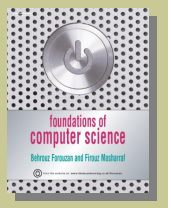


Algorithms

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University

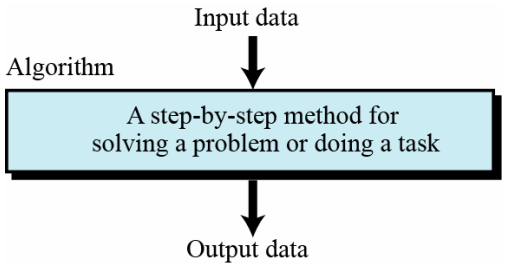
Outline

- Concept
- Three Constructs
- Algorithm Representation
- Basic Algorithms
- Recursion
- Summary

Concept

- An informal definition of an algorithm:

"a step-by-step method for solving a problem or doing a task"



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

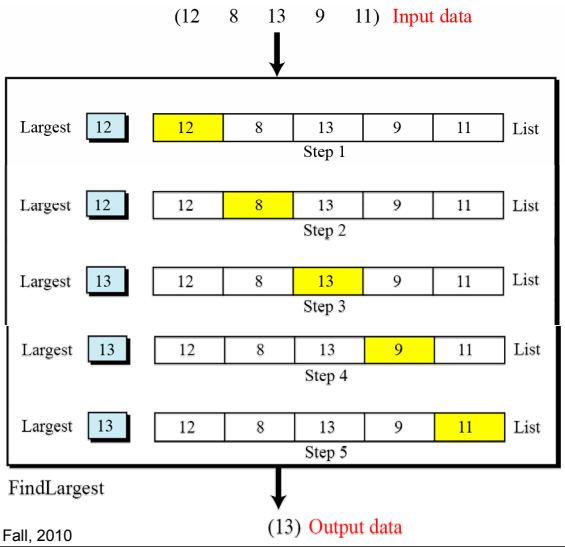
Concept

- Example

We want to develop an algorithm for finding the largest integer among a list of positive integers.

- The algorithm should be general and not depend on the number of integers.
- We first use a small number of integers (for example, five), and then extend the solution to any number of integers

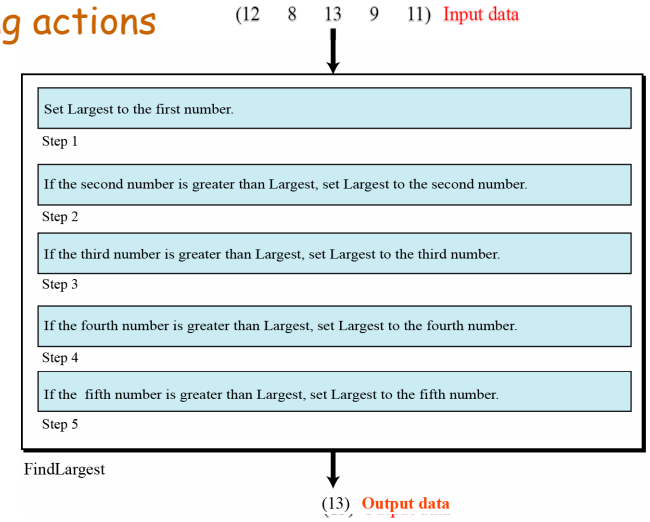
Concept



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Concept

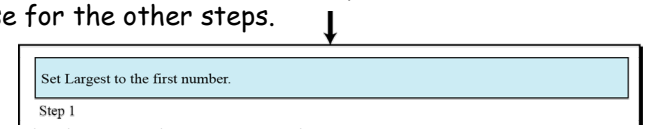
- Defining actions



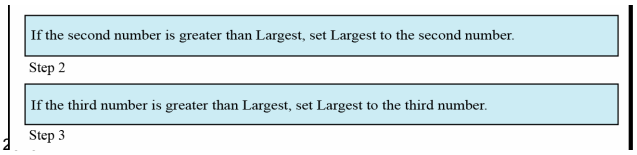
B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Concept

- Refinement
  - This algorithm needs refinement to be acceptable to the programming community.
  - Two problems.
    - First, the action in the first step is different than those for the other steps.



    - Second, the wording is not the same in steps 2 to 5.



Concept

- Refinement
  - Solution to problem 2:

    Changing the wording in steps 2 to 5 to "If the current integer is greater than Largest, set Largest to the current integer."

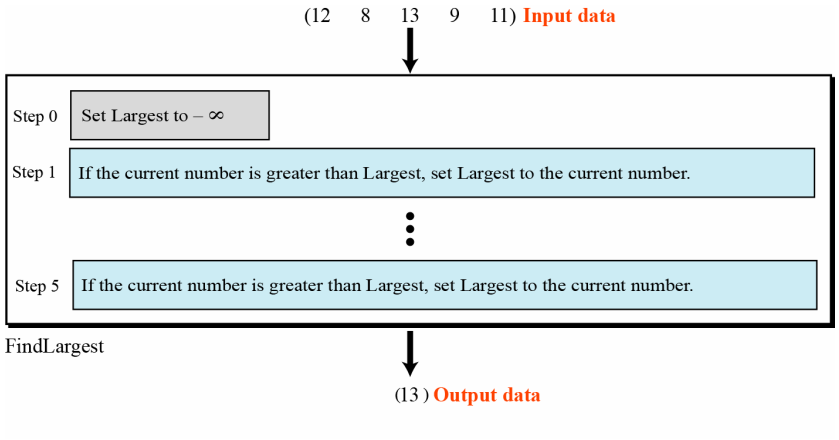
  - Solution to problem 1:

    If we initialize Largest to $-\infty$ (minus infinity), then the first step can be the same as the other steps.

In C, see <`limits.h`> for `INT_MIN`.
In C++, see <`limits`> for `std::numeric_limits<int>::min()`

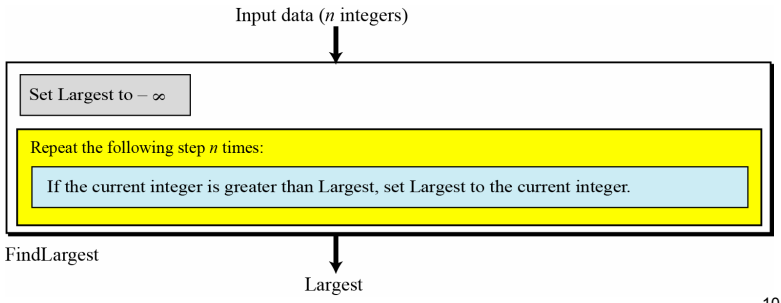
Concept

- The algorithm after refinement



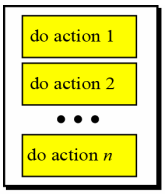
Concept

- Generalization
  - Is it possible to generalize the algorithm?
    - We want to find the largest of n positive integers, where n can be 1000, 000,000, or more.
    - We can tell the computer to repeat the steps n times.

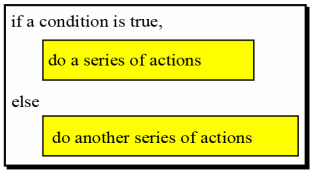


Three Constructs

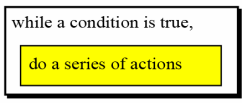
- Computer scientists have defined three constructs for a structured program or algorithm:
  - sequence
  - decision (selection)
  - repetition.



a. Sequence



b. Decision



c. Repetition

B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Three Constructs

- Sequence
  - An algorithm, and eventually a program, is a sequence of instructions, which can be a simple instruction or either of the other two constructs.
- Decision
  - Sometimes we need to test a condition. If the result of testing is true, we follow a sequence of instructions: if it is false, we follow a different sequence.
- Repetition
  - In some problems, the same sequence of instructions must be repeated.

Algorithm Representation

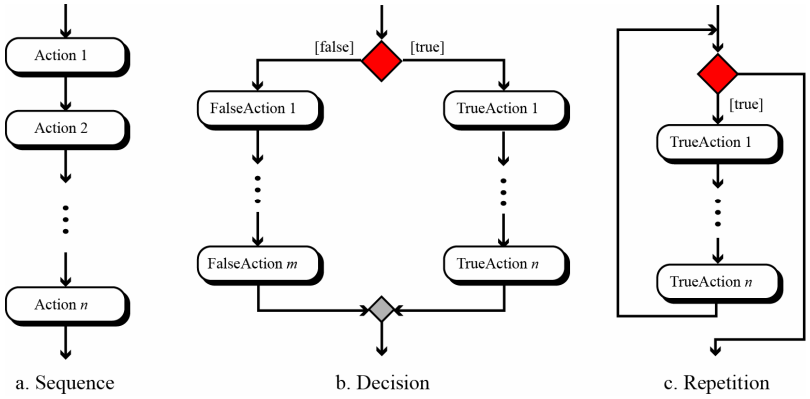
- So far, we have used figures to convey the concept of an algorithm.
- During the last few decades, tools have been designed for this purpose. Two of these tools, UML and pseudocode, are presented here.

Algorithm Representation

- Unified Modeling Language (UML)
  - UML is a pictorial representation of an algorithm.
  - It hides all the details of an algorithm in an attempt to give the "big picture" and to show how the algorithm flows from beginning to end.

Algorithm Representation

- UML for three constructs



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Algorithm Representation

- Pseudocode
  - Pseudocode is an English-language-like representation of an algorithm.
  - There is no standard for pseudocode—some people use a lot of detail, others use less.

    Some use a code that is close to English, while others use a syntax like the Pascal programming language.

Algorithm Representation

- Pseudocode for three constructs

```
action 1
action 2
. . .
action n
```

a. Sequence

```
if (condition)
{
   trueAction(s)
}
 else
{
   falseAction(s)
}
```

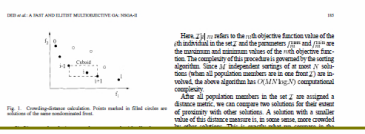
b. Decision

```
while (condition)
{
      Action(s)
}
```

c. Repetition

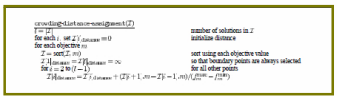
B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Algorithm Representation



crowding-distance-assignment($\mathcal{I}$)

$l = \lvert\mathcal{I}\rvert$	number of solutions in $\mathcal{I}$
for each i, set $\mathcal{I}[i]_{\text{distance}} = 0$	initialize distance
for each objective m	
$\mathcal{I} = \text{sort}(\mathcal{I}, m)$	sort using each objective value
$\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$	so that boundary points are always selected
for $i = 2$ to $(l-1)$	for all other points
$\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$	



Deb et al., 2002, NSGA-II

Algorithm Representation

Example 8.1

Write an algorithm in pseudocode that finds the sum of two integers.

Algorithm 8.1 Calculating the sum of two integers

```
Algorithm: SumOfTwo (first, second)
Purpose: Find the sum of two integers
Pre: Given: two integers (first and second)
Post: None
Return: The sum value
{
    sum ← first + second
    return sum
}
```


Algorithm Representation

Example 8.2

Write an algorithm to change a numeric grade to a pass/no pass grade.

Algorithm 8.2 Assigning pass / no pass grade

```
Algorithm: Pass/NoPass (score)
Purpose: Creates a pass/no pass grade given the score
Pre: Given: the score to be changed to grade
Post: None
Return: The grade
{
    if (score ≥ 70)     grade ← "pass"
    else                grade ← "nopass"
    return grade
}
```

Algorithm Representation

Example 8.3

Write an algorithm to change a numeric grade (integer) to a letter grade.

Note that this is only pseudo code.

In C/C++

```
int score = 20;
if (70 <= score <= 90)
   printf("Oh no");
```

Algorithm: **LetterGrade** (score)

Purpose: Find the letter grade corresponding to the given score

Pre: Given: a numeric score

Post: None

Return: A letter grade

```
{
    if (100 ≥ score ≥ 90)   grade ← 'A'
    if (80 ≥ score ≥ 89)    grade ← 'B'
    if (70 ≥ score ≥ 79)    grade ← 'C'
    if (60 ≥ score ≥ 69)    grade ← 'D'
    if (0 ≥ score ≥ 59)     grade ← 'F'
    return grade
}
```


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Algorithm Representation

Example 8.4

Write an algorithm to find the largest of a set of integers.

Algorithm: **FindLargest** (list)

Purpose: Find the largest integer among a set of integers

Pre: Given: the set of integers

Post: None

Return: The largest integer

```
{
    largest ← – ∞
    while (more integers to check)
    {
        current ← next integer
        if (current > largest)      largest ← current
    }
    return largest
}
```


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Algorithm Representation

Example 8.5

Write an algorithm to find the largest of the first 1000 integers in a set of integers.

```
Algorithm: FindLargest2 (list)
Purpose: Find and return the largest integer among the first 1000 integers
Pre: Given: the set of integers with more than 1000 integers
Post: None
Return: The largest integer
{
    largest ← – ∞
    counter ← 1
    while (counter ≤ 1000)
    {
        current ← next integer
        if (current > largest)        largest ← current
        counter ← counter + 1
    }
    return largest
}
```


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Algorithm Representation

- A more formal definition of an algorithm:

"An ordered set of unambiguous steps that produces a result and terminates in a finite time."

Algorithm Representation

- Ordered set
  - An algorithm must be a well-defined, ordered set of instructions.
- Unambiguous steps
  - Each step in an algorithm must be clearly and unambiguously defined.
  - If one step is to add two integers, we must define both "integers" as well as the "add" operation: we cannot for example use the same symbol to mean addition in one place and multiplication somewhere else

Algorithm Representation

- Produce a result
  - An algorithm must produce a result, otherwise it is useless.
  - The result can be data returned to the calling algorithm, or some other effect (for example, printing).
- Terminate in a finite time
  - An algorithm must terminate (halt). If it does not (that is, it has an infinite loop), we have not created an algorithm.

Basic Algorithms

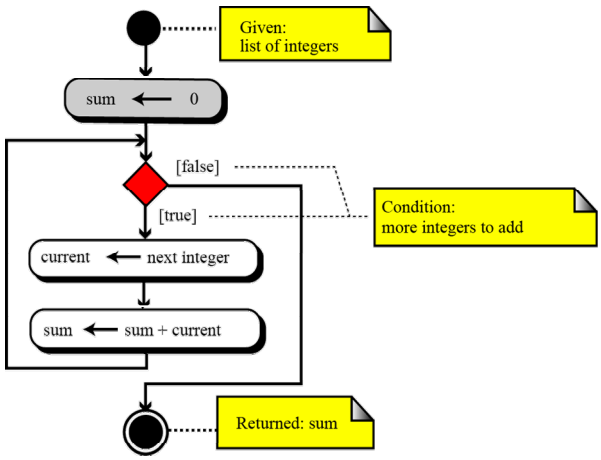
- Several algorithms are used in computer science so prevalently that they are considered "basic".
- We discuss the most common here. This discussion is very general: implementation depends on the language.

Basic Algorithms

- Summation
  - We can add two or three integers very easily, but how can we add many integers? The solution is simple: we use the add operator in a loop.
  - A summation algorithm has three logical parts:
    - Initialization of the sum at the beginning.
    - The loop, which in each iteration adds a new integer to the sum.
    - Return of the result after exiting from the loop.

Basic Algorithms

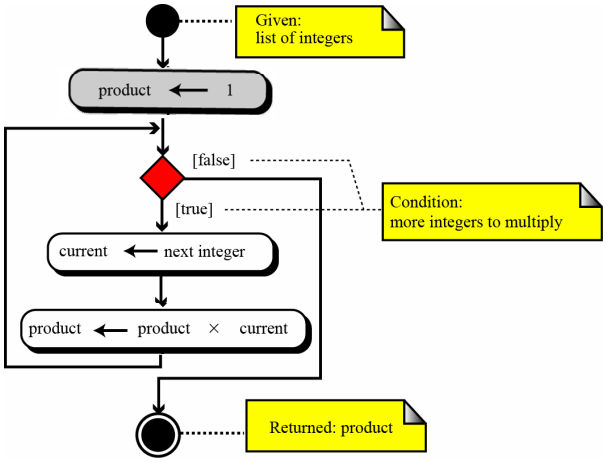
- Summation



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Algorithms

- Product



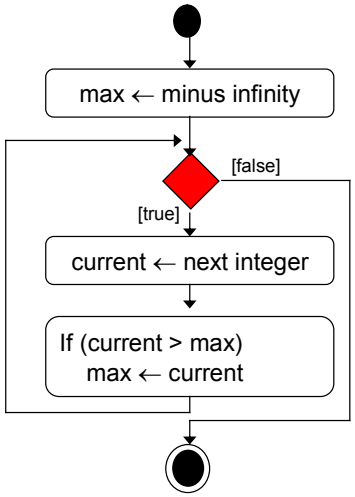
B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Algorithms

- Largest & smallest
  - Finding the largest
    - The idea is to write a decision construct to find the larger of two integers.
    - If we put this construct in a loop, we can find the largest of a list of integers.
  - Finding the smallest is similar, with two minor differences.
    - First, we use a decision construct to find the smaller of two integers.
    - Second, we initialize with a very large integer instead of a very small one.

Basic Algorithms

- Find the largest



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Sorting Algorithms

- Sorting
  - One of the most common applications in computer science is sorting, which is the process by which data is arranged according to its values.
  - If the data was not ordered, it would take hours and hours to find a single piece of information.
    - Imagine the difficulty of finding someone's telephone number in a telephone book that is not ordered.

Basic Sorting Algorithms

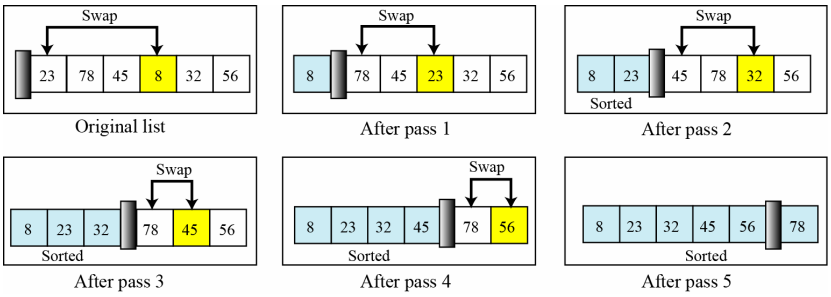
- In this section, we introduce three sorting algorithms:
  - selection sort
  - bubble sort
  - insertion sort
- These three sorting algorithms are the foundation of faster sorting algorithms used in computer science today.

Basic Sorting Algorithms

- Selection sort
  - In a selection sort, the list to be sorted is divided into two sublists—sorted and unsorted—which are separated by an imaginary wall.
  - We find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted sublist.
  - After each selection and swap, the imaginary wall between the two sublists moves one element ahead.

Basic Sorting Algorithms

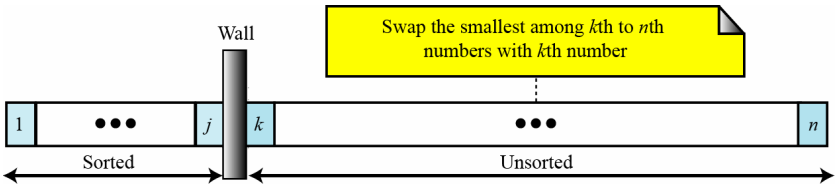
- Selection sort: an example



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Sorting Algorithms

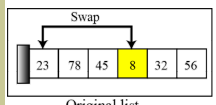
- Selection sort



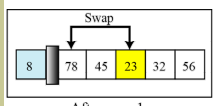
B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Sorting Algorithms

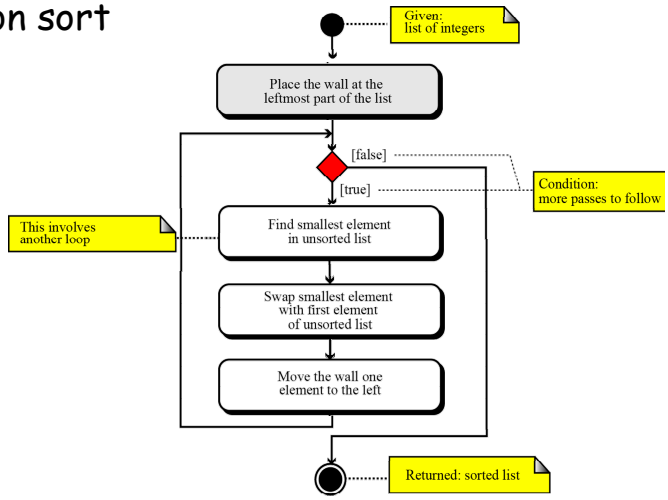
- Selection sort



Original list



After pass 1

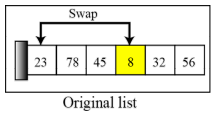


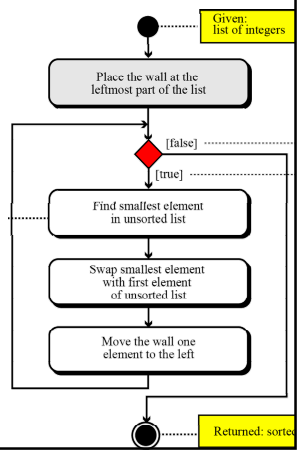
B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Sorting Algorithms

- Selection sort

```
const int DATA_SIZE = 6;
int data[DATA_SIZE] = {23, 78, 45, 8, 32, 56};

for (int wall=0; wall<DATA_SIZE-1; wall+=1)
{// only DATA_SIZE-1 (5) passes

        // Find the position of the minimal element
        int min_pos = wall;
        for (int next=min_pos+1; next<DATA_SIZE; next+=1)
        {
                if (data[next] < data[min_pos])
                {
                        min_pos = next;
                }
        }

        // Swap the first element and the minimal element
        int tmp = data[min_pos];
        data[min_pos] = data[wall];
        data[wall] = tmp;
}
```





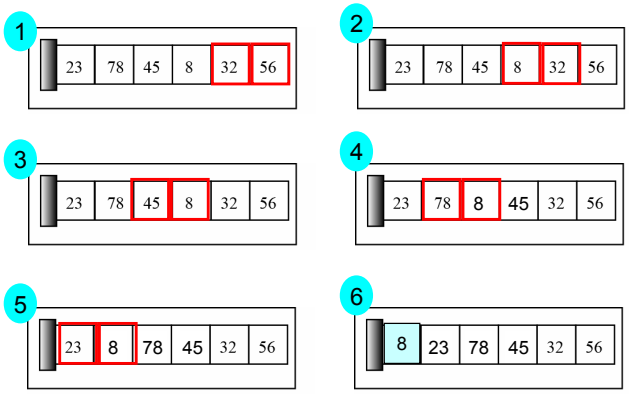
Basic Sorting Algorithms

- Bubble sort
  - The list to be sorted is also divided into two sublists—sorted and unsorted.
  - The smallest element is bubbled up from the unsorted sublist and moved to the sorted sublist.
  - After the smallest element has been moved to the sorted list, the wall moves one element ahead.

Basic Sorting Algorithms

- Bubble sort: an example

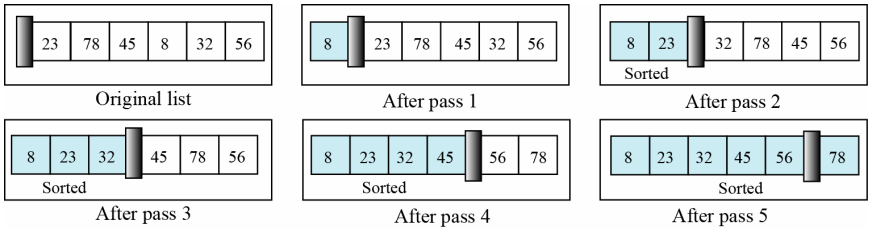
  "Floating"

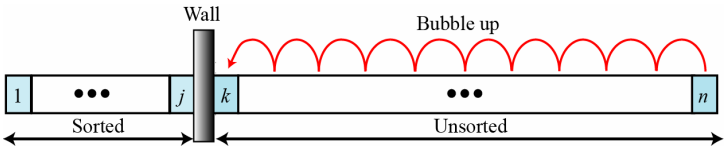


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Sorting Algorithms

- Bubble sort: an example



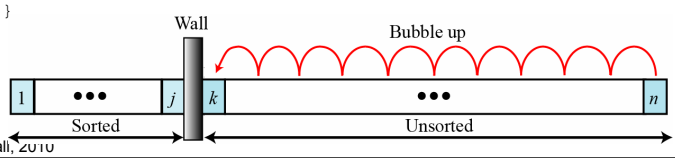


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Sorting Algorithms

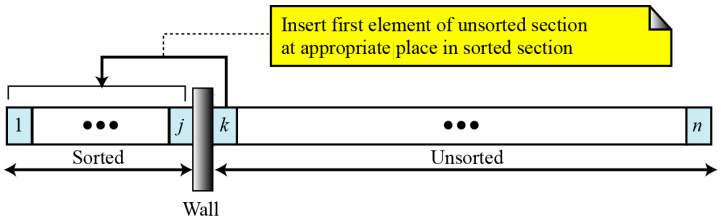
- Bubble sort

```
const int DATA_SIZE = 6;
int data[DATA_SIZE] = {23, 78, 45, 8, 32, 56};

for (int wall=0; wall<DATA_SIZE-1; wall+=1)
{// only DATA_SIZE-1 (5) passes

        // Floating the minimal element to the first position within the wall
        for (int current = DATA_SIZE-1; current > wall; current-=1)
        {
                if (data[current-1] > data[current])
                {
                        // data[current] <-> data[current-1]
                        int tmp = data[current-1];
                        data[current-1] = data[current];
                        data[current] = tmp;
                }
        }
}
```



Basic Sorting Algorithms

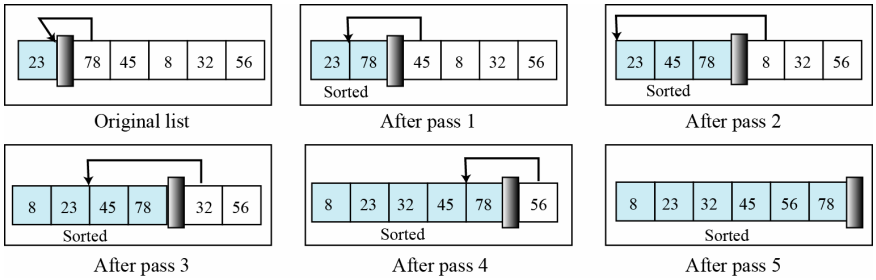
- Insertion sort
  - It is one of the most common sorting techniques, and it is often used by card players.
  - Each card a player picks up is inserted into the proper place in their hand of cards to maintain a particular sequence.



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Sorting Algorithms

- Insertion sort: an example

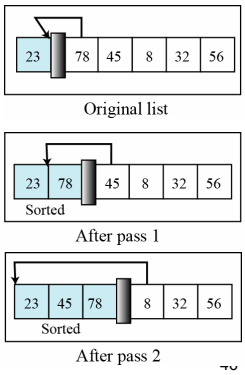


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Sorting Algorithms

- Insertion sort

```
for (int wall=1; wall<DATA_SIZE; wall+=1)
{// only DATA_SIZE-1 (5) passes

        // Find the position to insert
        int pos = 0;
        while (pos<wall && data[pos]<data[wall])
        {
                pos+=1;
        }

        // Shift the values after the found position
        int current = data[wall];
        for (int b=wall; b>pos; b-=1)
        {
                data[b] = data[b-1];
        }

        // Put the value in the position
        data[pos] = current;

}
```



Basic Sorting Algorithms

- Time complexity
  - Estimate how the computation time of your algorithm increases as the amount of data increases

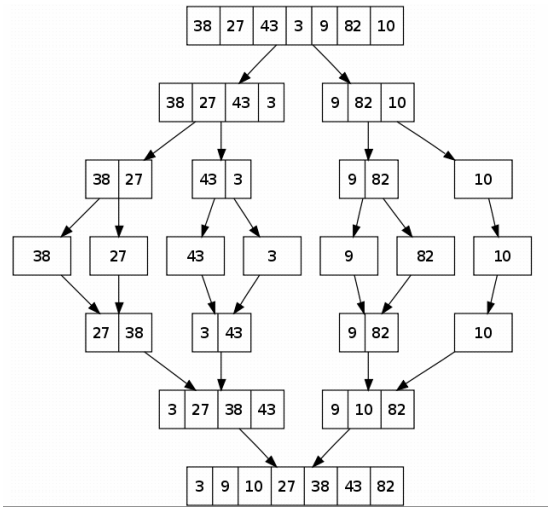
```
8   const int DATA_SIZE = 6;
9   int data[DATA_SIZE] = {23, 78, 45, 8, 32, 56};
10
11  for (int wall=0; wall<DATA_SIZE-1; wall+=1)
12  {// only DATA_SIZE-1 (5) passes
13
14      // Floating the minimal element to the first position within the wall
15      for (int current = DATA_SIZE-1; current > wall; current-=1)
16      {
17          if (data[current-1] > data[current])
18          {
19              // data[current] <-> data[current-1]
20              int tmp = data[current-1];
21              data[current-1] = data[current];
22              data[curr  Iterator current
23          }
24      }
25  }
```

$O(n^2) = (n^2 - n)/2 = (1 + (n-1))\times(n-1)/2 =$

wall	# inner iterations
0	5
1	4
2	3
3	2
4	1

Basic Sorting Algorithms

- Merge sort



Time

$O(n\log n) = (\log_2 7)\cdot n = 3n =$ $\left\{ \begin{array}{l} n \\ n \\ n \end{array} \right.$

http://en.wikipedia.org/wiki/File:Merge_sort_algorithm_diagram.svg

Basic Searching Algorithms

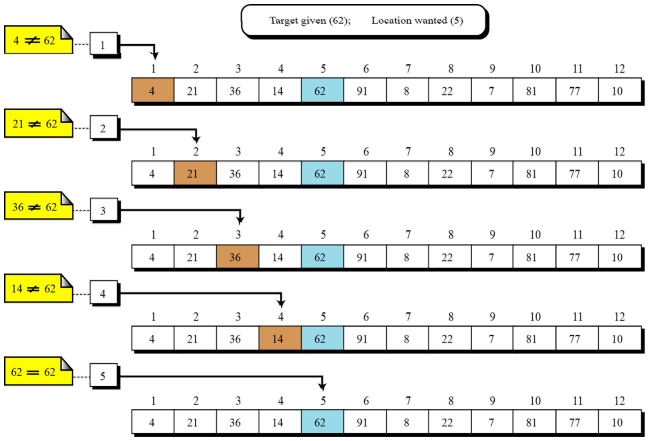
- Another common algorithm in computer science is searching, which is the process of finding the location of a target among a list of objects.
- In the case of a list, searching means that given a value, we want to find the location of the first element in the list that contains that value.
- There are two basic searches for lists: sequential search and binary search.
  - Sequential search can be used to locate an item in any list, whereas binary search requires the list first to be sorted.

Basic Searching Algorithms

- Sequential search
  - Generally, we use this technique only for small lists, or lists that are not searched often.
  - We start searching for the target from the beginning of the list. We continue until we either find the target or reach the end of the list.

Basic Searching Algorithms

- Sequential search: an example



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Searching Algorithms

```
int main()
{
    const int DATA_SIZE = 7;
    int data[DATA_SIZE] = {100, 52, 44, 72, 63, 17, 28};
    int key = 63;

    int found = 0;
    for (found=0; found<DATA_SIZE; found+=1)
    {
        if (data[found] == key)
        {
            break;
        }
    }

    if (found < DATA_SIZE)
    {
        cout << "We found " << key << " in data[" << found << "]." << endl;
    }
    else
    {
        cout << key << " is not found." << endl;
    }
```

Basic Searching Algorithms

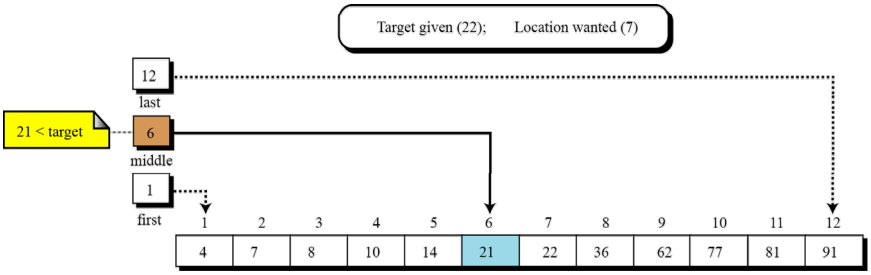
- Binary search
  - The sequential search algorithm is very slow. If we have a list of a million elements, we must do a million comparisons in the worst case. If the list is not sorted, this is the only solution.
  - If the list is sorted, however, we can use a more efficient algorithm called binary search.

Basic Searching Algorithms

- Binary search
  - A binary search starts by testing the data in the element at the middle of the list.
  - This determines whether the target is in the first half or the second half of the list. We eliminate half the list from further consideration.

Basic Searching Algorithms

- Binary search: An example



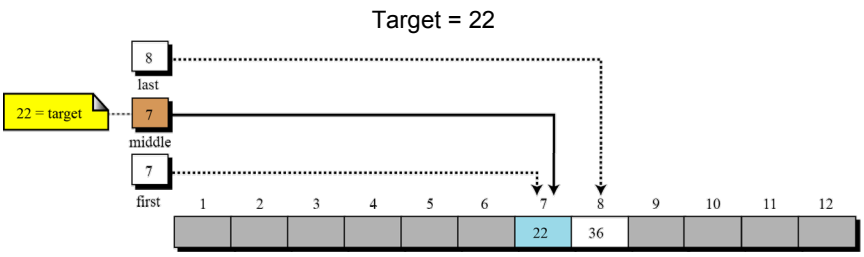
Basic Searching Algorithms

- Binary search: An example

Target = 22

Basic Searching Algorithms

- Binary search: An example



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Basic Searching Algorithms

```
const int DATA_SIZE = 12;
int data[DATA_SIZE] = {4, 7, 8, 10, 14, 21, 22, 36, 62, 77, 81, 91};
int key = 22;

int first = 0, last = DATA_SIZE-1, found = DATA_SIZE;
while (first<=last)
{
    int middle = (first+last)/2;
    if (data[middle] == key)
    {
        found = middle;
        break;
    }
    else if (data[middle] < key)
    {
        first = middle+1;
    }
    else // data[middle] > key
    {
        last = middle-1;
    }
}
```

Basic Searching Algorithms

```
if (found<DATA_SIZE)
{
    cout << "We found " << key << " in data[" << found << "]." << endl;
}
else
{
    cout << key << " is not found." << endl;
}
```


Basic Searching Algorithms

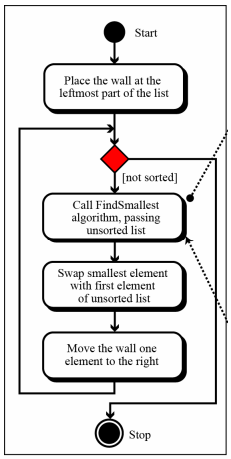
- Time complexity
  - Sequential search: $O(n)$
  - Binary search: $O(\log n)$
    - The premise is that the data are sorted. (Remember that you may pay $O(n\log n)$-time to sort the data.)

Subalgorithms

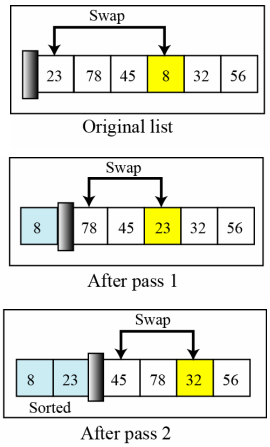
- The three programming constructs allow us to create an algorithm for any solvable problem.
- The principles of structured programming, however, require that an algorithm be broken into small units called subalgorithms.
- Each subalgorithm is in turn divided into smaller subalgorithms. A good example is the algorithm for the selection sort.

Subalgorithms

- An example of subalgorithms: selection sort



SelectionSort algorithm



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Subalgorithms

```
26
11      for (int wall=0; wall<DATA_SIZE-1; wall+=1) // only DATA_SIZE-1 (6) passes
12      {
13          int min_pos = wall;
14
15          // Find the position of the minimal element
16          for (int next=min_pos+1; next<DATA_SIZE; next+=1)
17          {
18              if (data[next] < data[min_pos])
19              {
20                  min_pos = next;
21              }
22          }
23
24          // Swap the first element and the minimal element
25          int tmp = data[min_pos];
26          data[min_pos] = data[wall];
27          data[wall] = tmp;
28      }
```

```
18      return min_pos;
19  }
```


Recursion

- In general, there are two approaches to writing algorithms for solving a problem.
- One uses iteration, the other uses recursion.
- Recursion is a process in which an algorithm calls itself.

Recursion

- To study a simple example, consider the calculation of a factorial.

Iterative definition

$$\text{Factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1) \times (n-2) \quad \cdots \quad 3 \times 2 \times 1 & \text{if } n > 0 \end{cases}$$

Recursive definition

$$\text{Factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times \text{Factorial}(n-1) & \text{if } n > 0 \end{cases}$$

B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Recursion

- Iterative vs. recursive algorithms

Algorithm 8.6 An iterative solution to the factorial pr

```
Algorithm: Factorial (n)
Purpose: Find the factorial of a number using a loop
Pre: Given: n
Post: None
Return: n!
{
    F ← 1
    i ← 1
    while (i ≤ n)
    {
        F ← F × i
        i ← i + 1
    }
    return F
}
```

Algorithm 8.7 Pseudocode for recursive solution of fact

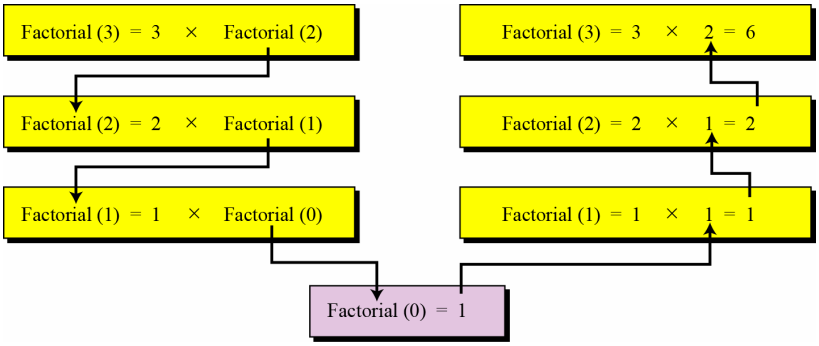
```
Algorithm: Factorial (n)
Purpose: Find the factorial of a number using recursion
Pre: Given: n
Post: None
Return: n!
{
    if (n = 0)      return 1
    else            return n × Factorial (n −1)
}
```

Recursion

```
int factorial_R(int n)
{
    if (n <= 1)
        return 1;
    else
        return n*factorial_R(n-1);
}
// ---------------------------------------------
int factorial_I(int n)
{
    int val = 1;
    for (int i=2; i<=n; i+=1)
    {
        val *= i;
    }
    return val;
}
// ---------------------------------------------
int main()
{
    cout << factorial_R(3) << ' ' << factorial_I(3) << endl;
```


Recursion

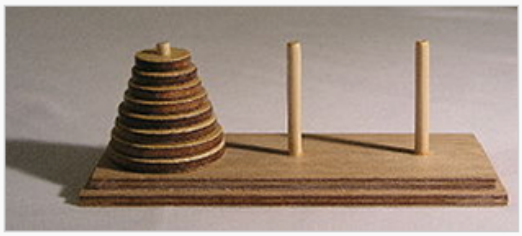
- Tracing the recursive solution to the factorial problem



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Recursion

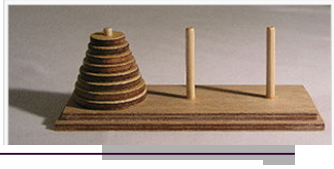
- Another example: Tower of Honai



A model set of the Towers of Hanoi (with 8 disks)

http://en.wikipedia.org/wiki/Tower_of_hanoi

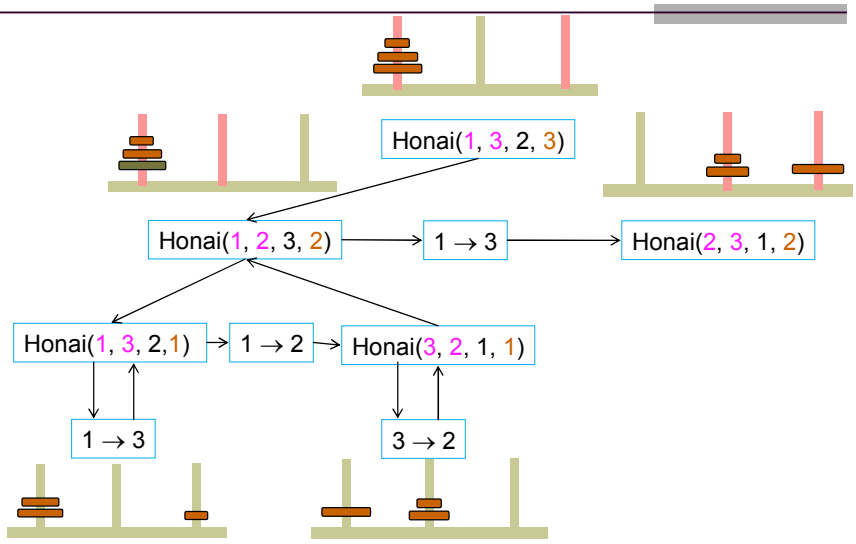
Recursion



```
void SolveHonaiTower(int from, int to, int aux, int num_circles)
{
    if (num_circles <= 1)
    {
        cout << "Move from " << from << " to " << to << '.' << endl;
    }
    else
    {
        SolveHonaiTower(from, aux, to, num_circles-1);
        cout << "Move from " << from << " to " << to << '.' << endl;
        SolveHonaiTower(aux, to, from, num_circles-1);
    }
}

int main()
{
    SolveHonaiTower(1, 3, 2, 3);

    system("pause");
    return 0;
}
```

```
D:\Teaching Course\_Introduction to Compute
Move from 1 to 3.
Move from 1 to 2.
Move from 3 to 2.
Move from 1 to 3.
Move from 2 to 1.
Move from 2 to 3.
Move from 1 to 3.
請按任意鍵繼續 . . .
```

Recursion



Recursion

- Other examples:
  - Greatest common divisor (gcd)

$$\gcd(x,y)=\begin{cases} x & \text{if } y=0 \\ \gcd(y, x \bmod y) & \text{otherwise} \end{cases}$$

  - Combination of n objects taken k at a time

$$C(n,k)=\begin{cases} 1 & \text{if } k=0 \text{ or } n=k \\ C(n-1,k)+C(n-1,k-1) & \text{if } n>k>0 \end{cases}$$

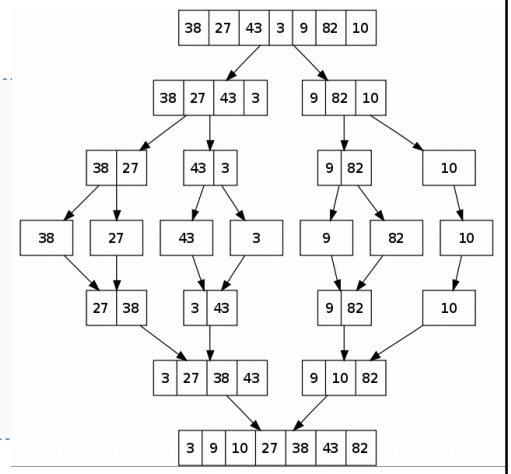
  - Fibonacci number

$$Fib(n)=\begin{cases} 0 & ,n=0 \\ 1 & ,n=1 \\ Fib(n-1)+Fib(n-2) & ,n>1 \end{cases}$$

Recursion

- Other examples:
  - Merge sort

```
function merge_sort(m)
    if length(m) ≤ 1
        return m
    var list left, right, result

    var integer middle = length(m) / 2
    for each x in m up to middle
        add x to left
    for each x in m after middle
        add x to right
    left = merge_sort(left)
    right = merge_sort(right)
    if left.last_item > right.first_item
        result = merge(left, right)
    else
        result = append(left, right)
    return result
```



http://en.wikipedia.org/wiki/Merge_sort

Recursion

- Recursion is convenient for thinking and programming.
- However, it is not costless. It takes both time and space to deal with function calls.
- If possible, try loops before recursions.

Summary

- An algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.
- Three constructs for a structured program/algorithm:
  - sequence
  - decision
  - repetition

Summary

- Algorithms can be represented in several ways:
  - Unified modeling language (UML)
  - Pseudocode
- Basic algorithms
  - Summation/product
  - Min/max
  - Sort
  - Search

Summary

- The principles of structured programming require that an algorithm be broken into small units called subalgorithms.
- There are two general ways to write algorithms: iterative and recursion.
  - An algorithm is recursive whenever the algorithm appears within the definition itself.