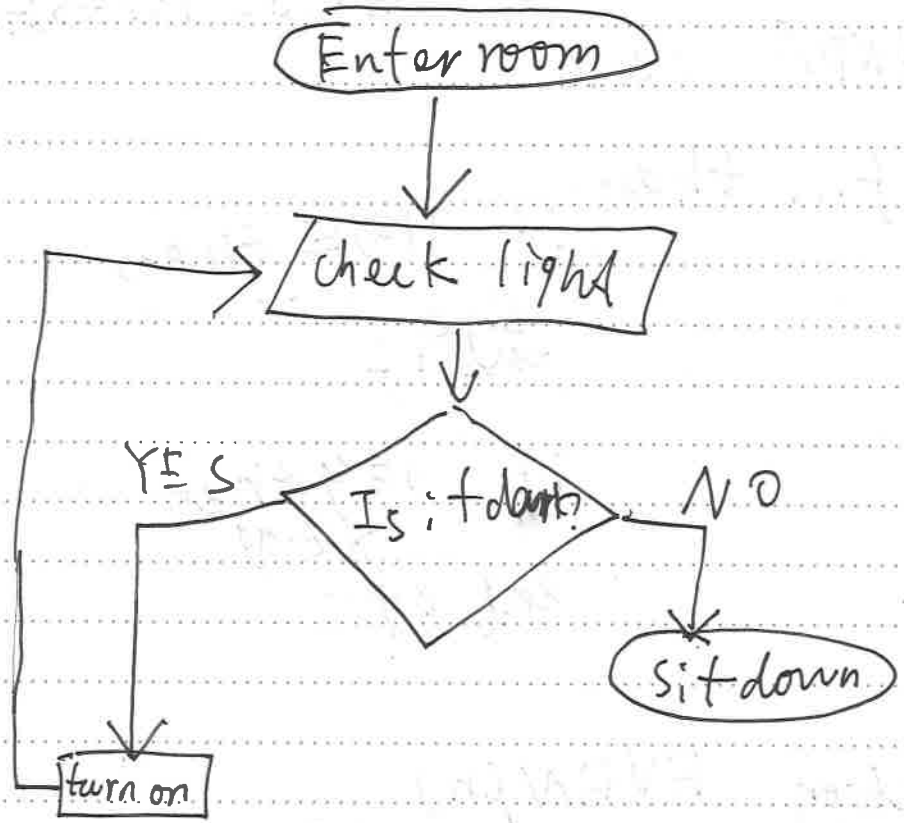
ADS 1

1.3. Flowcharts



2.1

oval — Terminal (start/end)

Parallelogram — I/O actions (gather or display data)

→ Control flow

diamond — Decisions (YES/NO)

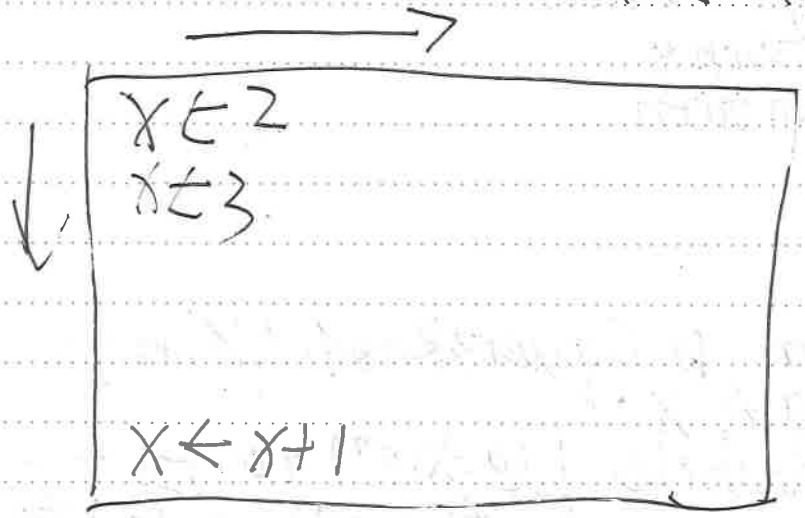
rectangle basic actions

2.1 Pseudocode

Assignment ←

$x \leftarrow 2$

$x \leftarrow$ TRUE



Arithmetic

$+ \quad - \quad \times \quad /$

Comparison

$= \quad \neq \quad < \quad > \quad \leq \quad \geq$

NOT ¬

¬TRUE = FALSE

AND ∧, OR ∨

If... then

```
if TRUE then
   ...
else
end if
```

```
if x=y then
    x←y
end if
```

```
function EVEN(n)
    if n mod 2 = 0
        return TRUE
    else
        return FALSE
    end if
end function
```

← body

```
x ← 1                          ← condition
for 2 ≤ i ≤ 10 do
    x ← x + i                  ] body
end for
```

```
x ← 1
y ← 0
while x < 11 do
    y ← x + y                  ] body
    x ← x + 1
end while
```

If $x^2 = n$, is x an integer?

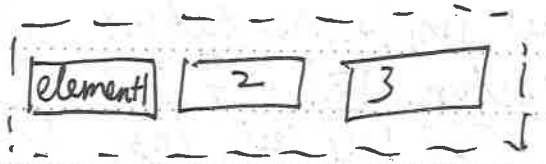
```
function IsXInteger(n)
    y ← FALSE
    for 1 ≤ i ≤ n do
        if i^2 = n then
            y ← TRUE
        end if
    end for
    return y
end function
```

```
function IsXInteger(n)
    y ← FALSE
    i ← 1
    while i^2 ≤ n then
        if i^2 = n then
            y ← TRUE
        end if
        i ← i + 1
    end while
    return y
end function
```

```
function SquareRoot(X, n)
    g ← X
    while ⌊(g × 10^n) + 0.5⌋ −
          ⌊(X/g × 10^n) + 0.5⌋ ≠ 0 do
        g ← 1/2 (g + X/g)
    end while
```

3.1 Vectors

a vector: v



sequential: ordered elements

cf. ordered set (total order)

Operation	Pseudocode
length	LENGTH[v]
select[k]	v[k]
store![o,k]	v[k] ← o
delete![k] ✗	the length is fixed
add![o] ✗	
Construct new(blank)	new Vector(n)
vector of length n	

3.2 Queues

a queue: tail E1 → E2 → E3 → E4 → E5 head

sequential: ordered elements

Extensible

add element to tail

remove element from head

Operation	Pseudocode
head	HEAD[q]
dequeue!	DEQUEUE[q]
enqueue![o]	ENQUEUE[o,q]
empty?	EMPTY[q]
Construct new(empty) queue	new Queue q

3.3 Stacks

a stack: E1 ⇄ E2 ⇄ E3 ⇄ E4

top

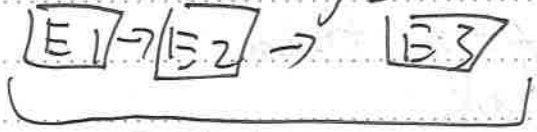
sequential: ordered elements

Extensible: add element to top

remove element from top

Operation	Pseudocode
push![o]	PUSH[o,s]
top	TOP[s]
pop!	POP[s]
empty?	EMPTY[s]
(construct new(empty) stack	new Stack s

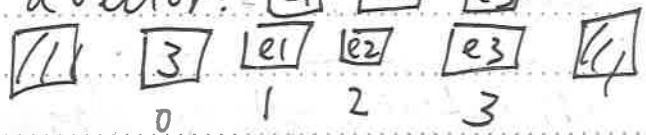
4.1 Dynamic arrays



Operation	Pseudocode
length	LENGTH[d]
select[k]	d[k]
store![o,k]	d[k] ← o
removeAt![k]	d[k] ≠ ϕ k ≤ LENGTH[d]
insertAt![o,k]	d[k] ← o k ≤ LENGTH[d]+1

Array implementation of Vector

a vector: E1 E2 E3

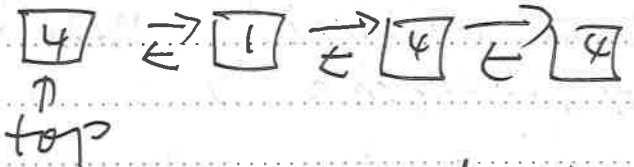


length
select[k]
store![o,k]
removeAt![2]

read[o]
read[k]
write![o,k]
write![Element 3,2]
write![3]
write![2,0]

insertAt![E4,2] (new array of size 5)
write![4,0]
write![E1,1]
write![E4,2]

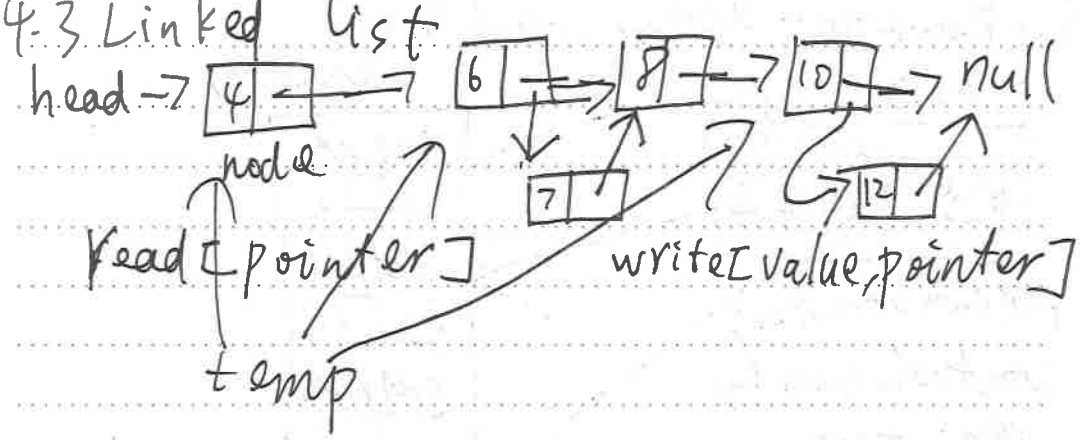
4.2.5 search stacks and queues.



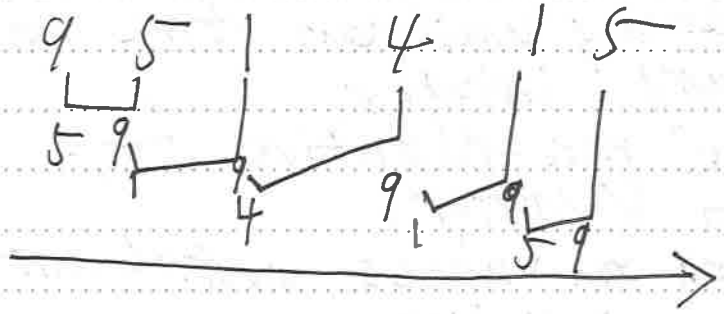
push to a second stack

queue: dequeue and enqueue,
special value: end of queue.

4.3 Linked List



5.1 Bubble sort




```
function Swap(vector, i, j)
    X ← vector[i]
    vector[j] ← V[i]
    V[i] ← X
    return vector
end function
```

```
function BubbleSort(vector)
    n ← LENGTH [vector]
    for 1 ≤ i ≤ n-1 do
        count ← 0
        for 1 ≤ j ≤ n-1 do
            if vector[j+1] < vector[j] then
                Swap(vector, j, j+1)
                count ← count + 1
            end if
        end for
        if count = 0 then
            break
        end if
    end for
    return vector
end function
```

5.2 Insertion sort

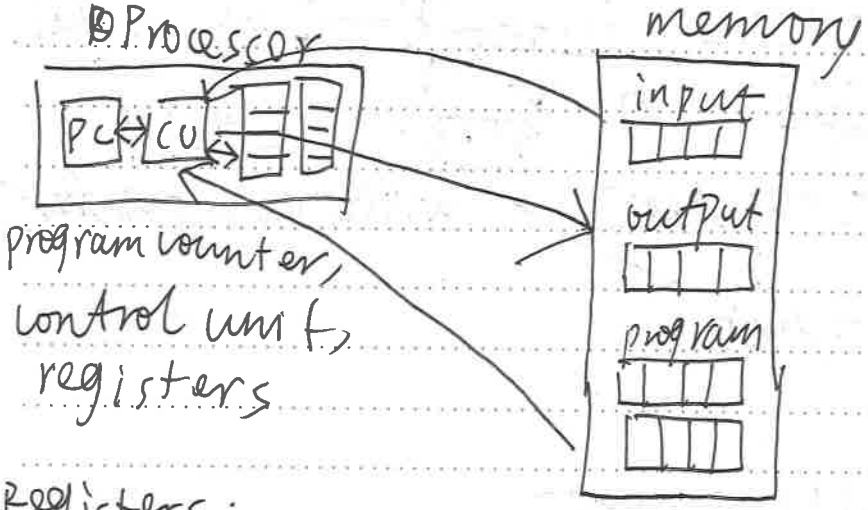
Shift for arrays, dynamic arrays, and vectors.

```
function Shift(array, i, j)
    if i ≤ j then
        return array
```

```
    end if
    store ← array[i]
    for 0≤k≤(i-j-i) do
        array[i-k] ← array[i-k-1]
    end for
    array[j] ← store
    return array
end function

function InsertionSort(vector)
    for 2≤i≤LENGTH(vector) do
        j ← i
        while(vector[i] < vector[j-1] ∧ (j>1)) do
            j ← j-1
        end while
        Shift(vector, i, j)
    end for
    return vector
end function
```

6.1 Random-access machine



registers:
- Each memory unit can store an arbitrary integer.
- Must be non-negative for program counter.
- Depending on values, control unit does an operation.

control unit:
- read, write, and copy values of memory units
- do simple arithmetic (add, subtract, multiple, divide)
- do condition operations (if_then_)
- each individual operation done in one time - stop

function Factorial (n)
a ← 1
for 1 ≤ i ≤ n do
a ← a × i
end for
~~end~~ return a
end function

Total: ~~4n+5~~ operations
4n+5

1. retrieve n
2. store n in register
3. store a —
4. store i —
5. check if i ≤ n
yes → 6
no → 9
6. multiply i and a and store result
7. increase i by 1
8. go to step 5
9. store a in output and stop

6.2 Growth of functions
exponential
$f(n) = 2^n$
$f(n-1) - f(n)$
2^n
doubling of the difference

~~polynomial~~ polynomial
$f(n) = n^2$
$f(n) = n$

logarithmic
$f(n) = \log_2 n$

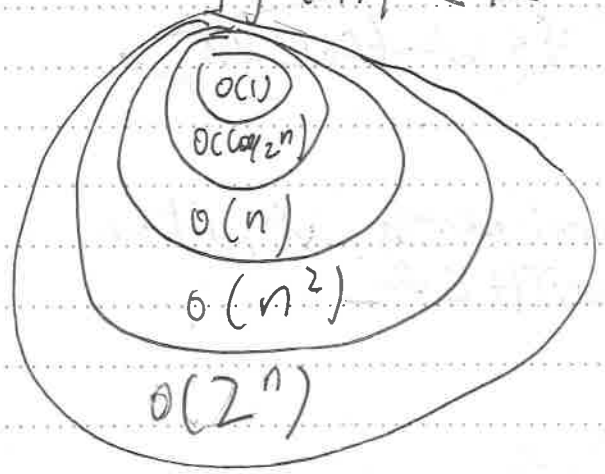
$2n+1$ linear
1 constant

$\log_2 \frac{n+1}{n} \leq \frac{1.5}{n}$

inverse linear

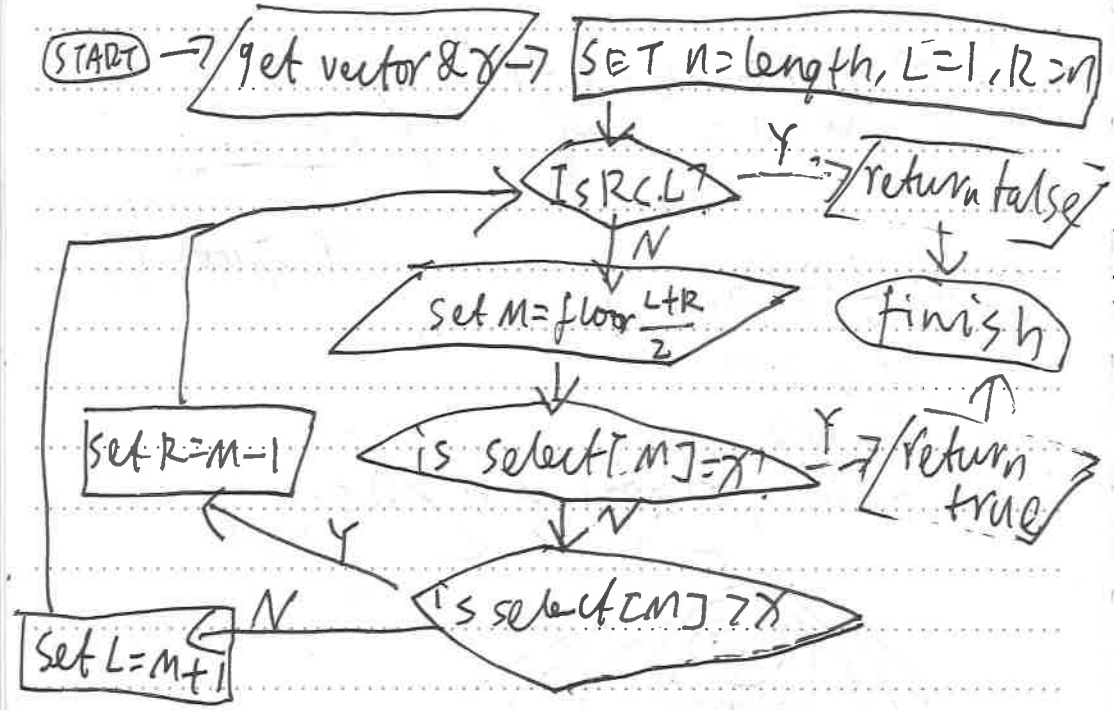
$O(\log_2 n) = O(\log_3 n)$
$\log_2 n = \log_3 n \,/\, \log_3 2$
$\log_3 2$ → constant

$f(n) \in O(g(n))$
$\exists k > 0$
$\exists n_0$
such that $\forall n > n_0$
$|f(n)| \leq k \cdot g(n)$



6.3.4 input size
size $n = O(\log n)$ bits

7.1 Binary search
mid-point = floor of (left + right)/2




```
function BinanySearch(v, item)
    n ← LENGTH[v]
    R ← n
    L ← 1
    while R ≥ L do
        M ← L (L+R)/2 ⌋
        if v(M) = item then
            return TRUE
        else if v[M] > item then
            R ← M-1
        else L ← M+1
        end if
    end while
    return FALSE
end function
worst-case complexity: O(logn)
```

8.1. Recursion

```
function GreatestCommonDivisor(a,b)
    if a=b then
        return a
    else if a>b then
        return GCD(a-b,b)
    else
        return GCD(a,b-a)
    end if
end function
```

```
function Search(v, L, item)
    n ← length[v]
    if L>n then
        return false
    else if v[L] = item then
        return true
    end if
```

```
    return Search(a, l+1, item)
end function

function LinearSearch(a, item)
    return Search(a, 1, item)
end function

function Sort(vector, r)
    if r≤1 then
        return vector
    end if
    for 1≤j≤r-1 do
        if vector[j+1]<vector[j] then
            Swap(vector, j, j+1)
        end if
    end for
    Sort(vector, r-1)
    return vector
end function
function BubbleSort(vector)
    n ← length[vector]
    return sort(vector, n)
end function
```

```
function Sort(vector, r)
    if r≤1 then
        return vector
    end if
    Sort(vector, r-1)
    j ← r, i ← r
    while(vector[i]<vector[j-1]) ∧ (j>1) do
        j ← j-1
    end while
    Shift(vector, i, j)
    return vector
end function
function InsertionSort(vector)
    n ← length[vector]
    return Sort(vector, n)
end function
function Shift(vector, i, j)
    if i≤j then
        return vector
    end if
    store ← vector[i]
    for 0≤k≤(i-j-1) do
        vector[i-k] ← vector[i-k-1]
    end for
```

```
    vector[j] ← store
    return vector
end function

function Search(v, item, L, R)
    if L>R then
        return false
    end if
    M ← L (L+R)/2 ]
    if v[M] = item then
        return true
    else if v[M] > item then
        R ← M-1
    else
        L ← M+1
    end if
    return Search(v, item, L, R)
end function
function Binary Search(v, item)
    n ← length [v]
    R ← n
    L ← 1
    return Search(v, item, L, R)
end function
```

9.1 Quick sort

9.2 Merge sort

```
function Merge(w, v)
    m ← length(w), n ← length(v)
    new Vector s(m+n)
    i ← 1,  j ← 1,  k ← 1
    while (i≦m) AND (j≦n) do
        if w[i] < v[j] then
            s[k] ← w[i]
            i ← i+1
        else
            s[k] ← v[j]
            j ← j+1
        end if
        k ← k+1
    end while
    while i≦m do
        s[k] ← w[i]
        i ← i+1,    k ← k+1
    end while
    while j≦n do
        s[k] ← v[j]
        j ← j+1 , k ← k+1
    end while
```

```
    return s
end function

function MergeSort (vector)
    n ← length[ vector ]
    if n=1 then
        return vector
    end if
    m ← floor( (n+1)/2 )
    new Vector L(m)
    new Vector R(n-m)
    L ← vector[ 1 : m ]
    R ← vector[ m+1 : n ]
    return Merge(MergeSort(L),
                 MergeSort(R) )
end function
```

	Worst-case Time	space
Bubble	$o(n^2)$	$O(1)$
Insertion	$o(n^2)$	$o(1)$
Quick	$O(n^2)$ *	$O(\log n)$
Merge	$o(n \log n)$	$o(n)$

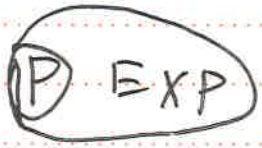
*: $O(n \log n)$ on average

10.2 Complexity classes

P: The set of all languages that can be decided by an algorithm in the RAM model with worst-case time complexity at most polynomial in the size of the input.

EXP: ~~_____~~ at most exponential* ~~_____~~

*: $O(2^{Poly(n)})$



NP: the set of all languages that can be decided by an algorithm in the RAM model that has ~~a~~ access to a proof (of polynomial size) with worst-case time complexity at most ~~polynomial~~ polynomial in the size of the input