Chapter 3.

Algorithms

Part I: Algorithms

Dept. of Software
Gachon University
Spring 2022

Contents

What is a Recipe ?

- A set of directions (steps) for making food or beverage.
  – Recipes tell you how to accomplish a task by performing a number of steps.
- e.g., to bake a cake the steps are:
  – preheat the oven; mix flour, sugar, and eggs thoroughly; pour into a baking pan; and so forth.

Good Recipe

- A good recipe has two parts
  - A list of ingredients with the amounts required
  - Step By Step Directions for Mixing & Handling ingredients

Grandma's Potato Salad

Ingredients

- 4 large potatos
- 1 small onion
- 3/4 C. Chopped Celery
- 2 hard boiled eggs, Chopped
- 2/3 Cup Mayonaise
- 2 tsp. salt
- Pepper

Directions

1. Boil potatos with skins on. When potatos are tender drain & cool.
2. When cool, peel and slice into large bowl.
3. Add remaining ingrediants. Mix all together.
4. Taste and add more salt if necessary.

DO NOT SKIMP on salt or mayonaise!

Adjusting/Applying Recipes

- A recipe is usually described for “general cases”
  - E.g., serving for 2 person
- The yield of the recipe can be changed/adjusted
  - (yield : number of servings the recipe makes)
- A recipe can be applied/extended to make different food or beverage.

Problems

- Problem 1. Given a list of positive numbers, return the largest number on the list.

general problems

- Problem 2. From a sequence {101, 12, 144, 98}, find the largest number.

A specific case of problem (input is specific)

- Problem 3. Find the ~~longest~~ shortest word in a sentence.

A specific case, too

Solutions for general problems → can be applied to solve lots of different types of problems

Algorithms

- An ***algorithm*** is a finite set of precise instructions for performing a computation or for solving a problem.

- Many other definitions:
  - *A well-defined computational procedure* that takes some value, or set of values, as input and produces some value, or set of values, as output.
  - A sequence of computational steps that transform the input into the output.
  - A tool for solving a well-specified computational problem.

Example

Describe an algorithm for finding the maximum value in a finite sequence of integers.

- **Solution:** Perform the following steps:
  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.
     – 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. If not, stop.
  4. When the algorithm terminates, the temporary maximum is the largest integer in the sequence.

Specifying Algorithms : Pseudocode

- Intermediate step between English language description and programming language implementation

- Example:
  - Finding the Maximum Element in a Finite Sequence

```
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}
```

Execute the Max Algorithm

- When you start up a piece of software, we say the program or its algorithm are being *run* or *executed* by the computer.

- Example: Let $\{a_i\}=\{7,12,3,15,8\}$. Find its maximum… *by hand*.
- Set $max = a_1 = 7$.
- Look at next element: $a_2 = 12$.
- Is $a_2 > max$? Yes, so change v to 12.
- Look at next element: $a_3 = 3$.
- Is 3>12? No, leave *max* alone….
- Is 15>12? Yes, *max* =15…

Properties of algorithms:

An algorithm must satisfies the following criteria. (Some important features)

- **Input** : an algorithm accepts zero or more inputs.
- **Output** : Information or data that goes out (produce at least one output).
- **Definiteness** : Every step in the computation defined precisely (unambiguous)
- **Correctness** : Correct output for every possible input.
- **Finiteness** : an algorithm terminates after a finite numbers of steps
- **Effectiveness** : Individual steps are all realizable (the instructions can be performed by using the given inputs in a finite amount of time)
- **Generality** : Works for many possible inputs.

Some Example Algorithm Problems

- Three classes of problems will be studied in this section.

1. ***Searching Problems***: finding the position of a particular element in a list.
2. ***Sorting problems***: putting the elements of a list into increasing order.
3. ***Optimization Problems***: determining the optimal value (maximum or minimum) of a particular quantity over all possible inputs.

Searching Problems

- **Definition**: The general *searching problem* is to locate an element x in an ordered list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list.
    - Given a list of n elements that are sorted into a definite order (*e.g.*, numeric, alphabetical),
    - And given a particular element x,
    - Determine whether x appears in the list,
    - and if so, return its *index* (*position*) in the list. (if not exist, return 0)
  - For example, a program that checks the spelling of words searches for them in a dictionary.
- We will cover - linear search and binary search.

Linear Search (sequential search)

- A linear search algorithm, that is, an algorithm that linearly searches a sequence for a particular element.
- Ex. Find '12' in {3, 6, 9, 12, 15, 18, 21, 24}

procedure *linear search*

(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)

$i := 1$

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

$i := i + 1$

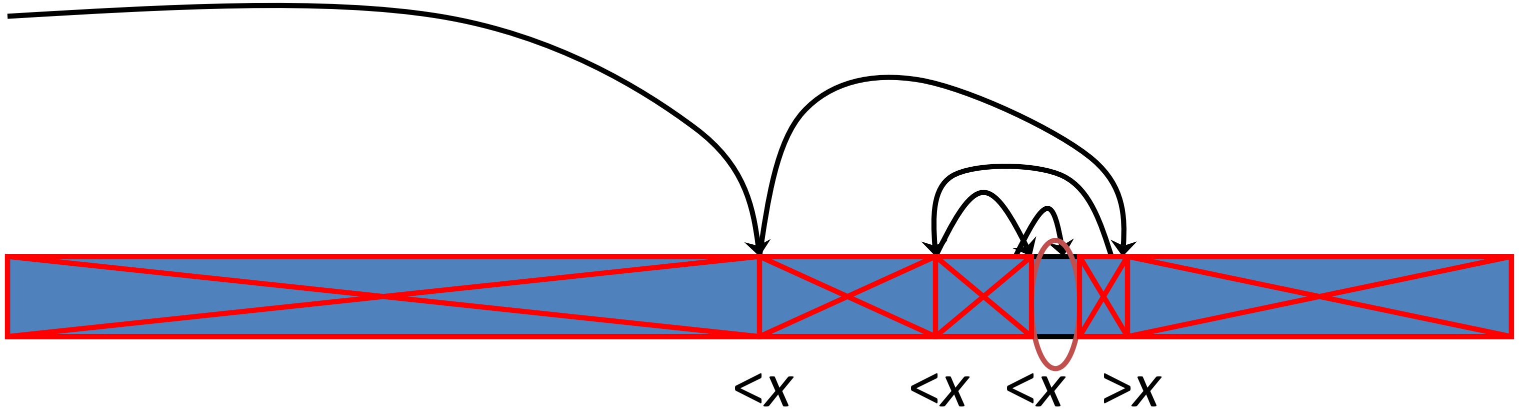
if $i \leq n$ **then** *location* := i

else *location* := 0

return *location* {index or 0 if not found}

Binary Search

- Basic idea: On each step, look at the *middle* element of the remaining list to eliminate half of it.
  - Assume the input is a list of items in increasing order.
- Ex. Find ‘18’ in { 3, 6, 9, 12, 15, 18, 21, 24 }



Binary Search

```
procedure binary search
(x:integer, a_1, a_2, …, a_n: distinct integers)
    i := 1  {left endpoint of search interval}
    j := n  {right endpoint of search interval}
    while i<j begin   {while interval has >1 item}
        m := ⌊(i+j)/2⌋  {midpoint}
        if x>a_m then i := m+1 else j := m
    end
    if x = a_i then location := i else location := 0
    return location
```

Binary Search Example

Example 3: The steps taken by a binary search for 19 in the list:

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

1. The list has 16 elements, so the midpoint is 8. The value in the 8th position is 10. Since 19 > 10, further search is restricted to positions 9 through 16.

   1 2 3 5 6 7 8 10   12 13 15 16 18 19 20 22

2. The midpoint of the list (positions 9 through 16) is now the 12th position with a value of 16. Since 19 > 16, further search is restricted to the 13th position and above.

   1 2 3 5 6 7 8 10   12 13 15 16   18 19 20 22

3. The midpoint of the current list is now the 14th position with a value of 19. Since 19 $\ngtr$ 19, further search is restricted to the portion from the 13th through the 14th positions .

   1 2 3 5 6 7 8 10   12 13 15 16   18 19   20 22

4. The midpoint of the current list is now the 13th position with a value of 18. Since 19> 18, search is restricted to the portion from the 18th position through the 18th.

   1 2 3 5 6 7 8 10   12 13 15 16   18   19   20 22

5. Now the list has a single element and the loop ends. Since 19=19, the location 16 is returned.

Linear Search vs. Binary Search

- Obviously, on sorted sequences, binary search is more efficient than linear search.
- How can we analyze the efficiency of algorithms?

We can measure the

- **time** (number of elementary computations) and
- **space** (number of memory cells) that the algorithm requires.

These measures are called **computational complexity** and **space complexity**, respectively. (we will study soon)

Sorting

- Sorting
  - The problem of ordering the element of a list
- Sorting is a common operation in many applications.
  - *e.g.* spreadsheets and databases

- Two sorting algorithms shown in textbook:
  - Bubble sort
  - Insertion sort

However, these are *not* very efficient, and you should not use them on large data sets!

Bubble Sort

- Example: Sort ***L*** = {3, 2, 4, 1, 5}

1st pass

3	2	2	2
2	3	3	3
4	4	4	1
1	1	1	4
5	5	5	5

2nd pass

2	2	2
3	3	1
1	1	3
4	4	4
5	5	5

3rd pass

2	1
1	2
3	3
4	4
5	5

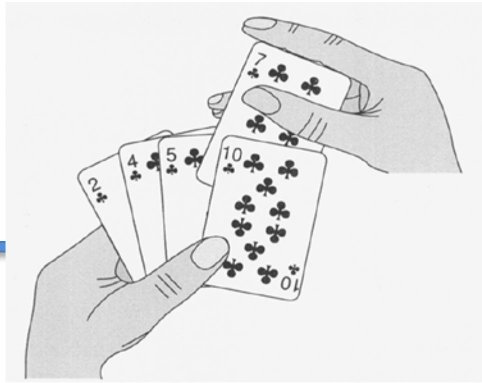
4th pass

1
2
3
4
5

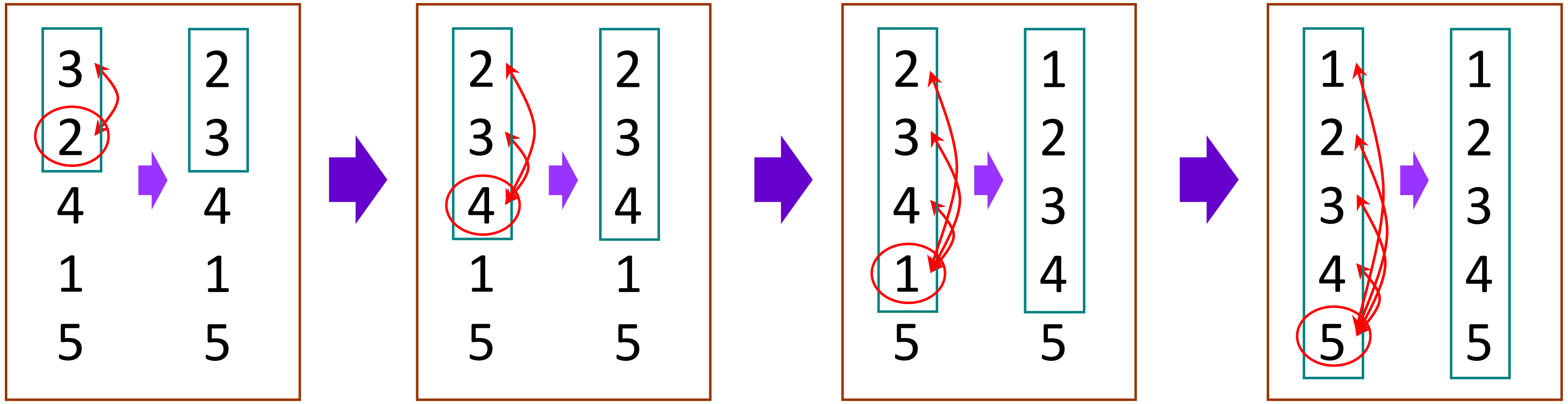
Bubble Sort

– Every pair of elements that are found to be out of order are interchanged.

Insertion Sort



- Example: Sort ***L*** = {3, 2, 4, 1, 5}



Examples of insertion sort

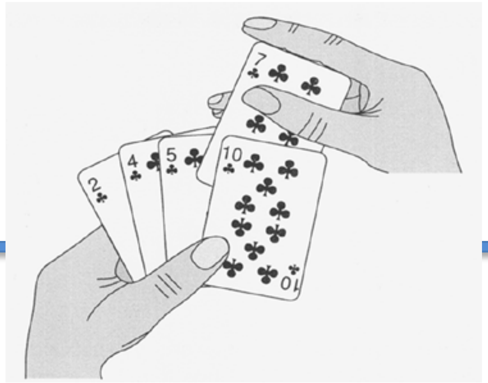
8 2 4 9 3 6

Jth Element

Examples of insertion sort

8 2 4 9 3 6

Insertion Sort



Question

- *Question:* Design an algorithm for making change (in U.S. money) of *n* cents with the following coins: quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent) , using the least total number of coins.
  For example, make change for 67 cents.

- We first select a quarter (leaving 42 cents).We next select a second quarter (leaving 17 cents), followed by a dime (leaving 7 cents), followed by a nickel (leaving 2 cents), followed by a penny (leaving 1 cent), followed by a penny.

Greedy Algorithm

- **Solution**: Greedy change-making algorithm for n cents. The algorithm works with any coin denominations $c_1, c_2, \ldots, c_r$.

procedure *change*($c_1, c_2, \ldots, c_r$: values of coins, where $c_1 > c_2 > \ldots > c_r$;
    n: a positive integer)
for $i := 1$ to r
    $d_i := 0$ [d_i counts the coins of denomination c_i]
    while $n \geq c_i$
        $d_i := d_i + 1$ [add a coin of denomination c_i]
        $n = n - c_i$
[d_i counts the coins c_i]

- Optimization problems can often be solved using a *greedy algorithm*, which makes the "best" choice at each step.
  - Making the "best choice" at each step does not necessarily produce an optimal solution to the overall problem, but in many instances, it does.

Section Summary

- Properties of Algorithms
- Algorithms for Searching and Sorting
- Greedy Algorithms