May 20

Divide and Conquer

1. Order Statistics

Given a collection of nums : Find the min / max / median / x% / k-th smallest num

Solution

#1 sort array , return A[k] TC: O(nlogk)

#2 based partition pos = partition(A[], 1, n)

① pos = k , return A[k]

② pos > k , k-th num in kth(A[], 1, pos-1)

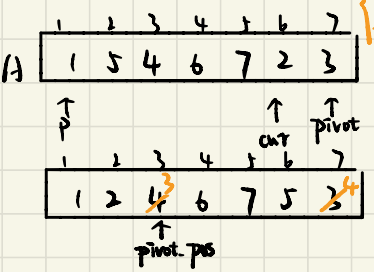
③ pos < k , k-th num in kth(A[], pos+1, n)

```
int quickselect(A[], begin, end, k){
    if(begin > end) return A[begin];
    idx = begin, pivot = A[begin];
    for(i = begin, ..., end){
        if(A[i] < pivot)
            swap(A[], idx++, i)
    swap(A[], idx, begin);
    if(idx + 1 = k) return A[idx]
    elif(idx+1 > k)
        return quickselect(A[], begin, idx-1, k);
    else
        return quickselect(A[], idx+1, end, k);
```

Two kinds of def kth(A[], begin, end, k)

① global num
② relatively

Find 3th smallest



kth(A[1...7], 3) ⇒ T(n) = T(n/2) + n ⇒ O(n)

pos = partition :3

kth(A[4, 7], 2)

T(n) = T(99n/100) + n ⇒ O(n)

T(n) = T(n-1) + n ⇒ O(n²)

- Find Top k : Given n number, return the smallest k numbers.

#1 : sort(A[]) , return A[1:k] O(nlogn)

#2 : kth algo(Above) , return A[1:k] O(n)* (Best)

#3 : make_heap(A[1...n]) min_heap , delete k time O(n + klogn) build
① small k , dominated by n
② large k , dominated by klogn

#4 : make_heap(A[1...k]) max_heap

```
for(i=k+1, ..., n)        O(n)
    if(getMax() > A[i]){
        deleteMax();      logk
        insert(A[i]);     logk
    }
```

O(k + (n-k)logk) build

O(nlogk) small k.
O(nlogn) large k. (k = n/2)

maintain max_heap of size k.

- Bit Int Multiplication

32-bits $0 \sim 2^{32}-1$ 10 digits

64-bits $0 \sim 2^{64}-1$

Input : 2 integers of n-bits

a | a_1 | a_2 |

$a = a_1 \cdot 2^{\frac{n}{2}} + a_2$

a | 1 | 1 | 0

$a = 1 \times 2^2 + 1 \times 2^1 + 0$ binary

```
  1 1 1 0   14
  0 1 0 1   5
-----------
    1 1 1 0
  1 1 1 0
-----------
  1 0 0 0 1 1 0   70
```

$O(n^2)$ SLOW!!!

b | b_1 | b_2 |

$\Rightarrow a_n \times b_n = (a_1 \cdot 2^{\frac{n}{2}} + a_2) \cdot (b_1 \cdot 2^{\frac{n}{2}} + b_2)$

$= \underbrace{a_1 b_1}_{P_1} \cdot 2^n + \underbrace{(a_2 b_1 + a_1 b_2)}_{P_3 - P_1 - P_2} \cdot 2^{\frac{n}{2}} + \underbrace{a_2 b_2}_{P_2}$ (Divide the bits by 2)

$T(n) = 4T(\frac{n}{2}) + n$ (sum 2 n-bits integer) $\Rightarrow O(n^2)$

if we can reduce 4 to 3, $T(n) = 3T(\frac{n}{2}) + n$, $O(n^2) \Rightarrow O(n^{\log_2 3})$

$P_1 = a_1 b_1$, $P_2 = a_2 b_2$, $P_3 = (a_1 + a_2) \cdot (b_1 + b_2)$

If n is an odd num:

= pad 0's to the next power of 2

$110 \rightarrow 0110$

$10001 \rightarrow 00010001$

Square Matrix Multiplication

Input : 2 nxn matrices

$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}_{n \times n}$ ($\frac{n}{2}$ rows, $\frac{n}{2}$ cols) $Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}_{n \times n}$, $Z = X \cdot Y = \begin{bmatrix} \quad \end{bmatrix}_{n \times n}$ $O(n^3)$ (Matrix Multiplication)

$XY = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix} = \begin{bmatrix} Q & R \\ S & T \end{bmatrix}$

$T(n) = 8T(\frac{n}{2}) + n^2 \Rightarrow O(n^3)$ In the left, we do 8 multiplications for $\frac{n}{2} \times \frac{n}{2}$ and four additions, each add takes $O(n^2)$

$P_1 = A(F-H)$, $P_2 = (A+B) \cdot H$, $P_3 = (C+D) \cdot E$, $P_4 = D(G-E)$ Strassen

$P_5 = (A+D) \cdot (E+H)$, $P_6 = (B-D)(G+H)$, $P_7 = (A-C)(E+F)$

$Q = P_4 + P_5 - P_2 + P_6$

$R = P_1 + P_2$

$S = P_3 + P_4$

$T = P_1 + P_5 - P_3 - P_7$

$\Rightarrow T(n) = 7T(\frac{n}{2}) + n^2 \Rightarrow O(n^{\log_2 7})$

- Counting Inversions

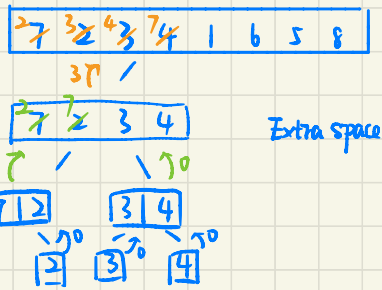
Inversion : $i<j$ && $A[i] > A[j] \Rightarrow (i, j)$ is an inversion

[1, 2, 3] # inversion 0

[2, 1, 3] # 1

[2, 3, 1] # 2

[3, 2, 1] # 3



Given a list of n items, find the # of inversions :

min # 0

max # $\binom{n}{2} = O(n^2)$

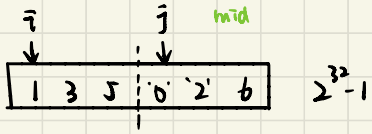
#1 Brute Force

```
count = 0
for (i = 1, ..., n)
    for (j = i+1, ..., n)
        if (A[i] > A[j])
            count++
```

#2 Divide and Conquer

inversion = #x + #y + # cross inversions

a) $\frac{n}{2} - i + 1$ (mid)



3 inv involves '0'

2 inv involves '2'

$T(n) = 2 \cdot T(\frac{n}{2}) + n \log n \Rightarrow n \log^2 n$?

```
CountInv(A[1...n], begin, end) {
    if (begin >= end) return 0;
    mid = (begin + end) / 2;
    x = CountInv(A[], begin, mid);
    y = CountInv(A[], mid+1, end);
    z = CrossInv(A[], begin, mid, end);
    return x+y+z;
}
```

merge(A[], begin, mid, end)

```
CrossInv(A[], begin, mid, end) {
    Sort(A[], begin, mid)
    Sort(A[], mid+1, end)
    p = begin, q = mid+1, cnt = 0;
    while (p <= mid && q <= end) {
        if (A[p] < A[q])
            p += 1
        else
            cnt += (mid - p + 1)
            q += 1
    }
    return cnt;
```

All num from p to mid form inverse pair with A[q]

```
int merge(A[], begin, mid, end) {
    int[] temp[] = new int[A.len]; // temp sort array
    int i = begin, j = mid+1, cnt = 0, idx = begin;
    while (i <= mid && j <= end) {
        if (A[i] <= A[j])
            temp[idx++] = A[i++]
        else
            temp[idx++] = A[j++];
            cnt = mid - i + 1;
    }
    // Fill the rest of i or j. Replace A[] with temp[]
```