Solving Recurrence (Part 2)
Divide-and-Conquer

CS-6^{th}

Instructor: Dr. Ayesha Enayet

Substitution Method

- The substitution method comprises two steps:
  1. Guess the form of the solution using symbolic constants.
  2. Use mathematical induction to show that the solution works, and find the constants.

Show that the $T(n)=T(n-1)+n$ has solution $O(n^2)$

- $T(n)=T(n-1)+n \rightarrow$ eq1
- Let’s take/guess $T(n)<=cn^2$
- $T(n-1)<=c(n-1)^2 \rightarrow$ eq2
- Substitute eq2 in eq1
- $T(n)<= c(n-1)^2+n$
- $=c(n^2-2n+1)+n$
- $=cn^2-2cn+c+n$
- $=cn^2 -(2cn-c-n)$
- $<=cn^2$

the last step holds as long as $(n(2c-1)-c)\geq 0$. We can use this to specify c and $n0$.
If we pick $c=1$, then we need $n-1\geq 0$. Which is possible as long as $n\geq 1=n0$.

Show that $T(n)=T(n/2)+\Theta(1)$ has solution $T(n)=O(\lg n)$

- $T(n)=T(n/2)+\Theta(1) \rightarrow$ eq1
- Let’s take/guess $T(n)<=c\lg n$
- $T(n/2)<=c\lg(n/2) \rightarrow$ eq2
- Substitute eq2 in eq1
- $T(n)<=c\lg(n/2) +\Theta(1)$
- $=c(\lg n-1)+ \Theta(1)$
- $=c\lg n-c+ \Theta(1)$
- $=c\lg n-(c-\Theta(1))$ we must choose c large enough to dominate lower order term $\Theta(1)$
- $T(n)<=c\lg n$

Exercise

- Show that $T(n)=4T(n/2)+n^2$ has solution $T(n)=\Omega(n^2 \lg n)$

Show that $T(n)=4T(n/2)+n^2$ has solution $T(n)=\Omega(n^2 \lg n)$

- $T(n)=4T(n/2)+n^2 \rightarrow$ eq1
- $T(n)>=c.n^2 \lg n$
- $T(n/2)>= c.(n/2)^2 \lg(n/2) \rightarrow$ eq2
- Substitute eq2 in eq1
- $T(n)>=4.c.n^2/4.\lg(n/2)+n^2$
- $=c.n^2(\lg n-1)+n^2$
- $=c.n^2 \lg n- c.n^2 +n^2$
- $=c.n^2 \lg n+ (1-c)\ n^2$ $\qquad$ $c<=1$
- $>= c.n^2 \lg n$

Back substitution: Find solution for $T(n)=T(n/2)+\Theta(1)$

- $T(n)=T(n/2)+\Theta(1) \rightarrow$ eq1
- $T(n/2)=T(n/4)+ \Theta(1) \rightarrow$ eq2 Substitute eq2 in eq1
- $T(n)= (T(n/4)+ \Theta(1)) +\Theta(1)$
- $T(n)=(T(n/8)+ \Theta(1))+\Theta(1) +\Theta(1)$
- $T(n)=(T(n/2^k))+k(\Theta(1))$
- $n/ 2^k=1 \rightarrow$ Base case /stopping condition
- $n= 2^k$
- $K=\log n$

$T(n)=(T(n/2^k))+k(\Theta(1))$
$T(n)=1+\lg n(\Theta(1))$
$T(n)=O(\lg(n))$

Find solution of $T(n)=T(n/2)+n$

- $T(n)=T(n/2)+n \rightarrow$ eq1
- $T(n/2)=T(n/4)+n/2 \rightarrow$ eq2 substitute eq2 in eq2
- $T(n)=[T(n/2^2)+n/2]+n$
- $T(n)=[T(n/2^3)+n/4]+n/2]+n$
- $T(n)=T(n/2^k)+n/2^{k-1}+ n/2^{k-2}\ldots n$
- Considering the base case
- $T(n/2^k)=1$
- $T(n)=1+n(1/2^{k-1}+ 1/2^{k-2}\ldots 1)$

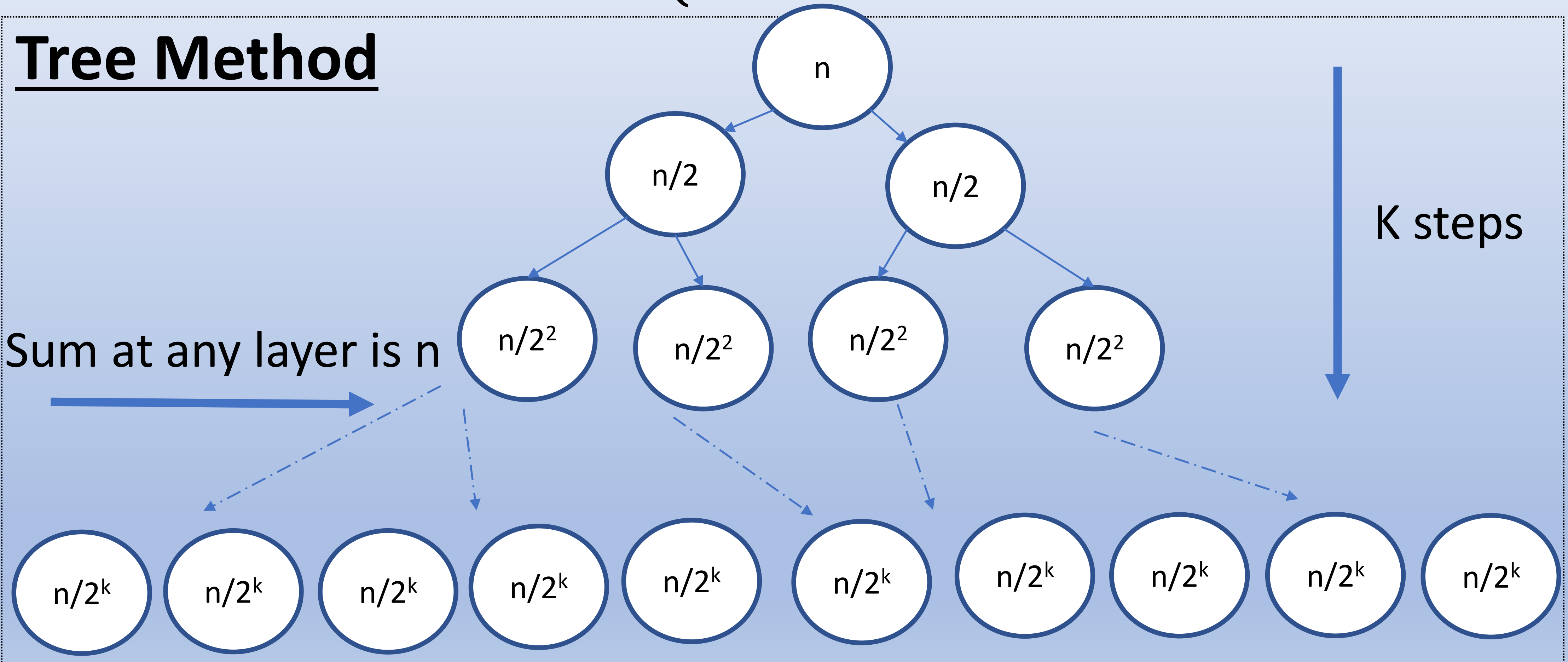
$T(n)=1+n(1+1)$
$T(n)=1+n(2)$
$T(n)=2n+1$
$T(n)=O(n)$

Divide-and-Conquer Algorithms

They break the problem into several subproblems that are similar to the original problem but smaller in size, solve the subproblems recursively, and then combine these solutions to create a solution to the original problem.

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

Tree Method



Solution

- $T(n)=n.k+n \rightarrow$ eq1
- $n/2^k=1$
- $k=\log n \rightarrow$ eq2
- $T(n)=n\lg n+n$
- $=O(n\lg n)$
- Or (alternate derivation) c.n is the cost at each level and it would take lgn+1 steps to reach the leave nodes
- $T(n)=cn(\lg n+1)=cn\lg n+cn=O(n\lg n)$ [reference Dr. Shah's handwritten notes)

Recurrence Relation: $\begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$

<u>Back Substitution Method</u>

- $T(n)=2T(n/2)+n \rightarrow$ eq1
- $T(n/2)=2T(n/2^2)+n/2 \rightarrow$ eq2
- Substitute equation 2 in 1
- $T(n)=2[2T(n/2^2)+n/2]+n$
- $T(n)=2[2T(n/2^2)+n/2]+n$
- $T(n)=2^2T(n/2^2)+n+n$
- $T(n)=2^2T(n/2^2)+2n \rightarrow$ eq3

- $T(n/2^2)=2T(n/2^3)+n/2^2 \rightarrow$ eq4
- Substitute eq4 in eq3
- $T(n)=2^2[2T(n/2^3)+n/2^2]+2n$
- $T(n)=2^3T(n/2^3)+n+2n$
- $T(n)=2^3T(n/2^3)+3n$
- $T(n)=2^kT(n/2^k)+3n$
- $T(n)=2^k(T(1))+kn$
- $T(n)=n.1+kn$
- $T(n)=n+nlgn$
- $=O(nlgn)$

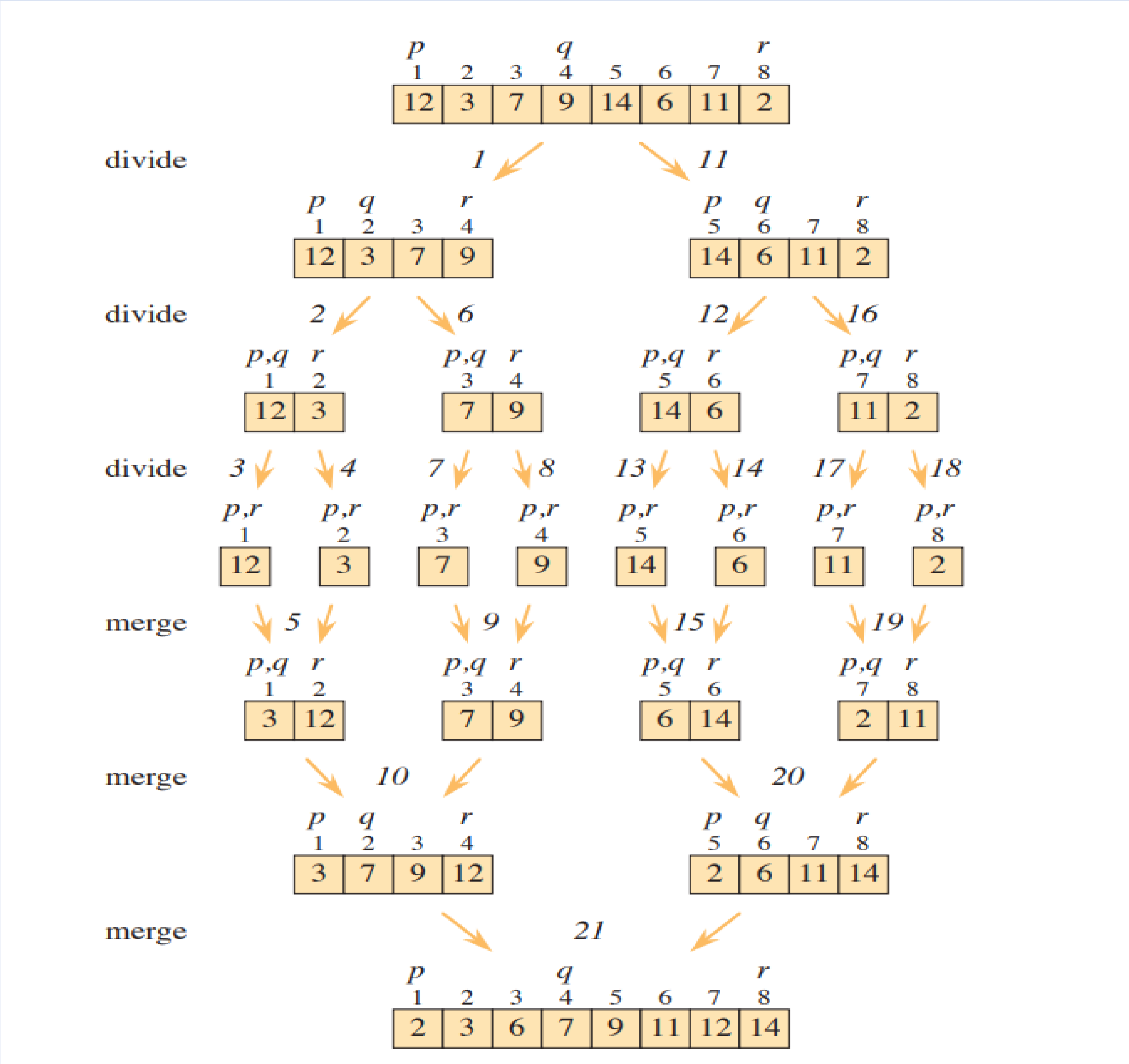
Assume $=T(n/2^k)=T(1)$

$n/2^k=1$

$n=2^k$

$K=lgn$

Merge Sort (Divide-and-Conquer)



Analysis of merge sort

Here's how to set up the recurrence for $T(n)$, the worst-case running time of merge sort on n numbers.

Divide: The divide step just computes the middle of the subarray, which takes constant time. Thus, $D(n) = \Theta(1)$.

Conquer: Recursively solving two subproblems, each of size $n/2$, contributes $2T(n/2)$ to the running time (ignoring the floors and ceilings, as we discussed).

Combine: Since the MERGE procedure on an n-element subarray takes $\Theta(n)$ time, we have $C(n) = \Theta(n)$.

Analysis of Merge sort

- In the following recurrence relation a is number of subproblems and n/b is size of the subproblem.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n < n_0 , \\ D(n) + aT(n/b) + C(n) & \text{otherwise} . \end{cases}$$

- T(n)=T(n/2)+ Θ(n) (check slide 7 onwards for the analysis)