**Course: Algorithm  
Prof. Prem Nair  
Student: Binh Van Tran  
ID: 986648  
Homework: Lab 3**

1. **Question 1 –** Write an algorithm such that the best running time is equal to the worst case running time

of *n* integers

The *for…loop* from index *zero* to *n – 1* is always be executed no matter where the maximum value is. So, the time complexity of this algorithm is always *O(n).*

1. **Question 2 –** Order the following functions based on their complexity

2n, 22n, 2n+1,

2n is in O(2n)

22n is in O(2n.2n)

2n+1 is in O(2.2n)

is in O(

The order should be: 2n, 2n+1, 22n and

1. **Question 3** – Mention one algorithm you know for each of the time complexities listed

O(1): Find maximum or minimum of a sorted array; Get value from an array by index

O(log*n*): Search a value in sorted array using binary search

O(n): Find maximum or minimum value of an unordered array; Search a value in array

O(nlog*n*): Quick sort, Heap sort, merge sort

O(n2): Insertion Sort, Selection Sort

O(n3): Any algorithm with 3 deep nested *for…loop*

O(2n): Fibonacci, finding subset

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

* **fib(n)**

We have T(1) = d, T(n) = T(n – 1) + T(n – 2).

Because this algorithm is not a Divide-And-Conquer one, so we cannot apply the master theorem to calculate the time complexity of T(n)

* **binarySearch**

We have:

T(1) = d,

T(n) = T() + c

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

bk = 20 = 1

Hence a = bk => T(n) is in O(nklog*n*) = O(n0log*n*) = O(log*n*)

1. **Practice Master theorem**

* Case 1: a < bk

T(n) = 3T(n/2) + n2

a = 3, b = 2, k = 2

bk = 4 > a

T(n) is in O(n2)

* Case 2: a = bk

Merge sort:

T(1) = d,

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

a = 2, b = 2, k = 1; bk = 21 = 2 = a

T(n) is in O(nklog*n*) = O(nlog*n*)

* Case 3: a > bk

T(n) = 4T(n/2) + (½)n

a = 4, b = 2, k = 1, bk = 2 < a

T(n) is in O(nlog4)