Introduction to Algorithms

Module Code: CELEN086

Lecture 6

(4/11/24)

Lecturer & Convenor: Manish Dhyani
Email: manish.dhyani@nottingham.edu.cn

Insertion sort (recap)

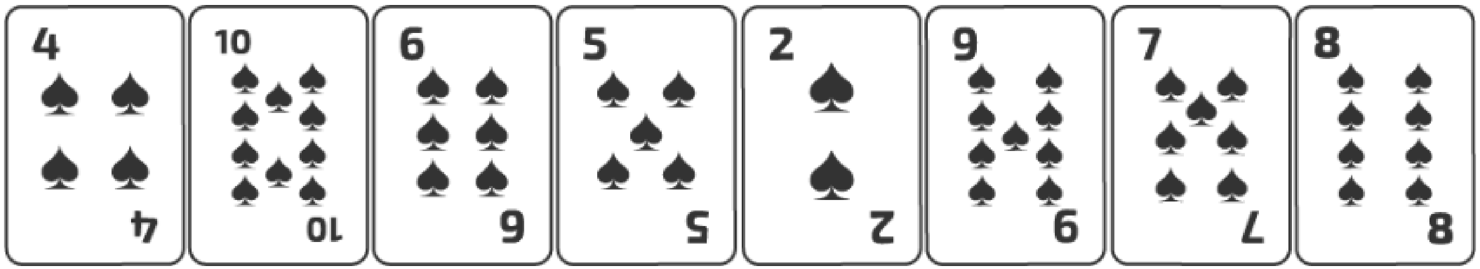
Insertion sort has two components:

- Make comparisons
- Insert element into the right position

Time complexity of insertion sort

Best case	Worst case	Average case
$O(n)$	$O(n^2)$	$O(n^2)$

Help me find the highest card



Divide & **conquer** in action

- If you have just one card,
  - turn it over and you're done.
- If you have more than one card:
  - Divide your cards into two equal stacks;
  - Ask a friend for the highest number in those stacks;
  - Use their solutions to find the highest number in the original stack.

A pair?

Written as (x, y)

x is the first item in the pair and y is the second

Let $(x', y') = \text{split}([1,2,3,4,5])$

x' is now the first value in the pair, y' the second.

Pair is a special case of tuple, a data-structure for lists of fixed finite lengths.

Especially useful for returning (or dealing with fixed length) multiple values. Supported in languages such as Haskell and ML …

Split : a List in the two sub list .

Algorithm: split(*list*)

Requires: A list of positive numbers *list*;

Returns: Two lists *list1* and *list2* that together make up *list*.

if isEmpty(*list*) **then**

    return (*Nil, Nil*)

else if isEmpty(tail(*list*)) **then**

    return (*list, Nil*)

else

    let *x* = value(*list*)

    let *y* = value(tail(*list*))

    let (L1, L2) = split(tail(tail(*list*)))

    return (*Cons*(*x,* L1)*, Cons*(*y, L2*))

endif

Practice

Calculate *split* [1,2,5,6]. Show the intermediate steps of the computation.

Algorithm: split(*list*)

Requires: A list of positive numbers *list*;

Returns: Two lists *list1* and *list2* that together make up *list*.

```
if isEmpty(list) then
    return (Nil, Nil)
else if isEmpty(tail(list)) then
    return (list, Nil)
else
    let x = value(list)
    let y = value(tail(list))
    let (L1, L2) = split(tail(tail(list)))
    return (Cons(x, L1), Cons(y, L2))
endif
```

First Call: split([1, 2, 5, 6])
- x = 1
- y = 2
- Recursive call: split([5,6])

Second Call: split([5, 6])
- x = 5
- y = 6
- Recursive call: split([ ])

Base Case: split([])
returns (Nil, Nil)

Building the Lists

Returning from Second Call:
- (L1, L2) = (Nil, Nil)
- list1 = [5]
- list2 = [6]

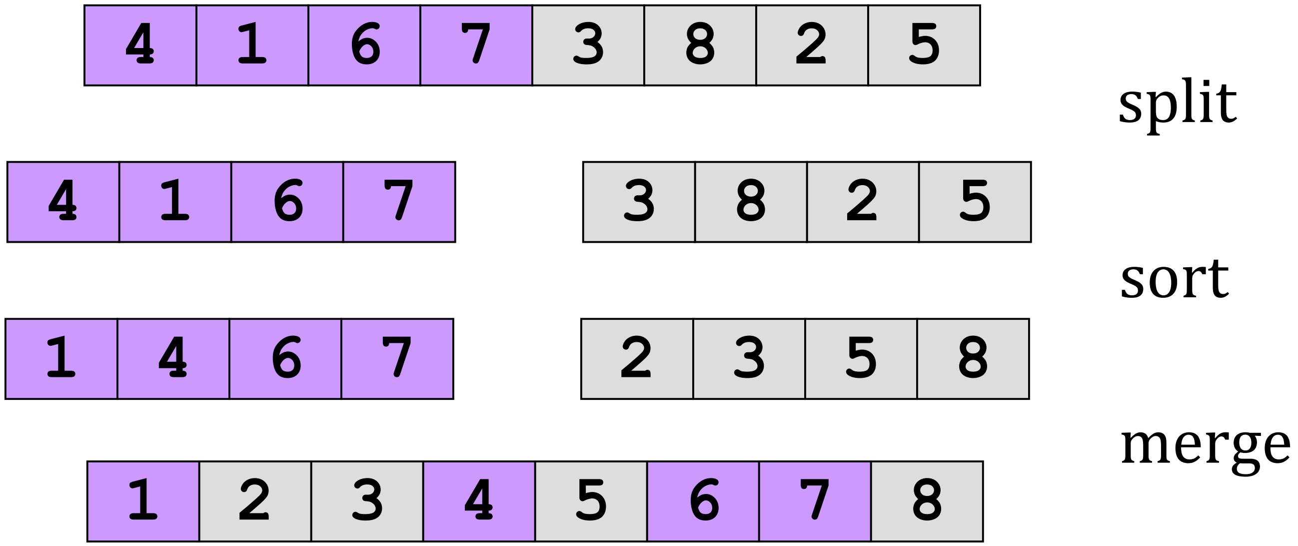
Returning from First Call:
- (L1, L2) = ([5], [6])
- list1 = [1, 5]
- list2 = [2, 6]

- Final Lists:
  - list1 = [1, 5]
  - list2 = [2, 6]

Merge sort

Merge sort has three components:

- Split the list into two sub-lists
- Sort each of the sub-lists (recursively)
- Merge the sub-lists



Divide-and-Conquer algorithm

Algorithm: merge

Requires:

list1

1	4	6	7

list2

2	3	5	8

Returns:

list

1	2	3	4	5	6	7	8

merge(list1, list2) = list

Idea of merging:

merge([1,4,6,7],[2,3,5,8]) =cons(1,merge([4,6,7],[2,3,5,8]))

Simpler instance of the same problem

=cons(1,cons(2, merge([4,6,7],[3,5,8])))

=...

=cons(1, cons(2, cons(...merge([empty list],[another list]))))

Base case

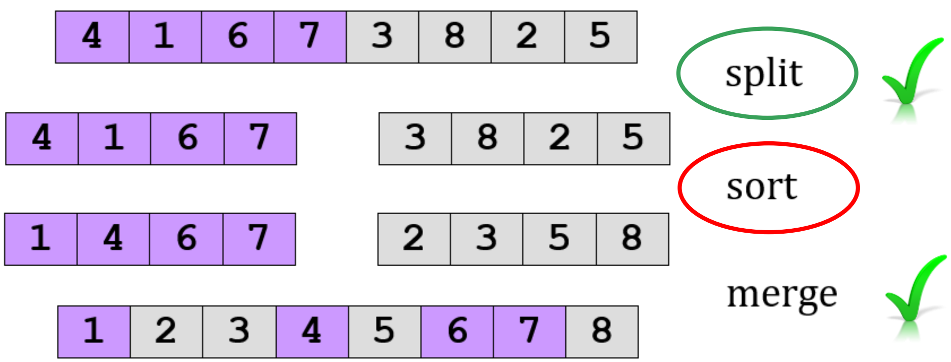
Algorithm: merge

Algorithm: merge(list1, list2)
Requires: two sorted lists
Returns: a merged (and sorted) list

```
if  isEmpty(list1)
     return  list2  // base case
elseif  isEmpty(list2)
     return  list1  // base case
elseif  value(list1)<value(list2)
     return  cons(value(list1),merge(tail(list1), list2) )
else
     return  cons(value (list2),merge(list1, tail(list2) ))
endif
```

You should trace it with the given example.

Algorithm: split



Algorithm: split(list) [Exercise; Seminar 6]
Requires: one list
Returns: two lists, each containing half elements of the original list

split(list) = (list1, list2)

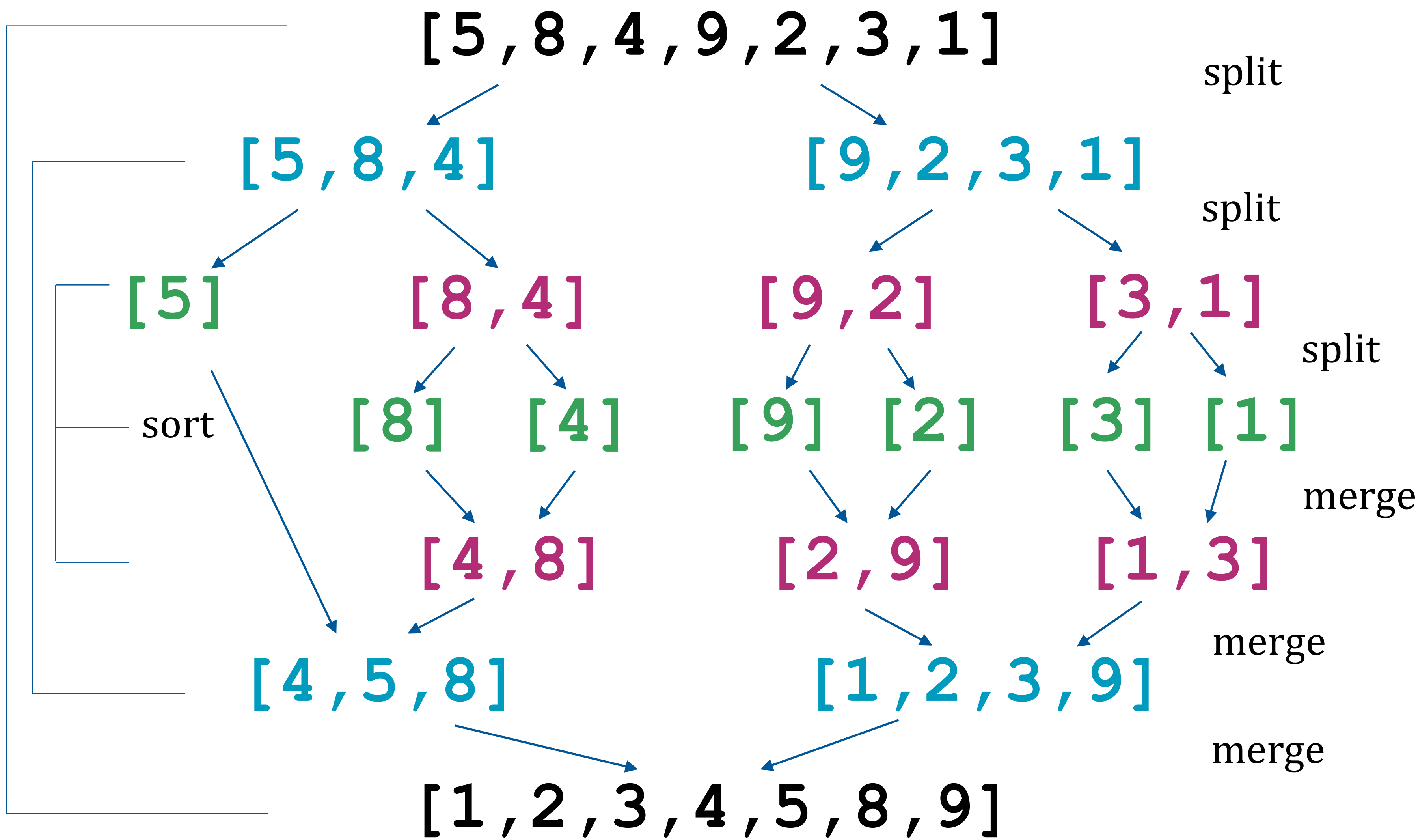
Requires: 4 1 6 7 3 8 2 5

Returns: 4 1 6 7   3 8 2 5

split algorithm will return multiple objects (lists) as an ordered pair.

Example: split([4,1,6,7,3,8,2,5])=([4,1,6,7], [3,8,2,5])

Example


[5,8,4,9,2,3,1]
split
[5,8,4]
[9,2,3,1]
split
[5]
[8,4]
[9,2]
[3,1]
split
sort
[8]
[4]
[9]
[2]
[3]
[1]
merge
[4,8]
[2,9]
[1,3]
merge
[4,5,8]
[1,2,3,9]
merge
[1,2,3,4,5,8,9]

Algorithm: mergeSort

Algorithm: mergeSort(list)
Requires: one list
Returns: a sorted list (with same elements)

```
1. if isEmpty(list) || isEmpty(tail(list))
2.      return list // list is already sorted (base case)
3. else
4.      let (L1,L2) = split(list)  // splitting list
5.      let S1 = mergeSort(L1) // sorting (recursive call)
6.      let S2 = mergeSort(L2) // sorting (recursive call)
7.      return merge(S1, S2) // merging
8. endif
```

Sub algorithms:

- split()
- merge()

You must trace it with the given example.

Merge sort has time complexity of $O(n \log_2 n)$ in its best/worst/average cases.

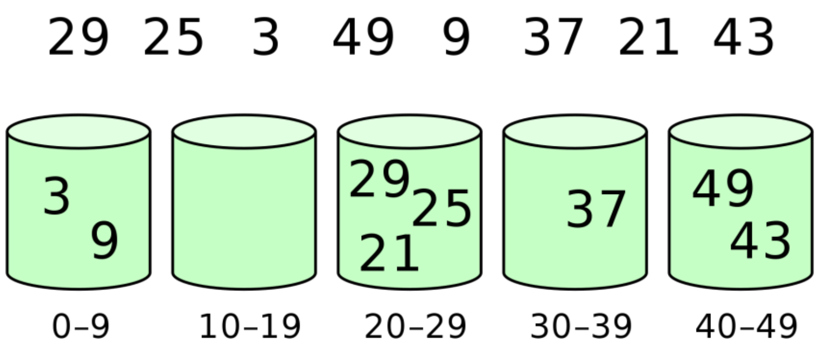
Bucket sort

If we know how elements from the target list are distributed, we can use bucket sort (also called bin sort).

Idea of bucket sort:

- Set up k empty "buckets".
- Scatter: Go over n elements in original list, putting each element in its corresponding bucket.
- Sort each non-empty bucket.
- Gather: Visit each bucket in order and gather all elements together.

(Source: Wikipedia)



[3,9] [21,25,29] [37] [43,49]

[3,9,21,25,29,37,43,49]

Time complexity $O(n+k)$ (best/average case)

$O(n^2)$ (worst case)

Recap

Searching schemes:

- Linear search
- Binary search

Sorting schemes:

- Insertion sort
- Merge sort
- Bucket sort

You should be able to:

- ✓ Search/sort a given list using specified scheme.
- ✓ Design algorithm for simple schemes. Trace the given algorithm for complex schemes.
- ✓ Know the time complexity and be able to explain it.

Note: For other sorting methods, such as selection sort and bubble sort, we will learn and design programs for them in Sem-2.

Mid-semester Exam Information

- Date & Time:

Wednesday 20 November 2024, 15:00 – 16:00
(Confirm your exam room from email sent by CPSO)

- Topic: Lectures 1-6

- Question formats:

20 questions
(15 Multiple-choice questions + 5 problem-solving questions)

$15 \times 1 + 5 \times 7 = 50$ marks. 25% towards your module final marks

Sample paper is available on Moodle.

Independent Learning Week

- ❑ No teaching activities (lecture/seminar)

  Office hours are scheduled as normal

- ❑ Two activities for this module

  Individual work

  Group work (3~5 students per group)

  Detailed instruction will be announced at the start of w/c 11/11/24.

(Optional, Advanced) Further reading...

- These are all **outside** the scope of our module but can give you a preview of things you may see in future years.
  - See http://en.wikipedia.org/wiki/Sorting_algorithm for the details and lots of algorithms I haven't discussed.
  - See http://en.wikipedia.org/wiki/Algorithmic_complexity for a reason to examine the speed of an algorithm.
  - See http://en.wikipedia.org/wiki/Big_O_notation for the notation we use and the speed of common algorithms.

Review Activity : Lecture 6