Discrete Mathematics: Lecture 8: 3.1 Algorithms

algorithms

Def. 1: algorithm: a finite sequence of precise instructions for performing a computation or for solving a problem

find the largest value in a finite sequence of integers

1. set the temporary maximum equal to the first integer in the sequence

2. compare the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this integer

3. repeat the previous step if there are more integers in the sequence

4. stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence

properties of algorithms

- input: information or data that comes in
- output: information or data that goes out
- definiteness: the steps of an algorithm is precisely defined

- correctness: an algorithm should produce the correct output values for each set of input values
- finiteness: an algorithm should produce the desired output after a finite number of steps for any input in the set
- effectiveness: each step of an algorithm in a finite amount of time
- generality: the algorithm should be applicable for all problems

pseudocode

pseudocode provides an intermediate step between an natural language description of an algorithm and an implementation of this algorithm in a programming language

find the maximum element in a finite sequence

procedure max($a_1, a_2, \ldots, a_n$: integers)

max := a_1

for i :=2 to n

    if max < a_i then max := a_i

return max {max is the largest element}

pseudocode

- procedure statements

    procedure name(L: list of integers)

- assignments

    variable := expression

- informal statements

    interchange a and b

- comments

    {x is the largest element in L}

pseudocode

- conditional constructions

```
if condition then statement

if condition then
        statement1
        statement2
        .
        statementn

if condition 1 then statement 1
else if condition 2 then statement 2
else if condition 3 then statement 3
:
else if condition n then statement n
else statement n+1
```

```
if score > 90 then
        grade := A
else if score > 80 then
        grade := B
```

pseudocode

- loop constructions

```
for variable := initial value to final value
        statement 1
        statement 2
        :
        statement n
```

```
for i :=2 to n
        if max < a_i then    max := a_i
```

```
while condition
        statement 1
        statement 2
        :
        statement n
```

```
while x<3
        a := a+1
```

pseudocode

- using procedures in other procedures

```
procedure_name(parameter)
```

- return statements

```
return x
```

algorithms

- searching algorithms
  - linear search
  - binary search

- sorting algorithms
  - bubble sort
  - insertion sort

- greedy algorithm

searching algorithms

- searching algorithms locate elements in an (ordered) list
- for example,

  finding a student name in the roster

  finding words in a dictionary

linear search

procedure **linear search**(x: integer, $a_1, a_2, \ldots, a_n$: integers)

i := 1

while (i≤ n and x ≠ a_i)

    i := i + 1

if i ≤ n then location := i

else location := 0

return location {location is the subscript of the term that equals x, or 0 if x is not found}

linear search

search for 3 in the list 1 2 5 3 6 7 8

1	2	5	3	6	7	8

while ($i \leq n$ and $x \neq a_i$)

$i := i + 1$

search for 9 in the list 1 2 5 3 6 7 8

1	2	5	3	6	7	8

if $i \leq n$ then location := i

else location := 0

binary search

procedure **binary search**(x: integer, $a_1, a_2, \ldots, a_n$: integers in increasing order)

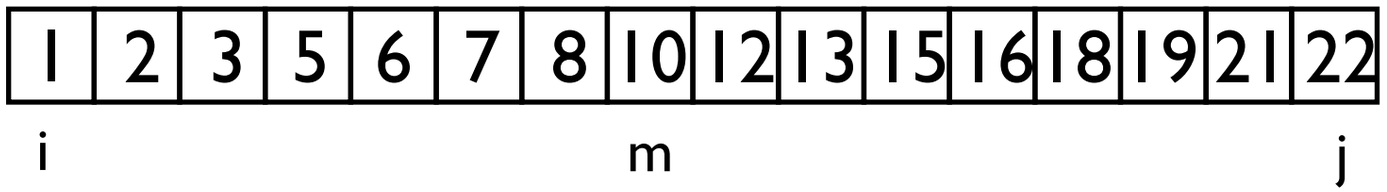
```
i := 1 {i is left endpoint of search interval}
j := n {j is right endpoint of search interval}
while (i < j)
        m := ⌊(i + j)/2⌋
        if x > a_m then i := m + 1
        else j := m
if x = a_i then location := i
else location := 0
return location {location is the subscript i of the term a_i equal to x, or 0 if
x is not found}
```

binary search

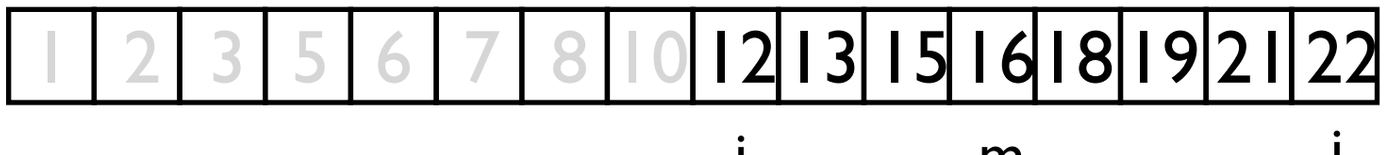
```
while (i < j)
        m := ⌊(i + j)/2⌋

        if x > a_m then i := m + 1
        else j := m
```

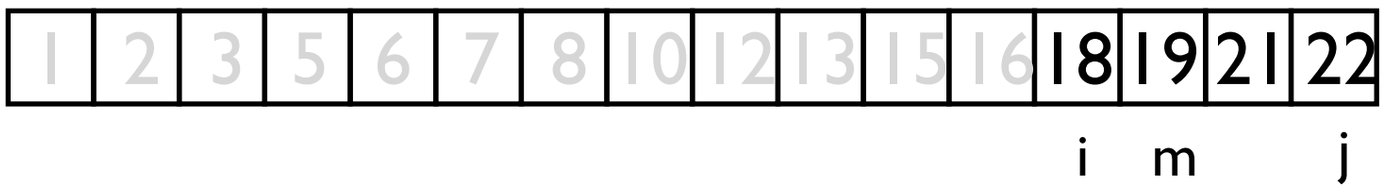
search for 19 in the list 1 2 3 5 6 7 8 10 12 13 15 16 18 19 21 22



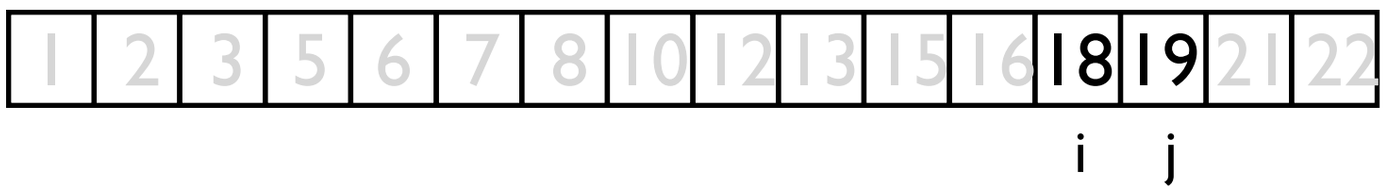
m := ⌊(i + j)/2⌋ = ⌊(1 + 16)/2⌋ = 8



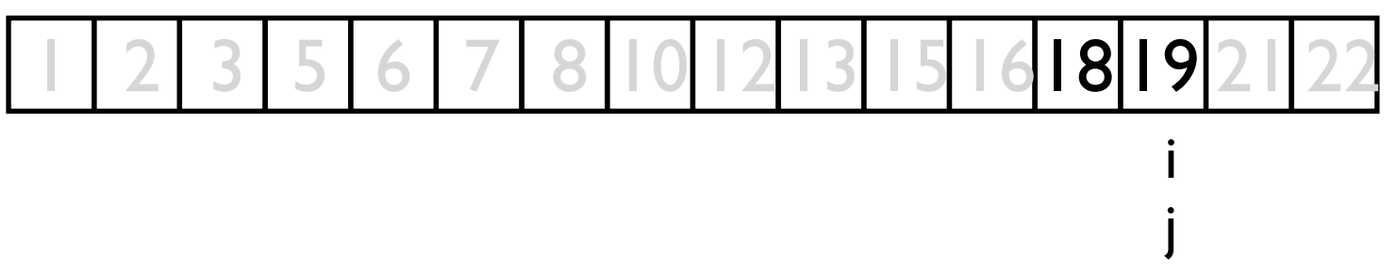
m := ⌊(9 + 16)/2⌋ = 12



m := ⌊(13 + 16)/2⌋ = 14



m := ⌊(13 + 14)/2⌋ = 13




```
if x = a_i then location := i
else location := 0
```

sorting algorithms

- sorting is putting the elements into a list in which the elements are in increasing order

bubble sort

- interchanging a larger element with smaller one following it

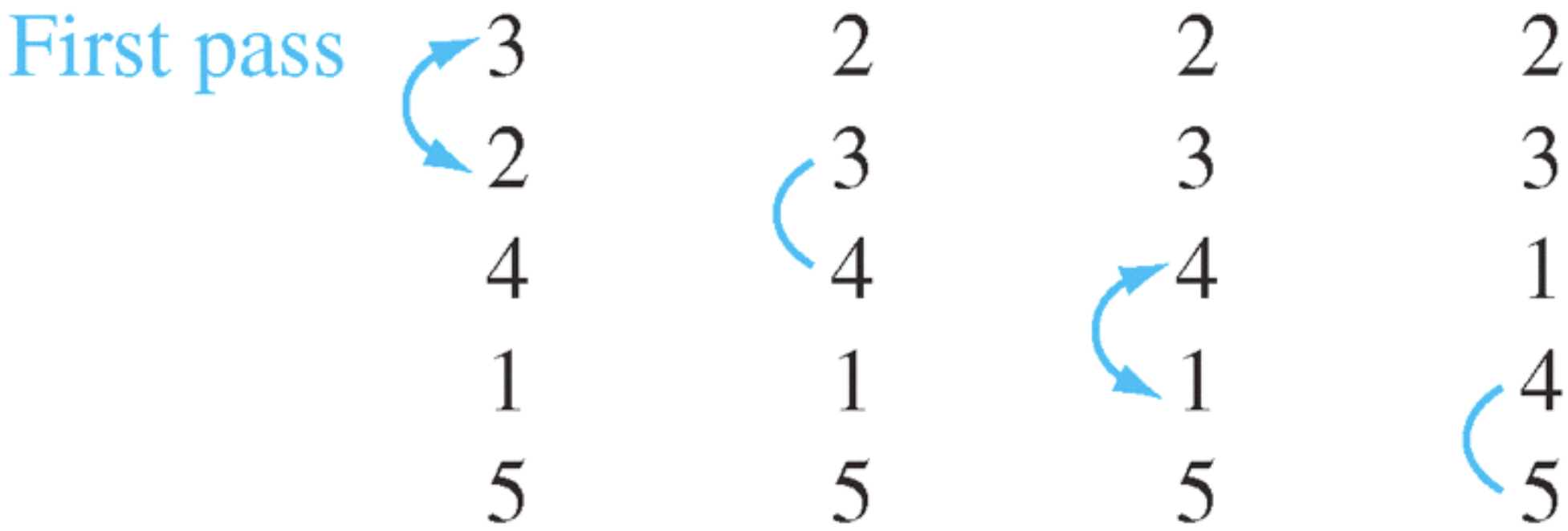
procedure bubble_sort ($a_1, a_2, \ldots, a_n$: real numbers, $n \geq 2$)

```
for i := 1 to n-1
      for j := 1 to n-i
            if a_j > a_{j+1} then interchange a_j and a_{j+1}
return max {a_1, ... a_n is in increasing order}
```

bubble sort

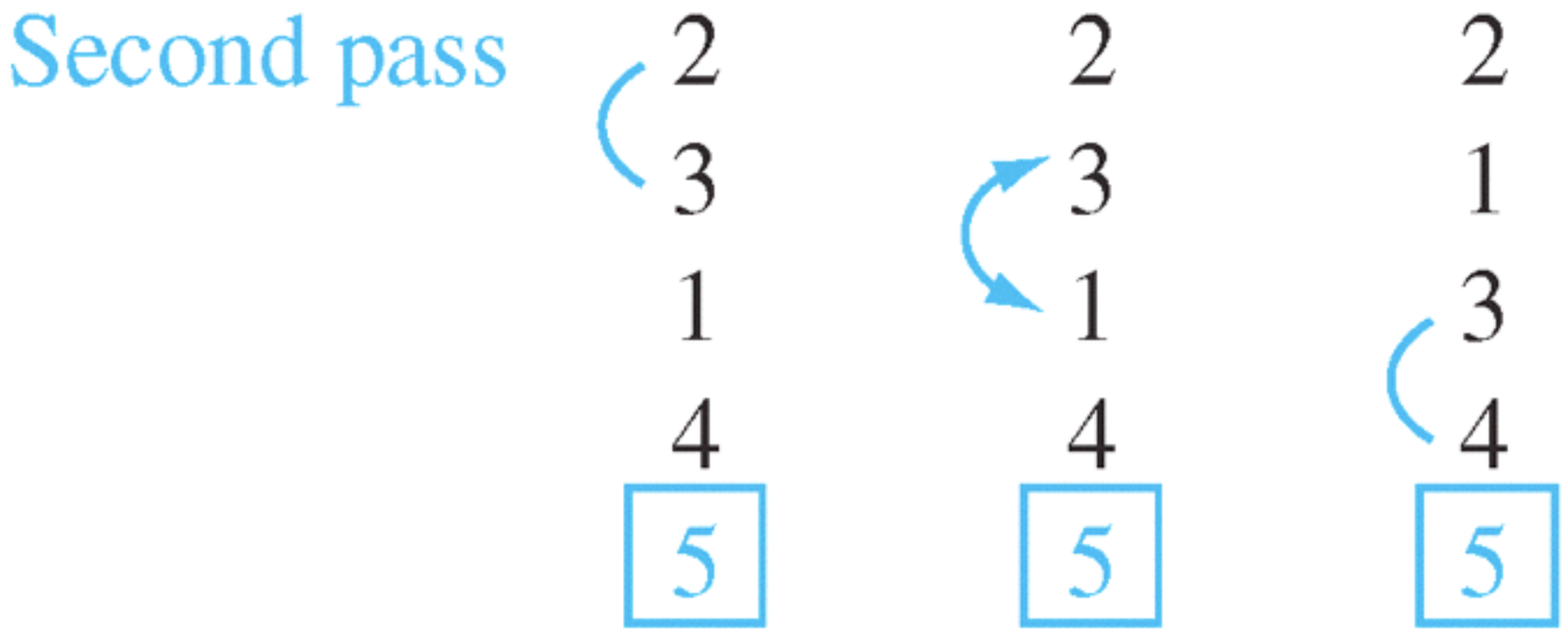
sort a list of numbers 3, 2, 4, 1, 5

if $a_j > a_{j+1}$ then interchange a_j and a_{j+1}



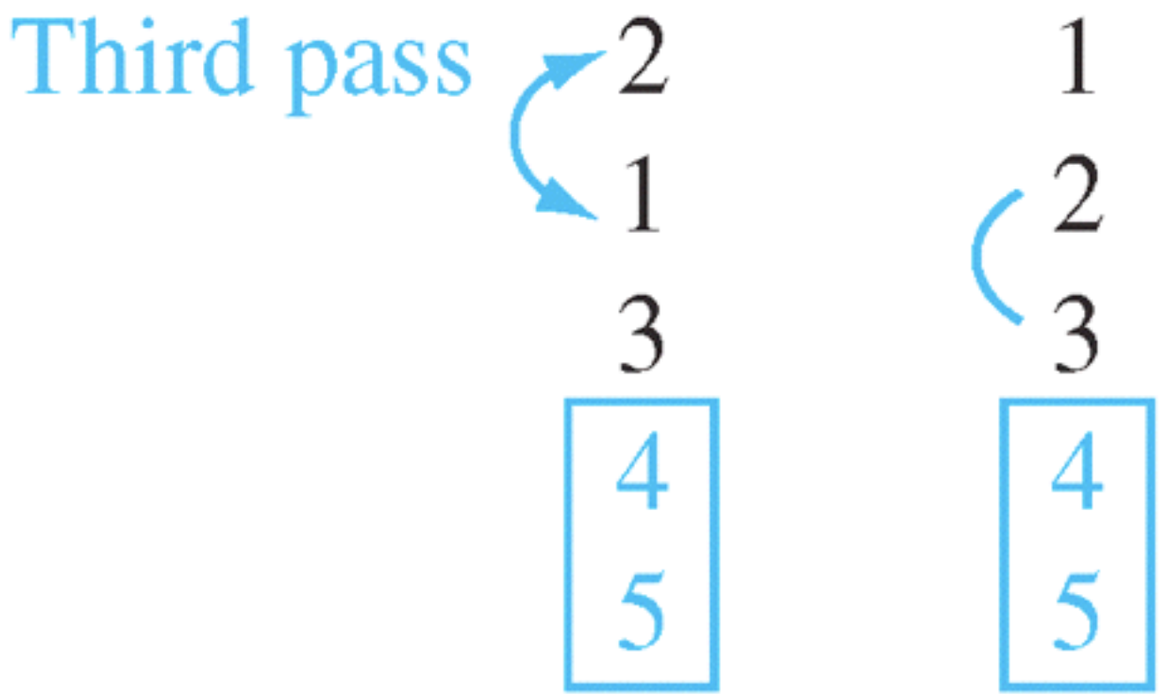
bubble sort

if $a_j > a_{j+1}$ then interchange a_j and a_{j+1}



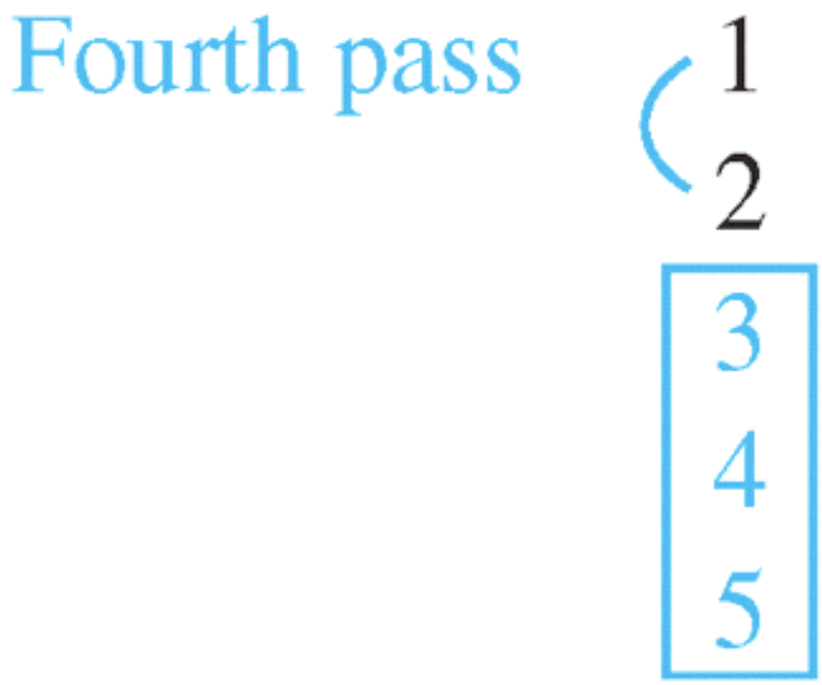
bubble sort

if $a_j > a_{j+1}$ then interchange a_j and a_{j+1}



bubble sort

if $a_j > a_{j+1}$ then interchange a_j and a_{j+1}



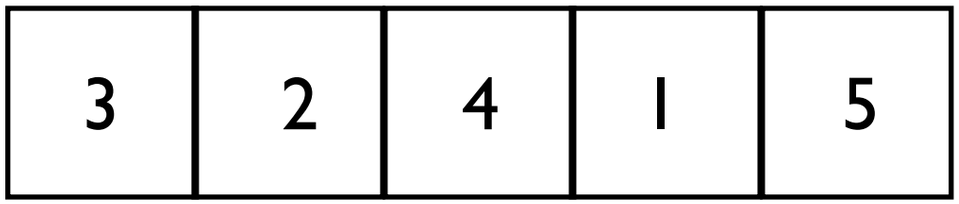
insertion sort

procedure insertion_sort ($a_1, a_2, \ldots, a_n$: real numbers, $n \geq 2$)

```
for i := 2 to n
        m = a_i
        j = i - 1
        while (j > 0 and a_j > m )
                a_{j+1} = a_j
                j := j - 1
        a_{j+1} := m
return {a_1, ... a_n is in increasing order}
```

Note: in our textbook, the algorithm is a little bit longer.

insertion sort



greedy algorithm

greedy algorithm makes the best choice at each step (locally optimal solution) according to a specified criterion with the hope of finding a global optimum

make 67 cents change with quaters(25 cents), dimes(10 cents), nickels(5 cents), and pennies(1 cent), and using the least total number of coins

2 quaters => 50 cents
1 dime => 10 cents
1 nickel => 5 cents
2 cents => 2 cents

greedy algorithm

procedure change ($a_1, a_2, \ldots, a_r$: different types of coins, $a_1 > a_2 > \ldots > a_r$
n: positive integer)

for i := 1 to r
    d_i := 0 { d_i counts the coins a_i used }
    while n ≥ a_i
        d_i := d_i + 1   { add a coin a_i}
        n := n - a_i

{d_i is the number of the coin a_i for i = 1, 2, ... r}

greedy algorithm

lemma:

If n is a positive integer, then n cents in change using quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent) using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies, and cannot have two dimes and a nickel. The amount of change in dimes, nickels, and pennies cannot exceed 24 cents

by using proof by contradiction....

- if three dimes, replace them with a quarter and a nickel
- if two nickels, replace them with one dime
- if five pennies, replace them with a nickel
- if two dimes and a nickel, replace them with a quarter

The Halting problem

- The halting problem was the first mathematical function proven to have no algorithm that computes it! We say, it is uncomputable.

- The function is Halts(P, I) :≡ "Program P, given input I, eventually terminates."

- Theorem: Halts is uncomputable!

  I.e., There does not exist any algorithm A that computes Halts correctly for all possible inputs.

  Its proof is thus a non-existence proof.

- Corollary: General impossibility of predictive analysis of arbitrary computer programs.

The Halting problem

Proof

- Given any arbitrary program H(P, I),
- Consider algorithm Breaker is defined as:

```
procedure Breaker (P: a program)
    halts := H(P, P)
    if halts then while T begin end
```

- Note that Breaker (Breaker) halts iff H(Breaker , Breaker) = F
- So H does not compute the function Breaker !