CSC 323 Algorithm Design and Analysis, Spring 2020
Instructor: Dr. Natarajan Meghanathan

Assignment 5: Comparison of two Recursive Algorithms to Determine the Minimum Element in an Array

Due by: Feb. 27th, 11.59 PM (in Canvas)

In Module 2, we saw a "Divide and Conquer" recursive algorithm to determine the maximum element in an array. This algorithm could be simply modified to determine the minimum element in an array. For more details about this approach, refer to the slides for Module 2.

We could also determine the minimum element in an array in a recursive fashion using a "Decrease by One" approach, as follows:
Min(A, 0 ... n-1) = Min(A[n-1], Min(A, 0 ... n-2)) for n > 1
Min(A, 0 ... n-1) = A[0] for n = 1
That is, the minimum among elements from index 0 to n-1 is the minimum of the element at index n-1 and the minimum among elements from index 0 to n-2. This recursion will terminate when we are to find the minimum among elements from index 0 to 0, which is A[0] itself.

The working of the "Decrease by One" approach is illustrated below with an example.

Let the array A = {34, 56, 2, 90, 8, 2, 34} be of seven (n = 7) elements, ranging from index 0 to 6.

Min(A, 0 ... 6) = Min(A[6], Min(A, 0 ... 5))
Min(A, 0 ... 5) = Min(A[5], Min(A, 0 ... 4))
Min(A, 0 ... 4) = Min(A[4], Min(A, 0 ... 3))
Min(A, 0 ... 3) = Min(A[3], Min(A, 0 ... 2))
Min(A, 0 ... 2) = Min(A[2], Min(A, 0 ... 1))
Min(A, 0 ... 1) = Min(A[1], Min(A, 0 ... 0))
Min(A, 0 ... 0) = A[0] = 34

Now, back substituting:
Min(A, 0 ... 1) = Min(A[1], Min(A, 0 ... 0)) = Min(56, 34) = 34
Min(A, 0 ... 2) = Min(A[2], Min(A, 0 ... 1)) = Min(2, 34) = 2
Min(A, 0 ... 3) = Min(A[3], Min(A, 0 ... 2)) = Min(90, 2) = 2
Min(A, 0 ... 4) = Min(A[4], Min(A, 0 ... 3)) = Min(8, 2) = 2
Min(A, 0 ... 5) = Min(A[5], Min(A, 0 ... 4)) = Min(2, 2) = 2
Min(A, 0 ... 6) = Min(A[6], Min(A, 0 ... 5)) = Min(34, 2) = 2

You are given a startup C++ code with the main function. You are required to implement the two recursive algorithms (call the function to implement the *divide and conquer* algorithm as: DivideByHalf and the function to implement the *decrease by one* algorithm as: DecreaseByOne). The way the code will be executed is it will ask the user for the array size, max value for an element in the array and an option (1 or 2). The code will run for 1000 trials and compute the average time of one of the two algorithm implementations depending on the value entered for option.

You will run the code for array size values of 1000, 10000 and 100000 for max value of 50000 in each case for each of the two options 1 and 2. The timers are setup to determine the running time in nano

seconds. For each trial, the array (of a specific array size) will be filled with random integers from 1 to 50000.

Tabulate the results for the running time of the two recursive algorithms for the above three values of array size and analyze the results. You should compare/explain the performance of the two algorithms and interpret the reasons for the difference or similarity in their performance, depending on the case.

Also, come up with a recurrence relation for the number of times the basic operation is executed in the Decrease by One algorithm, and solve the recurrence relation to evaluate the theoretical time complexity. Show all the work.

Submission:

(1 - 65 pts) Submit the .cpp file featuring the code for the implementations of the two recursive algorithms

(2 - 35 pts) Submit a single PDF file containing the following:

(a - 15 pts) Pseudo code for the "Decrease By One" recursive algorithm, write the recurrence relation for its basic operation and analyze its time complexity

(b - 5 pts) Justify the space-complexity of the "Decrease by One" algorithm is $\Theta(1)$, i.e., it is in-place.

(c - 7 pts) Tabulate the average run-time results for the two recursive implementations with respect to the different array sizes

(e - 8 pts) Compare/explain the performance of the two algorithms and interpret the reasons for the difference or similarity in their performance, depending on the case.