5 Design of Algorithms: By Divide and Conquer

5.1 Divide-and-conquer algorithms

Reading: CLRS 2.3

- A general template:

  function $D\&C(x)$

  if x is small and or simple then return $adhoc(x)$

  divide x into smaller instances $x_1, \dots, x_k$ where $k \geq 1$

  for $i \leftarrow 1$ to k

  $y_i \leftarrow D\&C(x_i)$

  combine the y_i's to obtain a solution y to x

  return y

  Remarks:

  - Relations between x and $x_1, \dots, x_k$, $y_1, \dots, y_k$ and y;
  - Time complexity $T(n) = \sum_{i=1}^{k} T(n_i) + D(n) + C(n)$, where $D(n)$ and $C(n)$ are time for "divide" and "combine", respectively;
  - Trade-off between $D(n)$ and $C(n)$;
  - The time complexity requirement reveals how "divide" and "combine" may be done.

$O(\log n)$	$T(n) = T(\frac{n}{2}) + 1$
$O(n)$	$T(n) = 2T(\frac{n}{2}) + 1$ or $T(\frac{n}{2}) + n$
$O(n \log n)$	$T(n) = 2T(\frac{n}{2}) + n$ or $T(\frac{n}{2}) + n \log n$
$O(n^2)$	$T(n) = 4T(\frac{n}{2}) + n$ or $2T(\frac{n}{2}) + n^2$

- Examples of D&C algorithms:

  Binary search: $T(n) = T(\frac{n}{2}) + O(1) \Rightarrow T(n) = O(\log n)$.

  Merge sort: $T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n \log n)$, where $D(n) = O(1)$ and $C(n) = O(n)$.

  Quick sort (best case): $T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n \log n)$, where $D(n) = O(n)$ and $C(n) = O(1)$.

  Integer multiplication: $T(n) = 3T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n^{\log 3}) = O(n^{1.585})$, where $D(n) = O(n)$ and $C(n) = O(n)$.

5.2 Matrix multiplication

Reading: CLRS 28.2

- Consider the multiplication of two $n \times n$ matrices. Using the definition of matrix multiplication, we need $\Theta(n^3)$. To use any divide-and-conquer idea, we have to first divide a matrix into several smaller matrices. Let $A_{n \times n}$ and $B_{n \times n}$ be the two matrices. Let $C_{n \times n} = A_{n \times n} \cdot B_{n \times n}$.

$$A_{n \times n} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B_{n \times n} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \qquad C_{n \times n} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

  Then $C_{11} = A_{11}B_{11} + A_{12}B_{21}$, $C_{12} = A_{11}B_{12} + A_{12}B_{22}$, $C_{21} = A_{21}B_{11} + A_{22}B_{21}$, and $C_{22} = A_{21}B_{12} + A_{22}B_{22}$. So the multiplication of two $n \times n$ matrices becomes eight multiplications of two $\frac{n}{2} \times \frac{n}{2}$ matrices, giving us $T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$. So $T(n) = \Theta(n^3)$ by master theorem. No improvement!

- In the late sixties, Strassen discovered a way to reduce eight multiplications to seven.

M_1	$(A_{12}-A_{22})(B_{21}+B_{22})$
M_2	$(A_{11}+A_{22})(B_{11}+B_{22})$
M_3	$(A_{11}-A_{21})(B_{11}+B_{12})$
M_4	$(A_{11}+A_{12})B_{22}$
M_5	$A_{11}(B_{12}-B_{22})$
M_6	$A_{22}(B_{21}-B_{11})$
M_7	$(A_{21}+A_{22})B_{11}$

C_{11}	$M_1+M_2-M_4+M_6$
C_{12}	M_4+M_5
C_{21}	M_6+M_7
C_{22}	$M_2-M_3+M_5-M_7$

Using the above idea in the divide-and-conquer algorithm, we get $T(n)=7T(\frac{n}{2})+\Theta(n^2)$, thus $T(n)=\Theta(n^{\log 7})=\Theta(n^{2.81})$ by master theorem.

- In the late seventies, the matrix multiplication algorithm is improved to $\Theta(n^{2.61})$ by Pan. In the late eighties, the algorithm is improved to $\Theta(n^{2.376})$. There is still a substantial gap to the $\Omega(n^2)$ lower bound.

5.3 Finding the kth smallest

Reading: CLRS 8.2 and 8.3

- Given a list of n numbers, find the kth smallest number among them.
- First try: Sort the list in increasing order ($\Theta(n\log n)$) and locate the kth element ($\Theta(1)$).
- Second try: Similar to Quick Sort.

  Function $Select(L,k)$
    if $|L|<50$
    then sort L and return the kth
    else choose any $p\in L$ as a pivot
      $L_1=\{a_i\in L|a_i<p\}$
      $L_2=\{a_i\in L|a_i=p\}$
      $L_3=\{a_i\in L|a_i>p\}$
      if $k\le|L_1|$ then return $Select(L_1,k)$
      else if $k\le|L_1|+|L_2|$
        then return p
        else return $Select(L_3,k-|L_1|-|L_2|)$

  Like Quick Sort, the time complexity of this algorithm heavily depends on the selection of the pivot in each recursion. If every time p happens to partition L evenly, then the time complexity is $\Theta(n)$. However, if the partition is extremely uneven, the time complexity degrades to $\Theta(n^2)$. Therefore, the worst-case time of the algorithm is $\Theta(n^2)$.

- Third try: Choose the pivot cleverly. Replace "choose any $p\in L$ as a pivot" in the above algorithm by the following code:

      divide L into $\lfloor\frac{|L|}{5}\rfloor$ sublists of (up to) 5 elements each
      sort each sublist into increasing order
      let M be the list of medians of all sublists
      $p\leftarrow Select(M,\lceil\frac{|M|}{2}\rceil)$

  It can be shown that there will be at most $\frac{3}{4}|L|$ elements in L_1 and at most $\frac{3}{4}|L|$ elements in L_3. Therefore, $T(n)\le O(1)$ for $n\le 49$ and $T(n)\le T(\frac{n}{5})+T(\frac{3n}{4})+O(n)$ for $n\ge 50$. By induction, $T(n)=O(n)$. Since $\Omega(n)$ is obvious a lower bound, this D&C algorithm is optimal.

- Theorem: Let $T(n)\le cn$ for $n\le 49$ and $T(n)\le T(\frac{n}{5})+T(\frac{3n}{4})+cn$ for $n\ge 50$. Show that $T(n)\le 20cn$.

Proof Induct on n. When $n \le 49$, $T(n) \le cn \le 20cn$. Assume that $T(i) \le 20ci$ for $i \le n-1$. Now consider $T(n)$.

$$\begin{aligned} T(n) &\le T(\frac{n}{5}) + T(\frac{3n}{4}) + cn \\ &\le 20c\frac{n}{5} + 20c\frac{3n}{4} + cn \\ &= 4cn + 15cn + cn \\ &= 20cn \end{aligned}$$

5.4 Exchanging two sections of an array

- Given an array A of n items. How can one exchange the first k items with the last $n-k$ items?
- A naive solution: copy the first k elements to an auxiliary array B; move the last $n-k$ elements to the first $n-k$ positions of A; and copy the k elements in B back to the last k positions of A.
- Assume only $\Theta(1)$ auxiliary memory is available. If the two sections have the same length, it is easy. For example, $A = (a,b,c,d,e,f)$ and $k = 3$. We can exchange the sections by using just one additional variable: swap a,d, swap b,e, and swap c,f.
- A D&C idea: Let $k = 3$. $(a,b,c,d,e,f,g,h,i,j,k) \to (d,e,f,g,h,i,j,k,a,b,c)$
- Algorithm:

  Swap(i,j,m) // Assume $i+m \le j$

  for $p \leftarrow 0$ to $m-1$

  swap $A[i+p]$ and $A[j+p]$

  Exchange(i,j,l,m) // Assume $i+l \le j$

  if $l = m$ *Swap*(i,j,l)

  else if $l < m$

  Swap$(i,j+m-l,l)$

  Exchange$(i,j,l,m-l)$

  else

  Swap(i,j,m)

  Exchange$(i+m,j,l-m,m)$

- Time complexity: Let $T(l,m)$ be the number of single swaps. If $l = m$, $T(l,m) = l$; if $l < m$, $T(l,m) = l + T(l,m-l)$; and if $l > m$, $T(l,m) = m + T(l-m,m)$.

  Prove by induction that $T(l,m) = l + m - gcd(l,m)$.

  Induct on $l+m$ (assuming $l,m > 0$). When $l+m = 2$, $l = m = 1$. So $T(l,m) = T(1,1) = 1$ by definition. On the other hand, $l+m-gcd(l,m) = 2 - gcd(1,1) = 1$. So the claim holds. In the inductive hypothesis, assume that for $l'+m' < l+m$, $T(l',m') = l'+m'-gcd(l',m')$. Now consider the case of $l+m$. If $l = m$, $T(l,m) = l = 2l - l = l+m-gcd(l,m)$. If $l < m$, $T(l,m) = l + T(l,m-l) = l + (l+m-l) - gcd(l,m-l) = l+m-gcd(l,m-l) = l+m-gcd(l,m)$. If $l > m$, $T(l,m) = m + T(l-m,m) = m + (l-m+m) - gcd(l-m,m) = l+m-gcd(l,m)$.

- *Exchange*$(1,k+1,k,n-k)$ solves the problem in time $T(k,n-k) = n - gcd(k,n)$.

5.5 Computing exponentiation

- Idea:

$$a^n = \begin{cases} a & \text{if } n = 1 \\ (a^{\frac{n}{2}})^2 & \text{if } n \text{ is even} \\ a \times a^{n-1} & \text{otherwise} \end{cases}$$

  For example, $a^{29} = aa^{28} = a(a^{14})^2 = a((a^7)^2)^2 = \cdots = a((a(a(a)^2)^2)^2)^2$. It takes a total of seven multiplications.

  For simplicity, we call this algorithm *expodac*(a,n).

- Time complexity analysis:

  Let $N(n)$ be the number of multiplications in $expodac(a,n)$.

$$N(n) = \begin{cases} 0 & \text{if } n = 1 \\ N(\frac{n}{2}) + 1 & \text{if } n \text{ is even} \\ N(n-1) + 1 & \text{otherwise} \end{cases}$$

  When $n > 1$ is odd, $N(n) = N(n-1)+1 = N(\frac{n-1}{2})+2 = N(\lfloor \frac{n}{2} \rfloor)+2$. When n is even, $N(n) = N(\frac{n}{2})+1 = N(\lfloor \frac{n}{2} \rfloor)+1$. Therefore, $N(\lfloor \frac{n}{2} \rfloor)+1 \le N(n) \le N(\lfloor \frac{n}{2} \rfloor)+2$. Define two functions N_i for $i = 1,2$ as follows.

$$N_i(n) = \begin{cases} 0 & \text{if } n = 1 \\ N_i(\lfloor \frac{n}{2} \rfloor) + i & \text{otherwise} \end{cases}$$

  It is easy to prove that $N_1(n) \le N(n) \le N_2(n)$. Since $N_1(n) = \Theta(\log n)$ and $N_2(n) = \Theta(\log n)$, then $N(n) = \Theta(\log n)$.

  Now, if each integer multiplication can be done in constant time, the time complexity of $expodac(a,n)$ is $\Theta(\log n)$. But what if the integers involved are so large that a multiplication can not be completed in constant time? Let $M(p,q)$ be the time needed to multiply two integers of sizes p and q (the numbers of figures). Let $T(p,n)$ be the time complexity of $expodac(a,n)$, where p is the size of a.

  Theorem/Exercise: The size of a^i is at least $i(p-1)$ but at most ip.

  Inspection of $expodac(a,n)$ yields the following definition of $T(p,n)$.

$$T(p,n) \le \begin{cases} 0 & \text{if } n = 1 \\ T(p,\frac{n}{2}) + M(\frac{pn}{2},\frac{pn}{2}) & \text{if } n \text{ is even} \\ T(p,n-1) + M(p,p(n-1)) & \text{otherwise} \end{cases}$$

  If $n > 1$ is odd, $T(p,n) \le T(p,n-1)+M(p,p(n-1)) \le T(p,\frac{n-1}{2})+M(\frac{p(n-1)}{2},\frac{p(n-1)}{2})+M(p,p(n-1)) = T(p,\lfloor \frac{n}{2} \rfloor)+M(p\lfloor \frac{n}{2} \rfloor,p\lfloor \frac{n}{2} \rfloor)+M(p,p(n-1))$. If n is even, $T(p,n) = T(p,\frac{n}{2})+M(\frac{pn}{2},\frac{pn}{2}) \le T(p,\lfloor \frac{n}{2} \rfloor)+M(p\lfloor \frac{n}{2} \rfloor,p\lfloor \frac{n}{2} \rfloor)+M(p,p(n-1))$. So in both cases,

$$T(p,n) \le T(p,\lfloor \frac{n}{2} \rfloor) + M(p\lfloor \frac{n}{2} \rfloor, p\lfloor \frac{n}{2} \rfloor) + M(p,p(n-1)).$$

  In general, $M(p,q) = \Theta(qp^{\alpha-1})$ where $p \le q$ and $\alpha = 2$ in the classic integer multiplication algorithm and $\alpha = \log 3$ in the divide-and-conquer algorithm. So $M(p\lfloor \frac{n}{2} \rfloor,p\lfloor \frac{n}{2} \rfloor) = \Theta((p\lfloor \frac{n}{2} \rfloor)^\alpha)$ and $M(p,p(n-1)) = \Theta(p^\alpha(n-1))$. So,

$$T(p,n) \le T(p,\lfloor \frac{n}{2} \rfloor) + \Theta(p^\alpha n^\alpha).$$

  By the iteration method or the master method, we have $T(p,n) \le \Theta(p^\alpha n^\alpha)$, thus $T(p,n) = O(p^\alpha n^\alpha)$.

  On the other hand, consider the last or the next to last multiplication in $expodac(a,n)$, depending on whether n is even or odd. It involves squaring $a^{\lfloor \frac{n}{2} \rfloor}$, which is of size at least $(p-1)\lfloor \frac{n}{2} \rfloor$. So,

$$T(p,n) \ge M((p-1)\lfloor \frac{n}{2} \rfloor, (p-1)\lfloor \frac{n}{2} \rfloor).$$

  Since $M((p-1)\lfloor \frac{n}{2} \rfloor,(p-1)\lfloor \frac{n}{2} \rfloor) = \Theta(((p-1)\lfloor \frac{n}{2} \rfloor)^\alpha)$, so $T(p,n) \ge \Theta(p^\alpha n^\alpha)$, thus $T(p,n) = \Omega(p^\alpha n^\alpha)$.

  Combining the two bounds, we have $T(p,n) = \Theta(p^\alpha n^\alpha)$, where $\alpha = 2$ if the classic algorithm is used and $\alpha = \log 3$ if the divide-and-conquer algorithm is used.

5.6 The closest-pair problem

Reading: CLRS 33.4

- Problem:

  Given n 2D points, find the two closest points.

  Remarks:

- Input is $(x_1, y_1), \ldots, (x_n, y_n)$ and output is $(x_i, y_i), (x_j, y_j)$.
- The distance between $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is $|p_1 p_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
- A brute-force algorithm: $\binom{n}{2} = O(n^2)$.

- A divide-and-conquer algorithm:

  Let X be point set P sorted by increasing x and Y be point set P sorted by increasing y.

  1. Divide: Use a vertical line l to bisect P into P_L and P_R. X and Y are thus correspondingly partitioned into X_L and X_R, Y_L and Y_R.
  2. Conquer: Use two recursive calls to find the closest pairs in P_L and P_R. Let δ_L (δ_R) be the distance between the two closest points in P_L (P_R). Define $\delta = \min\{\delta_L, \delta_R\}$.
  3. Merge: Find the closest pair in P. It can be the pair with distance δ found in the previous step, or a pair with one point in P_L and the other in P_R with a distance less than δ.

- Time complexity: $T(n) = T_P(n) + T_{DAC}(n)$.

  - Preprocessing: Sort P twice to construct X and Y. $\Rightarrow O(n \log n)$
  - Divide: Use the median in X to create the partition of P into P_L and P_R. Construct the partition of X into X_L and X_R. (This is easy.) Construct the partition of Y into Y_L and Y_R. (This can be tricky.) All of the above must be done in linear time. To check whether you have the right partitions, are P_L, X_L, and Y_L the same point set, and are P_R, X_R, and Y_R the same point set? $\Rightarrow O(n)$
  - Conquer: Two recursive calls on point sets of size $\frac{n}{2}$. $\Rightarrow 2T_{DAC}(\frac{n}{2})$
  - Merge: Many technical details to fill in. We wish to spend only linear time for the merge. Can we achieve this goal? $\Rightarrow O(n)$

  So $T_{DAC}(n) = 2T_{DAC}(\frac{n}{2}) + O(n) = O(n \log n)$. Overall, $T(n) = T_P(n) + T_{DAC}(n) = O(n \log n) + O(n \log n) = O(n \log n)$.

- Merge in linear time:

  Assume that

  - $P \Rightarrow P_L, P_R$, $X \Rightarrow X_L, X_R$, $Y \Rightarrow Y_L, Y_R$ by the vertical line l.
  - δ_L is the distance between the closest points in P_L and δ_R is the distance between the closest points in P_R.
  - $\delta = \min\{\delta_L, \delta_R\}$.

  Goal: Determine if there are two points, one in P_L and the other in P_R, with distance less than δ.

  An exhaustive search checks all pairs and may take $O(n^2)$. Can we just check $O(n)$ pairs and not miss any one with distance less than δ? Yes and here is why.

  Define a strip centered at l with width 2δ. Let P_S be the set of points in the strip and Y_S be P_S sorted by y. (Remember our linear time restriction: Can you create P_S and Y_S in linear time?)

  Claim: If there are points $p \in P_L$ and $q \in P_R$ with $|pq| < \delta$, then p and q must be in the strip.

  Pause: Can we check out all pairs in P_S to determine the one with the smallest distance?

  For each $p \in Y_S$, define a rectangle $R(p)$ of height δ and width 2δ, with the bottom edge of the rectangle passing p.

  Claim: If there are p and q with $|pq| < \delta$ and $q.y \geq p.y$, q must be in $R(p)$.

  Claim: There can be at most eight points in each $R(p)$.

  Why? Divide $R(p)$ ($\delta \times 2\delta$) into eight $\frac{\delta}{2} \times \frac{\delta}{2}$ squares. In each square, if there are two or more points, say q_1 and q_2, then

  $$|q_1 q_2| \leq \text{diagonal of the square} = \sqrt{2}\frac{\delta}{2} < \delta,$$

  which is impossible since q_1 and q_2 are on the same side of the vertical line l. So there can be at most one point in each square, with a total of eight points in $R(p)$.

  Claim: For any $p \in Y_S$, if there is q such that $|pq| < \delta$, then q must be one of the seven points following p in Y_S.

  Algorithm for merge:

```
m = |Ys|
mindist = |Ys[0]Ys[1]|
p = Ys[0]
q = Ys[1]
for i = 1 to m-1
  k = min {i + 7, m}
  for j = i + 1 to k
    dist = |Ys[i]Ys[j]|
    if dist < mindist
      mindist = dist
      p = Ys[i]
      q = Ys[j]
If mindist < delta
  return p and q as the closest points
else return the closest points found by
     the recursive calls
```