James Zafiri

04/15/23

CSC6013

**Module 05: Programming Assignment 05**

**Problem 1:**

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.

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]

Problem 1 Code

def Swap(A, i, j):

temp = A[i]

A[i] = A[j]

A[j] = temp

def SelectionSort(A):

for i in range(len(A)-1):

# counts per iteration

compare = 0

swap = 0

iterate = 0

# loop setting item to compare going backwards on list

for j in range(len(A)-i-1, 0, -1):

m = j

# loop to compare the items

for k in range(i, j):

compare += 1

if A[k] > A[m]:

m = k

# making sure item is not the same from prior loop

if m != j:

swap += 1

Swap(A, j, m)

iterate += 1

print(f"After iteration {iterate}: Array is {A}, with {compare} comparisons, and {swap} swaps.")

compare = 0

swap = 0

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]

SelectionSort(A1)

SelectionSort(A2)

SelectionSort(A3)

Problem 1 Output

After iteration 1: Array is [63, 44, 17, 77, 20, 6, 39, 84, 52, 99], with 9 comparisons, and 1 swaps.

After iteration 2: Array is [63, 44, 17, 77, 20, 6, 39, 52, 84, 99], with 8 comparisons, and 1 swaps.

After iteration 3: Array is [63, 44, 17, 52, 20, 6, 39, 77, 84, 99], with 7 comparisons, and 1 swaps.

After iteration 4: Array is [39, 44, 17, 52, 20, 6, 63, 77, 84, 99], with 6 comparisons, and 1 swaps.

After iteration 5: Array is [39, 44, 17, 6, 20, 52, 63, 77, 84, 99], with 5 comparisons, and 1 swaps.

After iteration 6: Array is [39, 20, 17, 6, 44, 52, 63, 77, 84, 99], with 4 comparisons, and 1 swaps.

After iteration 7: Array is [6, 20, 17, 39, 44, 52, 63, 77, 84, 99], with 3 comparisons, and 1 swaps.

After iteration 8: Array is [6, 17, 20, 39, 44, 52, 63, 77, 84, 99], with 2 comparisons, and 1 swaps.

After iteration 1: Array is [84, 52, 39, 6, 20, 17, 77, 44, 63, 99], with 9 comparisons, and 1 swaps.

After iteration 2: Array is [63, 52, 39, 6, 20, 17, 77, 44, 84, 99], with 8 comparisons, and 1 swaps.

After iteration 3: Array is [63, 52, 39, 6, 20, 17, 44, 77, 84, 99], with 7 comparisons, and 1 swaps.

After iteration 4: Array is [44, 52, 39, 6, 20, 17, 63, 77, 84, 99], with 6 comparisons, and 1 swaps.

After iteration 5: Array is [44, 17, 39, 6, 20, 52, 63, 77, 84, 99], with 5 comparisons, and 1 swaps.

After iteration 6: Array is [20, 17, 39, 6, 44, 52, 63, 77, 84, 99], with 4 comparisons, and 1 swaps.

After iteration 7: Array is [20, 17, 6, 39, 44, 52, 63, 77, 84, 99], with 3 comparisons, and 1 swaps.

After iteration 8: Array is [6, 17, 20, 39, 44, 52, 63, 77, 84, 99], with 2 comparisons, and 1 swaps.

After iteration 1: Array is [6, 84, 77, 63, 52, 44, 39, 20, 17, 99], with 9 comparisons, and 1 swaps.

After iteration 2: Array is [6, 17, 77, 63, 52, 44, 39, 20, 84, 99], with 8 comparisons, and 1 swaps.

After iteration 3: Array is [6, 17, 20, 63, 52, 44, 39, 77, 84, 99], with 7 comparisons, and 1 swaps.

After iteration 4: Array is [6, 17, 20, 39, 52, 44, 63, 77, 84, 99], with 6 comparisons, and 1 swaps.

After iteration 5: Array is [6, 17, 20, 39, 44, 52, 63, 77, 84, 99], with 5 comparisons, and 1 swaps.

**Problem 2:**

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.

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]

Problem 2 Code

def Swap(A, i, j):

temp = A[i]

A[i] = A[j]

A[j] = temp

def BubbleSort(A):

# to keep total count outside the loops

total\_compare = 0

total\_swap = 0

for i in range(len(A)-1):

# counts per iteration

compare = 0

swap = 0

# to see if another iteration is needed

swapped = False

for j in range(len(A)-i-1):

compare += 1

total\_compare += 1

# sees if next element is smaller

if A[j+1] < A[j]:

swap += 1

total\_swap += 1

Swap(A, j+1, j)

# set to True after each swap

swapped = True

print(f"After iteration {i}: Array is {A}, with {compare} comparisons and {swap} swaps.")

# if a swap did not occur on last iteration, break out of loop early

if not swapped:

break

print(f"\nTotal comparisons = {total\_compare} and total swaps = {total\_swap}")

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]

BubbleSort(A4)

BubbleSort(A5)

BubbleSort(A6)

Problem 2 Output

After iteration 0: Array is [44, 63, 17, 20, 77, 84, 6, 39, 52, 99], with 9 comparisons and 6 swaps.

After iteration 1: Array is [44, 17, 20, 63, 77, 6, 39, 52, 84, 99], with 8 comparisons and 5 swaps.

After iteration 2: Array is [17, 20, 44, 63, 6, 39, 52, 77, 84, 99], with 7 comparisons and 5 swaps.

After iteration 3: Array is [17, 20, 44, 6, 39, 52, 63, 77, 84, 99], with 6 comparisons and 3 swaps.

After iteration 4: Array is [17, 20, 6, 39, 44, 52, 63, 77, 84, 99], with 5 comparisons and 2 swaps.

After iteration 5: Array is [17, 6, 20, 39, 44, 52, 63, 77, 84, 99], with 4 comparisons and 1 swaps.

After iteration 6: Array is [6, 17, 20, 39, 44, 52, 63, 77, 84, 99], with 3 comparisons and 1 swaps.

After iteration 7: Array is [6, 17, 20, 39, 44, 52, 63, 77, 84, 99], with 2 comparisons and 0 swaps.

Total comparisons = 44 and total swaps = 23

After iteration 0: Array is [52, 6, 39, 20, 77, 17, 84, 44, 63, 99], with 9 comparisons and 7 swaps.

After iteration 1: Array is [6, 39, 20, 52, 17, 77, 44, 63, 84, 99], with 8 comparisons and 6 swaps.

After iteration 2: Array is [6, 20, 39, 17, 52, 44, 63, 77, 84, 99], with 7 comparisons and 4 swaps.

After iteration 3: Array is [6, 20, 17, 39, 44, 52, 63, 77, 84, 99], with 6 comparisons and 2 swaps.

After iteration 4: Array is [6, 17, 20, 39, 44, 52, 63, 77, 84, 99], with 5 comparisons and 1 swaps.

After iteration 5: Array is [6, 17, 20, 39, 44, 52, 63, 77, 84, 99], with 4 comparisons and 0 swaps.

Total comparisons = 39 and total swaps = 20

After iteration 0: Array is [6, 17, 20, 39, 44, 52, 63, 77, 84, 99], with 9 comparisons and 0 swaps.

Total comparisons = 9 and total swaps = 0

**Problem 3:**

**Calculating the p’th power of a real number**

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

at x = 5.4

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

**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?

Problem 3 Code

def power(x, p):

# to handle if the power is a zero

if p == 0:

return 1

num = x

# loops through each number to the specified power

for i in range(1, p):

# keeps multiplying itself by that number then returns when done

num \*= x

return num

def evaluate(A, x):

n = 0

# loops through each item of the array, returns added total when done

for i in range(len(A)):

# uses index of item to call power function using that value

n += A[i] \* power(x, i)

return n

A = [12.3, 40.7, -9.1, 7.7, 6.4, 0, 8.9]

x = 5.4

print(evaluate(A, x))

Problem 3 Output

227295.86317440012

Asymptotic Analysis

For Power:

If p == 0 🡪 1 : c1

return 1 🡪 1 : c2

num = x 🡪 1 : c3

for i in range(1, p) 🡪 n : c4

num \*= x 🡪 n : c5

return n 🡪 1 : c6

T(n) = c1(1) + c2(1) + c3(1) + c4(n) + c5(n) + c6(1)

= (c1 + c2 + c3 + c6)1 + (c4 + c5)n

= c7 + c8(n)

= c9(n)

O(n) <= c9(n)

For Evaluate:

n = 0 🡪 1 : c1

for i in range(len(A)) 🡪 n : c2

n += A[i] \* power(x, i) 🡪 n\*(n - 1)/2 : c3

T(n) = c1(1) + c2(n) + c3(n\*(n-1)/2)

= c1(1) + c2(n) + c3((1/2 n^2) – (1/2n))

= c1 + (c2 – c3)n + c4(n^2)

= c5(n^2)

O(n) <= c5(n^2)

**This algorithm is in the Big-Oh class O(n^2)**