Subject : .....

No. :

Date : ...../...../.....

堆疊 (Last in First Out)

two Application of stack
1. 代數運算式求解
2. search a Path

ex. 刪除字

```
while (not end of line){
    Read a new ch
    If (ch not '←')
        ADD(ch to ADT)
  else if ( ADT is not empty)
        Remove( last item from ADT)
  else
        Ignore '←'
}

while ( ADT is not empty){
    Retrieve (the last item from ADT)
    Put it in ch
    Remove( the last item from ADT)
}
```

```
7 isEmpty()
 push()
 pop()
 getTop()
 pop(ch)    //存到ch,再刪
```

Subject : .....

No. :

Date : ...../...../.....

ex. 括號檢查

```
balancedSoFar = true
while ( not end of string && balancedSoFar) {
    Read next character ch
    If( ch is '(')
        stack.push (ch)
    else if ( ch is ')') {
        If (!stack.isEmpty())
            stack.pop()
        else
            balancedSoFar = false
    }
}
If ( balancedSoFar && stack.isEmpty())
    return true
else
    return false
```

Implementations of the ADT stack

1. Array
2. Pointer more efficient
3. ADT List

321 → 231, 213, 123, 132, 321

先進後出

$\exists\ C_n^{2n} - (\text{push}\cdot\text{pop}\cdot\text{pop}\cdot\text{push}\cdot\text{push}\cdot\text{pop})$ 非

$-(\text{pop}\cdot\text{push}\cdot\text{push}\cdot\text{pop}\cdot\text{push}\cdot\text{pop})$ 非

Double A

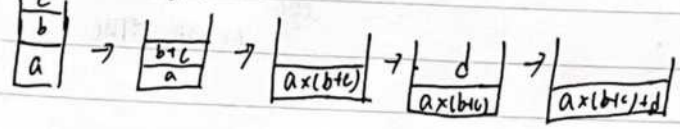
Subject : .....

No. :

Date : ...../...../.....

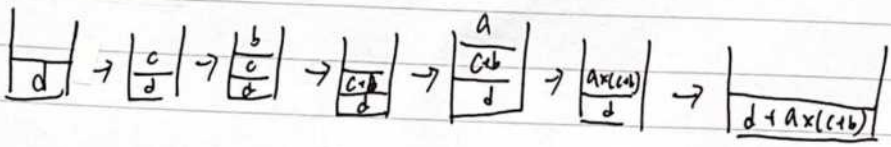
Postfix Evaluator.

a b c + x d +



prefix

+ x a + b c d



中序轉後序

1. 數字直接輸出
2. 遇到運算子, 比較優先度 (*、/ > +、-)
3. 遇到 ')', 輸出 stack 中運算子, 直到 '('

P → Z

		stack
push P	R. W	P
push R	X	PR
push X	X	PRX
pop X	X	PR
pop R		P
push W	S	PW
push S	T. Y	PWS
push T	X	PWST
pop T	X	PWS
pop S		PW
push Y		PWY
push Z		PWYZ

```
isPath (int orcity, int descity){
    stack astack;
    int topcity, nextcity;
    bool success;

    unvisitAll();
    astack.push(oricity)
    visited(oricity)
    astack.getTop(topcity)
    while (!astack.isEmpty() && topcity!=
        success = getNextCity(topcity, nextcity, des
        if (!success)
            astack.pop()
        else {
            astack.push(next city)
            visited(next city)
        }
        if (!astack.isEmpty)
            astack.getTop(topcity)
    } // while
```

Subject : .....

No. :

Date : ...../...../.....

```
if (astack.isEmpty())
    return false
else
    return true.
```

3 2

Double A

No.

DATE.

Queue (佇列)

A queue : New items enter at the "back" (rear) of the queue 後端
: Items leave from the "front" of the queue 前端
⇒ First in, first out

ADT queue operations
1. Create an empty queue
2. Destory a queue
3. isempty
4. Add a new item *enqueue
5. Remove *dequeue
6. Retrieve *getfront

str to num :

```
do {
    aQueue.dequeue(ch)
} while(ch is blank)
n=0
done = false
while (!done and ch is digit) {
    n = n*10 + integer of ch
    if (aQueue.isEmpty())
        done = true
    else
        aQueue.dequeue(ch)
}
```

Palindromes.

```
aQueue.createQueue();
aStack.createStack();
for(the next ch in str){
    aQueue.enqueue(ch);
    aStack.push(ch);
}
charEqual = true;
while (!aQueue.isEmpty() && charEqual){
    aQueue.getFront(front)
    aStack.setTop(top)
    if(front == top){
        aQueue.dequeue()
        aStack.pop()
    }
    else
        charEqual = False;
}
```

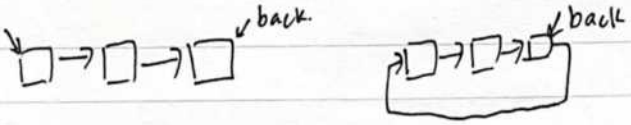
No.

DATE.

pointer-based queue

a reference to the front) linear linked list
a reference to the back

a reference to the back circular linked list



Linear

```
enqueue (const QueueItemType & new item){
        QueueNode *newPtr = new QueueNode;
        newptr->item = newitem;
        newptr->next = Null;
        if ( isEmpty()) ·
            frontPtr = newPtr;
        else
            backPtr->next = new Ptr;
        backptr = newPtr;
}

dequeue()  throw (QueueException){
    if (isEmpty())
        throw QueueException ("QueueException...");
    else {
        QueueNode *tempPtr = frontPtr;
        if (frontPtr == backPtr) {
                frontPtr = Null;
                backPtr = Null;
        }
        else
            frontPtr = frontPtr->next;

        tempPtr->next = Null
        delet tempPtr;
    }
}

getfront (const QueueItemType & Queuefront){
    if( isempty1())
        throw QueueException( "QueueException: ...");
    else
        Queuefront = frontPtr->item;
}
```

NAN PAO

Subject : .....

No. :

Date : ...../...../.....

Circular

```
enqueue (const QueueItemType & newItem) {
    QueueNode * newPtr = new QueueNode;
    newPtr → item = newItem;
    if( isEmpty() )
        newPtr → next = newPtr;
    else {
        newPtr → next = backPtr → next;
        backPtr → next = newPtr
    }
    backPtr = newPtr
}


dequeue ( ) throw (QueueException) {
  if(isempty())
        throw QueueException ("QueueException");
  else {
    QueueNode * tempPtr = backPtr → next;
    if ( backPtr == backPtr → next)
        backPtr = Null;
    else
        backPtr →next = tempPtr → next;
    tempPtr → next = Null;
    delete tempPtr;
  }
}
```

Array - based Queue

Circular

```
isEmpty() {
    return (!isFull) && (front == (back + 1) % max_queue);
}
enqueue( & item) throw( QueueException) {
  if (isFull == true)    throw ....;
  else {
    back = (back+1) % max_Queue;
    item [back] = newItem;
    if( front == (back+1) % max_Queue )  isFull = true;   } // else   } // enqueue
```

Double A

Subject : .....

No. :

Date : ...../...../.....

```
dequeue() {
 if (isEmpty())
        throw ...;
 else {
      front = (front +1) % max_Queue;
      if (isFull == true)
            isFull = False;
 }
}
```

Subject : Algorithm Efficiency

No. :

Date : ...../...../.....

Efficiency ⇒ Time & space

ex 1. traverse n個 node (link list)

⇒ n+1 個 comparisons, n+1 個 assignments, n writes

ex2. Towers of Hanoi with n disks

$2^{n}-1$ 個 moves

⇒ $M(n) = M(n-1) + 1 + M(n-1)$

ex3. How many time units does the nested loop take?

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

次數⇒ a:1, b=1, 5×1

a:2, b=1,2, 5×2

⋮

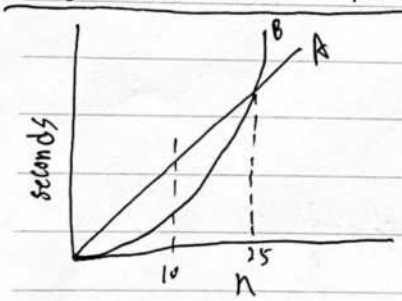
a:10, b=1~10, 5×10

⇒ 5*t*n*(n+1)/2

n=10 ⇒ 275t

n=100 ⇒ 25250t 不止10倍

Algorithm Growth Rates



B:指數 ⇒ n^2、$O(n^2)$、order-n^2

A:線性 ⇒ n、O(n)、order-n

Definition of the order of an algorithm

存在2個常數 k 和 n_0，使演算法能夠不超過 k*f(n)時間內解決大小不少於 n_0 的問題

Subject : .....

No. :

Date : ...../...../.....

ex.

$2.5n^2 - 2.5 * n$

$\exists\, \forall n \geq 10,\ (2.5n^2 - 2.5*n) \leq 1 * n^{10}$ $O(n^{10})$

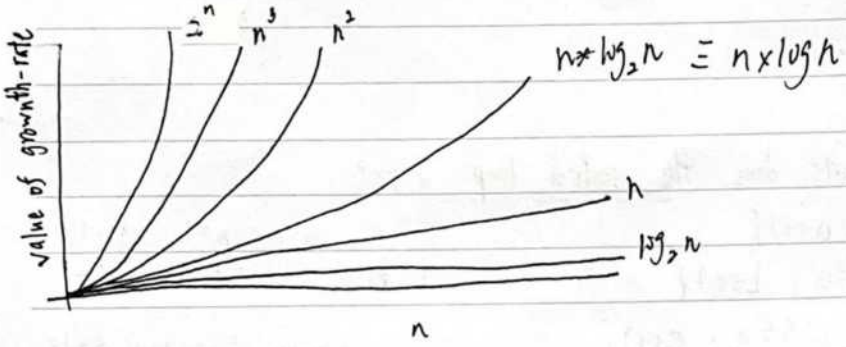
$\exists\, \forall n \geq 0,\ (2.5n^2 - 2.5*n) \leq 3 \times n^2$ $O(n^2)$

ex

$(n+1)*(c+a) + n*w$

$\forall n \geq n_0,\ (n+1)*((c+a) + n*w \leq k * f(n)$

$\forall n \geq 1,\ (n+1) \leq 2n$ $O(n)$



properties of growth-rate function

1. 忽略低位階
2. 忽略常數
3. $O(f(n)) + O(g(n)) = O(f(n)+g(n))$

* worst-case : Maximum
  Average-case : average
  Best-case : minimum

Sequential search

worst-case

2
8
1
9
5
7

$O(n)$

Average case

第幾步: 1 2

2 8
3 1
4 9
5 5
6 7

$(n+1)/2 \rightarrow 3.5$

$O(n)$

Best case

7

8
1
9
5
7

$O(1)$

Double A

No.

DATE.

Binary search of a sorted array

worst $O(\log_2 n)$

1 [6/2] =3
2 [3/2] =2
5 [2/2] =1
7
8
9

} 3次

$\Rightarrow n = 2^k$

average $O(\log_2 n)$

$\Rightarrow \log_2 10^6 = 19.9$

Best $O(1)$

* sequential search on sorted Data

worst	$O(n)$	$O(n)$
Average	$O(n)$	$O(n)$
Best	$O(1)$	$O(n)$
	(sorted)	(unsorted)

沒差太多

Categories of sorting algorithms

An internal sort : requires that the collection of data fit entirely in the computer's main memory

An external sort: The collection of data will not fit in the computer's main memory all at once, but must reside in secondary storage.

stable sort vs unstable sort

bubble
insertion
merge
radix

quick
heap

⇒ 相同值維持不變的排序 (stable)

NAN PAO

Comparison : Σ size = n.. 2 (size-1)

No.

DATE.

Bubble sort

23. 78. 45. 8. 32. 56

→ 8 | 23, 78, 45, 32, 56

→ 8. 23 | 32, 78, 45, 56

→ 8. 23. 32 | 45. 78. 56

→ 8. 23. 32 45 | 78. 56

→ 8. 23. 32. 45. 56 | 78

比較次数 & swap次数

22比較

selection sort

23. 78. 45. (8). 32. 56

→ 8 | 78. 45. (23) 32. 56

→ 8. 23 | 45. 78. (32). 56

→ 8. 23. 32 | 78. (45). 56

→ 8 23. 32. 45 | 78. (56)

→ 8. 23. 32. 45 56 | 78

swap 回数 Bubble sort 少

Insertion sort

固定 (23) | 78. 45. 8. 32. 56

→ 23. 78 | 45, 8. 32. 56

→ 23. 45. 78 | 8. 32. 56

→ 8. 23. 45. 78 | 32. 56

→ 8. 23. 32. 45. 78 | 56

→ 8. 23. 32. 45. 56. 78

No.

DATE.

2. 8a. 28. 14. 5a . 8b. 26. 2. 6. 29. 5b

1. bubble sort ⇒ 2. 5a. 5b. ----- ⇒ stable
2. selection sort ⇒ 2 ... 8b ... 8a ... ⇒ unstable
3. insertion sort ⇒ ⇒ stable

Bubble sort 複雜度

```
void bubbleSort (int A[], int n) {                    1+(n-1) assignments + n comparison (pass<n)
  bool sorted = false
  for (pass = 1; pass < n; pass++) {
    sorted = true                                     1+(n-pass) assignments + n-pass+1 comparison
    for (int index = 0; index < n-pass; index++) {
      if (A[index] > A[index+1])                      1 comparison
        swap(A[index], A[index+1])                    核心比較次數
        sorted = false
  }
}
```

⇒ unstable

✓ ⇒ best case O(n)

⇒ comparison : $n + \sum_{pass=1 \cdots n-1} \times (n-pass+1) + \sum_{pass=1 \cdots n-1} \times (n-pass)$

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

⇒ $(n-1)+(n-2)+(n-3) \cdots +1 = n*(n-1)/2 = 0.5n^2 - 0.5n \Rightarrow O(n^2)$

selection sort 複雜度 (最差)

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

Data exchanges: n-1 swap ⇒ O(n)

Comparison : $\sum_{size=n \cdots 2} (size-1) = (n-1)+(n-2)+\cdots+1 = n(n-1)/2 \Rightarrow O(n^2)$

NAN PAO

height: 3 height: 5 Height: 1

No.

DATE.

Insert sort 複雜度

```
void insertionSort (int A[], int n){
    for (unsorted=1; unsorted<n; ++unsorted){
        int loc = unsorted, nextItem= A[unsorted];
        for( ; loc>0 && (A[loc-1]>nextItem); --loc)
            A[loc] = A[loc-1];
        A[loc] = nextItem;
    }
}
```

>= unstable

* outer for loop: n-1 times
  inner for loop: at most unsorted times, unsorted = 1... n-1

worst case: 1+2+...+(n-1) = n*(n-1)/2 ⇒ $O(n^2)$

Shellsort

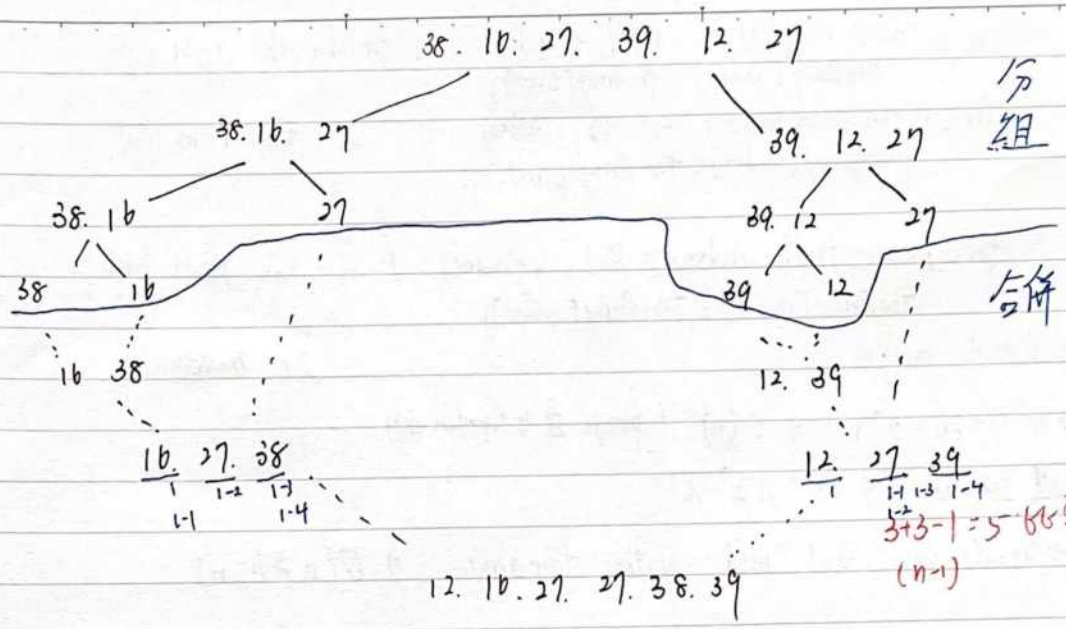
```
void Shellsort(int A[], int n){
    for(int h=n/2; h>0; h=h/2){
        for(int unsorted=h; unsorted<n; ++unsorted){
            int loc = unsorted;
            int nextItem = A[unsorted];
            for(; loc>=h && (A[loc-h]>nextItem); loc=loc-h)
                A[loc] = A[loc-h];
            A[loc]= nextItem;
        }
    }
}
```

⇒ not stable

Mergesort

- A recursive sorting alorithm
- Performance is independent of the initial order of the array items.
- strategy
  1. Divide an array into halves 先:分組
  2. sort each half 各自排序
  3. Merge the sorted halves into one sorted array 後:合併
  4. Divide-and-conquer

NAN PAO



=> stable

```
void mergesort(DataType theArray[], int first, int last) {
    if(first < last) {
        int mid = (first + last)/2
        mergeSort(theArray, first, mid);    // sort left
        merseSort(theArray, mid+1, last);   // sort right
        merse(theArray, first, mid, last);  // merge the two halves
    }
}
void merge(DataType theArray[], int first, int mid, int last) {
    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];
            ++first;
        }
    else {
        tempArray[index] = theArray[first2];
        ++first2;
    }
```

n-1次比較

No.

DATE:

```
for (; first1 <= last1; ++first1, ++index)    // finish the left half
    tempArray[index] = theArray[first1];
for (; first2 <= last2; ++first2, ++index)    // finish the right half
    tempArray[index] = theArray[first2];

for(index = first, index <= last; ++index)    // copy the result back
    theArray[index] = tempArray[index];
} // end merge
```

2n moves

⇒ $(n-1)+2n = 3*n-1$ ⇒ $O(n)$ (merge 這分function的)

⇒ 遞迴呼叫 ⇒ $3*n-2^x$ 次

⇒ extremely fast, but need another temp array ⇒ $O(n*\log_2 n)$

Quick sort

Strategy:

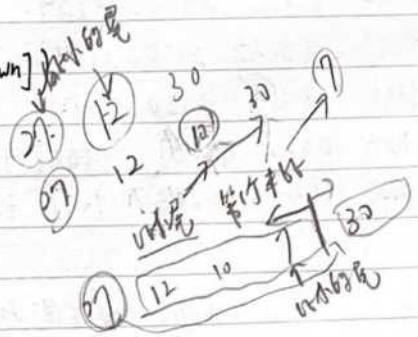
1. Choose a pivot
2. partition the array about the pivot
   - items < pivot
   - items >= pivot
   - pivot is now in correct sorted position
3. Sort left
4. Sort right

```
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);
    }
}
```

No.

DATE.

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



⇒ unstable

① 27	38	12	39	27	16
① 27	② 38	12	39	27	16
27	38 S2	12	39	27	16
27	12 S1	38 S2	39	27	16
27	12 S1	38	39 S2	27	16
27	12 S1	38	39	27 S2	16
27	12 S1	16 S1	39	27	38 S2
12	16 S1	27	39	27	38 S2

Partition

recursive call

⇒ Average case $O(n * \log_2 n)$

worst case $O(n^2)$

Radix Sort

base of a system of numbers

strategy: Decompose the sort key by the radix

- Treats a key as a character string
- Repeatedly assign the keys into group according to the i^{th} character

NAN PAO

NAN PAO

No.

DATE.

ex.

0123. 2154. 0222. 0004. 0283. 1560. 1061. 2150

(1560. 2150)、(1061)、(0222)、(0123. 0283)、(2154, 0004) queue 5

1560、2150、1061. 0222. 0123. 0283. 2154. 0004

(0004)、(0222、0123)、(2150、2154)、(1560、1061)、(0283) 十

0004. 0222. 0123、2150. 2154. 1560. 1061. 0283

(0004. 1061)、(0123. 2150、2154)、(0222. 0283) (1560) 百

0004. 1061、0123. 2150. 2154. 0222. 0283. 1560

(0004, 0123. 0222, 0283)、(1061. 1560)、(2150. 2154) 千

0004. 0123. 0222、0283、1061. 1560. 2150. 2154

* LSD (Least Sishificant Digit) 最右側相等分組、串接

* MSD (Most sishificant Digit) 最左側相等分組、X

↔

```
void radixSort(int A[], int first, int last) {
    int temp[
```

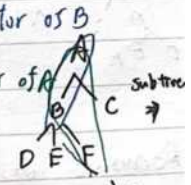
No.

DATE.

Tree.

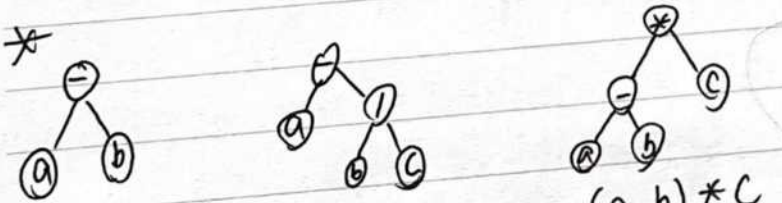
Terminology

1. Trees are composed of nodes and edges
2. Trees are hierarchical
   parent-child relationship between two nodes.
   Ancestor-descendant relationships among nodes.
3. subtree of a tree: Any node and its descendants.
4. General tree: A general tree T is a set of one or more nodes su
   that T is partitioned into disjoint subset:
   * A single node → the root
   * sets that are general trees, called subtree
5. parent of node X: The node directly above X in the tree.
6. Child of node X: A node direcly below X in the tree.
7. root: The only node in the tree with no parent.
8. Leaf: A node with no children.
9. siblings: Nodes with common parent.



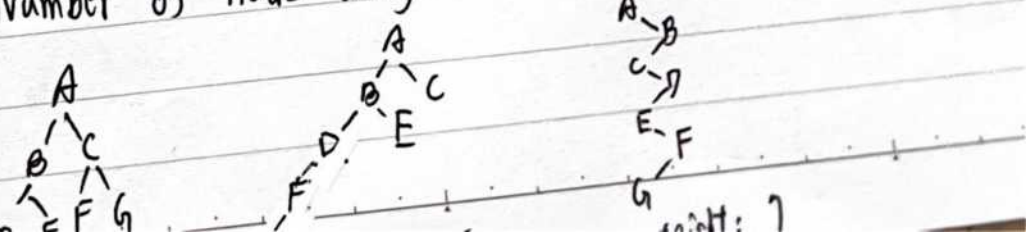
Binary tree

A binary tree is a set T of nodes such that either T is e
partitioned into three disjoint subsets:
* A single node r, the root
* left subtree, right subtree.



Height of a tree

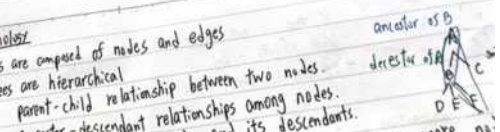
Number of node along the longest path from the



Tree.

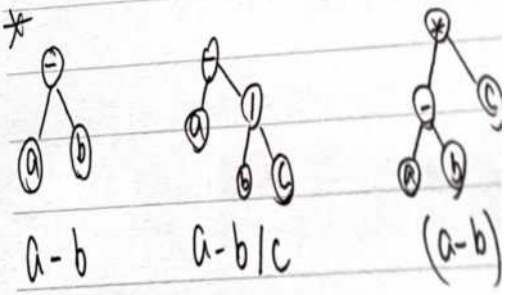
Terminology

1. Trees are composed of nodes and edges
2. Trees are hierarchical
   parent-child relationship between two nodes.
   Ancestor-descendant relationships among nodes.
3. subtree of a tree: Any node and its descendants.
4. General tree: A general tree T is a set of one or more no
   that T is partitioned into disjoint subset:
   * A single node = the root
   * sets that are general trees, called sub
5. parent of node X: The node directly above X in the
6. Child of node X: A node directy below X in
7. root: The only node in the tree with no par
8. Leaf: A node with no children.
9. siblings: Nodes with common parent.



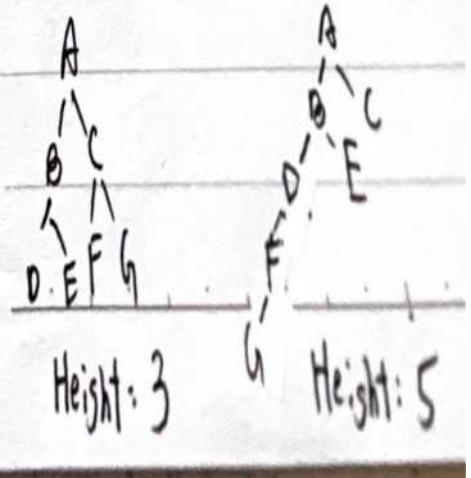
Binary tree

A binary tree is a set T of nodes such that
partitioned into three disjoint subsets:
* A single node r, the root
* left subtree, right subtree.



Height of a tree

Number of node along the l



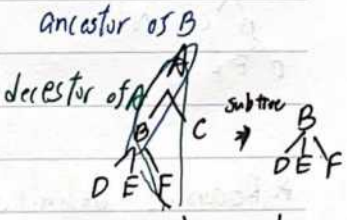
No.

DATE.

Tree.

Terminology

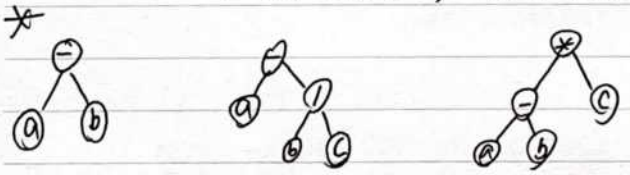
1. Trees are composed of nodes and edges
2. Trees are hierarchical
   parent-child relationship between two nodes.
   Ancestor-descendant relationships among nodes.



3. subtree of a tree: Any node and its descendants.
4. General tree: A general tree T is a set of one or more nodes such that T is partitioned into disjoint subset:
   * A single node → the root
   * sets that are general trees, called subtree
5. parent of node X: The node directly above X in the tree.
6. Child of node X: A node directy below X in the tree.
7. root: The only node in the tree with no parent.
8. Leaf: A node with no children.
9. siblings: Nodes with common parent.

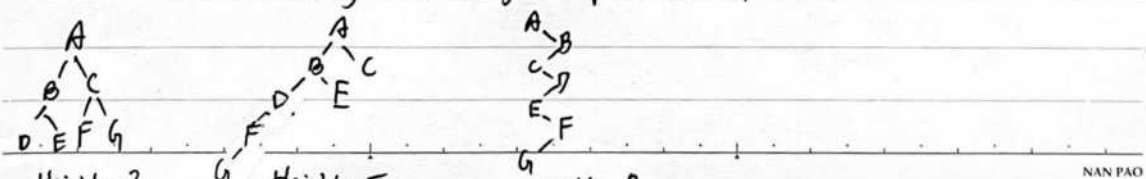
Binary tree

A binary tree is a set T of nodes such that either T is empty ✓ T is partitioned into three disjoint subsets:
* A single node r, the root
* left subtree, right subtree.



Height of a tree

Number of node along the longest path from the root to leaf

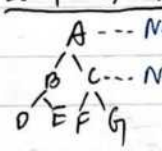


NAN PAO

No.

DATE

Level of node n



if tree is not empty, its height is equal to the maximum level of its nodes.

* if tree is empty, height = 0.

* Recursive definition of height

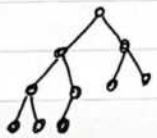
If T is empty, its height = 0

If T is not empty, height(T) = 1 + max{ height(T_L), height(T_R) }

Full Binary tree

每個點都滿

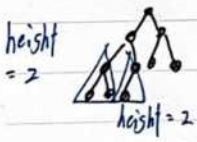
Complete Binary tree



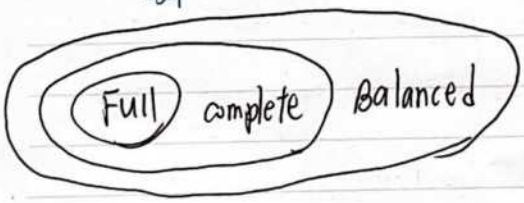
It is full to level h-1

Level h is filled from left to right.

Balanced Binary tree



A balanced Binary tree the height of any nodes 2 subtree differ by no more than 1.



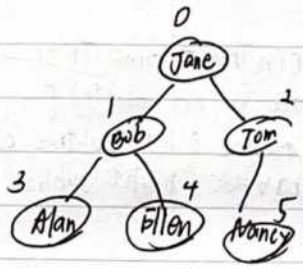
No.

DATE.

Array - based ADT Binary Tree

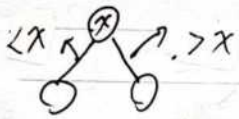
```
const int MAX_Node = 100;
class TreeNode {
    private:
      treeitemtype item;
      int leftchildren;
      int rightchildren;
};

TreeNode tree[MAX_NODES];
int root;
int free;
```



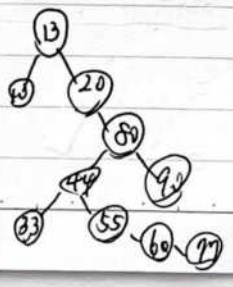
* leftchild = 2*parent + 1
rightchild = 2*parent + 2
parent = (child - 1)/2

Binary search Tree




```
search (int bst = Binarysearchtree, int searchkey = KeyType)
    if (bst is empty)
        not found
    else if (searchkey == rootkey)
        found
    else if (searchkey < rootkey)
        search ( bst's left subtree, searchkey)
    else
        search ( bst's right subtree, searchkey)

insertitem ( in treePtr = TreeNodePtr, in newItem = TreeItemType)
    if (search stop at x's left subtree)
        make x's leftchildPtr point to newItem
    else
        make x's rightchildPtr point to newItem
```




```
class TreeNode {
private:
    TreeItemType item;
    TreeNode    *leftchildPtr;
    TreeNode    *rightchildPtr;
};
```

No.

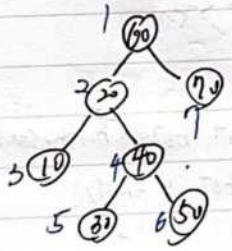
DATE.

Traversals of a Binary tree

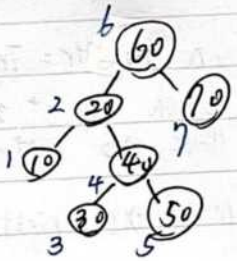
```
traverse ( in binTree = BinaryTree)
    if (bintree is not empty) {
        traverse (left subtree of binTree's root)
        traverse (right subtree of binTree's root)
    }
```

* Preorder visit root before visiting its subtree, before recursive calls
  Inorder visit root between visiting its subtree, between recursive calls
  postorder visit root after visiting its subtree, after recursive calls

```
pre( binary tree root) {
    binarytree treeptr = root;
    nodestack astack;
    while( !astack.empty() || treeptr != Null ){
      while ( treeptr != Null) {
          cout << treeptr -> data;
          astack.push (treeptr -> rightchild);
          treeptr = treeptr -> leftchild
      } // while
      astack.pop (treeptr);
    } // while
```




```
In ( binary tree root) {
    binarytree treeptr = root;
    nodestack astack;
    while (!astack.empty() || treeptr != Null) {
      while ( treeptr != Null) {
          astack.push(treeptr);
          treeptr = treeptr -> leftchild
      }
      astack.pop(treeptr);
      cout << treeptr -> data;
      treeptr = treeptr -> rightchild;
    }
```



NAN PAO