Design and Analysis of Algorithms

Analysis of Recursive Algorithms

Analyzing the Time Efficiency of Recursive Algorithms

- Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed.
- Solve the recurrence or, at least, ascertain the order of growth of its solution

Recursive Function

- Compute the factorial function $F(n) = n!$ for an arbitrary nonnegative integer n. Since
  - $n! = 1 \ldots\ldots (n-1) . n = (n-1)! . n$ for $n \geq 1$
  - and $0! = 1$ by definition,
  - we can compute $F(n) = F(n-1) . n$ with the following recursive algorithm.

- **ALGORITHM F(n)**
- //Input: A nonnegative integer n
- //Output: The value of n!
- if n = 0 return 1
- else return F(n − 1) ∗ n

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

- $T(n) = T(n-1) + 1$
- $T(n-1) = T(n-2) + 1$
- $T(n-2) = T(n-3) + 1$
- $T(n-3) = T(n-4) + 1$
- $T(n-4) = T(n-5) + 1$

- By Substitution
- $T(n) = T(n-2) + 1 + 1$
- $T(n) = T(n-3) + 3$
- $T(n) = T(n-4) + 4$
- $T(n) = T(n-5) + 5$
- …
- $T(n) = T(n-i) + i$

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

- For $T(n-i) = T(0)$
- $i = n$

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

Digits

- The following algorithm finds the number of binary digits in the binary representation of a positive decimal integer.

- **ALGORITHM BinRec(n)**
- //Input: A positive decimal integer n
- //Output: The number of binary digits in n’s binary representation
- if
  - n = 1 return 1
- else
  - return BinRec(n/2) + 1

Complexity

- **BinRec(n)**
- if
  - n = 1 return 1
- else
  - return BinRec(n/2) + 1

- $T(n) = T\left(\frac{n}{2}\right) + 1$
- $T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1$
- $T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1$
- $T\left(\frac{n}{8}\right) = T\left(\frac{n}{16}\right) + 1$
- $T\left(\frac{n}{16}\right) = T\left(\frac{n}{32}\right) + 1$

Complexity

- $T(n) = T\left(\frac{n}{2}\right) + 1$
- $T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1$
- $T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1$
- $T\left(\frac{n}{8}\right) = T\left(\frac{n}{16}\right) + 1$
- $T\left(\frac{n}{16}\right) = T\left(\frac{n}{32}\right) + 1$

- By Substitution
- $T(n) = T\left(\frac{n}{4}\right) + 2$
- $T(n) = T\left(\frac{n}{8}\right) + 3$
- $T(n) = T\left(\frac{n}{16}\right) + 4$
- $T(n) = T\left(\frac{n}{32}\right) + 5$
- …
- $T(n) = T\left(\frac{n}{i}\right) + \log i$

Complexity

- $T(n) = T\left(\frac{n}{i}\right) + \log i$
- For $T\left(\frac{n}{i}\right) = T(1)$
- $i = n$
- $T(n) = T\left(\frac{n}{n}\right) + \log n$
- $T(n) = 1 + \log n$
- $T(n) = \log n + 1 = \Theta(\log n)$

Merge Sort Algorithm

```
MergeSort(A, i, j)
    if j > i then
        mid ← (i  + j)/2
        MergeSort(A, i, mid )
        MergeSort(A, mid + 1, j )
        Merge(A, i, mid,  j )
```

- $T(n) = 2T\left(\frac{n}{2}\right) + n$
- $T(n/2) = 2T\left(\frac{n}{4}\right) + n/2$
- $T(n/4) = 2T\left(\frac{n}{8}\right) + n/4$
- $T(n/8) = 2T\left(\frac{n}{16}\right) + n/8$
- $T(n/16) = 2T\left(\frac{n}{32}\right) + \frac{n}{16}$

Telescoping Sum

- $T(n) = 2T\left(\frac{n}{2}\right) + n$
- $2T(n/2) = 2T\left(\frac{n}{4}\right) + 2 * n/2$
- $4T(n/4) = 2T\left(\frac{n}{8}\right) + 4 * n/4$
- $8T(n/8) = 2T\left(\frac{n}{16}\right) + 8 * n/8$
- $16T(n/16) = 2T\left(\frac{n}{32}\right) + 16 * \frac{n}{16}$
- $\ldots$
- $\frac{n}{2}T\left(\frac{n}{\frac{n}{2}}\right) = nT\left(\frac{n}{n}\right) + \frac{n}{2}\left(\frac{n}{\frac{n}{2}}\right)$

- Sum of equations
- $T(n) + 2T\left(\frac{n}{2}\right) + 4T\left(\frac{n}{4}\right) + 8T\left(\frac{n}{8}\right) + \ldots + \frac{n}{2}T\left(\frac{n}{\frac{n}{2}}\right) = 2T\left(\frac{n}{2}\right) + 4T\left(\frac{n}{4}\right) + 8T\left(\frac{n}{8}\right) + \ldots + \frac{n}{2}T\left(\frac{n}{n}\right) + n + n + n + \ldots + n$
- $T(n) = nT(1) + n + n + n + \ldots + n$
- $T(n) = n\log n$

Iterative Substitution

- $T(n) = 2T\left(\frac{n}{2}\right) + n$
- $T(n/2) = 2T\left(\frac{n}{4}\right) + n/2$
- $T(n/4) = 2T\left(\frac{n}{8}\right) + n/4$
- $T(n/8) = 2T\left(\frac{n}{16}\right) + n/8$
- $T(n/16) = 2T\left(\frac{n}{32}\right) + \frac{n}{16}$

- By Substitution
- $T(n) = 4T\left(\frac{n}{4}\right) + n + n$
- $T(n) = 8T\left(\frac{n}{8}\right) + n + n + n$
- $T(n) = 16T\left(\frac{n}{16}\right) + 4n$
- $T(n) = 32T\left(\frac{n}{32}\right) + 5n$
- …
- $T(n) = iT\left(\frac{n}{i}\right) + \log i\,(n)$

Iterative Substitution

- $T(n) = iT\left(\frac{n}{i}\right) + \log i\,(n)$
- For $T(n/i) = T(1)$
- $i = n$

- $T(n) = iT\left(\frac{n}{i}\right) + \log i\,(n)$
- $T(n) = nT(1) + \log n\,(n)$
- $T(n) = n + n\log n$
- $T(n) = n\log n + n = \Theta(n\log n)$

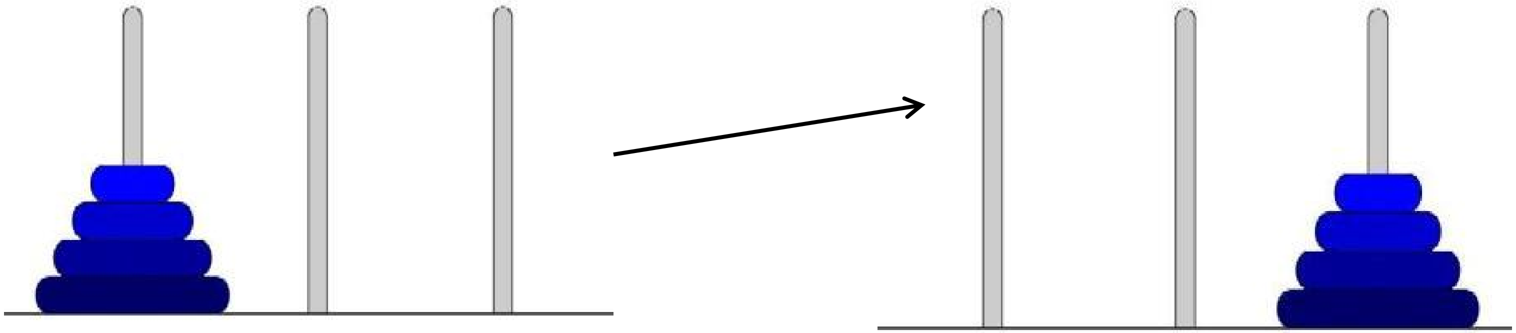
Quick Sort Algorithm

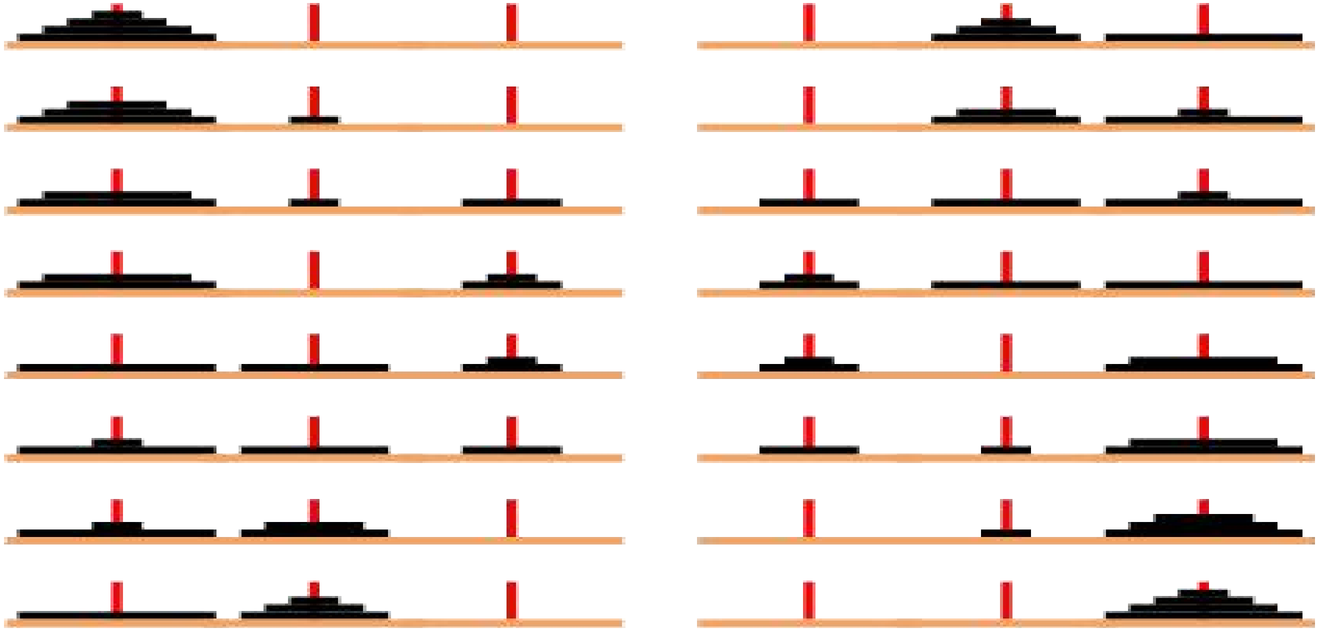
```
QuickSort(A, left, right) {
  if (right > left) then {
      pivot = Partition(A, left, right);
      QuickSort(A, left, pivot-1);
      QuickSort(A, pivot+1, right);
  }
}
```

- Best Case
- $T(n) = 2T\left(\frac{n}{2}\right) + n$
- Worst Case
- $T(n) = T(n-1) + T(1) + n \cong \mathrm{T(n-1) + n}$

Tower of Hanoi puzzle

- The game starts by having few discs stacked in increasing order of size. The number of discs can vary, but there are <u>only</u> three pegs.





Tower of Hanoi puzzle

Recursive Solution for the Tower of Hanoi with algorithm

Let’s call the three peg Src(Source), Aux(Auxiliary) and st(Destination).

1) Move the top N – 1 disks from the Source to Auxiliary tower
2) Move the Nth disk from Source to Destination tower
3) Move the N – 1 disks from Auxiliary tower to Destination tower. Transferring the top N – 1 disks from Source to Auxiliary tower can again be thought of as a fresh problem and can be solved in the same manner.

Tower of Hanoi puzzle

- $M(n) = M(n - 1) + 1 + M(n - 1)$ *for* $n > 1$.
- With the obvious initial condition $M(1) = 1$, *we have the following recurrence* relation for the number of moves $M(n)$:
  - $M(n) = 2M(n - 1) + 1$ *for* $n > 1$,
  - $M(1) = 1$.
- We solve this recurrence by the same method of backward substitutions:

Tower of Hanoi puzzle

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

Design and Analysis of Algorithm

Recurrence Tree

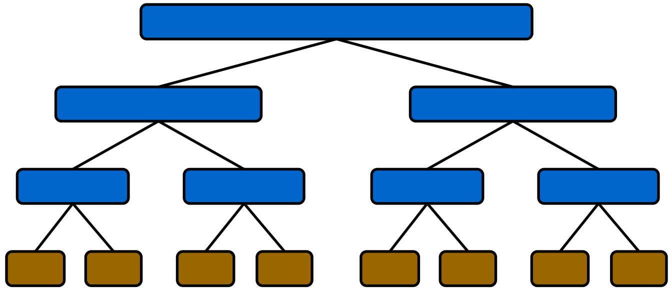
Complexity Analysis

- General Divide and Conquer
- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- *Merge Sort*
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

- Draw the recursion tree for the recurrence relation and look for a pattern:

$$T(n) = \begin{cases} 0 & \text{if } n < 2 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

depth h	T's	size
0	1	***n***
1	2	***n***/2
i	2^i	***n***/2^i
…	…	…



Time
n
n
n
…

Total time = ***n*** + ***n*** log ***n***

(last level plus all previous levels)

Master Theorem

Master Theorem

Master Theorem Formal

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \ where \ b > 1$$

1. $T(n) = \Theta(f(n)) \ if \ f(n) = n^{\log_b a+\epsilon}$
2. $T(n) = \Theta\left(f(n)\log_b^{k+1} n\right) \ if \ f(n) = n^{\log_b a}\log_b^k n$
3. $T(n) = \Theta\left(n^{\log_b a}\right) \ if \ f(n) = n^{\log_b a-\epsilon}$

$$\epsilon > 0$$

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

1. $T(n) = \Theta(f(n))\ if\ f(n) > n^{\log_b a}$
2. $T(n) = \Theta\left(n^{\log_b a} \log_b^{k+1} n\right)\ if\ f(n) = n^{\log_b a} \log_b^k n$
3. $T(n) = \Theta\left(n^{\log_b a}\right)\ if\ f(n) < n^{\log_b a}$

Master Theorem: f(n) is polynomial

$$T(n) = aT\left(\frac{n}{b}\right) + n^d$$

- Comparison part
- $f(n) = n^{\log_b a}$: $<$ and $>$ $for\ other\ cases$
- $f(n) = n^{\mathrm{d}}$
- $n^{\mathrm{d}} = n^{\log_b a}$
- $\mathrm{d} = \log_b a$

Master Theorem: f(n) is polynomial

- $T(n) = aT\left(\frac{n}{b}\right) + n^d$

1. $T(n) = \Theta(n^d)\ if\ d > \log_b a$
2. $T(n) = \Theta(f(n)\log_b n)\ if\ d = \log_b a$
3. $T(n) = \Theta(n^{\log_b a})\ if\ d < \log_b a$

- $T(n) = \Theta(f(n)\log n)\ if\ f(n) = n^{\log_b a}\log_b^k n$
- $T(n) = \Theta\left(n^{\log_b a}\log_b^{k+1} n\right)\ if\ f(n) = n^{\log_b a}\log_b^k n$

Master Theorem f(n) is log-polynomial

$$T(n) = aT\left(\frac{n}{b}\right) + n^d \log_b^k n$$

1. $T(n) = \Theta\left(n^{\log_b a}\right)\ if\ d < \log_b a$
2. $T(n) = \Theta\left(f(n)\log_b^{k+1} n\right)\ if\ f(n) = n^{\log_b a}\log_b^k n$

4.6 Master Theorem Proof

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- $T(n) = a^2T\left(\frac{n}{b^2}\right) + af(n/b) + f(n)$
- $T(n) = a^3T\left(\frac{n}{b^3}\right) + a^2f(n/b^2) + af(n/b) + f(n)$
- $T(n) = a^3T\left(\frac{n}{b^3}\right) + \sum_{j=0}^{3-1} a^j f(n/b^j)$
- …
- $T(n) = a^iT\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j f(n/b^j)$

4.6 Master Theorem Proof

- $T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j f(n/b^j)$
- $if\ \ f(n) = n^{\log_b a - \epsilon}$
- $T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}$
- $T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} \frac{a^j n^{\log_b a - \epsilon}}{b^{\mathrm{j}\log_b a - \epsilon}}$

4.6 Master Theorem Proof

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

4.6 Master Theorem Proof

- $T(n) = a^i T\left(\frac{n}{b^i}\right) + n^{\log_b a - \epsilon} \sum_{j=0}^{i-1} b^{\epsilon j}$
- $T\left(\frac{n}{b^i}\right) = T(1)$
- $\frac{n}{b^i} = 1$
- $i = \log_b n$
- $T(n) = a^i T\left(\frac{n}{b^i}\right) + \frac{n^{\log_b a - \epsilon}(b^{\epsilon i} - 1)}{b^{\epsilon} - 1}$
- By using value of i
- $T(n) = a^{\log_b n} T\left(\frac{n}{n}\right) + n^{\log_b a - \epsilon}(b^{\epsilon \log_b n} - 1)$

4.6 Master Theorem Proof

- $T(n) = a^{\log_b n} T\left(\frac{n}{n}\right) + n^{\log_b a - \epsilon}(b^{\epsilon \log_b n} - 1)$
- $T(n) = a^{\log_b n} + n^{\log_b a - \epsilon}(n^{\epsilon} - 1)$
- $T(n) = n^{\log_b a} + n^{\log_b a} - n^{\log_b a - \epsilon}$
- $T(n) = n^{\log_b a} + n^{\log_b a}(1 - \frac{1}{n^{\epsilon}})$
- $T(n) = \Theta(n^{\log_b a})$

4.6 Master Theorem: Case 2

- $T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j f(n/b^j)$
- $if\ \ f(n) = n^{\log_b a+\epsilon}$
- $T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a+\epsilon}$
- $T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} \frac{a^j n^{\log_b a+\epsilon}}{b^{\mathrm{j}\log_b a+\epsilon}}$

4.6 Master Theorem

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

4.6 Master Theorem

- $T\left(\frac{n}{b^i}\right) = T(1)$
- $\frac{n}{b^i} = 1$
- $i = \log_b n$
- $T(n) = a^i T\left(\frac{n}{b^i}\right) + n^{\log_b a - \epsilon} \sum_{j=0}^{i-1} 1/b^{\epsilon j}$
- $T(n) = a^i T\left(\frac{n}{b^i}\right) + n^{\log_b a + \epsilon}$
- By using value of i
- $T(n) = a^{\log_b n} T\left(\frac{n}{n}\right) + n^{\log_b a + \epsilon}$

4.6 Master Theorem

- $T(n) = a^{\log_b n} T\left(\frac{n}{n}\right) + n^{\log_b a+\epsilon}$
- $T(n) = a^{\log_b n} + n^{\log_b a+\epsilon}$
- $T(n) = \Theta(n^{\log_b a+\epsilon})$
- $T(n) = \Theta(f(n))$

Case 3

- Home Task