1 Analyze Best, Average, and Worst Case runtimes for the following search algorithm.

Function search(array of size n, target):

for each element in array:

if element == target:

return "Element found"

return "Element not found"

1a) Best Case: What is the best-case scenario for this linear search algorithm? Explain the situation where it would perform most efficiently.

Answer –

The best case scenario runtime for is O(1), This would happen when the target is in the first index in the array.

1b) Average Case: How does the average-case scenario consider different possible positions of the target element within the array? Provide an explanation of how the algorithm performs on average.

Answer –

The average case scenario runtime is Θ (n/2). This can be seen by imagining the searched array as different distances away from a location, if these distances are 0, 1, 2, 3, 4, 5, and 6 units away the average distance away is 3 units. Now if the distances stretch to “n” units away the average distance is n/2 units away.

For example, if you are given the array [1,4,6,2,7,5], and you are searching for 2 it will take 4 checks in the array to find 2 since it is t the third index. But if you are looking for 4 it will only take 2 checks in the array both of which are close to the average search time in this array of 3.

1c) Worst Case: What is the worst-case scenario for this linear search algorithm? Describe the input arrangement that would lead to the highest time complexity.

Answer –

The worst-case scenario runtime is Ω (n+1). This only happens when the target that is being searched for does not exit in the array.

2 Consider a scenario where you need to find the sum of all even numbers between 1 and a given positive integer n.

2a) Write pseudocode to solve this problem. Make sure to clearly define the variables and provide step-by-step instructions for your algorithm.

Answer –

Function SumEvens(positive integer n)

Integer sum = 0

for each even integer i between 0 and n

sum = sum + i

return sum

2b) After writing the pseudocode, analyze the time complexity of each line in terms of the input size n. Provide a brief explanation for each line's time complexity analysis.

Answer –

Function SumEvens(positive integer n) Time complexity of 1

Integer sum = 0 Time complexity of 1

for each even integer i between 0 and n Time complexity of n

sum = sum + i Time complexity of 1

return sum Time complexity of 1

2c) Finally, calculate and provide the overall time complexity of your algorithm using big-O notation.

Answer – The final time complexity is the sum of the individual time complexities of each line. Final time complexity = 1+1+ n(1+1) = 2n+2

3 Sort the groups of functions into increasing order of asymptotic (big-O) complexity and explain the answer.

Group 1 unsorted

* f1(n) = O(2^n)
* f2(n) = O(n!)
* f3(n) = O(n^3)
* f4(n) = O(n^2)
* f5(n) = O(n log n)

Group 1 sorted

* f5(n) = O(n log n)
* f4(n) = O(n^2)
* f3(n) = O(n^3)
* f1(n) = O(2^n)
* f2(n) = O(n!)

Explanation - O(n log n) has the smallest time complexity because after n0 = 0 O(n^2) grows faster. O(n^2) is next because after n0 = 1 O(n^3) grows faster. O(n^3) is next because after n0 = 2 O(2^n) grows faster. And O(n!) is the largest time complexity because after n0 = 4 O(n!) grows faster then O(2^n).

Group 2 unsorted

* f1(n) = 2^2^1000000
* f2(n) = 2^100000n
* f3(n) = n choose 2
* f4(n) = n\*sqrt(n)

Group 2 sorted

* f1(n) = 2^2^1000000
* f4(n) = n\*sqrt(n)
* f3(n) = n choose 2
* f2(n) = 2^100000n

Explanation – The reason f1 is the fasted is because it is a constant running time so every equation will eventually take longer than it. f4 is next because it can be simplified to n^(3/2) this is a lower order than n^2 but higher than a constant runtime. f3 is next, the equation for n choose 2 is (n(n+1))/2 which is asymptotic with O(n^2), giving it a larger runtime than n^(3/2). And finally f2 has the largest asymptotic runtime since after n0 = 4 O(n!) grows faster than O(n^2)

4 - Proofs

a) Prove that f(n) = 3𝑛^2 + 2n + 1 is in O(𝑛^2 ). Show the constants c and n0 that satisfy the definition of Big O.

prove that the upper bound runtime is n^2

Answer –

By definition f(n) is O(n^2) if there exists some c>0, and n0 >= 0 such that 0 <f(n) <= c\*n^2 for all n >= n0. To prove this we will substitute f(n) into the expression.

0 <3𝑛^2 + 2n + 1<= c\*n^2 then divide by the highest power in f(n), 0<1+2/(3n)+1/(3n^2)<=c/3, since n is now only in the denominator as n increases this term decreases 1+2/(3n)+1/(3n^2) now choose n0 to be 1 so we get 2<=c/3 so we only need to find a c which validates this expression. That c is 6. Since there is a c that exists for which 0<f(n)<=c\*n^2 for all n>=1 this proves f(n) is O(n^2)

b) Prove that f(n) = 𝑛^3 + 4𝑛^2 + 2n is in Θ(𝑛^3 ). Show the constants c1, c2, and n0 that satisfy the definition of Theta notation.

prove the average runtime complexity is n^3

Answer –

By definition for f(n) to be Θ(𝑛^3) there must exist c1>0, c2>0, and n0>=0 such that 0<c1\*n^3<=f(n)<=c2\*n^3 for all n>=n0. Substitute f(n),

0<c1\*n^3<= 𝑛^3 + 4𝑛^2 + 2n <=c2\*n^3, divide by the highest term,

0<c1<= 1+ 4/n + 2/n^2 <=c2 choose n0 to be 1, 0<c1<= 1+ 4 + 2 <=c2, 0<c1<=7<=c2, choose c1 to be 6 and c2 to be 8. Since there exists a c1, c2 for which 0<c1\*n^3<=f(n)<=c2\*n^3 for all n>=1 this proves f(n) is Θ(𝑛^3 ).

c) Prove that f(n) = 2𝑛^2 + 7n is in Ω(𝑛^2 ). Show the constants c and n0 that satisfy the definition of Omega notation.

Prove that the lower bound time complexity is n^2

Answer –

By definition f(n) is Ω (𝑛^2) if there exists a c>0 and n0>=0 such that f(n)>=c\*n^2>=0 for all n>=n0. Now substitute f(n) into the expression. 2𝑛^2 + 7n >=c\*n^2>=0 now divide by the highest power in f(n), 1+7/2n >=c/2>=0 choose n0 to be 1, 9/2>=c/2>=0 a c that validates this is 8. Since there is a c that exists for which f(n)>=c\*n^2>=0 for all n>=1 this proves f(n) is Ω(𝑛^2)

d) Given two functions g(n) = 𝑛^2 and h(n) = 𝑛^3 , determine whether g(n) = O(h(n)). Provide a proof for your answer.

Answer –

By definition for g(n) to be O(h(n)) then there must exist some c>0 and no>=0 such that 0<g(n)<=c\*h(n). Substitute g(n) and h(n) into the expression. 0<n^2<=c\*n^3, divide by the highest power in g(n), 0<1<=c\*n. choose n0 to be 1, 0<1<=c a c that validates this is 2. Since there is a c that exists for which 0<g(n)<=h(n) for all n>=1 this proves g(n) is O(h(n)).

e) For the function f(n) = 5𝑛^2 + 3n log n + 10, determine the tightest possible bound using Big O, Omega, and Theta notations. Explain your reasoning.

Answer –

Big O(n^2)

By definition f(n) is O(n^2) if there exists c>0 and n0>=0 such that 0<f(n)<=c\*n^2 for all n>=n0

Substitute f(n), 0<5𝑛^2 + 3n log n + 10<=c\*n^2, divide by the highest power,

0<1 + (3 log n)/(5n) + 10/5n^2<=c/5, choose n0 to be 1, 0<3<=c/5, A c that validates this expression is 15. Since there is a c>0 which validates this expression for all n>=1 f(n) is O(n^2).

Big Ω(n^2)

By definition f(n) is Ω(n^2) if there exists some c>0 and n0>=0 such that 0<c\*n^2<=f(n) for all n>=n0. Substitute f(n), 0<c\*n^2<= 5𝑛^2 + 3n log n + 10, divide by the highest power,

0<c/5<= 1 + (3 log n)/(5n) + 10/5n^2, choose n0 to be 1, 0<c/5<=3 a c that validates this is c = 15

Big Θ(n^2) c1=4, c2=6

Remember, proofs for asymptotic notations often involve finding appropriate constants (c and n0) that satisfy the definitions of Big O, Theta, and Omega. Provide clear explanations and reasoning in your answers