Justin Cabral

CS 5084

Assignment 5

Divide and Conquer

Text

Description automatically generated

func median(v1, v2, n)

* If n == 1

Return min(array1(v1 + k), array2(v2 + k))

* K <- ceiling( n / 2 )

// This means that our median must be in array1[v1, …, n] and array2[0, …, v2]

* If array1(v1 + k) < array2(v2 + k)

Return median(k, v1 + floor( n/2 ), v2)

// Othereise the median must be in array1[0, … , v1] and array2[v2, …, n]

* Else

Return (k, v1, v2 + floor( n/2 ))

func getMedianValue(array1, array2)

* n = (array1 length + array 2 length) / 2
* V1 = 0
* V2 = 0
* Return median(v1, v2, n)

Text

Description automatically generated

SignificantInversion(array) {

If | array | = 1

Return {array, 0}

arrayFirst : first half of the array

arrayLast : last half of the array

a1, inverse1 = SignificantInverse(arrayFirst) // Returns a sorted arrayFirst as a1 and number of significant inversions of inverse1

a2, inverse2 = SignificantInverse(arrayLast) // Returns a sorted arrayLast as a2 and number of significant inversions of inverse2

i = 1

j = 1

split inverse = 0

array2 = new empty array

// The cardinality of a1 is |a1| and the cardinality of a2 is |a2|

While (I <= |a1| and j <= |a2|)

If (a1[i] > 2 \* a2[j])

split inverse = (split inverse + |a1| - i + 1)

If (a1[i] > a2[j])

array2 = append(a2[j])

j = j + 1

else

array\_2 = append(a1[i])

i = i + 1

if (i <= |a1|)

array2 = append(a1)

if(j <= |a2|)

array2 = append(a2)

significant inversion = inverse1 + inverse2 + split inverse

return {array2, sign\_inverson)

Text

Description automatically generated

Text, letter

Description automatically generated

EquivalenceTest(‘cards’) // set of cards

x =|cards|

If (x == 1)

Return one card

If (x == 2)

// If the first & second card are the same then return either card.

If (card 1 == card 2)

Return card 1

set\_1 = the first half of cards

set\_2 = the second half of cards

If EquivalenceTest(set\_1)

// If cards are equal in set\_1 then we return a card from the set

Return card from set\_1

// Now we compare the card that was returned with the set of ‘cards’

// If the cards are equal in set\_2 are equivalent

If EquivalenceTest(set\_2)

Return a card from set\_2

// Now we compare the returned with the set of cards from all ‘cards’

// Return that card if the count of the equivalent card is more than half of the cards in the set ‘cards’

Text

Description automatically generated

Consider a local\_minimum(T, v) where v is either the root, or Xv is less than the value of the parent v.

If v is a leaf or Xv is less than the valuers of the children of v

Then return v

Else

Let y be a child of v such that Xy < Xv

local\_minimum(T, y)

// end of if statement

// function end

Text

Description automatically generated

My approach:

* First, we must probe the nodes found in the middle
* Check if any nodes found in the middle are a local minimum
* If we are to find any middle nodes that are local minimum, stop the algorithm
* Elsewise, among the middle nodes try and find the minimum value node m
* Given that the local minimum is not m, this means a node X exists, either to the right or to the left of m’s such that vx < vm
* Let vx be in G\_0 which is half of the grid
* G\_0 is a (n \* n / 2) grid
* Once again, we apply the probing procedure to G\_0 half grid
* Using this divide and conquer algorithm, we only use 3n/2 probes