**Question 1: Algorithm in which best-case time is equal to worst-case time:**

Ans: Bubble Sort Algorithm

Algorithm BubbleSort(A, n)

Input Array A of n elements

Output: Array A sorted in Ascending order

M<-n-1

for i <-0 to m

for j<-0 to m

if A[j] > A[j+1] then

var.swap(j, j+1)

swap(i, j)

temp <-A[i]

A[i] <-A[j]

A[j] <-temp

**Analysis**

Because there are two nested loops, both depending on n, time complexity is ϴ(n2). This is the same for both best and worst case time because the comparisons have to be made for the whole array (even when the input is already sorted).

**Question 2. Order based on their complexity.**

2^n , 2^(n + 1), 2^(2n), 2^( 2^n )

**Question 3.**

O(log n) : Binary Search,

O(n log n): Merge Sort,

O(n) : Finding maximum value in an integer array using binary search algorithm

O(1): Linear Search,

O(n^2): Selection Sort,

O(n^3): Matrix Multiplication,

O(2^n): Fibonacci

**Question 4.**  Apply Master Theorem and determine the time complexity of

fib(n) shown in slide 48.

Fib(n) = Fib(n-1)+Fib(n-2)

**Method1**

Let Fib(n) = T(n)

T(n) = T(n-1)+T(n-2)+c

a = 1, b=?

As per Master theorem b must be known and should be greater than 1, so we cannot apply Master theorem here.

Finding time complexity for Fib(n).

T(n) = T(n-1) +T(n-2) + c

T(n) = T(n-1) + T(n-1) + c i.e T(n-1) ~ T(n-2)

T(n) = 2T(n-1)+c

T(n) = 2(2T(n-2) + c ) + c

T(n) = 4T(n-2) + 3c

T(n) = 4(2T(n-3) + c ) + 3c

T(n) = 8T(n-3) + 7c

T(n) = 2^3T(n-3) + (2^3-1)c

2^k(n-k) + (2^k-1)c

Let’s find k value for which n-k = 1

2^k(1) + (2^k-1)c

2^k+(2^k-1)c

We assume this is also true for n=k

2^n+(2^n-1)c

2^n + 2^nc - c

2^n(1+c)-c

This means T(n) ~ 2^n

So the time complexity of fib(n) is O(2^n)

**Method2**

Let’s say the number of recursive calls to find Fibonacci number n is f(n)

f (2) = 2 = 2^2 -1

f (3) = 4 = 2^3-1

f (4) = 8 = 2^4 -1

f (5) = 14 ~ 16 = 2^5 -1

f (6) = 24 ~ 32 = 2^6-1

f(n) ~ 2^(n-1)

We can say the upper bound of the function f(n) is 2^(n-1)

So, the time complexity for the Fibonacci series is O(2^n).

**Question 5 Solve the recurrence**

T (1) = 1

T(n) = 2T(n/2) + c

Hint: Assume n = 2^k.

T(1) = 1

T(n) = 2T(n/2) + c

Hint: Assume n = 2^k.

T(n) = 2T(n/2) + C

n = 2^k

n = 32 = 2^5

T(32) = 2T(16) + C

2T (16) = 2^2 T(8) + 2C

2^2T(8) = 2^3 T(4) + 2^2C

2^3T(4) = 2^4 T(2) + 2^3C

2^4T(2) = 2^5 T(1) + 2^4C

T (32) = 2^5 + C {1 + 2 + …+ 2^4}

= 2^5 + C(2^5 – 1)

= n + C (n – 1)

= O(n)

**Question 5. Practice Master theorem.**

1) Find Max(A, l, h)

if (l==h) return A[l]

else return (Mathmax (FindMax(A, l, [l + h / 2]) , (FindMax (A, [l + h /2])))

a = 2, b = 2, k = 0, c = 1, b^k = 2^0 = 1

a > b^k

ϴ (n logb a)

ϴ (n log22)

ϴ (n1) = ϴ(n)

2) MergeSort(A, l, h)

If (l == h) return array with A[l]

MergeSort(A, l, [l+h/2], h)

return merge(L, R)

a = 2, b = 2, c = 1, k = 1, b^k = 2^1 = 2

a = b^k

T(n) = ϴ (n^k logn)

= ϴ (n^1 logn)

= ϴ (n logn)

3)We couldn’t find an existing algorithm that falls under the category a< b^k

So, we have assumed the following and did the analysis.

T(n) = 2T (n/4) + nc

a = 2, b = 4, k = 1, b^k = 4^1 = 4

a < b^k

T(n) = ϴ(n^k)

= ϴ(n^1)

= ϴ(n)