MergeSort

Algorithm : Design & Analysis

[5]

In the last class…

- Insertion sort
- Analysis of insertion sorting algorithm
- Lower bound of local comparison based sorting algorithm
- General pattern of divide-and-conquer
- Quicksort
- Analysis of Quicksort

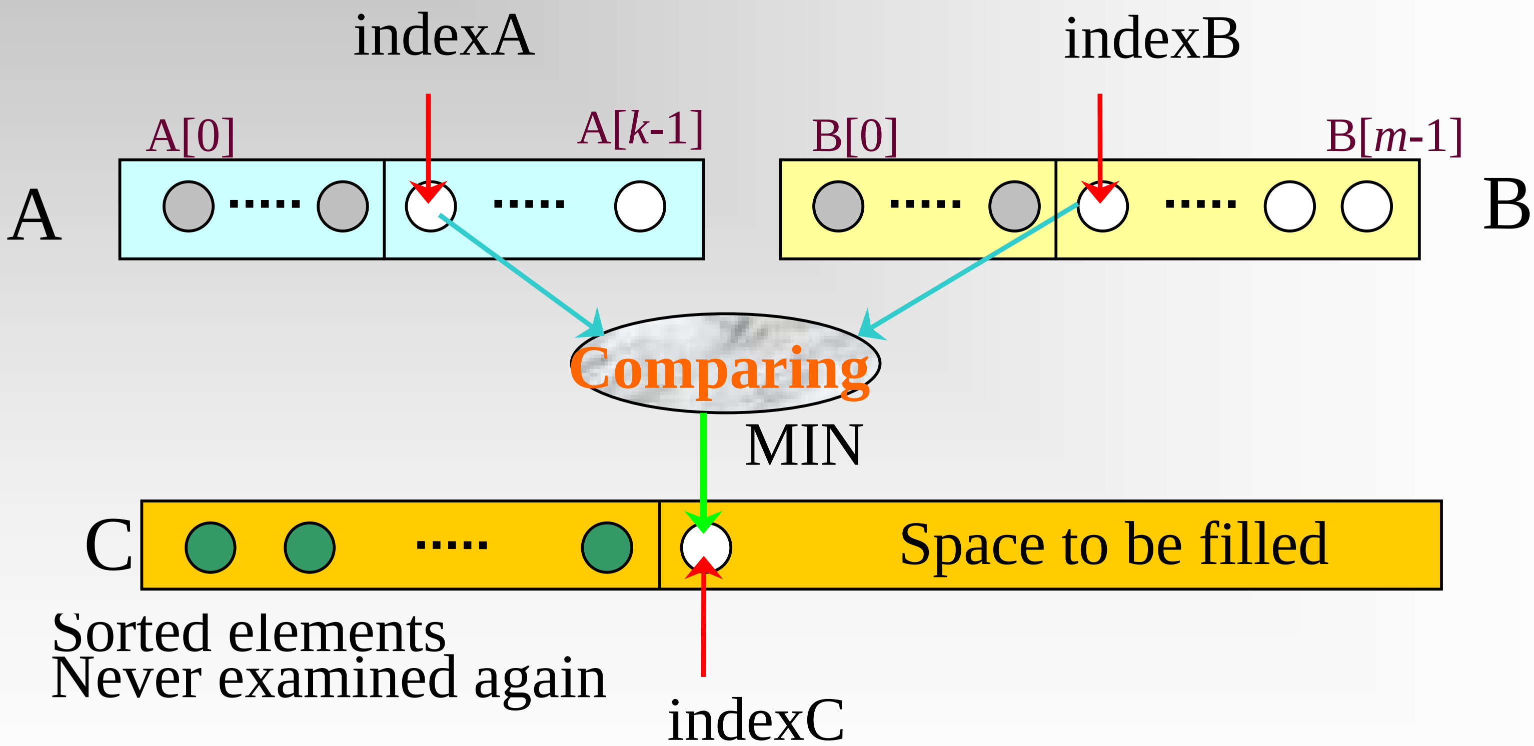
Mergesort

- Mergesort
- Worst Case Analysis of Mergesort
- Lower Bounds for Sorting by Comparison of Keys
  - Worst Case
  - Average Behavior

MergeSort: the Strategy

- Easy division
  - No comparison is done during the division
  - Minimizing the size difference between the divided subproblems
- Merging two sorted subranges
  - Using *Merge*

Merging Sorted Arrays



Merge: the Specification

- Input: Array *A* with *k* elements and *B* with *m* elements, each in nondecreasing order of their key.
- Output: *C*, an array containing $n=k+m$ elements from *A* and *B* in nondecreasing order. *C* is passed in and the algorithm fills it.

Merge: the Recursive Version

merge(A,B,C)

```
if (A is empty)
    rest of C = rest of B
else if (B is empty)
    rest of C = rest of A
else
    if (first of A ≤ first of B)
        first of C =first of A
        merge(rest of A, B, rest of C)
    else
        first of C =first of B
        merge(A, rest of B, rest of C)
return
```

Base cases

Worst Case Complexity of Merge

- Observations:
  - After each comparison, one element is inserted into Array C, ***at least*.**
  - After entering Array C, an element will never be compared again
  - After the last comparison, at least two elements have not yet been moved to Array C. So at most n-1 comparisons are done.
  - Worst case is that the last comparison is conducted between A[k-1] and B[m-1]
- In worst case, ***n-1*** comparisons are done, where $n=k+m$

Optimality of Merge

- Any algorithm to merge two sorted arrays, each containing k=m=n/2 entries, by comparison of keys, does at least n-1 comparisons in the worst case.
  - Choose keys so that:

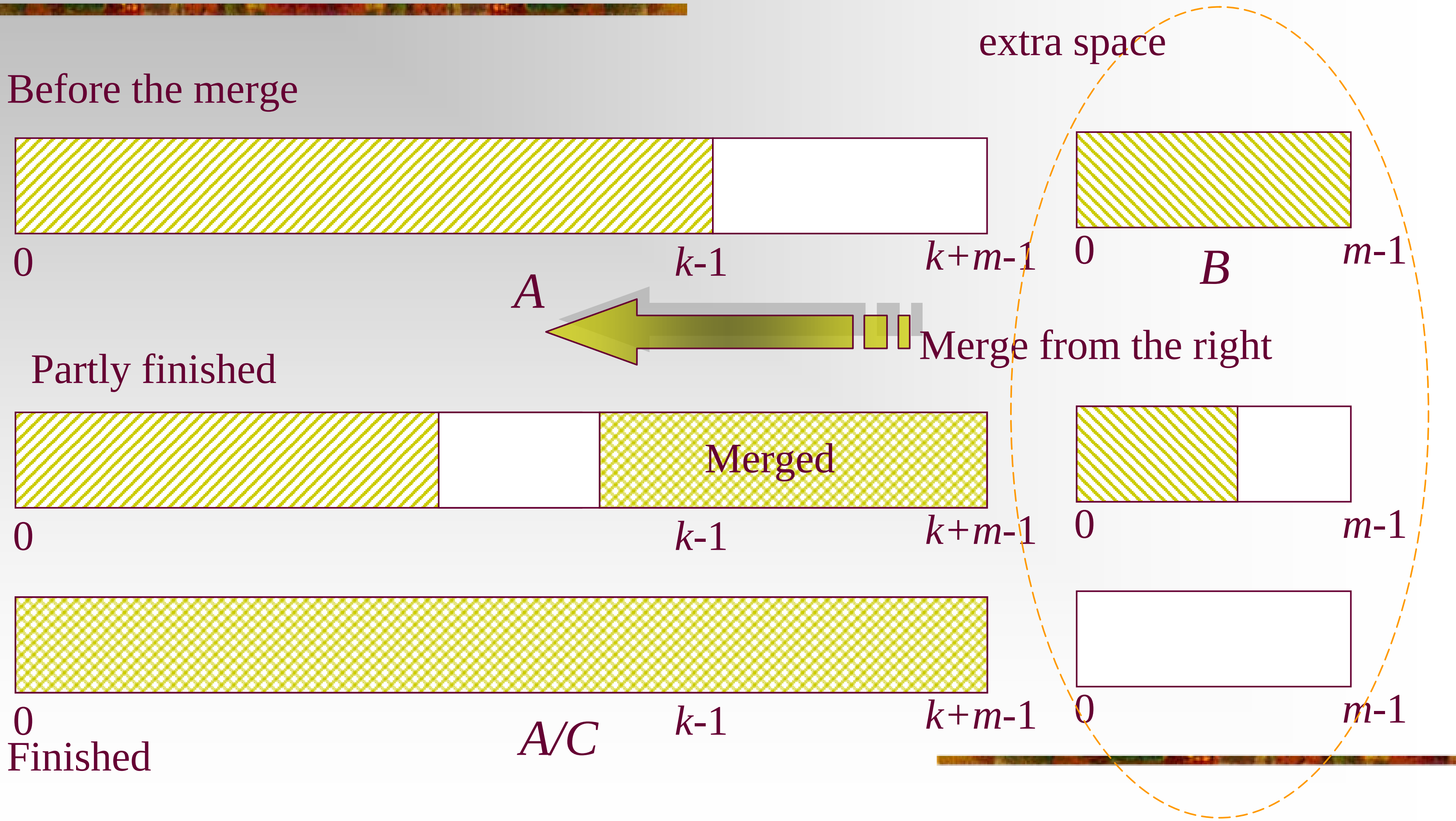
    $$b_0<a_0<b_1< a_1<\ldots<b_i<a_i<b_{i+1},\ldots,<b_{m-1}<a_{k-1}$$
  - Then the algorithm must compare a_i with b_i for every i in [0,m-1], and must compare a_i with b_{i+1} for every i in [0, m-2], so, there are n-1 comparisons.

Valid for $|k-m|\leq 1$, as well.

Space Complexity of Merge

- A algorithm is “in space”, if the extra space it has to use is in $\Theta(1)$
- Merge ***is not*** a algorithm “in space”, since it need enough extra space to store the merged sequence during the merging process.

Overlapping Arrays for Merge



MergeSort

- Input: Array *E* and indexes *first*, and *last*, such that the elements of *E*[*i*] are defined for *first*≤*i*≤*last*.
- Output: E[*first*],…,E[*last*] is a sorted rearrangement of the same elements.
- Procedure

```
void mergeSort(Element[] E, int first, int last)
    if (first<last)
        int mid=(first+last)/2;
        mergeSort(E, first, mid);
        mergeSort(E, mid+1, last);
        merge(E, first, mid, last)
    return
```

Analysis of Mergesort

- The recurrence equation for Mergesort
  - $W(n)=W(\lfloor n/2 \rfloor)+W(\lceil n/2 \rceil)+n-1$
  - $W(1)=0$

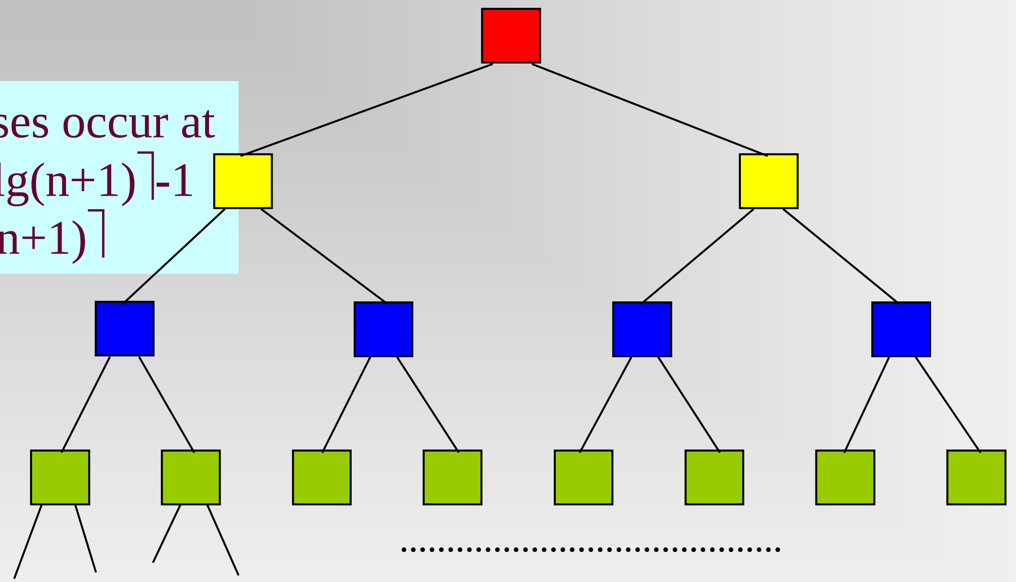
Where n=last-first+1, the size of range to be sorted

- The *Master Theorem* applies for the equation, so:

$$W(n) \in \Theta(n \log n)$$

Recursion Tree for Mergesort

Base cases occur at depth $\lceil \lg(n+1) \rceil - 1$ and $\lceil \lg(n+1) \rceil$



n-1 Level 0

n-2 Level 1

n-4 Level 2

n-8 Level 3

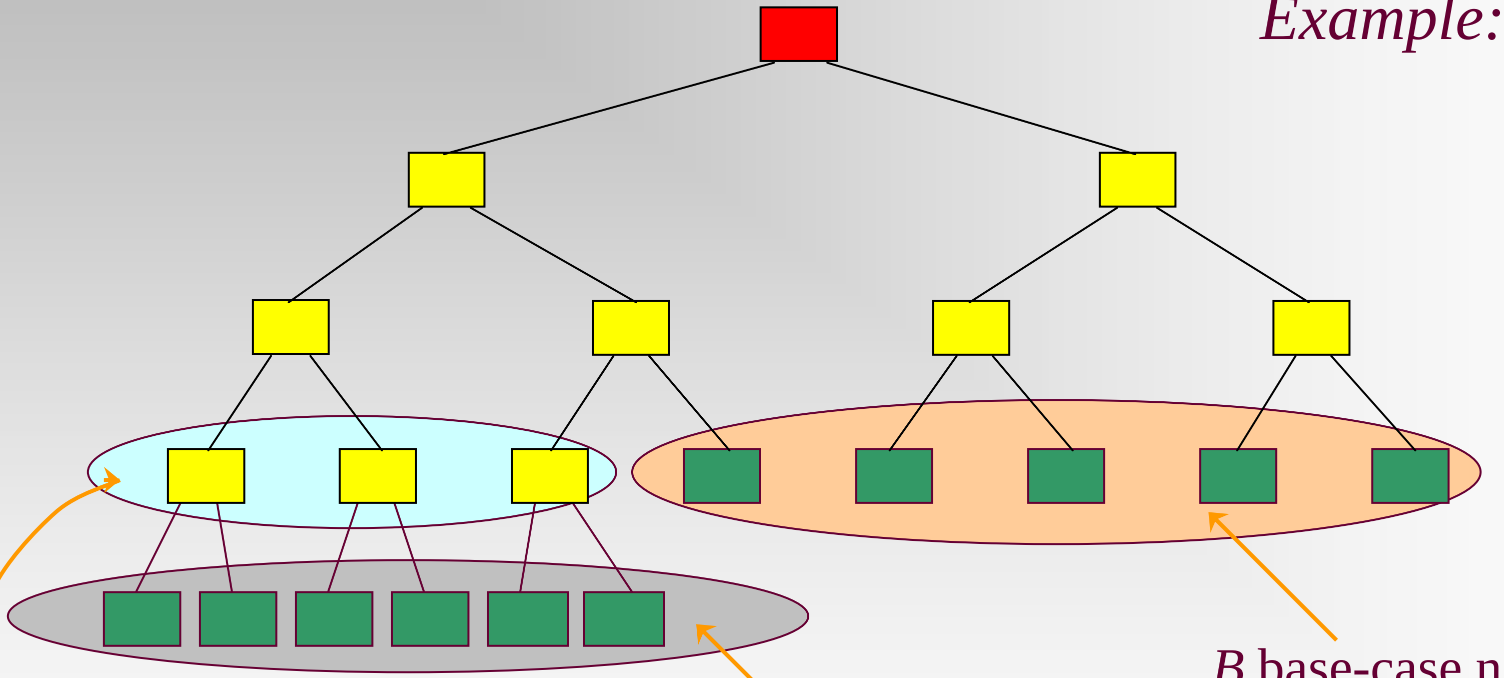
Note:
nonrecursive costs on level k is $n\text{-}2^k$ for all level without basecase node

T(n)	n-1
T(n/2)	n/2-1
T(n/4)	n/4-1
T(n/8)	n/8-1

$k/2$ may be $\lceil k/2 \rceil$ or $\lfloor k/2 \rfloor$

Non-complete Recursive Tree

Example: n=11



2^{D-1} nodes

B base-case nodes on the second lowest level

n-B base-case nodes

No nonbase-case nodes at this depth

Since each nonbase-case node has 2 children, there are $(n-B)/2$ nonbase-case nodes at depth D-1

Number of Comparison of Mergesort

- The maximum depth D of the recursive tree is $\lceil \lg(n+1) \rceil$.
- Let B base case nodes on depth D-1, and n-B on depth D, (Note: base case node has nonrecursive cost 0).
- (n-B)/2 nonbase case nodes at depth D-1, each has nonrecursive cost 1.
- So:

$$W(n) = \sum_{d=0}^{D-2} (n - 2^d) + \frac{n-B}{2} = n(D-1) - (2^{D-1} - 1) + \frac{n-B}{2}$$

$$Since\ \ (2^D - 2B) + B = n,\ that\ \ is\ \ B = 2^D - n$$

$$So, W(n) = nD - 2^D + 1$$

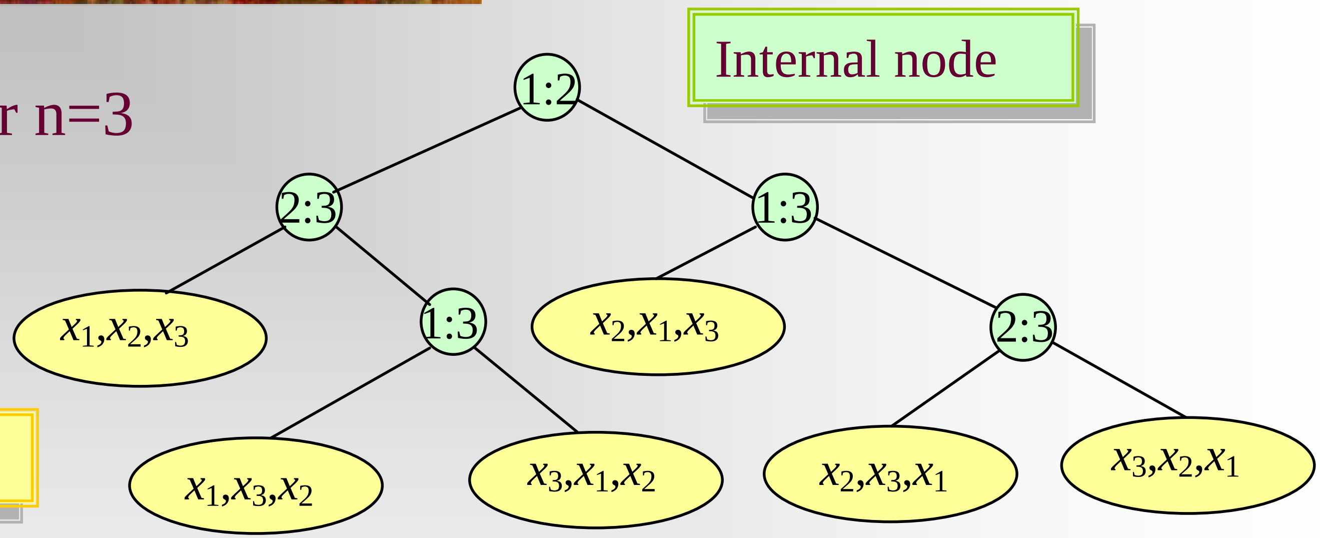
$$Let\ \ \frac{2^D}{n} = 1 + \frac{B}{n} = \alpha,\ then\ 1 \le \alpha < 2, \quad D = \lg n + \lg \alpha$$

$$So, W(n) = n\lg n - (\alpha - \lg \alpha)n + 1$$

- **$\lceil$nlg(n)-n+1$\rceil$ ≤ number of comparison ≤ $\lceil$nlg(n)-0.914n$\rceil$**

Decision Tree for Sorting

A example for n=3



- Decision tree is a 2-tree.(Assuming no same keys)
- The action of Sort on a particular input corresponds to following on path in its decision tree from the root to a leaf associated to the specific output

Characteristics of the Decision Tree

- For a sequence of n distinct elements, there are n! different permutation, so, the decision tree has at least n! leaves, and exactly n! leaves can be reached from the root. So, for the purpose of lower bounds evaluation, we use trees with exactly n! leaves.
- The number of comparison done in the ***worst case*** is the height of the tree.
- The ***average*** number of comparison done is the average of the lengths of all paths from the root to a leaf.

Lower Bound for Worst Case

- ***Theorem***: Any algorithm to sort n items by comparisons of keys must do at least $\lceil \lg n! \rceil$, or approximately $\lceil n\lg n - 1.443n \rceil$, key comparisons in the worst case.
  - Note: Let $L = n!$, which is the number of leaves, then $L \le 2^h$, where h is the height of the tree, that is $h \ge \lceil \lg L \rceil = \lceil \lg n! \rceil$
  - For the asymptotic behavior:

$$\lg(n!) \ge \lg[n(n-1)\ldots\left(\left\lceil \frac{n}{2} \right\rceil\right)] \ge \lg\left(\frac{n}{2}\right)^{\frac{n}{2}} = \frac{n}{2}\lg\left(\frac{n}{2}\right) \in \Theta(n \lg n)$$

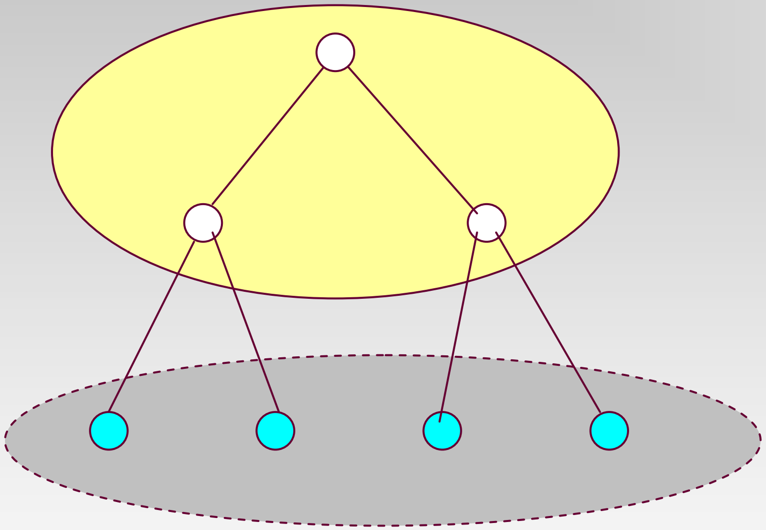
derived using:

$$\lg n! = \sum_{j=1}^{n} \lg(j)$$

2-Tree

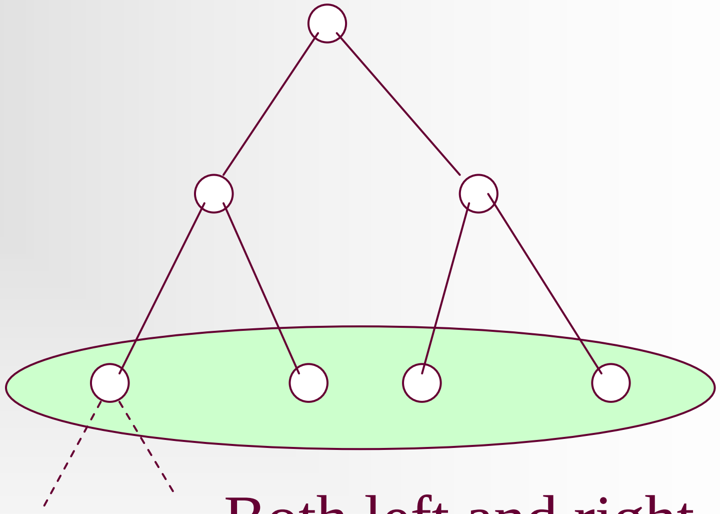
- 2-Tree

internal nodes



external nodes
no child
any type

- Common Binary Tree



Both left and right
children of these
nodes are empty tree

External Path Length(EPL)

- The ***EPL* of a 2-tree** *t* is defined as follows:
  - [Base case] 0 for a single external node
  - [Recursion] *t* is non-leaf with sub-trees *L* and *R*, then the sum of:
    - the external path length of *L*;
    - the number of external node of *L*;
    - the external path length of *R*;
    - the number of external node of *R*;

Properties of EPL

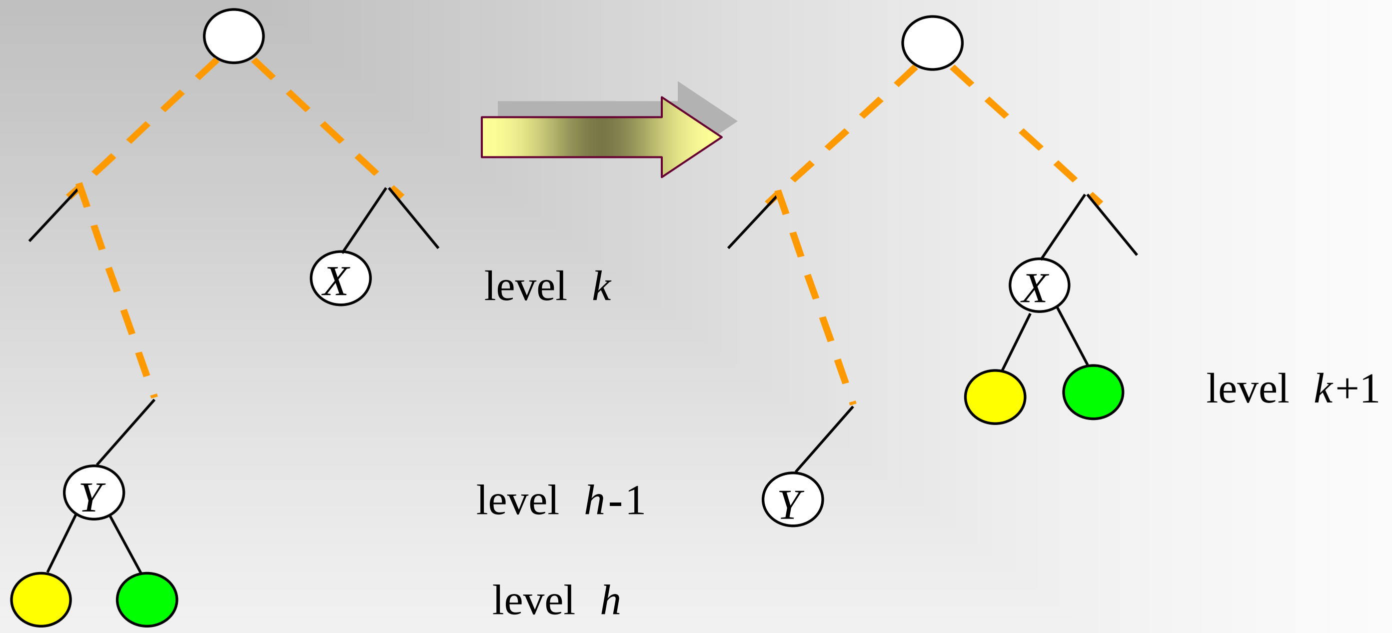
- Let t is a 2-tree, then the *epl* of t is the sum of the paths from the root to each external node.
- *epl* $\geq m\lg(m)$, where m is the number of external nodes in t
  - $epl=epl_L+epl_R+m\geq m_L\lg(m_L)+m_R\lg(m_R)+m$,
  - note $f(x)+f(y)\geq 2f((x+y)/2)$ for $f(x)=x\lg x$
  - so,
    $epl \geq 2((m_L+m_R)/2)\lg((m_L+m_R)/2)+m = m(\lg(m)-1)+m$
    $=m\lg m$.

Lower Bound for Average Behavior

- Since a decision tree with L leaves is a 2-tree, the average path length from the root to a leaf is $\frac{epl}{L}$.
- The trees that minimize *epl* are as balanced as possible.

  to be proved
- Recall that $epl \geq L\lg(L)$.
- **Theorem**: The average number of comparison done by an algorithm to sort n items by comparison of keys is at least $\lg(n!)$, which is about $n\lg n - 1.443n$.

Reducing External Path Length



Assuming that *h*-*k*>1, when calculating *epl*, *h*+*h*+*k* is replaced by (*h*-1)+2(*k*+1). The net change in *epl* is *k*-*h*+1<0, that is, the *epl* decreases.

So, more balanced 2-tree has smaller epl.

Mergesort Has Optimal Average Performance

- We have proved that the average number of comparisons done by an algorithm to sort n items by comparison of keys is at least about $n\lg n$-1.443n
- The **worst** complexity of mergesort is in $\Theta(n\lg n)$
- But, the average performance can not be worse the the worst case performance.
- So, mergesort is optimal as for its average performance.

Home Assignment

- pp.212-
  - 4.24
  - 4.25
  - 4.27
  - 4.29
  - 4.30
  - 4.32