**Aug 3 Recursion, Sorting**

**Concepts**

Function

Recursion

Sorting

% Script to compare reverse and reverseR:

% To compare the execution times for both versions of string reversion, we

% will reverse a long string many times and take the average of the total

% execution time.

% generate a long string

N=100; % number of strings

% print table header

fprintf(' len Nonrecursive Recursive\n');

fprintf('-------------------------------------\n');

for n=1:N

s='';

len=n\*10; % length of string is 10 times the string number

for i=1:len

% simply assign a to each position of the string

% s will be a long string of a's

s(i)='a';

end

numIter=100; % number of iterations

% time the nonrecursive version

tic

% reverse s repeatedly

for i=1:numIter

Reverse(s); % ignore the result

end

timeN=toc;

% time the recursive version

tic

% reverse s repeatedly

for i=1:numIter

Reverse(s); % ignore the result

end

timeR=toc;

% display execution time

fprintf('%5d %15.5f %15.5f\n',len,timeN/numIter,timeR/numIter);

end

% The recursive version is a little bit faster than the nonrecursive

% version. However, if we used vectorized code in the nonrecursive

% version, then the nonrecursive version should be faster.

function m=FactorialR(n)

% Return the factorial of n.

% n is a nonnegative integer.

% m is the factorial of n.

if n==0 % base case: 0!=1

m=1;

else % recursive case

m=n\*FactorialR(n-1);

end

function t=Reverse(s)

% Nonrecursive version of string reversion.

% s is the string to be reversed.

% t is the reverse of s.

% determine the length of s

len=length(s);

% Initialize the reverse as the empty string.

t='';

% Use a for loop to reverse s by starting at the end of s, work backward to

% In general, if the position of s is i, then the position of t is len-i+1;

for i=len:-1:1

t(len-i+1)=s(i);

end

function t=ReverseR(s)

% Recursive version of string reversion.

% s is the string to be reversed.

% t is the reverse of s.

% determine the length of s

n=length(s);

% base case: (n==1) s is the reverse of itself

if n==1

% simply assign s (which is the reverse of s) to t

t=s;

else

% obtain the reverse of the last n-1 characters of s recursively

subReverse=ReverseR(s(2:n));

% append the first character of s to the reverse

t=[subReverse s(1)];

end

**Sorting**

Helper method for **swapping**

function x = swap(x,i,j)

% Swap x(i) and x(j)

temp = x(i);

x(i) = x(j);

x(j) = temp;

**Selection sort**

function x = clgSelectionSort(x)

len = length(x);

for george = 1:len-1

juan = george;

for alice = juan+1:len

if (x(alice) < x(juan))

juan = alice;

end

end

if (juan ~= george)

x = swap(x,juan,george);

end

end

**Bubble sort**

function x = clgBubbleSort(x)

len = length(x);

for george = 1:len-1

for alice = len:-1:george+1

if (x(alice-1) > x(alice))

x = swap(x, alice-1, alice);

end

end

end

**Insertion sort**

function x = clgInsertionSort(x)

len = length(x);

for george = 2:len

toBeInserted = x(george);

alice = george;

while (alice > 1) && (x(alice - 1) > toBeInserted)

x(alice) = x(alice - 1);

alice = alice - 1;

end

x(alice) = toBeInserted;

end

**Quick sort**

function x = clgQuickSort(x)

len = length(x);

if len >= 2

george = floor(len/2);

pivot = x(george);

left = [];

right = [];

for alice = 1:len

if alice ~= george % not omittable

if x(alice) < pivot

left = [left x(alice)];

else

right = [right x(alice)];

end

end

end

x = [clgQuickSort(left) pivot clgQuickSort(right)];

end

**Bucket sort**

function x = clgBucketSort(x,bucketSize)

% Assumes that x consists of integers ranging from 1 to bucketSize only.

len = length(x);

bucket = zeros(1,bucketSize);

for i = 1:len

bucket(x(i)) = bucket(x(i))+1;

end

k = 1;

for i = 1:bucketSize

for j = 1:bucket(i)

x(k) = i;

k = k+1;

end

end

Radix sort

Merge sort

Heap sort

Cocktail sort

Comb sort

Counting sort

Shell sort