**HOMEWORK 2: ALGORITHM EFFICIENCY**

**A – Theory part:**

**A1.** Using Big O notation, indicate the time requirement of each of the following tasks in the worst case. Describe any assumptions that you make.

a. After arriving at a party, you shake hands with each person there.

- Time complexity: O(n)

- Assumption: The time complexity assumes that 'n' represents the number of people at the party, and shaking hands with each person takes constant time.

b. Each person in a room shakes hands with everyone else in the room.

- Time complexity: O(n^2)

- Assumption: The time complexity assumes that 'n' represents the number of people in the room. As each person needs to shake hands with every other person, the time required is proportional to the square of the number of people.

c. You climb a flight of stairs.

- Time complexity: O(n)

- Assumption: The time complexity assumes that 'n' represents the number of stairs in the flight. Climbing one step takes constant time, and the time required is directly proportional to the number of stairs.

d. After entering an elevator, you press a button to choose a floor.

- Time complexity: O(1)

- Assumption: The time complexity assumes that pressing a button to choose a floor takes constant time, regardless of the number of floors in the building.

e. You ride the elevator from the ground floor up to the nth floor.

- Time complexity: O(1)

- Assumption: The time complexity assumes that riding the elevator from the ground floor to the nth floor takes constant time, regardless of the number of floors in the building.

f. You read a book twice.

- Time complexity: O(n)

- Assumption: The time complexity assumes that 'n' represents the number of pages in the book. Reading the book twice requires going through each page twice, which takes linear time proportional to the number of pages.

**A2.** List the following growth-rate functions in order of growth:

A picture containing text, screenshot, font, white

Description automatically generated

🡪 1 – log2(n) – n – nlog2(n) – – – – – n!

**A3.** Describe the running time of the following pseudocode in Big-O notation in terms of the variable n. Assume all variables used have been declared.

A picture containing text, font, line, number

Description automatically generated

“int i = 0” 🡪 1

“... i < k; ++i) 🡪 n (with k = n)

“cost = cost + (i\*k)” 🡪 1 (for loop runs k times, with k = n, so this one takes n)

“answ = foo(n)” 🡪 1

🡪 f(n) = 1 + n + n + 1 = 2 + 2\*n

🡪 O()

A picture containing text, font, line, number

Description automatically generated

“int i = 0” 🡪 1

“... i < n; ++i)” 🡪 n

“if ... 🡪 1000 + (2 +) ( for loop runs n times, so this one take n\*(1000 + (2 +))

else ...”

🡪 f(n) = 1 + n\*(1000 + (2 +)) = 1 + 1002\*n +.

🡪 O()

A picture containing text, font, line, number

Description automatically generated

“int i = 0” 🡪 1

“...; i < n + 100; ++i) 🡪 n + 100

“int j = 0” 🡪 1 ( the first for loop runs n + 100 times, so this one takes n)

“; j < i\*n; ++j)” 🡪 [(n+101)\*(n+100)/2] \* n ( the first for loop runs n + 100 times, so this one takes (n+100)\* [(n+101)\*(n+100)/2] \* n)

“sum = sum + j” 🡪 [(n+101)\*(n+100)/2] \* n ( the first for loop runs n + 100 times, so this one takes (n+100)\* [(n+101)\*(n+100)/2] \* n)

“int k = 0” 🡪 1 ( the first for loop runs n + 100 times, so this one takes n)

“... k < n + n + n; ++k)” 🡪 3n ( the first for loop runs n + 100 times, so this one takes (n+100)\* 3n)

“c[k] = c[k] + sum” 🡪 1 ( the first for loop runs n + 100 times, so this one takes (n+100)\* 3n)

🡪 f(n) = 1 + n + 100 + n + 2\*(n+100)\* [(n+101)\*(n+100)/2] \* n + n + 2\*(n+100)\* 3n

🡪 O()

A picture containing text, font, line, screenshot

Description automatically generated

“int j = 4” 🡪 1

“ j < n; j = j + 2” 🡪 (n-4)/2

“val = 0” 🡪 1 ( for loop runs (n-4)/2 times, so this one take (n-4)/2 )

“int i = 0” 🡪 1 ( for loop runs (n-4)/2 times, so this one takes (n-4)/2)

“...; i < j; ++i)” 🡪 3+ 4 + 5 + ... (n-4)/2 = (n-4)\*(n-2)/4 - 3

“val = val + i \* j” 🡪 1 ( the first for loop runs (n-4)/2 times and the second for loop runs ((n-4)\*(n-2) - 12)/4 times, so this one takes ((n-4)\*(n-2) - 12)\*(n-4)/8)

“int k = 0” 🡪 1 ( the first for loop runs (n-4)/2 times and the second for loop runs ((n-4)\*(n-2) - 12)/4 times, so this one takes ((n-4)\*(n-2) - 12)\*(n-4)/8)

“...; k < n; ++k)” 🡪 n ( the first for loop runs (n-4)/2 times and the second for loop runs ((n-4)\*(n-2) - 12)/4 times, so this one takes n\*((n-4)\*(n-2) - 12)\*(n-4)/8)

“val++” 🡪 n ( the first for loop runs (n-4)/2 times and the second for loop runs ((n-4)\*(n-2) - 12)/4 times, so this one takes n\*((n-4)\*(n-2) - 12)\*(n-4)/8)

🡪 f(n) = 1 + 4\*(n-4)/2 + (n-4)\*(n-2)/4 – 3 + n\*((n-4)\*(n-2) - 12)\*(n-4)/4

🡪 O()

A picture containing text, font, line, white

Description automatically generated

“int i = 0” 🡪 1

“...; i < n \* 1000; ++i)” 🡪 1000n

“sum = (sum \* sum)/(n \* i)” 🡪 1 ( for loop runs 1000n times, so this one takes 1000n )

“int j = 0” 🡪 ( the first for loop runs 1000n times, so this one takes 1000n )

“...; j < i; ++j)” 🡪 1 + 2 + ... 999n = (999n + 1) \* 999n / 2 ( the first for loop runs 1000n times, so this one takes 1000n \* (999n + 1) \* 999n / 2)

“sum += j \* i” 🡪 1 ( the first for loop runs 1000n times, so this one takes 1000n \* (999n + 1) \* 999n / 2)

🡪 f(n) = 1 + 1000n\*3 + 1000n \* (999n + 1) \* 999n

🡪 O()

**A4.** Assume that each of the following expressions has a running time of T(n) and the input size is n. Specify the highest-order operand in the expression and the corresponding Big-O.

a. 5 + 0.001 + 0.025n

- The highest-order operand is .

- The corresponding Big O notation: O().

b. 500n + 100+ 50nlog10(n)

- The highest-order operand is .

- The corresponding Big O notation: O().

c. 100n + 0.01

- The highest-order operand is .

- The corresponding Big O notation: O().

d. 2n + + 0.5

- The highest-order operand is .

- The corresponding Big O notation is O().

e. 0.3n + 5+ 2.5

- The highest-order operand is .

- The corresponding Big O notation is O().

f. 0.01n + 100

- The highest-order operand is .

- The corresponding Big O notation is O().

g. nlog3(n) + nlog2(n)

- The highest-order operand is nlog2(n).

- The corresponding Big O notation is O(nlog2(n)).

h. 0.01log2(n) +

- The highest-order operand is log2(n).

- The corresponding Big O notation is O(log2(n)).

i. 2log2(n) + 2log5(n)

- The highest-order operand is log2(n).

- The corresponding Big O notation is O(log2(n)).

**A5.**

**A picture containing text, screenshot, font, algebra

Description automatically generated**

- The complete recurrence relation for the running time of the mystery function can be defined as follows:

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

Where T(n - 2) represents the running time for the recursive call with n decremented by 2, T(n / 2) represents the running time for the recursive call with n divided by 2, and O(1) represents the constant time for the if-else condition and other operations within the function.

- The base case is defined as:

T(0) = O(1)

This base case signifies that when the input n reaches 0 or becomes negative, the running time is constant and does not depend on the input size.

**A6.** For each of the following statements, identify whether it is correct or not.

a. f(n) = is O()

f(n) = = . In Big O notation, we drop the constant factor and consider the dominant term => O(). So this statement is correct.

b. f(n) = is O()

f(n) = = . But << ( is very small compared to ) 🡪 O(). So this statement is incorrect.

**B. Coding part:**

**B1.**

A screenshot of a computer program

Description automatically generated with medium confidence

- The SUMPOWER function has a time complexity of O(n^2) because it calls the POWER function, which has a time complexity of O(n), within a loop that iterates n times.

- The SUMPOWER2 function has a time complexity of O(n) because it uses a loop that iterates n times and performs constant-time operations (multiplication and addition) in each iteration.

- Therefore, SUMPOWER2 is expected to run faster than SUMPOWER for larger values of n because it has a more efficient time complexity.