Algorithms

Jordi Planes

Escola Politècnica Superior
Universitat de Lleida

2016

Syllabus

What's been done

- Formal specification
- Computational Cost
- Transformation recursion $\rightarrow$ iteration
- Divide and conquer
- Sorting
- Multiple call transformation

Syllabus

What we'll do today

- Formal specification
- Computational Cost
- Transformation recursion $\rightarrow$ iteration
- Divide and conquer
- Sorting
- Multiple call transformation

Divide and Conquer

Divide and Conquer

The divide and conquer strategy solves a problem by:

1. Breaking it into subproblems
2. Recursively solving these subproblems
3. Appropriately combining their answers.

Sorting

Sorting

How do you sort a set of play cards?

Sorting

Give the pseudocode, find the computational cost and trace an example for:

Sorting

Simple sorting

- Bubble sort

- Pancake sorting

Sorting

No so Simple sorting

- Insertion sort
- Selection sort
- Heap sort

Sorting

Divide and conquer

- Merge Sort
- Quick Sort

Merge sort

```
mergesort( a:[1...n] ) :
    if n > 1 then
        return merge( mergesort( [1...n/2] ),
                      mergesort( [n/2+1...n] )
    else
        return a
```

Merge sort

Exercises

Design function `merge( x, y )`, where x and y are two sorted arrays, and returns the merging of them.

Merge sort

Exercises

Design function `merge( x, y )`, where x and y are two sorted arrays, and returns the merging of them.

```
merge( x[1...k], y[1...l] ) :
    if k = 0 : return y[1...l]
    if l = 0 : return x[1...k]
    if x[1] ≤ y[1] :
        return x[1] + merge( x[2...k], y[1...l])
    else :
            return x[1] + merge( x[1...k], y[2...l])
```

Merge sort

Find the computational cost and trace an example

Quick sort

```
function quicksort(A, lo, hi) :
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p)
        quicksort(A, p + 1, hi)
```

Quick sort

```
function partition(A, lo, hi) :
    pivot ← A[hi]
    i ← lo
    for j ← lo to hi − 1 do
        if A[j] <= pivot then
            swap( A[i], A[j] )
            i ← i + 1
    swap( A[i], A[hi] )
    return i
```

Quick sort

Find the computational cost and trace an example

Binary search

```
function binarysearch( a, el, lower, upper ) :
    if lower <= upper then
        mid ← (lower+upper)/2
        if el = a[mid] return true
        if el < a[mid]
            return binarysearch( a, el, lower, mid−1 )
        else
            return binarysearch( a, el, mid+1, upper )
    return false
```

Binary search

Find the computational cost and trace an example

Exercises

Work in groups.

- Maximum sum subarray
- Compute median
- Remove duplicates from an array
- Find if exists index equals value ($a[i] = i$)
- Skyline
- Matrix multiplication

Transformation

Review

Associative and conmutative: Tail call transformation.

```
function iterative( x ) is
    a ← trivial( x )
    while not base case ( x ) do
        a ← compute( x, a )
        x ← reduce( x )
    return a
```

Examples: addition, product

Review

Not Associative or not conmutative: transformation using inverse.

```
x' = x
while not base case
    x' = reduce( x' )

a = trivial( x' )
while x ≠ x'
    x' = inverse reduce( x' )
    a = compute( a, x' )

return a
```

Example: substraction

Review

Not inverse: transformation usign stack.

```
while not base case
    push to stack x
    reduce( x )

a = trivial( x )
while stack not empty
    a = compute( a, top )
    pop

return a
```

Example: integer division

Multiple recursion

Merge sort

```
mergesort( a:[1...n] ) :
    if n > 1 then
        return merge( mergesort( [1...n/2] ),
                      mergesort( [n/2+1...n] )
    else
        return a
```

Merge sort transformation

```
mergesort( a )
    for all item i in a
        push( [i] )
    while |Q| > 1
        push( merge( pop(), pop() ) )
    return pop()
```

Quick sort transformation

```
function quicksort(A, lo, hi) :
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p)
        quicksort(A, p + 1, hi)
```

Quick sort transformation

```
function quicksort(A:[1...n]) :
    push( <1,n> )
    while stack not empty :
        <low,up> ← pop()
        if ( low < up ) then
            m ← (low + up / 2)
            partition( A, low, up, m )
            push( <m,up> )
            push( <low,m−1>)
```

Exercises

Find the recursive function and transform to iterative:

1. Binary tree height

Exercises

Find the recursive function and transform to iterative:

1. Binary tree height

```
function height( tree ) is
  if empty tree then
    return 0
  return 1 + max( height( tree.left )
                  height( tree.right) ) )
```

Exercises

Find the recursive function and transform to iterative:

1. Hanoi towers

Exercises

Find the recursive function and transform to iterative:

1. Hanoi towers

1. move $n-1$ discs from A to B. This leaves disc n alone on peg A
2. move disc n from A to C
3. move $n-1$ discs from B to C so they sit on disc n