SRI AKILANDESWARI WOMEN'S COLLEGE, WANDIWASH

DESIGN AND ANALYSIS OF ALGORITHM

Class : III UG Computer Science

Mrs. R.PADMASHREE

Assistant Professor Department of Computer Science

SWAMY ABEDHANADHA EDUCATIONAL TRUST, WANDIWASH

WHAT IS AN ALGORITHM?

- An algorithm is “a finite set of precise instructions for performing a computation or for solving a problem”
  - A program is one type of algorithm
    - All programs are algorithms
    - Not all algorithms are programs!
  - Directions to somebody’s house is an algorithm
  - A recipe for cooking a cake is an algorithm
  - The steps to compute the cosine of 90° is an algorithm

Some algorithms are harder than others

- Some algorithms are easy
  - Finding the largest (or smallest) value in a list
  - Finding a specific value in a list
- Some algorithms are a bit harder
  - Sorting a list
- Some algorithms are very hard
  - Finding the shortest path between Miami and Seattle
- Some algorithms are essentially impossible
  - Factoring large composite numbers

ALGORITHM 1: MAXIMUM ELEMENT

- Given a list, how do we find the maximum element in the list?

- To express the algorithm, we'll use pseudocode
  - Pseudocode is kinda like a programming language, but not really

ALGORITHM 1: MAXIMUM ELEMENT

- Algorithm for finding the maximum element in a list:

procedure max (a_1, a_2, …, a_n: integers)
max := a_1
for i := 2 **to** n
  if max < a_i **then** *max* := a_i

{*max* is the largest element}

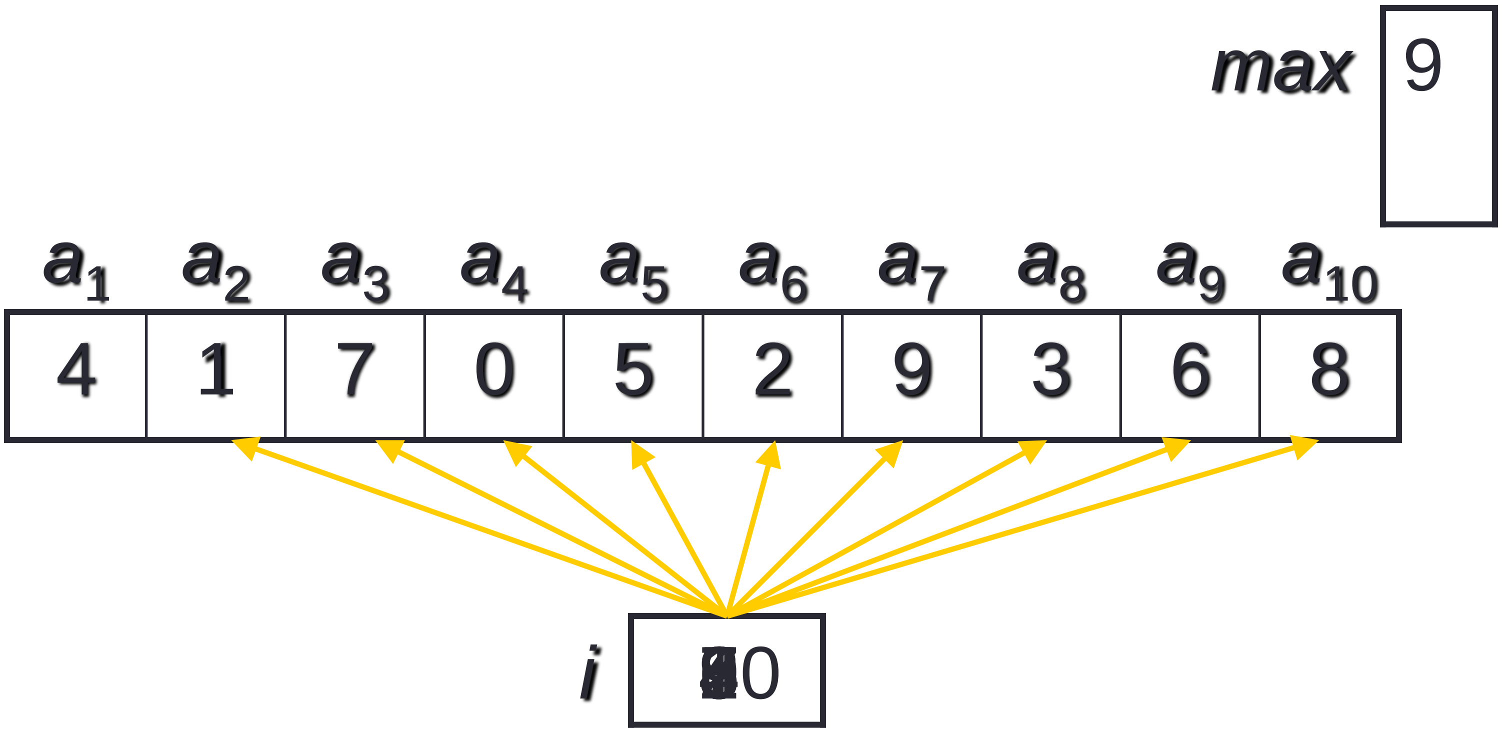
ALGORITHM 1: MAXIMUM ELEMENT

procedure max (a_1, a_2, …, a_n: integers)

$max := a_1$

for i := 2 **to** n

if max < a_i **then** $max := a_i$



PROPERTIES OF ALGORITHMS

- Algorithms generally share a set of properties:
  - Input: what the algorithm takes in as input
  - Output: what the algorithm produces as output
  - Definiteness: the steps are defined precisely
  - Correctness: should produce the correct output
  - Finiteness: the steps required should be finite
  - Effectiveness: each step must be able to be performed in a finite amount of time
  - Generality: the algorithm *should* be applicable to all problems of a similar form

ALGORITHM 2: LINEAR SEARCH

- Given a list, find a specific element in the list
  - List does NOT have to be sorted!

procedure linear_search (x: integer; $a_1, a_2, \ldots, a_n$: integers)

$i := 1$

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

$i := i + 1$

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

else *location* := 0

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

ALGORITHM 2: LINEAR SEARCH, TAKE 2

procedure linear_search (x: integer; $a_1, a_2, \ldots, a_n$: integers)

$i := 1$

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

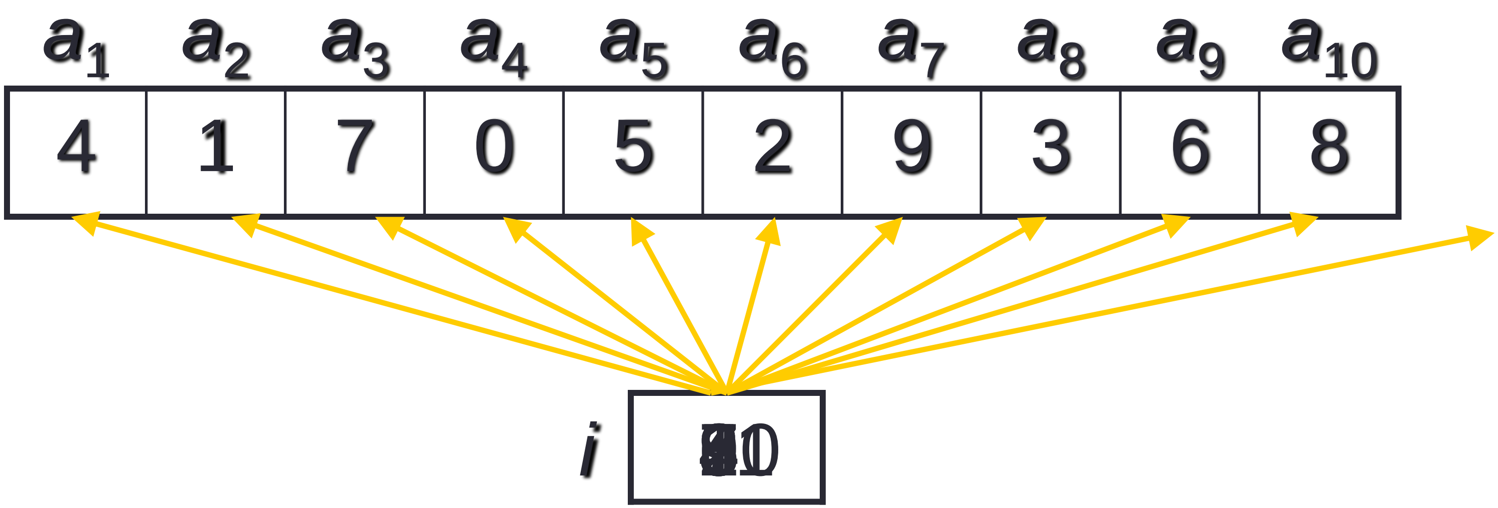
    $i := i + 1$

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

else *location* := 0

x: 11

location: 0



ALGORITHM 3: BINARY SEARCH

procedure binary_search (x: integer; $a_1, a_2, \ldots, a_n$: increasing integers)

$i := 1$ { i is left endpoint of search interval }

$j := n$ { j is right endpoint of search interval }

while $i < j$

begin

m := $\lfloor (i+j)/2 \rfloor$ { m is the point in the middle }

if x > a_m **then** $i := m+1$

else $j := m$

end

if $x = a_i$ **then** *location* := i

else *location* := 0

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

Algorithm 3: Binary search, take 1

procedure binary_search (x: integer; a_1, a_2, …, a_n: increasing integers)

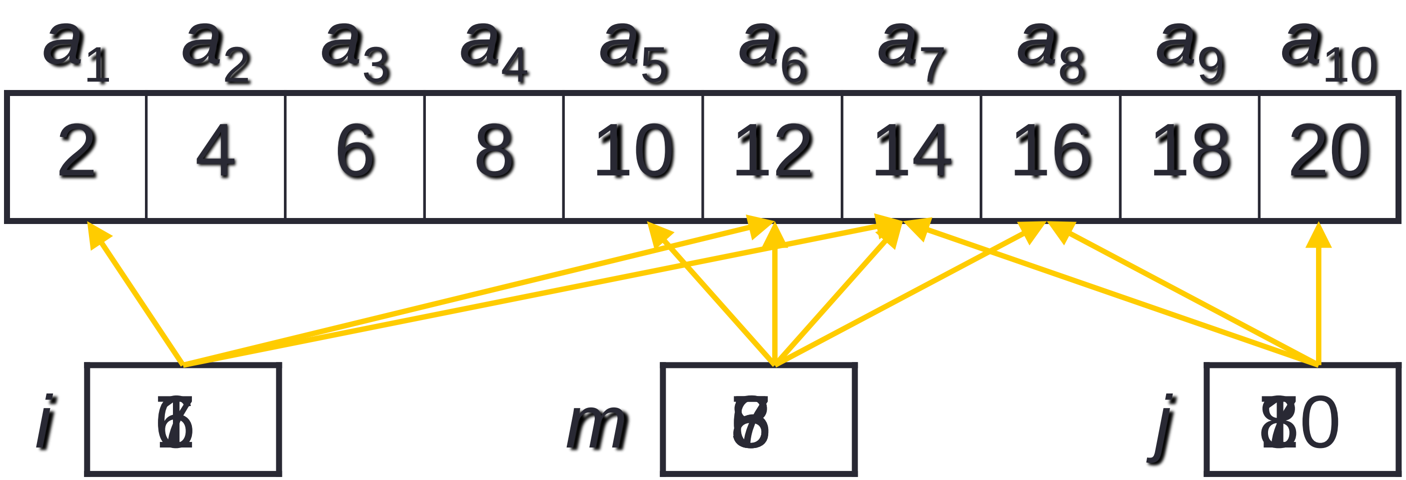
$i := 1$
$j := n$

while $i < j$
begin
  $m := \lfloor (i+j)/2 \rfloor$
  if $x > a_m$ **then** $i := m+1$
  else $j := m$
end

if $x = a_i$ **then** $location := i$
else $location := 0$

x: 14

$location$: 7



ALGORITHM 3: BINARY SEARCH, TAKE 2

procedure binary_search (x: integer; $a_1, a_2, \ldots, a_n$: increasing integers)

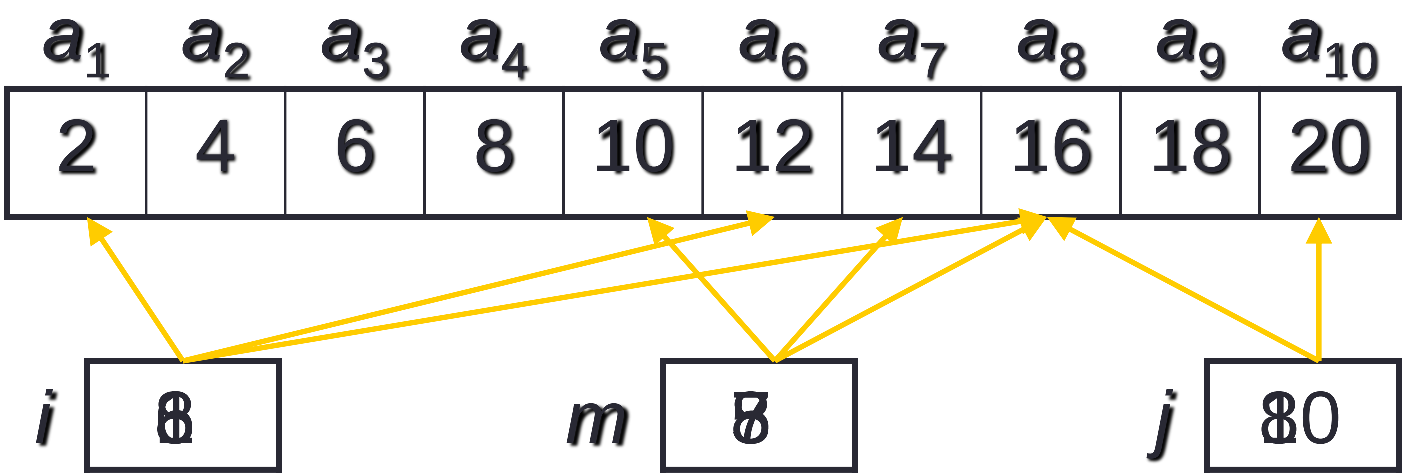
i := 1
j := n

while $i < j$
begin
  m := $\lfloor (i+j)/2 \rfloor$
  if x > a_m then i := m+1
  else j := m
end

if $x = a_i$ then *location* := i
else *location* := 0

x: 15

location: 0



BINARY SEARCH RUNNING TIME

- How long does this take (worst case)?

- If the list has 8 elements
  - It takes 3 steps
- If the list has 16 elements
  - It takes 4 steps
- If the list has 64 elements
  - It takes 6 steps

- If the list has *n* elements
  - It takes $\log_2 n$ steps

SORTING ALGORITHMS

- Given a list, put it into some order
  - Numerical, lexicographic, etc.

- We will see two types
  - Bubble sort
  - Insertion sort

ALGORITHM 4: BUBBLE SORT

- One of the most simple sorting algorithms
  - Also one of the least efficient
- It takes successive elements and “bubbles” them up the list

procedure bubble_sort ($a_1, a_2, \ldots, a_n$)

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

{ $a_1, \ldots, a_n$ are in increasing order }

BUBBLE SORT RUNNING TIME

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

- Outer for loop does n-1 iterations
- Inner for loop does
  - n-1 iterations the first time
  - n-2 iterations the second time
  - …
  - 1 iteration the last time
- Total: $(n\text{-}1) + (n\text{-}2) + (n\text{-}3) + \ldots + 2 + 1 = (n^2\text{-}n)/2$
  - We can say that's "about" n^2 time

INSERTION SORT RUNNING TIME

for j := 2 **to** n **begin**
    i := 1
    while $a_j > a_i$
        $i := i + 1$
    $m := a_j$
    for k := 0 **to** j-i-1
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
end { $a_1, a_2, \ldots, a_n$ are sorted }

- Outer for loop runs n-1 times
- In the inner for loop:
  - Worst case is when the while keeps i at 1, and the for loop runs lots of times
- Total is $1 + 2 + \ldots + n\text{-}2 = (n\text{-}1)(n\text{-}2)/2$
  - We can say that's "about" n^2 time

COMPARISON OF RUNNING TIMES

- Searches
  - Linear: n steps
  - Binary: $\log_2 n$ steps
  - Binary search is about as fast as you can get
- Sorts
  - Bubble: n^2 steps
  - Insertion: n^2 steps
  - There are other, more efficient, sorting techniques
    - In principle, the fastest are heap sort, quick sort, and merge sort
    - These each take take $n * \log_2 n$ steps


Thank you