Discussion 3: Merge sort, Quick Sort, together with Divide and Conquer

CS101 Fall 2024

CS101 Course Team

Oct 2024

Merge Sort

Core idea: Merge sorted sequences together to sort the whole array.
Recursion relation: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.
Time complexity: $\Theta(n \log n)$.
Space complexity: $\Theta(n)$. (Not in-place)
Stable: Rely on breaking ties by choosing the front one.

Quick Sort

Core idea: Distinguish those greater than pivot and less than pivot.

Recursion relation

Choosing i-th largest as pivot: $T(n) = T(i) + T(n-i) + \Theta(n)$.
Randomized quick-sort: choosing each as pivot uniformly.
A deterministic way: Median-of-Median.

Time complexity: $\Theta(n \log n)$ (Average case), $\Theta(n^2)$ (Worst case).
Space complexity: $\Theta(1)$. (In-place).
Not stable: Choosing non-distinct element as pivot.

Variant algorithms

- Counting Inversions:
  Trivial idea: Check all (i, j) pairs whether $a_i > a_j$.
  Based on merge-sort: Count "Cross-Inversions" when merging.
- N-th element:
  Trivial idea: Sort then find, takes $\Theta(n \log n)$.
  Based on quick-sort: Consider the number of 2 parts after dividing.

Master Theorem

Master Theorem

Given $T(n) = aT(\frac{n}{b}) + f(n)$, $T(1) = 1$.

- $f(n) = o(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a})$
- $f(n) = \Theta(n^{\log_b a} log^k n) \Rightarrow T(n) = n^{log_b a} \log^{k+1} n$.
- $f(n) = \Omega(n^{\log_b a} + \epsilon)$ with some $\epsilon > 0 \Rightarrow T(n) = \Theta(f(n))$

Example:

1. $T(n) = 2T(\frac{n}{2}) + \Theta(n)$:
   $a = 2, b = 2, n^{\log_b a} = \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$
2. $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$:
   $a = 7, b = 2, n^{\log_b a} = \omega(n^2) \Rightarrow T(n) = \Theta(n^{\log_2 7})$

Recursion Tree & Expansion

Recall

$T(n) = 2T(\frac{n}{2}) + \Theta(n) \Rightarrow \exists$constant c, $T(n) \leq 2T(\frac{n}{2}) + cn$.

$$\begin{aligned} T(n) &\leq 2[2T(\frac{n}{4}) + c\frac{n}{2}] + cn = 4T(\frac{n}{4}) + 2cn \\ &\leq 4[2T(\frac{n}{8}) + c\frac{n}{4}] + 2cn = 8T(\frac{n}{8}) + 3cn. \\ \Rightarrow T(n) &\leq 2^k T(\frac{n}{2^k}) + kcn \end{aligned}$$

Substitute by $k = \log_2 n$, we got $T(n) \leq nT(1) + cn\log_2 n = O(n \log_2 n)$.

Recursion Tree: Visualization Understanding of Expansion.
Time = depth $\times$ (average) time for each layer.

Mathematical Skill

23 MidTerm

$T(n) = T(0.99n) + \Theta(1)$ where $T(0) = 0$ and $T(1) = 1$.

From senior high school: $T(n) = f(T(n-1), \cdots)$
Substitution: $t(m) = T(0.99^{-m})$ i.e. $m = \log_{\frac{1}{0.99}} n$.
Then $t(m) = t(m-1) + \Theta(1) \Rightarrow t(m) = \Theta(m) \Rightarrow T(n) = \Theta(\log n)$.

Another

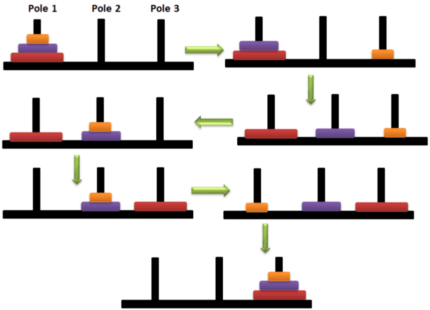
$T(n) = aT(\frac{n}{b}) + f(n)$.

Other Algorithms using divide and conquer

- Strassen Matrix Multiplication: $O(n^{\log_2 7})$. for $n \times n$ matrix (Recent: $n^\omega, \omega < 2.371339$, `https://arxiv.org/abs/2404.16349`)
- Fast Fourier Transform (FFT): $O(n \log n)$ for n-th degree polynomial.

Example: Hanoi Puzzle

1. Moving disks over 3 rods.
2. Only move one disk each time.
3. Never place a larger disk on top of a smaller one.
4. Example of moving 3 disks in 7 steps.



Example: Hanoi Puzzle

1. Always obey the rule that the larger disks remain on the bottom of the stack.
2. For the problem with n disks, If we ignore the largest disks, the problem will be reduced to size of $n-1$. If we successfully solve the problem with $n1$, that means we move all $n1$ towers in one stack, then we can move the biggest disk to the right place.
3. we can find it costs $2^n - 1$ steps to moving n disks.

Example: Hanoi Puzzle

Pseudocode:

```
function moveTower(currentHeight, fromPole, toPole, withPole)
    if currentHeight = 0 then return
    end if
    moveTower(currentHeight - 1, fromPole, withPole, toPole)
    moveDisk(height, fromPole, toPole)
    moveTower(currentHeight - 1, withPole, toPole, fromPole)
      return
end function
      totalNumberOfMove ← 2^n
```

We have $T(n) = 2T(n-1) + 1$, It's easy to find that we have time complexity $T(n) = O(2^n)$ same as the number of steps we need.