CSCE 2100: Computing Foundations 1

Iteration vs. Recursion

Tamara Schneider
Jorge-Reyes Silveyra
Fall 2012

Sorting

<u>Goal</u>: Permute a list of n elements, such that they are sorted in increasing (or decreasing) order

- Example: $(4, 2, 7, 3, 5, 3)$
- Sorted list: $(2, 3, 3, 4, 5, 7)$

What type of lists can be sorted?

- "Less than" order must be defined
- Lexicographic order: order on strings

There are iterative and recursive algorithms.

Lexicographic Ordering

- Dictionary, alphabetic ordering
- Compare strings $x = x_1x_2 \ldots x_m$ and $y = y_1y_2 \ldots y_n$.
- We say that if one of the following is true:
  - Either is a proper prefix of , i.e. m<n and for i=1,2,…,m: $x_i=y_i$, or
  - For some i>0, $x_j=y_j$ for j=0,2,..,i-1 and $x_i<y_i$
- What about the empty string ε?
- Sort `base`, `ball`, `mound`, `bat`, `glove`, `batter`

Lexicographic Order Example

`ball-base-bat-batter-glove-mound`

`ball < base`	$x_1=y_1$, $x_2=y_2$, $x_3<y_3$
`base < bat`	$x_1=y_1$, $x_2=y_2$, $x_3<y_3$
`bat < batter`	$x_1=y_1$, $x_2=y_2$, $x_3=y_3$ (proper prefix)
`batter < glove`	$x_1<y_1$
`glove < mound`	$x_1<y_1$

Definition: Permutation

- A rearrangement of the elements of an ordered list
- Each element occurs *exactly* as many times as it occurred in the original list.
- Is $(4, 5, 3, 4)$ a permutation of $(4, 4, 3, 5)$?
- Is $(4, 3, 3, 2)$ a permutation of $(3, 4, 4, 2)$?

Definition: Sorting

Operation of converting an arbitrary list $(a_1,a_2,a_3,\ldots a_n)$ into a list $(b_1,b_2,b_3,\ldots b_n)$, such that

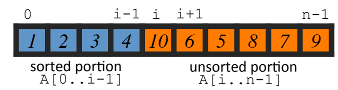
1. $(b_1,b_2,b_3,\ldots b_n)$ is in sorted order
2. $(b_1,b_2,b_3,\ldots b_n)$ is a permutation of the original list

Iteration

- Repetition of a mathematical or computational procedure applied to the result of a previous application.
- Example: use of looping constructs
  - for-statement
  - while-statement

Iterative Sorting: Selection Sort

Sort an array of size n in increasing order
`A[0] < A[1] < ... < A[n-2] < A[n-1]`
Assume the array consists of a contiguous sorted and contiguous unsorted portion
`A[0..i-1]` sorted
`A[i..n-1]` not sorted



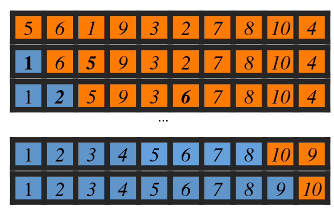
Iterative Sorting: Selection Sort

At each iteration, add the smallest element of the unsorted portion to the end of the sorted portion

- smallest element at index `small`
- exchange `A[i]` and `A[small]`

`A[0..i]` is sorted and `A[i+1..n-1]` is not sorted yet

Iterative Sorting: Selection Sort

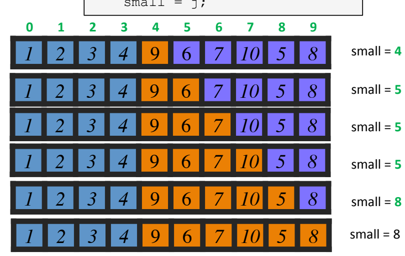


Selection Sort - Implementation

```
void SelectionSort(int A[], int n){
  int small, temp;
  for(int i=0; i<n-1; i++){
    small = i; //index of smallest so far
    for(int j=i+1; j<n; j++) //in unsorted part
      if(A[j] < A[small]) //if < smallest so far
        small = j; //then it's the smallest
    //swap A[i] with A[small]
    temp = A[small];
    A[small] = A[i];
    A[i] = temp;
  } //for
} //SelectionSort
```

Inner Loop of Selection Sort

```
small = i;
for(int j=i+1; j<n; j++)
  if(A[j] < A[small])
    small = j;
```



Selection Sort - Framework

```
#include <stdio.h>
const int MAX=100;
int A[MAX];
void SelectionSort(int A[], int n);
void main(){
  int n;
  //read and store input in A
  for(n=0; n<MAX && scanf("%d",&A[n])!=EOF; n++);
  //sort array
  SelectionSort(A, n);
  //print sorted array
  for(int i=0; i<n; i++)
    printf("%d\n", A[i]);
}
```

Iterative Sorting: Selection Sort

Examples

- Sort []
- Sort [5]
- Sort [5,4,3,2,1]
- Sort [1,8,4,2,9]

Recursion

- Solution of a problem is obtained by using the solutions of smaller instances of the problem
- Recursive functions call themselves
- Cleaner code for some applications

Concepts and Definitions

Self-Definition: A concept is defined or built in terms of itself

- No circularity
- Finite number of steps to smaller cases lead to base case

Basis Induction:

- Test for a basis case
- Inductive case

Inductive / Recursive Definitions

- **Basis rule(s)**, base case(s)
- **Inductive rule(s)** to build larger instances of concept from smaller ones
- Example: list
  - Basis rule: Empty list is a list
  - Inductive rule: element followed by a list is a list
- Inductive definitions ≠ Proofs by induction!!!

Recursive Definition of Factorial

Basis: 1! = 1

Induction: n! = n × (n-1)!

Example:

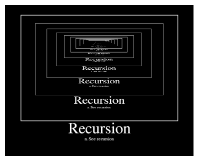
$$\left.\begin{aligned}5! &= 5 \times (5-1)! \\ &= 5 \times 4! \\ &= 5 \times 4 \times (4-1)! \\ &= 5 \times 4 \times 3! \\ &= 5 \times 4 \times 3 \times (3-1)! \\ &= 5 \times 4 \times 3 \times 2! \\ &= 5 \times 4 \times 3 \times 2 \times (2-1)! \\ &= 5 \times 4 \times 3 \times 2 \times 1!\end{aligned}\right\}\text{Induction}$$

$$= 5 \times 4 \times 3 \times 2 \times 1 \quad \text{(Basis)}$$

Recursive Functions

A function that calls itself

- Direct: directly calls itself
- Indirect: a chain of functions calls that results in calling itself (aka *mutual recursion*)



Recursive Factorial Implementation

Basis: 1! = 1.
Induction: n! = n × (n-1)!

```
int factorial(int n){
  if(n <=1 )return 1; //basis
  else return n * factorial(n-1); //induction
}
```

Recursive Definition: Lexicographic Order

Basis:

- ε < w for any string w ≠ ε
- If characters c<d, then for any string w and x: cw < dx

Induction:
If w < x for strings w and x, then for any character c: cw < cx

base	<	batter
ase	<	atter
se	<	tter

bat	<	batter
at	<	atter
t	<	tter
ε	<	ter

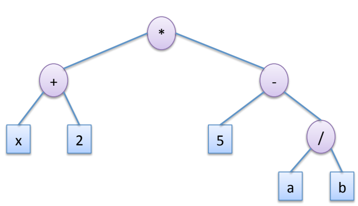
Recursive Definition: Arithmetic Expressions

Basis: Arithmetic expressions
1. Variables
2. Integers
3. Real Numbers

Induction: If E_1 and E_2 are arithmetic expressions, then the following are also arithmetic expressions:
1. $(E_1 + E_2)$
2. $(E_1 - E_2)$
3. $(E_1 \times E_2)$
4. (E_1 / E_2)
5. If E is an arithmetic expression, then so is (-E)

Expression Trees

- Expression trees can be used to represent recursively defined arithmetic expressions
- Example: (x + 2) * (5 – (a / b))

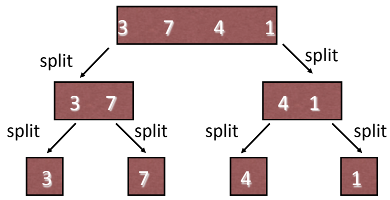


Recursive Sorting: MergeSort

- A Divide-and-Conquer Algorithm
- Break a problem into subproblems and solve them
- Combine solved subproblems into solution to problem
- Conditions:
  - Subproblem must be simpler than the original problem
  - After a finite number of subdivisions, a small subproblem that can directly be solved must be encountered

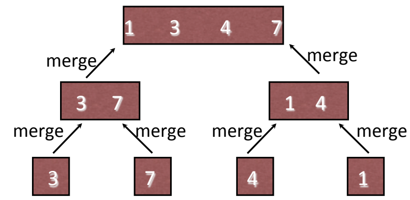
MergeSort – Recursive Sorting

- **Split** array at each recursive step into two arrays of half the size



MergeSort – Recursive Sorting

- **Merge** two sorted smaller arrays into a larger array



Recursive Sorting - Merge Sort

```
void mergeSort( *double A[] ) {  // assume length is
                                 // power of 2
   int n = A.size();
   if (n > 1) {
     double* B = new double[n/2];
     double* C = new double[n/2];
     split (A, B, C);
     mergeSort( B );
     mergeSort( C );
     merge( B, C, A );
   }
}
```

Summary

- Iteration
- Iterative Sorting
- Recursion
- Recursive / Inductive Definitions
- Recursive Sorting