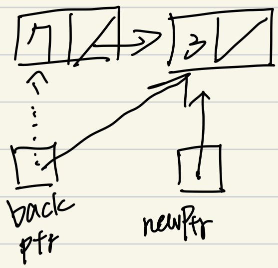
Queue

11024212 周[illegible]

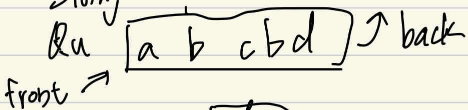
back rear 後端　　佇列 = 排隊
front 前端



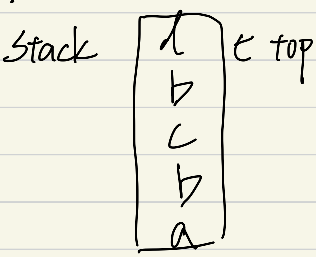
enqueue 新增 放後 enter
dequeue 移除 移前 delete
get front 擷取

newPtr -> next = NULL
backPtr -> next = newPtr
backPtr = newPtr

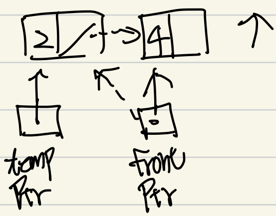
string. abcbd



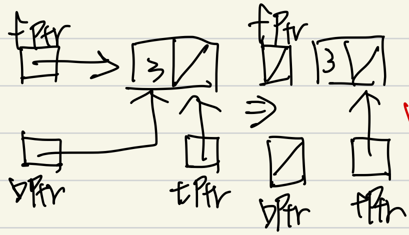
全空：前端指標
非空：後端指標下一個



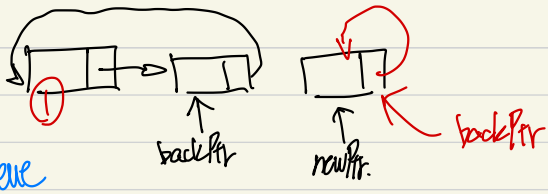
temp = front
frontPtr = frontPtr -> next (dequeue)



queue Front = frontPtr -> item
dequeue
擷取後移除



fPtr = NULL
bPtr = NULL



enqueue

If (!Empty()) 都要放到 newPtr

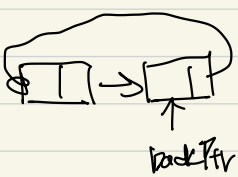
newPtr->next = newPtr

else

newPtr->next = backPtr->next 原本指第1個

backPtr->next = newPtr

backPtr = newPtr



dequeue

if (backPtr == backPtr->next) only one

backPtr = NULL

else

backPtr->next = tempPtr->next

tempPtr->next = NULL

delete tempPtr

Array

避免浪費 搬動 環狀陣列

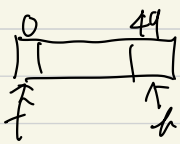
front = (back+1) % Max_Queue

利用 count

空 count = 0

全滿 count = Max

加入(full) 檢查



back = Max_Queue - 1

isfull = (front == (back+1) % Max_Queue)

dequeue

if(isEmpty())

throw

else

front = (front+1) % Max_Queue

if(isfull == True)

isfull = FALSE

enqueue 新增 改+1

alist.insert(getLength() + 1)

dequeue

alist.remove(1) 移除最前

getFront(queue front) 擷取

alist.retrieve(1, queue front)

Position - ADT

List ← 最彈性

Stack

Queue

Arrival	duration	Departure	waiting	
20	5	25	0	AWT = 10/4 = 2.5
22	4	29	3	MWT = 6
23	2	31	6	MQL = 2 排隊
30	3	34	1	AQL = 11+11/4 = 1

event list 事件清單 分 Arrival Departure

(排隊的窗口)

Queue

empty

0

20

Eventlist (準備進入&預計離開)

A 20 5

empty

D 25

演算法 Big-O

鏈結串列.

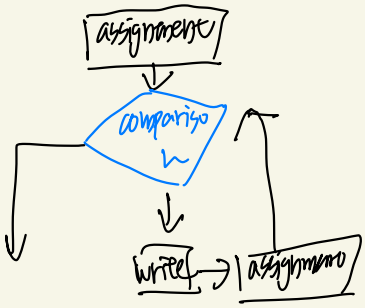
矛盾.

實作 Implementation

電腦 Computer

資料 Dataset.

$(n+1)\cdot(c+a)+n\times w$ → 1 moves.

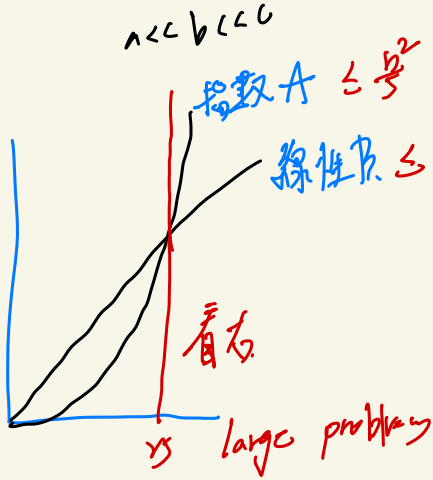



```
for (a=1. a<=n .a++)
{ (b=1 b<= a b++)
  {(c=1 c<=5 c++)
```

for $a=1$ to n ⇒ $n\times(n+1)\div 2$

⇒ $(2.5n^2+2.5n)/4$ ⇒ $n=10$ 要5次

$n=100$ 要1250次



成長函數

$O(n^2)$ ⇐ order n^2

$O(n)$ ⇐ order n

Big O 希望我最主的效率 (會不會超過 $k \times f(n)$)

$2.5n^2 + 2.5n$ is $O(?)$

$\forall n \geq n_0$ 找 $\leq k f(n)$

✓ $\forall n \geq 10$ $\leq 1 \times n^{10}$

$\forall n \geq n_0$, $(2.5n^2 + 2.5n) \leq k \times n^2$ Better.

$\forall n \geq 0$, " $\leq 3 \times n^2$

↑

(0, 1, 2).

忽略低次方

忽略常數.

complexity = Big O.

只在意最慢.

$O(n^2 + n^3)$ n^2 + n^3

time 相加.

A $O(1)$ 速度最快

$\sim n^0$

B $O(\log_2 n)$

C $O(n)$

D $O(n \log_2 n)$

E $O(n^2)$

F $O(n^3)$

G $O(2^n)$

Worst-case 多

Average-case 平均

Best-case 少

Sequential Search. 循序搜尋

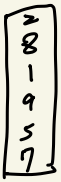
strategy.

Efficiency.

Worst case - O(n)

Averge case - O(n)

Best case O(1)



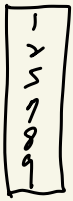
1
2
3
4
5
6
次數

(n+1)/2 = 3.5

0.5n 3.5n

Binary search 二元搜尋

O(log2^n)



6/2 = 3
3/2 = 2
2/2 = 1

n = 2^k

log2 16^6 = 19.9

100萬 only 20次

O(n) need 100萬.

二元 > 循序

有無排序不影響 循序的複雜度

	sorted	unsorted	found
v WC	O(n)	O(n)	O(n)
v AC	O(n)	O(n)	O(n)
BC	O(1)	O(n)	O(1)

内部排序 是否能在电脑記憶体執行
外部排序 ↑無法放入大量資料.

stable sort VS Unstable

bubble	selection
insertion	quick
merge	heap
radix	

相同值維持不變到排序

<u>Bubble</u> sort 高矮列隊

最小往上漂 swap
n n-1回合 一直換

selection sort 大隊接力 換棒

n n-1回合

只需換一次

bubble 取only 寫法 stable

selection unstable

Insertion sort

將前2個做為 range

插入數

n n-1回合

No swap

insertion stable

Bubble Sore

$n + \sum_{p=1 \sim n-1} (n-pass+1) + \sum_{p=1 \sim n-1} (n-pass)$

for (/

for ( ) => $O(n^2)$ => worst case

用不一定每次要 swap

if (a[n] > a[n+1]) ← major comparison

↑ >= unstable

swap

加上 bool sorted Best case $O(n)$

Selection Sort 最大的方法 Unstable

void selection Sort

找最大位置 → wc $O(n^2)$ bc $O(n^2)$

將最大→最後 swap Bc $O(1)$

只做搬動 or 資料量少時可使用

Insertion Sort 效率較好

$n \times (n+1)/2 + (n+1) = 1$ 千

n-1 件回合

wc $O(n^2)$

bc $O(n)$

逐一往後搬動

直到 insertion 的位置

>= unstable

第 1 已排

未排 1 → 已排

shell sort

可以改間隔 >1

Not stable

選擇排序

從未排好數字找最小

把最小丟到未排好數字最左，並標記排好

合併 可內可外

將大數列利用拆半的方法，拆到 only 1，再做合併

利用 recursion.

```
void MergeSort (vector<int> &array, int 前, int end)
    if(前<end)
        int mid = (前+end)/2
        MergeSort (array).
```

合併放在遞迴中

判斷 左1與右大小

小就放下.

最後放剩下

基数排序法

先看个位数 → 十位没有补0依此类推

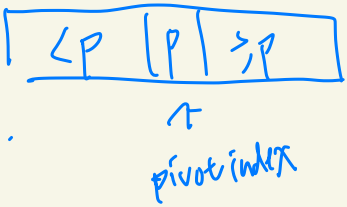
先找最大数，求总位数

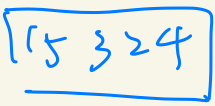
%10取余数

快排

item pivot

item >= pivot





我以第i为标准

i 最左 < 标准

j 最右 >= 标准

i == j stop. 标准互换

我今为左右找到