Nathaniel Daniel

Nicolescu

CS 477

21 September 2020

HW3

1.

a. This algorithm searches an array for duplicates.

b. $\Sigma_{i = 0}^{n - 2} \Sigma_{j = i + 1}^{n - 1} 1$

$\Sigma_{i = 0}^{n - 2}[(n - 1) - (i + 1) + 1]$

$\Sigma_{i = 0}^{n - 2}(n - 1 - i)$

$\Sigma_{i = 0}^{n - 2} n - \Sigma_{i = 0}^{n - 2} 1 - \Sigma_{i = 0}^{n - 2} i$

$n(n-1) - (n - 1) - [(n - 2)(n - 1) / 2]$

$n^2 - n - n + 1 - (n^2 - 3n + 2)/2$

$n^2 - 2n + 1 - 0.5n^2 - 1.5n + 1$

$0.5n^2 - 0.5n + 2$

$O(n^2)$

2.

a. See problem2.cpp.

Output:

Min: 1

Max: 9

The algorithm works by dividing the array in half repeatedly until each subdivision has less than 3 elements, like so:

1 4 9 3 4 9 5 6 9 3 7

1 4 9 3 4   9 5 6 9 3 7

1 4   9 3 4   9 5 6   9 3 7

1 4   9   3 4   9   5 6   9   3 7

Once each array has less than 2 elements, it checks to see if the first or last element are greater than max or less than min. If so, it updates the min/max values.

b. The output will be the same whether or not there are multiple of the same min/max-values in the array. Using the above array, multiple 9's are in the array. The largest value is updated to be 9 when the computer evaluates the second array subdivision. After, when the other array subdivisions that contain 9 are evaluated, the max var is not updated as 9 is not a greater value than 9.

c.

T(n) = if(n == 1) => 3

if(n == 2) => 5

n => T(floor(n / 2)) + T(ceil(n / 2))

$T(n) = 2T(n/2) + 2$

$T(n) = 2(2T(n/4) + 2) + 2$

$T(n) = 4T(n/4) + 4 + 2$

$T(n) = 4(2T(n/8) + 2) + 4 + 2$

$T(n) = 8T(n/8) + 8 + 4 + 2$

$T(n) = 2^iT(n/2^i) + 2^i + \ldots + 2$

Let $n = 2^{k+1}$.

$T(n) = (2^{k+1}/2)T(2^{k+1}/2^k) + 2^k + \ldots + 2$

$2^k + \ldots + 2 = 2(1-2^k)/(1 - 2) = 2(2^k - 1) = 2(n/2 - 1)$

$T(n) = (2^{k+1}/2)T(2^{k+1}/2^k) + 2(n/2 - 1)$

$T(n) = (n/2)T(2) + 2(n/2 - 1)$

$T(n) = (n/2)(5) + 2(n/2 - 1)$

$T(n) = 5n/2 + n - 2$

$T(n) = 7n/2 - 2$

$O(n)$

3. See problem3.cpp.

   Output:

   Original: 1 4 9 3 4 9 5 6 9 3 7 2

   Sorted: 1 2 3 3 4 4 5 6 7 9 9 9

4. N/A

5.

   a. $\Sigma_{i=0}^{n-2} \Sigma_{j = i+1}^{n - 1} \Sigma_{k = i}^{n} 3$

   $3 \Sigma_{i=0}^{n-2} \Sigma_{j = i+1}^{n - 1}(n - i + 1)$

   $3 \Sigma_{i=0}^{n-2} \Sigma_{j = i+1}^{n - 1}(n - i + 1)$

   $3 \Sigma_{i=0}^{n-2}[(n-1) - (i + 1) + 1](n - i + 1)$

   $3 \Sigma_{i=0}^{n-2} (n - 1 - i)(n - i + 1)$

   $= 3 * [(n + 1)(n - 1) + n(n - 2) + (n - 1)(n - 3)+ \ldots + (1)(3)]$

   $= 3 \Sigma_{j=0}^{n-1}(j + 2)j$

   $= 3 \Sigma_{j=0}^{n-1} j^2 + 2j$

$= 3 \Sigma_{j=0}^{n-1} j^2 + 6 \Sigma_{j=0}^{n-1} j$

$= 3 * ([(n - 1)n(2n-1) / 6] + [2(n-1)n / 2])$

$= 3n(n -1)(2n+5)/6$

$\approx n^3$

$O(n^3)$

b. A[j,i]/A[i,i] is repeatedly calculated in the inner loop. Moving it outside of the loop in a temporary variable is more efficient.