Algorithms - 4 Divide and conquer

Divide: into a number of subproblems that are smaller instances of the same problem
Conquer: solve subproblems recursively
Base case: if the subproblems are small enough, solve by brute force
Combine: merge subproblem solutions to give a solution to the original problem

Recurrence Relations

> transform recurrence relations in to closed form - asymptotic solution
>
> **substitution**- need to have a good guess
>
> > guess: the form of solution
> > check: mathematical induction to find the constants and show it works)
> >
> > > name the constant in the asymptotic notation
> >
> > e.g.
> > Given : T(n) = 2T(n/2) + cnlogn
> > **Guess:** $T(n) \leq dnlog^2n$
> > **Inductive hypothesis(IH):**
> > IH: $T(k) \leq dk(logk)^2$ k<n and for positive constant d, c
> > $T(n) = 2T(n/2) + cnlogn \leq 2d \times \frac{n}{2}(log\frac{n}{2})^2 + cnlogn$
> > $= dn(logn - log2)^2 + cnlogn$
> > $= dn(logn)^2 - 2dnlogn + dn + cnlogn$
> > $= dn(logn)^2 - (2d - c)nlogn + dn$
> > Since our base case is $n \geq 2 = n_0 \implies dn \leq dnlogn$
> > $T(n) \leq dn(logn)^2 - (2d - c)nlogn + dnlogn$
> > $= dn(logn)^2 - (d - c)nlogn$
> > Only if we choose d that $d - c \geq 0 \implies d \geq c$
> > we can make sure that: $T(n) \leq dn(logn)^2$
> > There for $T(n) \leq dnlog^2n$
> >
> > if we need to prove θ, we need to prove O, Ω separatly
>
> **recursion**
>
> > recursion tree：
> > e.g $T(n) = \begin{cases} 2T(\frac{n}{2}) + O(n) & else \\ O(1) & if\ n = 1 \end{cases}$

depth	#node	node size	time complexity
0	1	n	$c \cdot n$
1	2	n/2	$c \cdot n$
...	...		...
i	2^i	n/2^i	$c \cdot n$
...	...		...
lgn	n	n/n	$c_2 \cdot n$

So total time complexity is $c_2 n + \sum_{i=1}^{logn} c \cdot n = nlogn(n)$

master method

$T(n) = aT(n/b) + f(n)$
a = # recursive calls >= 1 b > 1
f(n) >= 0 (for divide/combine)

case	f(n)	T(n)	situation
1	$O(n^{log_b a - \epsilon})$ constant $\epsilon > 0$	$\theta(n^{log_b a})$	cost dominated by the leaves
2	$\theta(n^{log_b a} log^k n)$ constant $k \geq 0$	$\theta(n^{log_b a} log^{k+1} n)$	cost is same at each level $log_b n$ levels
3	$\Omega(n^{log_b a + \epsilon})$ constant $\epsilon > 0$ and f(n) satisfies the regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and sufficiently large n	$\theta(f(n))$	cot dominated by the root

simplified:
$T(n) = aT(n/b) + \theta(n^d)$
$a \geq 1, b > 1, d \geq 0, assume n = b^k$

case	f(n)	T(n)	situation
1	$log_b a > d \quad or \quad a > b^d$	$\theta(n^{log_b a})$	cost dominated by the leaves
2	$log_b a = d \quad or \quad a = b^d$	$\theta(n^d logn)$	cost is same at each level $log_b n$ levels
3	$log_b a < d \quad or \quad a < b^d$	$\theta(n^d)$	cot dominated by the root

simplified ++: $\theta(n^d) \rightarrow n^d$
prove:

$$T(n) = \begin{cases} aT(\frac{n}{b}) + n^d & else \\ O(1) & if\ n = 1 \end{cases}$$

WLOG: $n = b^k$

depth	#node	node size	time complexity(non recursive cost)
0	a^0	$\frac{n}{b^0}$	$n^d = (\frac{n}{b^0})^d$

depth	#node	node size	time complexity(non recursive cost)
1	a^1	$\frac{n}{b^1}$	$n^d = (\frac{n}{b^1})^d$
...	...		...
i	a^i	$\frac{n}{b^i}$	$n^d = (\frac{n}{b^i})^d$
...	...		...
lgn	$a^k = a^{log_b n}$	$\frac{n}{b^k} = 1$	1

work for each level: $a^i(\frac{n}{b^i})^d = n^d(\frac{a}{b^d})^i$
total work : $n^d \sum_{i=0}^{log_b n} (\frac{a}{b^d})^i$
notice: a -> represent the increase of subproblem, b^d -> represents the decrease in cost of the subproblem

case3: $(\frac{a}{b^d})^i < 1 \iff log_b a < d$

> $a = (\frac{a}{b^d})^0 \leq \sum_{i=0}^{log_b n} (\frac{a}{b^d})^i \leq \sum_{i=0}^{\inf} (\frac{a}{b^d})^i$
>
> Thus $\sum_{i=0}^{log_b n} (\frac{a}{b^d})^i = \theta(1), T(n) = \theta(n^d)$

case2: $(\frac{a}{b^d})^i < 1 \iff log_b a < d$

> T(n) = $\sum_{i=0}^{log_b n} (\frac{a}{b^d})^i = n^d \sum_{i=0}^{log_b n} 1^i = n^d(log_b n + 1)$
> Thus $T(n) = \theta(n^d logn)$

case1: $(\frac{a}{b^d})^i > 1 \iff log_b a > d$

> $T(n) = n^d \sum_{i=0}^{log_b n} (\frac{a}{b^d})^i \in \theta(n^d \frac{a^{log_b n}}{b^{d log_b n}}) = \theta(n^d \frac{a^{log_b n}}{n^d}) = \theta(a^{log_b n}) = \theta(a^{\frac{log_a n}{log_a b}} = a^{log_a n log_b a} = n^{log_b a})$
> Notice in first step :
> sum of the geometric series, $a \neq b^d$, and $a/b^d - 1$ is a constant:
> $\sum_{i=0}^{log_b n} (a/b^d)^i = \frac{(a/b^d)^{log_b n+1} - 1}{a/b^d - 1} \in \theta((a/b^d)^{log_b n}))$

Problem - Matrix Multiplication

> Sol1: normally: $O(n^3)$
> Sol2: Recursively compute the matrix multiplication each matrix -> 4 squares - $O(n^3)$
>
> > $$T(n) = \begin{cases} 8T(\frac{n}{2}) + O(n^2) & else \\ O(1) & if\ n = 1 \end{cases}$$
> > Still $O(n^3)$
>
> Sol3: Strassen's Matrix
>
> > A11 A12 B11 B 12 = C11 C12
> > A21 A22 * B21 B22 = C21 C22
> >
> > we can just calculate 7 sub multiplication
> > P1 = A11(B12 - B22)
> > P2 = (A11 + A12)B22
> > P3 = (A21 + A22)B11
> > P4 = A22(B21 - B11)
> > P5 = (A11 + A22)(B11 + B22)
> > P6 = (A12 - A22)(B21 + B22)
> > P7 = (A11 - A21)(B11 + B12)

> > C11 = P5 + P4 - P2 + p6
> > C12 = P3+ P4
> > C21 = P3 + P4
> > C22 = P5 + P1 - P3 - P7
> >
> > $$T(n) = \begin{cases} 7T(\frac{n}{2}) + O(n^2) & else \\ O(1) & if\ n = 1 \end{cases}$$
> > $n^{2.80} \le O(n^{log_2 7}) \le n^{2.81}$
>
> Recurrence
> substitution method: guess and check method

Problem - Integer Multiplication

> original: x, y - n digit $O(n^2)$
> using merge and sort:
>
> > x = $2^{n/2} x_L + x_R$
> > y = $2^{n/2} y_L + y_R$
> > $x \cdot y = (2^{n/2} x_L + x_R)(2^{n/2} y_L + y_R) = x_L y_L 2^n + (X_R y_L + x_L y_R) 2^{n/2} + y_R x_R$
> > $T(n) = 4T(n/2) + O(n) -> O(n^2)$
> >
> > change:
> > $P_1 = x_L y_L, P_2 = x_R y_R, P_3 = (x_L + x_R)(y_L + y_R), x \cdot y = P_1 2^n + (P_3 - P_1 - P_2) 2^{n/2} + P_2$
> > $T(n) = 3T(n/2) + O(n) = O(n^{log_2 3})$

Problem - Closest Pair of Points

> Given a set of points, find the closest pair of points
>
> brute search: O(n^2)
> divide and conquer:
>
> ```
> CLOSET_PAIR_REC(px) # px are points sorted by x-value
>     if |px| <= 3
>         solve and return
>     CONSTRUCT Qx and Rx
>     (q1, q2) = CLOSEST_PAIR_REC(Qx)
>     (r1, r2) = CLOSEST_PAIR_REC(Rx)
>     theta = min{d(q1, q2), d(r1, r2)}
>     x_mean = max x-coordianate in Qx
>     CONSTRUCT Y = (s1, s2, ...sm) where si =(xi, yi) and |xi-xmean| <= theta # sorted by y-value
>     for each si in Y
>         for j = 1 to 7 # not tight ? may be smaller?
>             if d(si, si+j) < theta
>                 (s1', s2') = (si, si+j)
>                 theta = d(s1', s2')
> ```
>
> $T(n) = 2T(b/2) + O(nlongn) => O(nlog^2 n)$
> To improve: ? just input y, not x, y
>
> > O(nlogn)
>
> https://sites.math.rutgers.edu/~ajl213/CLRS/Ch33.pdf

> 3D
> https://people.csail.mit.edu/indyk/6.838-old/handouts/lec17.pdf
> http://euro.ecom.cmu.edu/people/faculty/mshamos/1976ShamosBentley.pdf
> https://www.cse.iitd.ac.in/~ssen/cs852/scribe/scribe2/lec.pdf

I'th Order Statistic

> given an array, we want to find ith number

Deterministic Linear Selection

```
DETERMINISTICSELECT(A, n, i)
    # find the ith smallest of n items in A
    divide the elements of the input array A into groups of 5 # theta(n)
    find the medium of each group of 5 items and put them into another array B # theta(n)
    x = DETERMINISTICSELECT(B, n/4 , n/10) # T(n/5)
    partition A-{x} into two sets A1, A2 such that # theta(n)
    A1 = {k|k<x}
    A2 = {k|k>x}
 # ?
 if i = |A1| + 1 return x
    else
    if i < |A1| +1 return DETERMINISTICSELECT(A1,|A1|,1)
        else return DETERMINISITICSELECT(A2, |A2|, i-|A1|-1)
```

> time analysis:
>
> everytime we can split at least 3n/10 - 6 (don't need to check any more)
> $T(n) \leq O(n) + T(n/5) + T(7n/10 + 6)$
> $T(n) = O(n)$