1

Stacks Last in first out

isEmpty(): boolean {query} 是否為空

push (in newItem : StackItemType) 新增一筆
  throw stackException

pop() throw StackException 移除最近一筆

getTop (out stackTop : StackItemType) {query}
  throw StackException 擷取最近一筆

pop (out stackTop : StackItemType)
  throw StackException 擷取後移除最近一筆

readAndCorrect (out aStack: Stack)
while (not end of line)
{ read a new character ch
 if (ch is not '←')
  aStack.push (ch)
else if (! aStack.isEmpty())
  aStack.pop

}

displayBackward (in aStack Stack)
 while (! aStack.isEmpty()){
  aStack.pop (ch)
  write ch
}

while (not end of string && balanced So far){
 read new ch
  if (ch is '{')
   aStack.push (ch)
  else if (ch is '}')
   if (! aStack.isEmpty())
    aStack.pop()
   else balanced So far = false

}
 if (balanced So far && aStack.isEmpty())
  return true.
 else
  return false

or use counter.
{ => counter +1
} => counter -1

if counter曾小於0
or 結束 counter 不是0 則錯.

2

n個車廂.

3 2 1 → 2 3 1 / 2 1 3 / 1 2 3 / 1 3 2 / 3 2 1 / ~~3 1 2~~.

push, push, push, pop, pop, pop.
total number of permutations: C(6, 3) $C\binom{2n}{n}$

incorrect.
push, pop, pop, push, push, pop 空堆疊.
push, pop, pop pop pop push
push 2(n-1) pop 4(n+1)

$CN_n = C\binom{2n}{n} - C\binom{2n}{n+1} = \frac{1}{n+1}C\binom{2n}{n}$

a - (b + c * d) / e

ch	Stack	postfix Exp.
a		a
-	-	a
(	- (	a
b	- (	ab
+	- (+	ab
c	- (+	abc
*	- (+ *	abc
d	- (+ *	abcd
)	- (+	abcd * Move operators
	- (	abcd * + from stack to.
	-	abcd * + postfix Exp until '('
/	- /	abcd * +
e	- /	abcd * + e → abcd * + / e.

3

Queue

New items enter at the back or rear of the queue
Items leave from the front of the queue
First-in, first out (FIFO) property

Remove the item that was added earlist 移除.
Retrieve the item that was added earlist 擷取

enqueue (in newItem : QueueItemType) 新增.
throw QueueException

dequeue = 移除.

getFront (out queueFront : QueueItemType) 擷取
{query}

dequeue (out queueFront : QueueItemType)
擷取後移除.

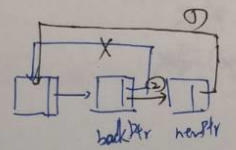
aQueue.createQueue()
: .enqueue (5)
: .enqueue (2)
: .enqueue (7)
: .getfront (queueFront)
: .dequeue (:)
: .dequeue (:)

front
↓
5
5 2 從前取
5 2 7 ← back

5 2 7 (queueFront is 5)
2 7 (: is 5)
7 (: is 2)

4

```
void Queue :: enqueue ( const QueueItemType & newItem) {
  QueueNode *newPtr = new QueueNode ;        環狀佇列
  newPtr -> item = newItem ;                 新增
  if (isEmpty ())                            // 0 -> 1 node
    newPtr -> next = newPtr ;                // point to itself
  else {                                     // k -> k+1 nodes , k>0
    newPtr -> next = backPtr -> next ;       // point to the front.   ①
    backPtr -> next = newPtr ;               // put behind the back.  ②
  } // else.
  backPtr = newPtr ;  new node at the back.
}
```



dequeue 環狀佇列 移除.

```
if (isEmpty ())
    throw ... ;
else {
  QueueNode *tempPtr = backPtr -> next      // tempPtr 第一個. the front.
  if ( backPtr == backPtr -> next )         // one node.
    backPtr = NULL;
  else backPtr -> next = tempPtr -> next.   // the next front
  tempPtr -> next = NULL;                   // defensive strategy
  delete tempPtr ;                          // release space
} // else
```

5

```
void Queue :: enqueue (const QueueItemType & newItem) {
  QueueNode * newPtr = new QueueNode;
  newPtr -> item = newItem;      // set data portion of new Node
  newPtr -> next = NULL;          新節點內容及指標.
  if (isEmpty ())
    frontPtr = newPtr;
  else backPtr -> next = newPtr;
  backPtr = newPtr;
}


void Queue :: dequeue () throw (QueueException) {
  if (isEmpty ())
    throw QueueException ("QueueException: ...");
  else {
    QueueNode * tempPtr = frontPtr;
    if (frontPtr == backPtr) { //only one node.
      frontPtr = NULL;
      backPtr = NULL;
    }
    else. frontPtr = frontPtr -> next.
    tempPtr -> next = NULL; // defensive strategy.
    delete tempPtr;
  }
}
```

6

```
n = 10
n = 100
for (a = 1; a <= n; a++)
  for (b = 1; b <= a; b++)
    for (c = 1; c <= 5; c++)
      cout << a << b << c << endl;
```

a = 1	b = 1	c = 5 x 1
a = 2	b = 1, 2	c = 5 x 2
a = 3	b = 1, 2, 3	c = 5 x 3
:	:	: ← n

how many times units does the nested loop take.

$\Sigma (5+5\times a)$ for $a = 1$ to $n \Rightarrow 5 * t * \boxed{n*(n+1)/2}$

$t*(2.5n^2+2.5n) \rightarrow n=10: 275t,\ n=100: 25250t$

(×10, ÷×10)

Growth Rate

Bubble sort

```
for (pass == 1 ; pass < n ; pass++)        第幾回合.

  for (int index = 0; index < n - pass ; index ++){ 第幾筆資料
    if (A[index] > A[index+1])
      swap (A[index], A[index+1]);
  }
}
```

比較次數.

$n + \Sigma_{pass=1 \dots n-1}(n-pass+1) + \Sigma_{pass=1 \dots n-1}(n-pass)$

$= n + 2[n*(n-1) - n*(n-1)/2] + (n-1) = n^2+n-1 \rightarrow O(n^2)$

核心比較次數.

$(n-1) + (n-2) + (n-3) + \dots + 1$

$= n*(n-1)/2 = 0.5n^2 - 0.5n \rightarrow O(n^2)$

7

Selection Sort

```
void selectionSort (int A[], int n){
  for (last = n-1; last >0; last --){
    int largest = indexoflargest (A, last+1);
    swap (A[largest], A[last]);
  }
}
```

```
int indexoflargest (int A[], int size){
  int indexSoFar = 0;
  for (index = 1; index < size; index++)
    if (A[indexSoFar] < A[index])
      indexSoFar = index;
  return indexSoFar;
}
```

Merge Sort.

```
void mergeSort (DataType theArray[], int first, int last){
  if (first < last){
    int mid = (first + last)/2;
    mergeSort (theArray, first, mid);
    mergeSort (theArray, mid+1, last);
    merge (theArray, first, mid, last);
  }
}
```

```
void merge (DataType theArray[], int first, int mid, int la
} DataType tempArray[MAX_SIZE];
  int first1 = first, last1 = mid;
  int first2 = mid+1, last2 = last;
  int index = first;
  for (; (first1 <= last1) && (first2 <= last2); ++index)
    if (theArray[first1] < theArray[first2]){
      tempArray[index] = theArray[first1];
      ++first1;
    } else {
      tempArray[index] = theArray[first2];
      ++first2;
    }
  for (; first1 <= last1; ++first1, ++index)
    tempArray[index] = theArray[first1];
  for (; first2 <= last2; ++first2, ++index)
    tempArray[index] = theArray[first2];
  for (index = first; index <= last; ++index)
    theArray[index] = tempArray[index];
}
```

8

Quick Sort.

```
void quickSort (DataType theArray[], int first, int last){
  int pivotIndex;
  if (first < last){
    partition (theArray, first, last, pivotIndex);
    quickSort (theArray, first, pivotIndex - 1);
    quicksort (theArray, pivotIndex+1, last);
  }
}
```

```
void partition (DataType theArray[], int first, int last, int & pivotIndex){
  choosePivot (theArray, first, last);
  DataType pivot = theArray[first];
  int lastS1 = first;
  int firstUnknown = first + 1;
  while (firstUnknown <= last){
    if (A[firstUnknown] < pivot){
      ++lastS1;
      swap (A[lastS1], A[firstUnknown]);
    }
    ++firstUnknown;
  }
}
```

Radix Sort

```
void radixSort (int A[], int first, int last){
  int temp[MAX_D][MAX_SIZE], maxData;
  int counter[10] = {0}, i, j;
  for (maxData = A[first], i = first+1; i <= last; i++)
    if (maxData < A[i])  maxData = A[i];
  for (int base = 1; (maxData/base) > 0; base *= 10){
    for (i = first; i <= last; i++){
      int LSD = (A[i]/base) % 10;
      temp[LSD][counter[LSD]] = A[i];
      counter[LSD]++;
    }
    ...
```

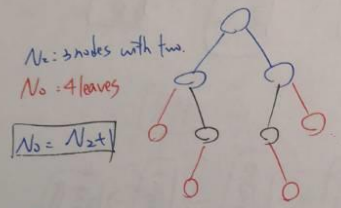
```
for (base = 1; (maxData/base) > 0; base *= 10)
  int k=0;
  for (i=0; i<10; i++)
    if (counter[i] > 0){
      for (int j=0; j<counter[i]; j++, k++)
        A[k] = temp[i][j];
      counter[i] = 0;
    } // if
} // for.
```

Height of a tree.
- Number of nodes along the longest path from the root to a leaf.

minimum height of n nodes

$n \le 2^h - 1 \Rightarrow \log_2(n+1) \le \log_2(2^h)$

$h \ge \log_2(n+1) \Rightarrow h = \lceil \log_2(n+1) \rceil$



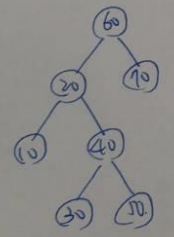

```
search (in bst: BinaryST , in searchKey: keyType)
  if (bst is empty)
    Item not found
  else if (searchKey == root key)
    Item is found
  else if (searchKey < root key)
    search (bst's left subtree, searchKey)
  else
    search (bst's right subtree, searchKey)
```

```
class TreeNode{
private:
    TreeItemType item;          // data portion.
    TreeNode * leftChildPtr;    // 左子節點
    TreeNode * rightChildPtr;   // 右.
};
TreeNode * root;
```

delete

M is the leftmost node in X's right subtree
M will have no more than one child (0 or 1)
M's key is called the inorder successor of X's key.

Copy the item that is in M to X
Remove the node M from the tree



前序 Preorder : 60 20 10 40 30 50 70
中序 Inorder : 10 20 30 40 50 60 70
後序 Postorder 10 30 50 40 20 70 60.