Homework

Ex 1: Factorial of the number n

Traditional solution:

Function Factorial\_Iterative(n)

result ← 1 1

For i ← 1 to n n

result ← result × I 1

End For

Return result 1

End Function

T(n)=1+n.1+1=n+2

=> O(n)

Recursive solution:

Function Factorial\_Recursive(n)

If n = 0 Then 1

Return 1 1

Else

Return n × Factorial\_Recursive(n - 1) n

End If

T(n)=n+1

=> O(n)

Ex 2: Merge-Sort

Recursive solution:

Merge (S1,S2)

result ← empty sequence 1

i ← 1, j ← 1 1

while i ≤ length(S1) and j ≤ length(S2) do log n

if S1[i] ≤ S2[j] then

append S1[i] to result

i ← i + 1

else

append S2[j] to result

j ← j + 1

while i ≤ length(S1) do log n

append S1[i] to result

i ← i + 1

while j ≤ length(S2) do log n

append S2[j] to result

j ← j + 1

return result 1

MERGE-SORT(S)

if length(S) ≤ 1 then n

return S 1

else

mid ← ⌊length(S) / 2⌋ n-1

S1 ← S[1 .. mid] n/2

S2 ← S[mid+1 .. length(S)] n/2

sorted\_S1 ← MERGE-SORT(S1) n/2

sorted\_S2 ← MERGE-SORT(S2) n/2

return MERGE(sorted\_S1, sorted\_S2) nlog(n)

T(n)=nlog(n)+1+1+log(n)+log(n)+log(n)+1+n+1+n-1+(n/2)\*4+nlog(n)

O(nlogn)

Ex 3: Compute power

Recursive solution:

RECURSIVE-POWER(b, n)

if n = 0 then 1

return 1 1

else

return b × RECURSIVE-POWER(b, n - 1) n+n+n+1

T(n)=3n+2

=>O(n)

Traditional solution:

ITERATIVE-POWER(b, n)

result ← 1 1

for i ← 1 to n do n

result ← result × b n

return result 1

T(n)=1+1+n=n+2

=>O(n)

Ex 4: A smart solution

Traditional solution:

ITERATIVE-POWER(b, n)

result ← 1 1

for i ← 1 to n do n times

result ← result × b n multiplications

return result 1

T(n)=2n+2

=>O(n)

Recursive solution:

SMART-POWER(b, n)

if n = 0 then

return 1 1

half ← SMART-POWER(b, ⌊n / 2⌋) T(n/2)

if n mod 2 = 0 then 1

return half × half 1 multiplication

else

return b × half × half 2 multiplications

T(n)=T(n/2)+c(constant)

=>O(logn)

Ex 5: Recursive Squaring

Traditional solution:

ITERATIVE-POWER(x, n)

result ← 1 1

for i ← 1 to n do n times

result ← result × x n multiplications

return result 1

T(n)=2n+2

=>O(n)

Recursive solution:

FAST-POWER(x, n)

if n = 0 then

return 1

temp ← FAST-POWER(x, ⌊n / 2⌋)

if n mod 2 = 0 then

return temp × temp

else

return x × temp × temp

T(n) = T(n/2)+c(constant)

=> O(log n)

Ex 6: Fibonacci sequence problem

Traditional solution:

ITERATIVE-FIB(n)

1. if n = 0 or n = 1 then 1

2. return 1 1

3. prev ← 1 1

4. curr ← 1 1

5. for i ← 2 to n do n - 1

6. temp ← curr 1

7. curr ← curr + prev 1

8. prev ← temp 1

9. return curr 1

=>O(n)

Recursive solution:

RECURSIVE-FIB(n)

1. if n = 0 or n = 1 then 1

2. return 1 1

3. return RECURSIVE-FIB(n - 1) + RECURSIVE-FIB(n - 2) → 2^n

=>O(2^n)

Ex 7: Linear search problem

Traditional solution:

LinearSearch(Names, Numbers, TargetName):

n = length of Names 1

FOR i FROM 0 TO n-1: n + 1

IF Names[i] == TargetName: n

RETURN Numbers[i] 1

ENDFOR

RETURN "Not Found" 1

T(n) = 2n + 4

O(n)

Recursive solution:

FUNCTION RecursiveSearch(Names, Numbers, TargetName, index):

n = length of Names 1

IF index >= n: 1

RETURN "Not Found" 1

IF Names[index] == TargetName: 1

RETURN Numbers[index] 1

ELSE:

RETURN RecursiveSearch(Names, Numbers, TargetName, index + 1) T(n-1)

O(n)

Ex 8: Find Largest Problem

Traditional solution:

FUNCTION FindLargest(List):

n = length of List 1

IF n == 0: RETURN null 1

largest\_so\_far = List[0] 1

FOR i FROM 1 TO n-1: n

IF List[i] > largest\_so\_far: n - 1

largest\_so\_far = List[i] n - 1

ENDFOR

RETURN largest\_so\_far 1

T(n) = 1 + 1 + 1 + n + (n - 1) + (n - 1) + 1 = 3n  
O(n)

Recursive solution:

FUNCTION RecursiveLargest(List, n):

IF n == 1: 1

RETURN List[0] 1

ELSE:

largest\_in\_rest = RecursiveLargest(List, n-1) T(n-1)

IF largest\_in\_rest > List[n-1]: 1

RETURN largest\_in\_rest 1

ELSE:

RETURN List[n-1] 1

O(n)

Ex 9: Binary Search problem

Traditional solution:

FUNCTION BinarySearch(SortedList, Target):

low = 0 1

high = length of SortedList - 1 1

WHILE low <= high: log(n) + 1

mid = low + (high - low) / 2 log(n)

IF SortedList[mid] == Target: log(n)

RETURN mid 1

ELSE IF SortedList[mid] < Target: log(n)

low = mid + 1 log(n)

ELSE:

high = mid - 1 log(n)

ENDWHILE

RETURN -1 1

T(n) = 1 + 1 + (log(n)+1) + 4\*log(n) + 1 = 5log(n) + 4  
O(log n)

Recursive solution:

FUNCTION RecursiveBinarySearch(List, Target, low, high):

IF low > high: 1

RETURN -1 1

mid = low + (high - low) / 2 1

IF List[mid] == Target: 1

RETURN mid 1

ELSE IF List[mid] < Target: 1

RETURN RecursiveBinarySearch(..., mid + 1, high) T(n/2)

ELSE:

RETURN RecursiveBinarySearch(..., low, mid - 1) T(n/2)

O(n)

Ex 10: Computing Prefix Averages Problem

Traditional solution:

FUNCTION EfficientPrefixAverages(X):

n = length of X 1

A = new array of size n n

current\_sum = 0 1

FOR i FROM 0 TO n-1: n + 1

current\_sum = current\_sum + X[i] n

A[i] = current\_sum / (i + 1) n

ENDFOR

RETURN A 1

T(n) = 1 + n + 1 + (n + 1) + n + n + 1 = 4n + 4  
O(n)

Recursive solution:

FUNCTION RecursivePrefixAvgHelper(X, A, i, current\_sum):

IF i == length of X: 1

RETURN 1

new\_sum = current\_sum + X[i] 1

A[i] = new\_sum / (i + 1) 1

RecursivePrefixAvgHelper(X, A, i + 1, new\_sum) T(n-1)

O(n)

Ex 11: Greatest Common Divisor of Two Integers

Traditional solution:

FUNCTION GCD\_Iterative(m, n):

WHILE n != 0: log(n) + 1

temp = n log(n)

n = m MOD n log(n)

m = temp log(n)

ENDWHILE

RETURN m 1

T(n) = (log(n)+1) + 3\*log(n) + 1 = 4log(n) + 2  
O(log n)

Recursive solution:

FUNCTION GCD\_Recursive(m, n):

IF n == 0: 1

RETURN m 1

ELSE:

RETURN GCD\_Recursive(n, m MOD n) T(n')

O(log n)

Ex 12: Selection Sort

Traditional solution:

FUNCTION SelectionSort(List):

n = length of List 1

FOR i FROM 0 TO n-2: n

min\_index = i n - 1

FOR j FROM i+1 TO n-1: (n²-n)/2

IF List[j] < List[min\_index]: (n²-n)/2

min\_index = j (n²-n)/2

ENDFOR

swap(List[i], List[min\_index]) n - 1

ENDFOR

RETURN List 1

 T(n) = (3/2)n^2

O(n²)

Recursive solution:

FUNCTION RecursiveSelectionSort(List, i):

n = length of List

IF i >= n - 1: 1

RETURN 1

min\_index = i 1

FOR j FROM i + 1 TO n - 1: n-i

IF List[j] < List[min\_index]: n-i-1

min\_index = j

ENDFOR

swap(List[i], List[min\_index]) 1

RecursiveSelectionSort(List, i + 1) T(n-1)

O(n^2)