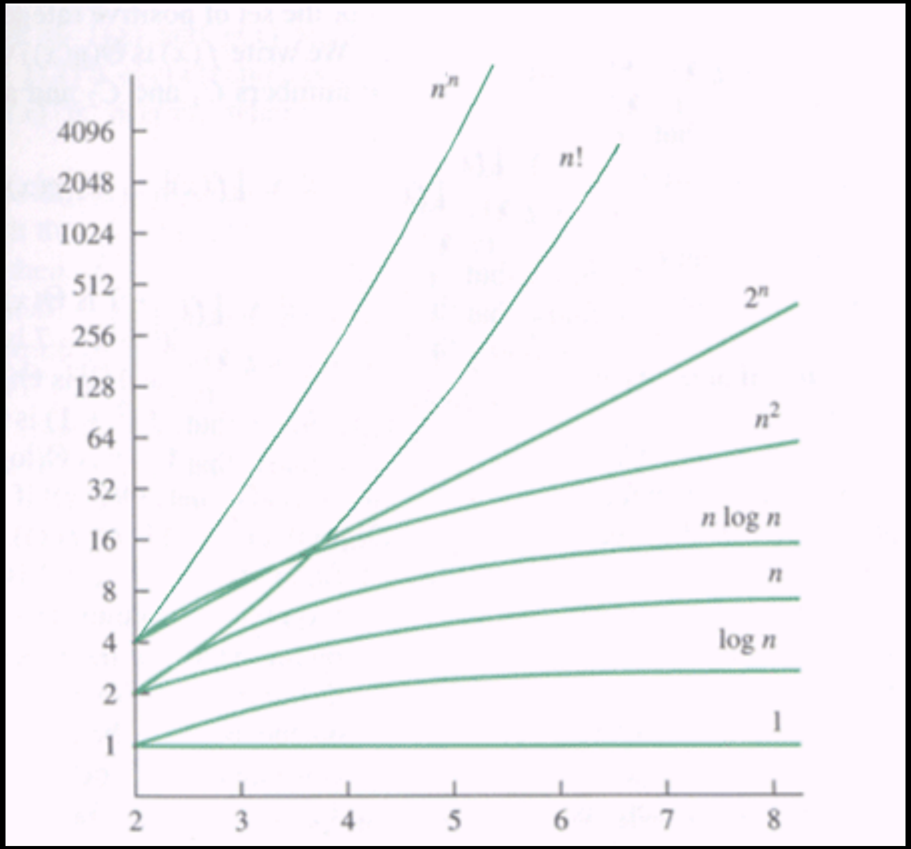
İTÜ

Algo

Q1: Asymptotic Comparison

- Order the following functions by asymptotic growth rate:
  - $n^2 + 5n + 7$
  - $\log_2 n^3$
  - 95^{17}
  - $2^{\log_2 n}$
  - n^3
  - $nlog_2n + 9n$
  - $4 \log_2 n$
  - $\log_2 n + 3n$

Solution1



- 95^{17}
- $\log_2 n^3$
- $4\log_2 n$
- $2^{\log_2 n}$
- $\log_2 n + 3n$
- $nlog_2 n + 9n$
- $n^2 + 5n + 7$
- n^3

$$1 << \log\ n << n << n\ \log\ n << n^2 << 2^n << n!$$

Q2: Limit Method

- Determine a tight inclusion of the form **f(n) ∈ Δ(g(n))** for following functions using **limit method**.

$$\mathrm{f(n)} = \log(n^2)\ , \qquad g(n) = logn\ + 8$$

Solution 2

Using limit method we can set up a limit quotient between f and g functions as follows:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \text{then } f(n) \in \mathcal{O}(g(n)) \\ c > 0 & \text{then } f(n) \in \Theta(g(n)) \\ \infty & \text{then } f(n) \in \Omega(g(n)) \end{cases}$$

1. We look for algebraic simplifications first.
2. If **f** and **g** both diverge or converge on zero or infinity, then we need to apply **l'Hôpital's Rule**.

Solution 2

l'Hôpital's Rule:

Let **f** and **g**, if the limit between the quotient $\frac{f(n)}{g(n)}$ exists, it is equal to the limit of the derivative of the denominator and the numerator.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{f(n)'}{g(n)'}$$

Solution 2

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \text{then } f(n) \in \mathcal{O}(g(n)) \\ c > 0 & \text{then } f(n) \in \Theta(g(n)) \\ \infty & \text{then } f(n) \in \Omega(g(n)) \end{cases}$$

$$\lim_{n\to\infty} \frac{\log(n^2)}{\log(n\,) + 8} = \lim_{n\to\infty} \frac{\frac{2}{n\ln 10}}{\frac{1}{n\ln 10}} = \lim_{n\to\infty}(2) = 2$$

$$0 < \lim_{n\to\infty} \frac{\log(n^2)}{\log(n\,)+8} = 2 < \infty$$

We can say **f(n) = Θ(g(n))**

Q3: Limit Method

- Determine a tight inclusion of the form **f(n) ∈ Δ(g(n))** for following functions using **limit method**.

$$f(n) = 2^n \ , \qquad g(n) = 3^n$$

Solution 3

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^{\mathrm{n}}}{3^{\mathrm{n}}}$
- **l'Hôpital's Rule:**

$$\frac{(2^n)'}{(3^n)'} = \frac{(\ln 2)2^n}{(\ln 3)3^n}$$

- Both numerator and denominator still diverge. We'll have to use an algebraic simplification.

Solution 3

- $\lim_{n\to\infty} \frac{2^n}{3^n} = \left(\frac{2}{3}\right)^n$

$$\lim_{n\to\infty} \alpha^n = \begin{cases} 0 & \text{if } \alpha < 1 \\ 1 & \text{if } \alpha = 1 \\ \infty & \text{if } \alpha > 1 \end{cases}$$

We can say $2^n \in O(3^n)$

ITÜ İSTANBUL TEKNİK ÜNİVERSİTESİ 1773

Algo

Q4: Selection Sort Time Complexity Study

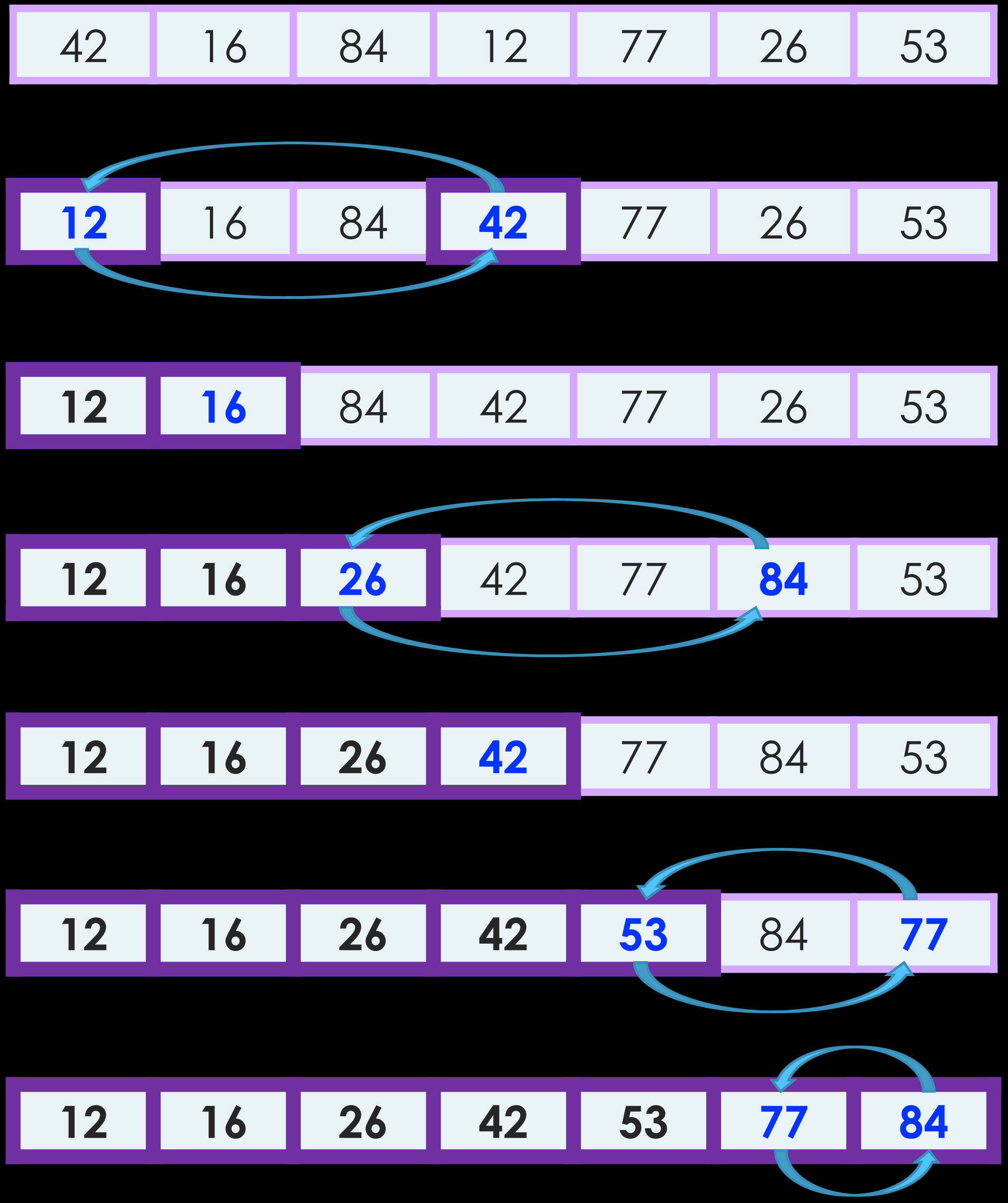
Although the statement adequately describes the sorting problem, **it is not an algorithm** because it leaves several questions unanswered.

For example, it does not tell us where and how the elements are initially stored or where we should place the result.

This is our first attempt:

We assume that the elements are stored in an array a, such that i^{th} integer is stored in the i^{th} position a[i], $1 \le i \le n$.

1. **for** *i*:= *1* **to** *n* **do**
2. **{**
3. Examine *a[i]* to *a[n]* and suppose
4. the smallest element is *a[j];*
5. Interchange *a[i]* with *a[j];*
6. **}**



The array, before the selection sort operation begins.

The smallest number (**12**) is swapped into the first element in the structure.

In the data that remains, **16** is the smallest; and it does not need to be moved.

26 is the next smallest number, and it is swapped into the third position.

42 is the next smallest number, and it is already in the correct position.

53 is the next smallest number in the data that remains; and it is swapped to the appropriate position.

Of the two remaining data items, **77** is the smaller; the items are swapped.
The selection sort is now complete.

Example: Selection Sort

To turn the algorithm into a pseudocode program, two subtasks remain:

- **Finding the smallest element**.

This task is solved by **assuming that *a[i]* is the minimum**, and then **comparing *a[i]* with *a[i+1], a[i+2],...,*** and whenever a smaller element is found, regarding it as a **new minimum**. Eventually the last element a[n] is compared with current minimum and we are done.

- **Defining “interchange” of a[i] with a[j]:**

$t := a[i];$

$a[i] = a[j];$

$a[j] := t;$

t: local variable defined for the swapping process of a [i] and a[j]

Example: Selection Sort

```
Algorithm SelectionSort (a,n)
// Sort the array a[1:n] in non decreasing order
      for i:= 1 to n-1 do
       {
          j := i;
          for k := i+1 to n do
              { if  a[k] < a[j] then j := k;  }
          t:=a[i]; a[i]:=a[j]; a[j]:= t;
       }
```

Time complexity of Selection Sort

Statement	Steps/ execution (s/e)	Frequency		Total steps	
		if-t	*if-f*	*If-t*	*if-f*
1. Algorithm SelectionSort (a,n)	0	-	-	0	0
2. {					
3. for *i*:= *1* **to** *n-1* **do**	1				
4. { *j* := *i*;	1				
5. **for** *k* := *i*+*1* **to** *n* **do**	1				
6. { **if** (*a*[*k*] < *a*[*j*]) **then**	1				
7. *j* := *k*; }	1				
8. *t*:=*a*[*i*]; *a*[*i*]:=*a*[*j*]; *a*[*j*]:= *t*;}	3				
9. }					
Total					

What is the time complexity?
Do worst (if-true) and best (if-false) time differ?

Time complexity of Selection Sort

Statement	Steps/ execution (s/e)	Frequency		Total steps	
		if-t	*if-f*	*If-t*	*if-f*
1. Algorithm SelectionSort (a,n)	0	-	-	0	0
2. {					
3. for *i*:= *1* **to** *n-1* **do**	1	n	n	n	n
4. { *j* := *i*;	1				
5. **for** *k* := *i+1* **to** *n* **do**	1				
6. { **if** (*a[k]* < *a[j]*) **then**	1				
7. *j* := *k*; }	1				
8. *t*:=*a[i]*; *a[i]*:=*a[j]*; *a[j]*:= *t*;}	3				
9. }					
Total					

What is the time complexity?
Do worst (if-true) and best (if-false) time differ?

Time complexity of Selection Sort

Statement	Steps/ execution (s/e)	Frequency		Total steps	
		if-t	*if-f*	*If-t*	*if-f*
1. Algorithm SelectionSort (a,n)	0	-	-	0	0
2. {					
3. **for** *i:= 1* **to** *n-1* **do**	1	n	n	n	n
4. { *j := i;*	1	n-1	n-1	n-1	n-1
5. **for** *k := i+1* **to** *n* **do**	1				
6. { **if** *(a[k] < a[j])* **then**	1				
7. *j := k;* }	1				
8. *t:=a[i]; a[i]:=a[j]; a[j]:= t;*}	3				
9. }					
Total					

What is the time complexity?
Do worst (if-true) and best (if-false) time differ?

Time complexity of Selection Sort

Statement	Steps/ execution (s/e)	Frequency		Total steps	
		if-t	*if-f*	*If-t*	*if-f*
1. Algorithm SelectionSort (a,n)	0	-	-	0	0
2. {					
3. **for** *i*:= *1* **to** *n-1* **do**	1	n	n	n	n
4. { *j* := *i*;	1	n-1	n-1	n-1	n-1
5. **for** *k* := *i+1* **to** *n* **do**	1	n*(n-1)	n*(n-1)	n*(n-1)	n*(n-1)
6. { **if** (*a[k]* < *a[j]*) **then**	1				
7. *j* := *k*; }	1				
8. *t:=a[i]; a[i]:=a[j]; a[j]:= t;* }	3				
9. }					
Total					

What is the time complexity?
Do worst (if-true) and best (if-false) time differ?

Time complexity of Selection Sort

Statement	Steps/ execution (s/e)	Frequency *if-t*	*if-f*	Total steps *If-t*	*if-f*
1. Algorithm SelectionSort (a,n)	0	-	-	0	0
2. {					
3. for *i:= 1* **to** *n-1* **do**	1	n	n	n	n
4. { *j := i;*	1	n-1	n-1	n-1	n-1
5. **for** *k := i+1* **to** *n* **do**	1	n*(n-1)	n*(n-1)	n*(n-1)	n*(n-1)
6. { **if** (*a[k] < a[j]*) **then**	1	$(n-1)^2$	$(n-1)^2$	$(n-1)^2$	$(n-1)^2$
7. j := k; }	1				
8. t:=a[i]; a[i]:=a[j]; a[j]:= t;}	3				
9. }					
Total					

[Note: $(n-1)^2 = (n-1) \times (n-1)$]

What is the time complexity?

Do worst (if-true) and best (if-false) time differ?

Time complexity of Selection Sort

Statement	Steps/ execution (s/e)	Frequency		Total steps	
		if-t	*if-f*	*If-t*	*if-f*
1. Algorithm SelectionSort (a,n)	0	-	-	0	0
2. {					
3. for *i:= 1* **to** *n-1* **do**	1	n	n	n	n
4. { *j := i;*	1	n-1	n-1	n-1	n-1
5. **for** *k := i+1* **to** *n* **do**	1	n*(n-1)	n*(n-1)	n*(n-1)	n*(n-1)
6. { **if** (*a[k] < a[j]*) **then**	1	$(n-1)^2$	$(n-1)^2$	$(n-1)^2$	$(n-1)^2$
7. j := k; }	1	$(n-1)^2$	0	$(n-1)^2$	0
8. t:=a[i]; a[i]:=a[j]; a[j]:= t;}	3				
9. }					
Total					

[Note: $(n-1)^2 = (n-1) \times (n-1)$]

What is the time complexity?

Do worst (if-true) and best (if-false) time differ?

Time complexity of Selection Sort

Statement	Steps/ execution (s/e)	Frequency		Total steps	
		if-t	*if-f*	*If-t*	*if-f*
1. Algorithm SelectionSort (a,n)	0	-	-	0	0
2. {					
3. for *i:= 1* **to** *n-1* **do**	1	n	n	n	n
4. { *j := i;*	1	n-1	n-1	n-1	n-1
5. **for** *k := i+1* **to** *n* **do**	1	n*(n-1)	n*(n-1)	n*(n-1)	n*(n-1)
6. { **if** *(a[k] < a[j])* **then**	1	$(n-1)^2$	$(n-1)^2$	$(n-1)^2$	$(n-1)^2$
7. *j := k;* }	1	$(n-1)^2$	0	$(n-1)^2$	0
8. *t:=a[i]; a[i]:=a[j]; a[j]:= t;*}	3	n-1	n-1	3n-3	3n-3
9. }					
Total					

[Note: $(n-1)^2 = (n-1) \times (n-1)$]

What is the time complexity?

Do worst (if-true) and best (if-false) time differ?

Time complexity of Selection Sort

Statement	Steps/ execution (s/e)	Frequency		Total steps	
		if-t	*if-f*	*If-t*	*if-f*
1. Algorithm SelectionSort (a,n)	0	-	-	0	0
2. {					
3. **for** *i:= 1* **to** *n-1* **do**	1	n	n	n	n
4. { *j := i;*	1	n-1	n-1	n-1	n-1
5. **for** *k := i+1* **to** *n* **do**	1	n*(n-1)	n*(n-1)	n*(n-1)	n*(n-1)
6. { **if** *(a[k] < a[j])* **then**	1	$(n-1)^2$	$(n-1)^2$	$(n-1)^2$	$(n-1)^2$
7. *j := k;* }	1	$(n-1)^2$	0	$(n-1)^2$	0
8. *t:=a[i]; a[i]:=a[j]; a[j]:= t;*}	3	n-1	n-1	3n-3	3n-3
9. }					
Total				$3n^2+3$	$2n^2+5n+3$

[Note: $(n-1)^2 = (n-1) \times (n-1)$]

What is the time complexity?

Do worst (if-true) and best (if-false) time differ?

Convergent Power Sum

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \leq O(1), \text{ for } 0 < x < 1$$

- A polynomial is asymptotically equal to its leading term as x → ∞

$$\sum_{i=0}^{d} a_i\, x^i = \theta\left(x^d\right)$$

$$\sum_{i=0}^{d} a_i\, x^i = o\left(x^{d+1}\right)$$

$$\sum_{i=0}^{d} a_i\, x^i \sim a_d\, x^d$$

Sums of Powers

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

- for n → ∞

$$\sum_{i=1}^{n} i^d \sim \frac{1}{d+1} n^{d+1}$$

- Or equivalently

$$\sum_{i=1}^{n} i^d = \frac{1}{d+1} n^{d+1} + o\left(n^{d+1}\right)$$

Examples

$$\begin{cases} \sum_{i=1}^{n} i \sim \frac{n^2}{2} \\ \sum_{i=1}^{n} i^2 \sim \frac{n^3}{3} \end{cases}$$

- 2^{nd} order asymptotic expansion

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

Recurrence Equations

Approximate Solution of Recurrence Relations

Recurrence Equations (over integers)

- Homogenous of degree d

$n > d$

$$\boxed{x_n = a_1 x_{n-1} + a_2 x_{n-2} + \ldots + a_d x_{n-d}}$$

- Given $\begin{cases} \text{constant coefficients} & a_1, \ldots, a_d \\ \text{initial values} & x_1, x_2, \ldots, x_d \end{cases}$

A Useful Theorem

- $c > 0, d > 0$
- If

$$T(n) = \begin{cases} c_0 & n=1 \\ aT\left(\dfrac{n}{b}\right) + cn^d & n>1 \end{cases}$$

- then

$$T(n) = \begin{cases} \theta\left(n^{\log_b a}\right) & a > b^d \\ \theta\left(n^d \log_b n\right) & a = b^d \\ \theta\left(n^d\right) & a < b^d \end{cases}$$

Proof

$$T(n) = cn^d\, g(n) + a^{\log_b n}\, d$$

- Is solution

$$g(n) = 1 + \frac{a}{b^d} + \left(\frac{a}{b^d}\right)^2 + \ldots + \left(\frac{a}{b^d}\right)^{\log_b n - 1}$$

Cases

(1) $a > b^d \Rightarrow g(n) \sim \left(\frac{a}{b^d}\right)^{\log_b n-1}$

is last term so

$$T(n) = \theta\left(a^{\log_b n} d\right) = \theta\left(n^{\log_b a}\right)$$

(2) $a = b^d \Rightarrow g(n) = \log_b n$

so $T(n) = \theta\left(n^d \log_b n\right)$

(3) $a < b^d \Rightarrow g(n)$ upper bound by $0(1)$

so $T(n) = \theta\left(n^d\right)$

Solving Recurrences

- The recurrence has to be solved in order to find out T(n) as a function of n
- General methods for solving recurrences:
  - **Substitution Method**
  - **Recursion-tree Method**

The Substitution Method

- The substitution method for solving recurrences:
  - do a few substitution steps in the recurrence relationship until you can guess the solution (the formula) and prove it with math induction

Q5: Recurrence Relations

- Give tight asymptotic bounds for $T(n)$ in each of the following recurrences.

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

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

Solution 5a

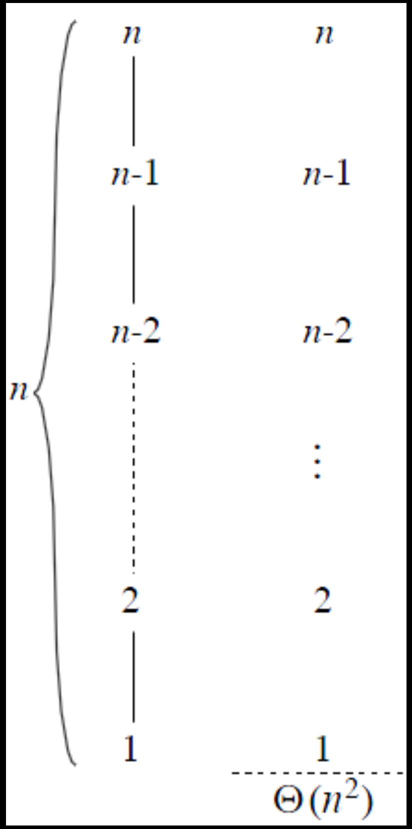
- Give tight asymptotic bounds for $T(n)$ in each of the following recurrences.

  a. $T(n) = T(n-1) + n$

Lower bound (Ω):

$T(n) \geq cn^2\ for\ some\ c > 0$
$T(n) \geq c(n-1)^2 + n$
$= cn^2 - 2cn + c + n \geq cn^2$

$true\ if\ 0 < c < \frac{1}{2} and\ n \geq 0$



Solution 5a

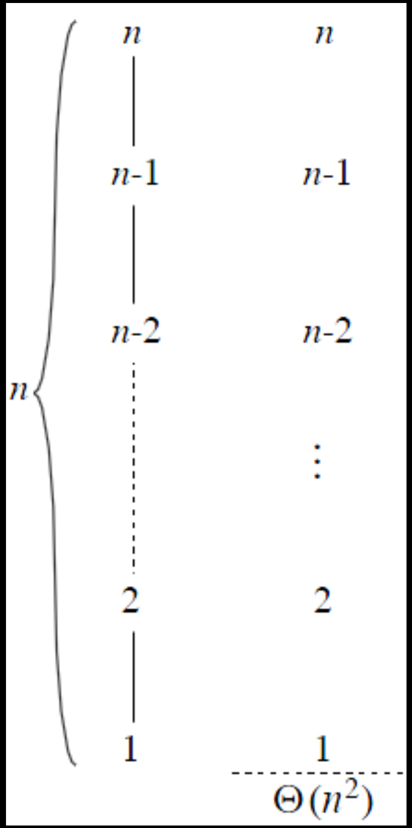
- Give tight asymptotic bounds for $T(n)$ in each of the following recurrences.

  a. $T(n) = T(n-1) + n$

  Upper bound (O):

  $T(n) \leq cn^2 \ for \ some \ c > 0$
  $T(n) \leq c(n-1)^2 + n$
  $= cn^2 - 2cn + c + n \leq cn^2$

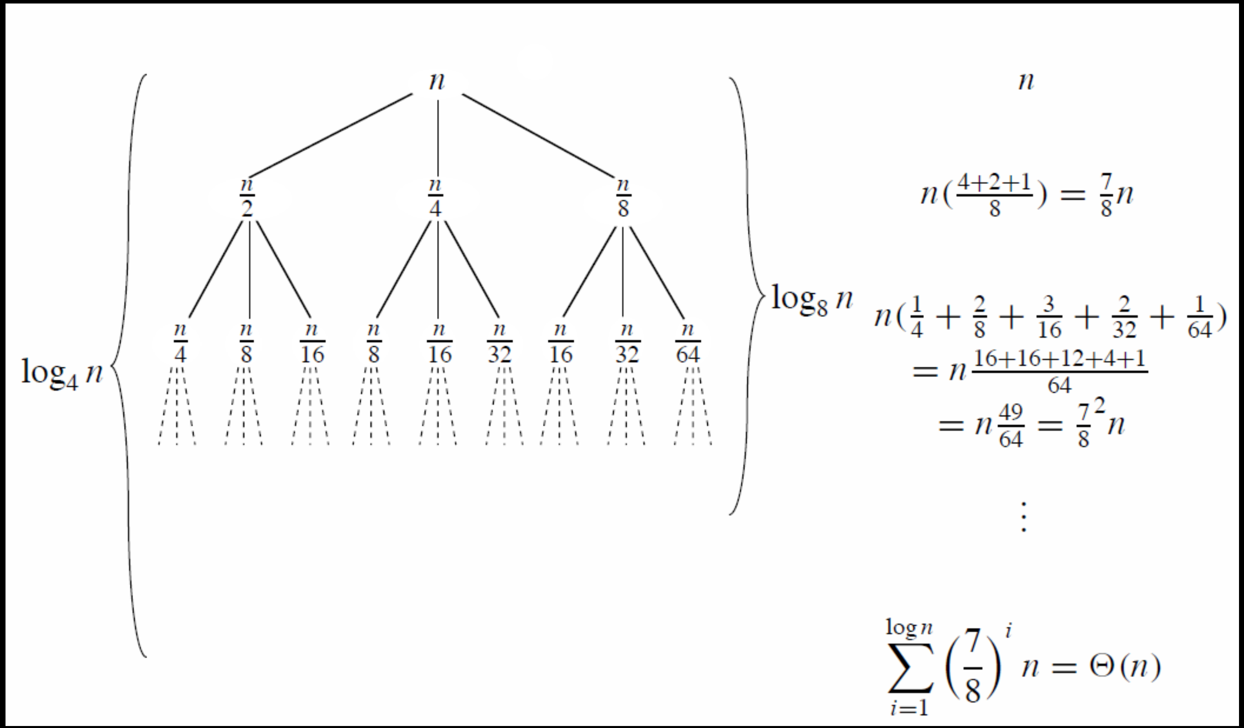
  $true \ if \ c = 1 \ and \ n \geq 1$



Solution 5b

- Give tight asymptotic bounds for $T(n)$ in each of the following recurrences.

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



Solution 5b

- Give tight asymptotic bounds for $T(n)$ in each of the following recurrences.

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

  Upper bound (*O*):

  $$T(n) \leq \frac{cn}{2} + \frac{cn}{4} + \frac{cn}{8} + n = \frac{7cn}{8} + n \leq cn$$

  $true\ if\ c \geq 8$

Solution 5b

- Give tight asymptotic bounds for $T(n)$ in each of the following recurrences.

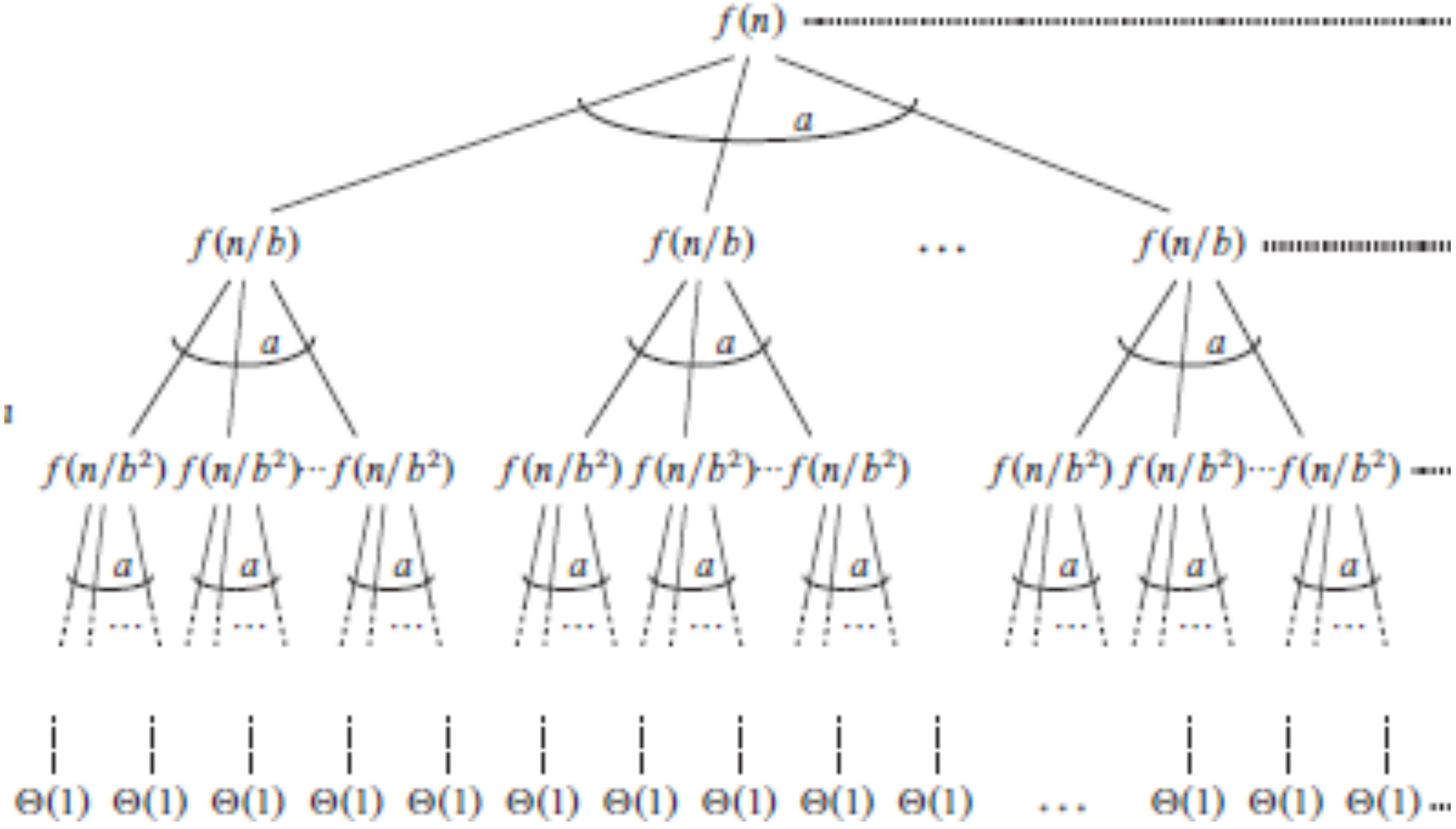
  b. $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{8}\right) + n$

Lower bound (*Ω*):

$$T(n) \geq \frac{cn}{2} + \frac{cn}{4} + \frac{cn}{8} + n = \frac{7cn}{8} + n \geq cn$$

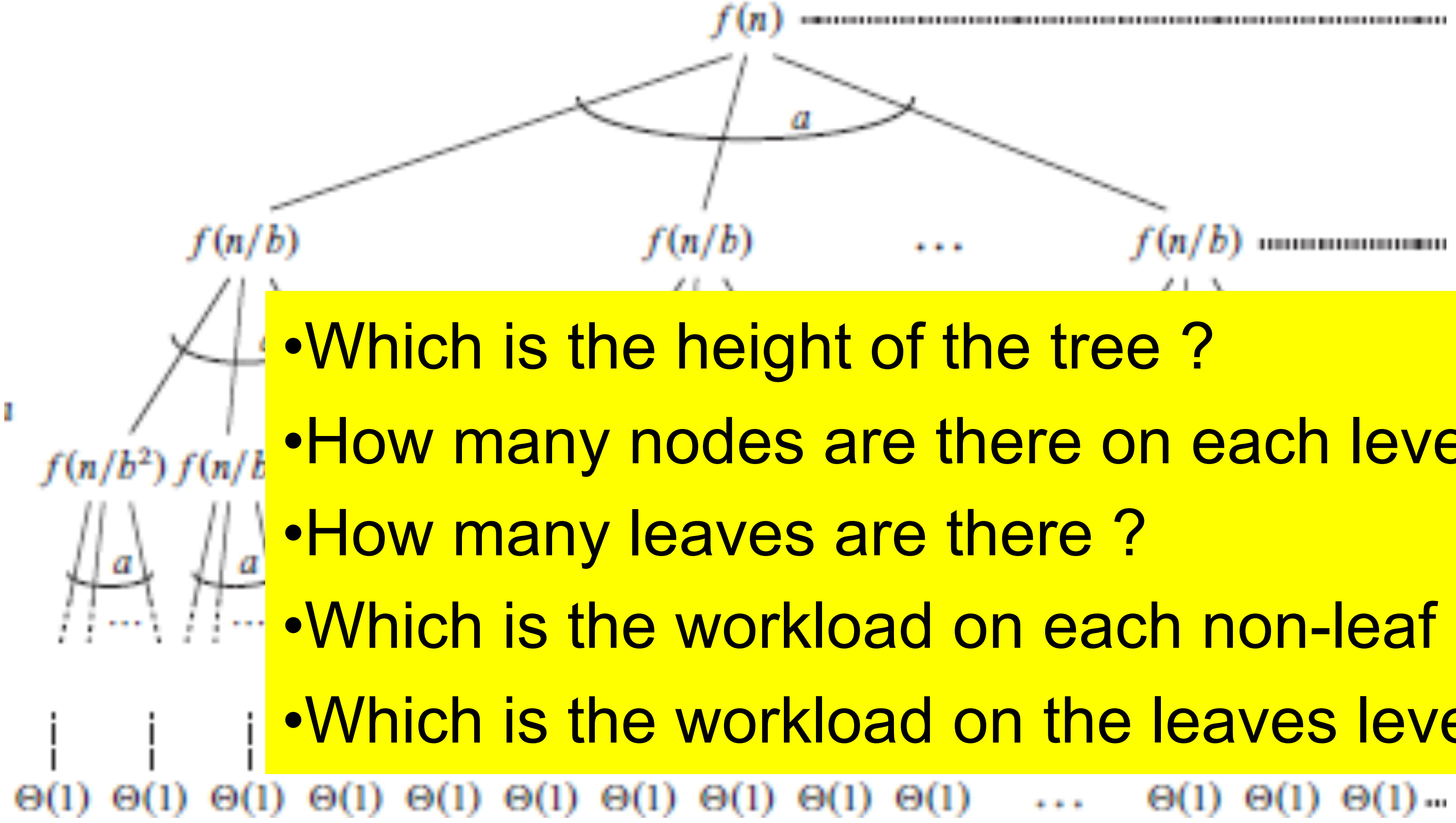
$true\ if\ 0 < c \leq 8$

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



Recursion tree

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



- Which is the height of the tree ?
- How many nodes are there on each level ?
- How many leaves are there ?
- Which is the workload on each non-leaf level ?
- Which is the workload on the leaves level ?

Recursion tree

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

From the recursion tree:

$$T(n) = n^{\log_b a} + \sum_{i=0}^{\log_b n-1} a^i \cdot f(\frac{n}{b^i})$$

Workload in leaves

Sum for all levels

Workload per level i

Applying the math

$$T(n) = n^{\log_b a} + c \cdot n^k \cdot \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^k}\right)^i$$

$$3 \text{ cases for the geometric series:}$$

$$if\ a = b^k : S(n) = \log_b n; \quad n^{\log_b a} = n^k; \quad T(n) = O(n^k \cdot \log_b n)$$

$$if\ a < b^k : S(n) = const; \quad n^{\log_b a} < n^k; \quad T(n) = O(n^k)$$

$$if\ a > b^k : S(n) = \frac{n^{\log_b a}}{n^k}; \quad n^{\log_b a} > n^k; \quad T(n) = O(n^{\log_b a})$$

T(n)=aT(n/b)+f(n), f(n)=c*n^k

We just proved the Master Theorem:

The solution of the recurrence relation is:

$$T(n) = \begin{cases} O(n^{\log_b a}), if\ a > b^k \\ O(n^k \log_b n), \quad if\ a = b^k \\ O(n^k), \quad if\ a < b^k \end{cases}$$

Q6: Recurrence Relations

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

b. $T(n) = 7T\left(\frac{n}{2}\right) + n^2$

Solution 6a

- Give tight asymptotic bounds for $T(n)$ in each of the following recurrences.

a) $T(n) = 16T\left(\frac{n}{4}\right) + n^2$ $a = 16, b = 4, f(n) = n^2$

$n^2 = \Theta\left(n^{\log_4 16}\right)$, ***case* 2**:

$T(n) = \Theta(n^2 \log_2 n)$

Solution 6b

- Give tight asymptotic bounds for $T(n)$ in each of the following recurrences.

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2 \qquad a = 7, b = 2, f(n) = n^2$$

$$n^2 = O\left(n^{\log_2 7 - \varepsilon}\right), \boldsymbol{case\ 1}:$$

$$T(n) = \Theta(n^{\log_2 7})$$

Example: Mergesort

input list L of length N

if N=1 *then* return L

else do

let L_1 be the first $\left\lfloor \frac{N}{2} \right\rfloor$ elements of L

let L_2 be the last $\left\lceil \frac{N}{2} \right\rceil$ elements of L

$M_1 \leftarrow$ Mergesort (L_1)

$M_2 \leftarrow$ Mergesort (L_2)

return Merge (M_1, M_2)

Time Bounds of Mergesort

- Initial Value $T(1) = c_1$

for $N > 1$ $$T(N) \le T\left(\left\lfloor \frac{N}{2} \right\rfloor\right) + T\left(\left\lceil \frac{N}{2} \right\rceil\right) + c_2 N$$

for some constants $c_1, c_2 \ge 1$

Merge Sort

How long does mergesort take?

- Bottleneck = merging (and copying).
  - merging two files of size N/2 requires N comparisons
- T(N) = comparisons to mergesort N elements.
  - to make analysis cleaner, assume N is a power of 2

$$\mathrm{T}(N) = \begin{cases} 0 & \text{if } N = 1 \\ \underbrace{2T(N/2)}_{\text{sorting both halves}} + \underbrace{N}_{\text{merging}} & \text{otherwise} \end{cases}$$

Claim. T(N) = N $\log_2$ N.

- Note: same number of comparisons for ANY file.
  - even already sorted
- We'll give several proofs to illustrate standard techniques.

Merge Sort

$$T(N) = \begin{cases} 0 & \text{if } N = 1 \\ \underbrace{2T(N/2)}_{\text{sorting both halves}} + \underbrace{N}_{\text{merging}} & \text{otherwise} \end{cases}$$