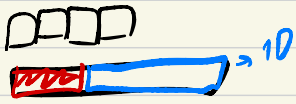
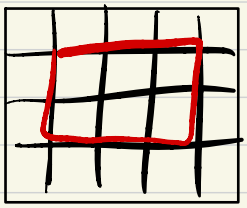
Algo → Qs 2D

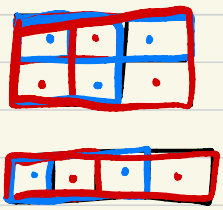
Data Structure

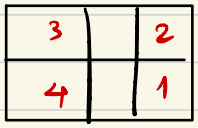


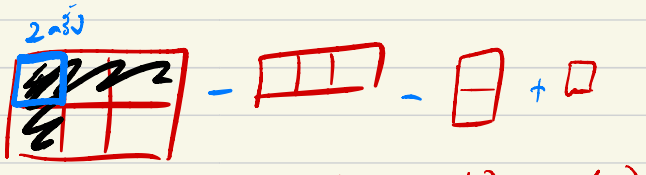


$O(n^2)$ per query

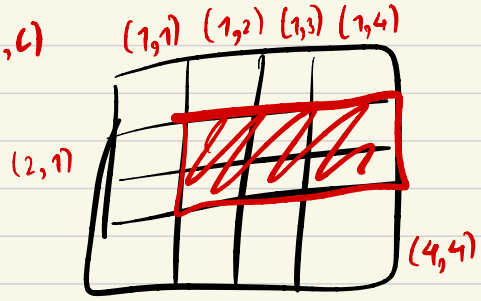
$O(1)$ per query
$O(n^2)$ precompute
$O(n^2 + Q)$; Q := number of queries

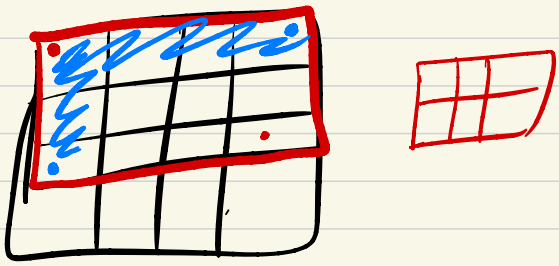


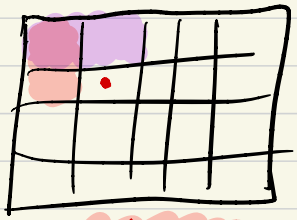




(R,C)

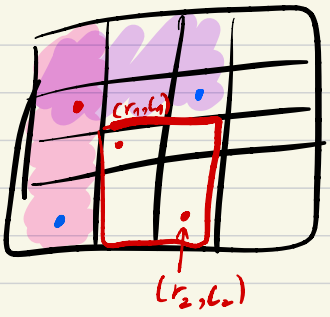




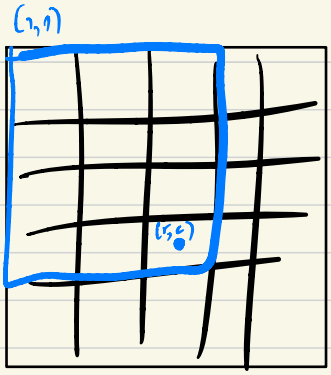


```
void precompute(){
    for(r; 1->R){
        for(c; 1->C){
            qs(r)(c) = qs(r-1)(c) + qs(r)(c-1) - qs(r-1)(c-1) + arr(r)(c)
        }
    }
}
```

```
int sum(int r1, int c1, int r2, int c2){
    if(r1 > r2 or c1 > c2) return -1
    return qs(r2)(c2) - qs(r2)(c1-1) - qs(r1-1)(c2) + qs(r1-1)(c1-1)
}
```

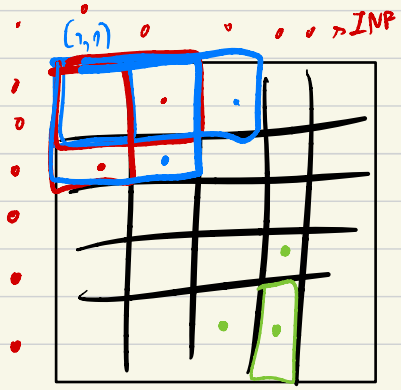


$(r_2, c_1 - 1)$
$(r_1 - 1, c_2)$
$(r_1 - 1, c_1 - 1)$



หา min ตั้งแต่ (1,1) จนถึง (r,c)

$O(R \times C)$ per query [Brute Force]



mi (r)(c) := ค่า min ของตารางตั้งแต่ช่อง (1,1) จนถึง (r,c)

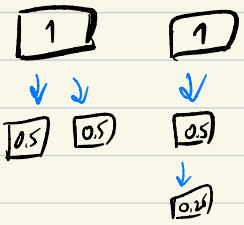
```
void precompute (){
    for (r: 0→R){
      for (c: 0→C){
          if (r==0 or c==0) mi(r)(c) =INF
          else mi(r)(c) = min ({ arr(r)(c), mi(r-1)(c), mi(r)(c-1)})
      }
    }
```

```
int query (int r, int c){
    return mi(r)(c)
}
```

min / max

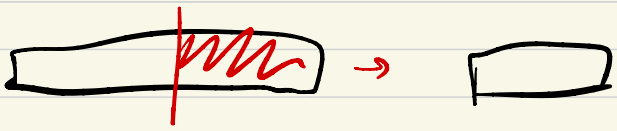
* min ไม่มี inverse

Divide and Conquer (D&C)



แบ่งปัญหาใหญ่ ให้เป็นปัญหาเดิม แต่ขนาดเล็กลง

Binary Exponentiation →

- Binary Search $O(\log n)$ 
- Binary Exponentiation $O(\log n)$
- Merge Sort $O(n \log n)$

Binary Exponentiation

a^n เช่น $2^8 = \overbrace{2 \times 2 \times \dots \times 2}^{8 \text{ ตัว}}$ $O(n)$

$2^8 = 2^4 \times 2^4$ $b = 2^4 \rightarrow b \times b$

$2^4 = 2^2 \times 2^2$

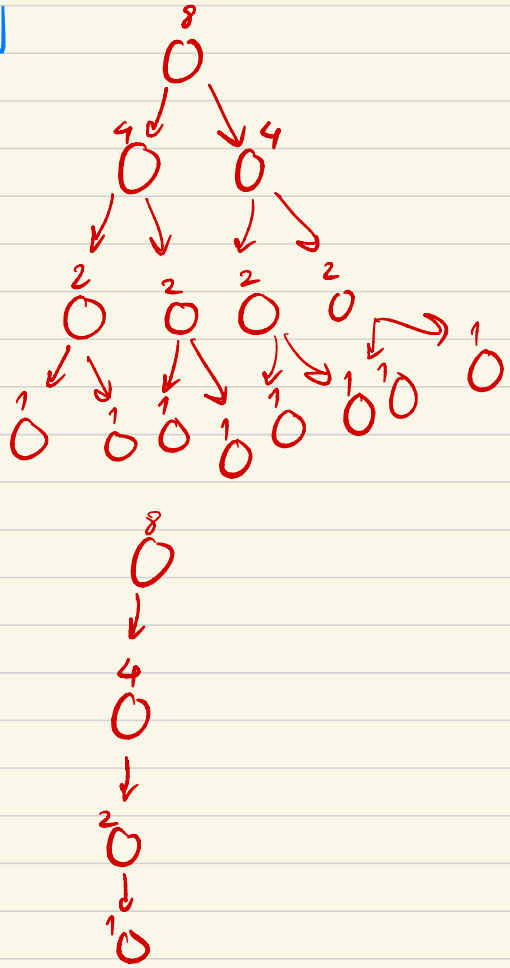
$2^2 = 2^1 \times 2^1$ $8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

$f(n) = a^n$

$$f(n) = \begin{cases} 1 ; n = 0 \\ a ; n = 1 \\ f(n/2) \times f(n/2) ; n \text{ เป็นเลขคู่} \\ f(n/2) \times f(n/2) \times a ; n \text{ เป็นเลขคี่} \end{cases}$$

```
define long long ll
ll expo(ll a, ll n){
    if (n==0) return 1
    if (n==1) return a
    ll b = expo( a, n/2)
    if( n%2 ==0){
        return  expo(a,n/2) x expo(a,n/2)
    }else{
        return  expo (a,n/2) x expo(a,n/2) xa
    }
}
```

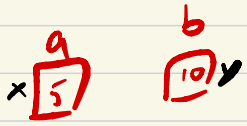
$\log_2 N$



Pass by reference
Merge sort Linda

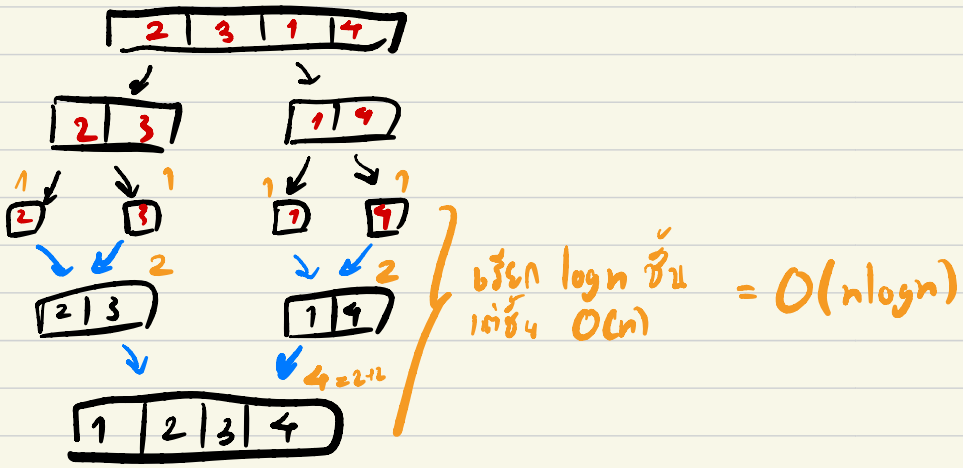
Pass by Reference

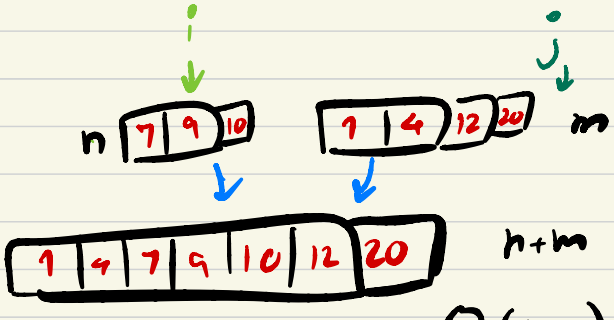
```
void swap( int &x , int &y){
    int tmp = x
        x = y
        y = tmp
}
```



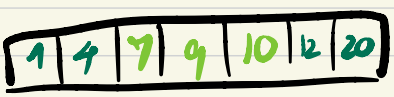
a=5, b=10
swap(a,b)
print (a,b) // 10,5

Merge Sort → O(nlogn)





$O(\frac{nm}{2})$



$O(n+m)$

```
int arr[]

void merge_sort (int l, int r){    //l = leftmost index
                                   //r = rightmost index
    if(l >= r) return
    int mid = (l+r)/2
    merge_sort(l, mid) , merge_sort(mid+1, r)

    int i=l, j=mid+1, tmp[r-l+1]=[2], k=0   // k 는 tmp
    while ( i < mid and j <r){
        if(arr[i] ≤ arr[j]){
            tmp[k] = arr[i]
            i++
        }else{
            tmp[k] = arr[j]
            j++
        }
        k++
    }
    while( i<mid ){
        tmp[k] = arr[i]
        k++, i++
    }
    while (j<r){
        tmp[k] = arr[j]
        k++, j++
    }
    for( k=0; k < r-l+1 ; k++){
        arr[m] = tmp[k]
    }
}


merge_sort (0, n-1)
```

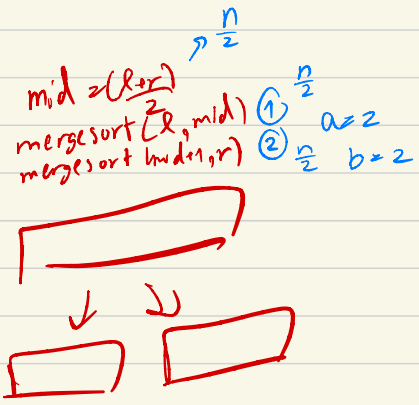
Master Theorem

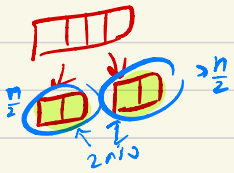
$$T(n) = \begin{cases} O(1) ; n=1 \\ a\,T(\frac{n}{b}) + O(n^d) ; n>1 \end{cases}$$

$O(n^d)$; $a<b^d$

$O(n^d \log n)$; $a=b^d$

$O(n^{\log_b a})$; $a>b^d$

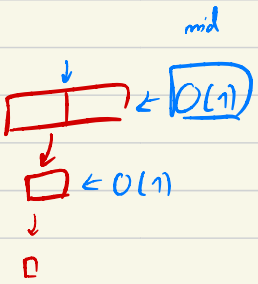




mergesort $\begin{cases} O(1); n=1 \\ 2T(\frac{n}{2}) + O(n) ; n>1 \end{cases}$

$a=2, b=2, d=1 \rightarrow a=2, b^d=2 \rightarrow$ case ② ที่ $a=b^d$

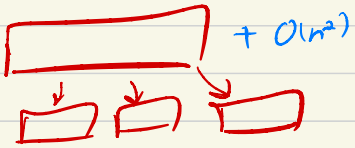
$O(n^1 \log n) = O(n \log n)$ ✓



binary search $\begin{cases} O(1) ; n=1 \\ 1T(\frac{n}{2}) + O(1^0) ; n>1 \end{cases}$

$a=1, b=2, d=0 \rightarrow a=1, b^d=1 \rightarrow$ case ② ที่ $a=b^d$

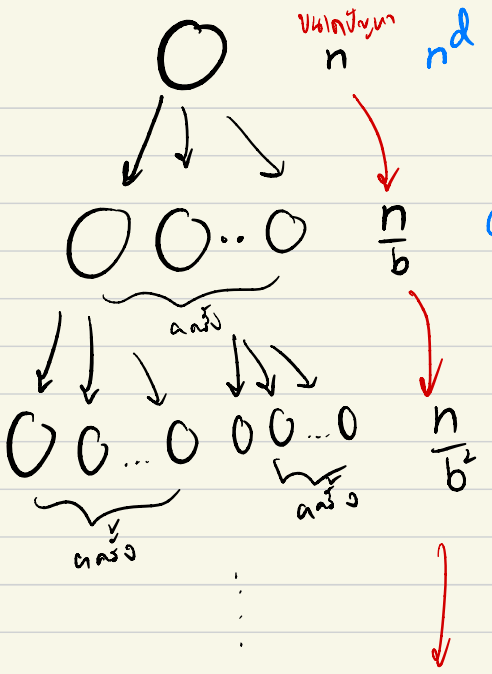
$O(n^0 \log n) = O(\log n)$



$T(n) \begin{cases} O(1); n=1 \\ 3T(\frac{n}{2}) + O(n^2) ; n>1 \end{cases}$

$a=3, b=2, d=2 \rightarrow a=3, b^d=4$ เข้า case ① $a<b^d$

$O(n^d) = O(n^2)$



$$= n^d + a\left(\frac{n}{b}\right)^d + a^2\left(\frac{n}{b^2}\right)^d + \dots + a^h\left(\frac{n}{b^h}\right)^d$$

$$= n^d + \frac{a n^d}{b^d} + \frac{a^2 n^d}{b^{2d}} + \dots + \frac{a^h n^d}{b^{hd}}$$

$$= n^d \left[1 + \frac{a}{b^d} + \frac{a^2}{b^{2d}} + \dots + \frac{a^h}{b^{hd}} \right]$$

(0, 1, 2, 3, ..., $h = \log_b n$)

$$= n^d \cdot \sum_{h=0}^{\log_b n} \left(\frac{a}{b^d}\right)^h$$ → 3 กรณี

→ $\frac{a}{b^d} < 1$ แสดงว่า $\sum_{h=0}^{\log_b n} \left(\frac{a}{b^d}\right)^h = O$ ได้ จะได้ $O(n^d)$

→ $\frac{a}{b^d} = 1$ แสดงว่า $\sum_{h=0}^{\log_b n} \left(\frac{a}{b^d}\right)^h = 1^0 + 1^1 + \dots + 1^h = \log_b n$ จะได้ $O(n^d \log n)$

→ case (3) ต่อหน้าต่อไป

อนุกรม → เลขคณิต → + เช่น 1, 3, 5, 7 ผลต่างร่วม แสดงว่า $r = 2$

→ เรขาคณิต → × เช่น 1, 3, 9, 27, 81, ... แสดงว่า $r = 3$

Case 3

$\frac{a}{b^d} > 1$ แสดงว่าเป็นอนุกรมเรขาคณิตที่ $a_1 = 1$, $r = \frac{a}{b^d}$

จาก $S_n = \frac{a_1(r^n - 1)}{r-1}$; n = จำนวนพจน์

$$S_n = \frac{1\left[\left(\frac{a}{b^d}\right)^{\log_b n+1} - 1\right]}{\left(\frac{a}{b^d}\right) - 1}$$

$$S_n = \frac{\frac{a^{\log_b n+1}}{b^{d\log_b n+1}} - 1}{\frac{a}{b^d} - 1}$$

$$S_n = \frac{a^{\log_b n+1} \cdot b^d}{b^{d\log_b n+1} \cdot a}$$

$$S_n = \frac{a^{\log_b n}}{b^{\log_b n^d}}$$

จาก $y^{\log_y x} = x$; $S_n = \frac{a^{\log_b n}}{n^d}$

จาก $n^d \cdot \sum_{k=0}^{\log_b n} \left(\frac{a}{b^d}\right)^k$ จะได้ $n^d \cdot \frac{a^{\log_b n}}{n^d} = a^{\log_b n}$

จาก $y^{\log_k x} = x^{\log_k y}$; $n^{\log_b a}$

สรุปได้ว่า เมื่อ $\left(\frac{a}{b^d}\right) > 1$ จะได้ $O(n^{\log_b a})$