CSC 6013 – Spring 1 2023

Coding Assignment for Brute Force Algorithms:

Selection Sort, Bubble Sort, and Polynomial Evaluation

Submit your work as a docx or pdf file through Blackboard.

1) a) Write Python code for the selection sort algorithm to sort an array into ascending order, but modify the code in the class notes to do three things:

i) After k iterations through the outer loop, the k **LARGEST** elements should be sorted rather than the k **SMALLEST** elements.

ii) On each iteration through the outer loop, count the number of times two array elements are compared and the number of times two array elements are swapped. Reinitialize these counters to zero at the beginning of each iteration.

iii) After each iteration through the outer loop, print three things: the partially sorted array, the number of comparisons on this iteration, and the number of swaps on this iteration. After the kth iteration, you should see that the k largest elements have been placed into the last k slots of the array.

b) Check your algorithm on the problem instances:

A1 = [63, 44, 17, 77, 20, 6, 99, 84, 52, 39]

A2 = [84, 52, 39, 6, 20, 17, 77, 99, 63, 44]

A3 = [99, 84, 77, 63, 52, 44, 39, 20, 17, 6]

2) a) Write Python code for the bubble sort algorithm to sort an array into ascending order, but with the possibility of an early exit if there are no swaps on some iteration through the outer loop. Modify the code in the class notes to do four things:

i) On each iteration through the outer loop, count the number of times two array elements are compared and the number of times two array elements are swapped. Reinitialize these counters to zero at the beginning of each iteration.

ii) After each iteration through the outer loop, print three things: the partially sorted array, the number of comparisons on this iteration, and the number of swaps on this iteration. After the kth iteration, you should see that at least the k largest elements have "bubbled up" into the last k slots of the array.

iii) If there are no swaps on some iteration through the outer loop, the array is now sorted, so terminate the algorithm.

iv) When the algorithm concludes (after n-1 iterations or after an early exit), display the total number of comparisons of array elements and the total number of swaps required to sort the array.

b) Check your algorithm on the problem instances:

A4 = [44, 63, 77, 17, 20, 99, 84, 6, 39, 52]

A5 = [52, 84, 6, 39, 20, 77, 17, 99, 44, 63]

A6 = [6, 17, 20, 39, 44, 52, 63, 77, 84, 99]

3) Polynomial evaluation

A polynomial can be represented as an array in which the coefficient of the k'th power of the variable is stored in array entry A[k]. For example, the array whose values are:

k	0	1	2	3	4	5	6
A[k]	12.3	40.7	-9.1	7.7	6.4	0	8.9

would represent the polynomial

$$f(x) = 12.3 + 40.7x - 9.1x^2 + 7.7x^3 + 6.4x^4 + 8.9x^6.$$

To evaluate this function at x = 5.4 we would "plug in" 5.4 for each occurrence of the variable x and perform the indicated arithmetic.

a) Calculating the p'th power of a real number

Write a Python function

power (x, p)

to calculate the value of x^p for any real number x and any non-negative integer p.
The brute-force function should use a for loop to repeatedly multiply the value of x.
Do not use Python's exponentiation operator ** to evaluate x**p.

b) Evaluating a polynomial

Write a Python function

evaluate (A, x)

that determines the value of f(x) for the polynomial that is represented by the corresponding array A of coefficients. For each term of the polynomial, this function should make a call to the power() function that you wrote in part3a.

c) Run your code to evaluate the polynomial

$$f(x) = 12.3 + 40.7x - 9.1x^2 + 7.7x^3 + 6.4x^4 + 8.9x^6$$

at x = 5.4

d) Asymptotic analysis

Determine the maximum number of multiplications that are needed for any polynomial of degree n (not just for this instance with a polynomial of degree 6). For your initial work, you can identify a sum of terms. Be sure to include the work (number of multiplications) performed on each call to the power() function. For your final answer, give a simple expression in terms of a power of n (there should be no terms involving p and no summations Σ). What is the Big-Oh class for this algorithm?