

## Statement Purpose:

Purpose of this lab is to introduce the concepts of Big-O running time of the Algorithms to the students. The students are also familiarized with the time and space complexity of algorithms. They are also taught how to analyze an algorithm and calculate the worst case (Big-O) running time of the algorithm.

## Activity Outcomes:

The students will learn how to analyze small algorithms with respect to time complexity. In this process, they will analyze an algorithm and find the worst case running time of that algorithm. They will also understand, if running time of an algorithm is known for an input, how they will find the running time of the same algorithm for another input.

## Theory Review (10 Minutes):

### **Example 1: (Linear Search - Taking $O(n)$ time to run)**

Consider the following piece of code for searching an element stored in variable 'value' in an unsorted array 'a' of size 'n'.

```
public static boolean LinearSearch(int[] a, int value) {
    for (int i = 0; i < a.length ; i++) {
        if (a[i] == value)
            return true;
    }
    return false;
}
```

Since the array 'a' is unsorted, the only way to be sure that the 'value' is not in the array is to look at every single value of the array. Hence, in worst case, the algorithm will run all the way to the end of the array which is of length 'n'. So, the worst case running time of the above linear search algorithm for an unsorted array of length 'n' is  $O(n)$ .



**Example 2: (Binary Search - Taking  $O(\log n)$  time to run)**

Suppose the elements of the array 'a' are sorted in any order, for example in ascending order to be specific. Then we can use the following binary search code to find the 'value' in the array 'a' of length 'n'.

```
public static boolean BinarySearch(int[] a, int value) {
    int low = 0;
    int high = a.length - 1;
    while (low < high) {
        int mid = (low + high) / 2;
        if (a[mid] == value)
            return true;
        else if (value > a[mid])
            low = mid + 1;
        else
            high = mid - 1;
    }
    return false;
}
```

This algorithm will try to find the 'value' at the middle index of the array. If it is found, the algorithm terminates by returning 'true' otherwise, it checks where the 'value' is present in the array. If the value is present in the second half (i.e. right half) of the array, it discards the remaining first half (i.e. left half) by making the variable  $low = mid + 1$ . If the value is present in the first half (i.e. left half) of the array, it discards the remaining second half (i.e. right half) by making the variable  $high = mid - 1$ .

In this attempt, the algorithm divides the input to half each time until it is left with only one element.

Hence the worst case running time of binary search algorithm on a sorted array of length 'n' is  $O(\log_2 n)$  which is much faster than that of linear search algorithm which is  $O(n)$ .



**Example 3: (Inserting an element in sorted linked list - Taking  $O(n)$  time to run)**

Suppose we want to insert an element stored in variable 'data' into a sorted linked list. The method insert() (studied in Linked List topic) will insert an element stored in variable 'data' in a sorted linked list. In this attempt, it finds the correct position in the linked list by moving 'helpPtr' from one node to the next node using 'while' loop.

In worst case, required value will be inserted at the end of the linked list. In this case, we have to traverse the entire linked list all the way to the last node. Hence the running time of the algorithm for inserting an element at the correct location in a sorted linked list of size 'n' is  $O(n)$ .

**Practice Activity with Lab Instructor: (10 Minutes)****Example 1: (Algorithms Analysis)**

For an  $O(2^n)$  algorithm, a friend tells you that it took 17 seconds to run on her data set on a  $O(2^n)$  algorithm. You run the same program, on the same machine, and your data set with  $n = 7$  takes 68 seconds. What size was her data set?

**Solution:**

Let running time of the algorithm on an input of size 'n' be  $T(n)$ .

Then  $T(n) = C (2^n)$  for some constant C

For data set with  $n = 7$ , it gives

$$T(7) = C (2^7)$$

$$68 = C (128)$$

$$C = 68/128 = 17/32$$

Now, to find the size of data set for our friend, we again use the same formula

$$T(n) = C (2^n)$$

Putting the value of C in it, we get

$$T(n) = 17/32 (2^n)$$

$$17 = 17/32 (2^n)$$

$$2^n = 17(32/17)$$

$$2^n = 32$$



Taking  $\log_2$  on both sides, we get

$$\log_2 (2^n) = \log_2 32$$

$$\log_2 (2^n) = \log_2 (2)^5$$

$$n (\log_2 2) = 5 (\log_2 2) \quad (\text{Since } \log m^n = n \log m)$$

$$n = 5 \quad (\text{Since } \log_2 2 = 1)$$

Thus size of our friend's data set is 5.

### Example 2: (Algorithm Analysis)

Consider the following code.

```
int function5(int A[], int B[], int n) {  
    int i, j, sum = 0;  
    for (i=0; i<n; i++)  
        for (j=0; j<n; j++)  
            if (A[i] == B[j])  
                sum++;  
    return sum;  
}
```

Determine the running time for this function in terms of the variable  $n$  ( $n > 0$ ). The answers should simply be in terms of Big-O. Also justify your answer.

### Solution:

The outer *for* loop runs for  $i = 0$  to  $n-1$  i.e. a total of ' $n$ ' times. Similarly the inner *for* loop runs for  $j = 0$  to  $n-1$  i.e. a total of ' $n$ ' times. Since these are nested loops and the inner loop runs for every value of outer loop variable ' $i$ ', the worst case running time of this function is  $O(n \times n)$  i.e.  $O(n^2)$ .



**LAB EXERCISES: (60 Minutes)**

1. For an  $O(n^3)$  algorithm, one data set with  $n = 3$  takes 54 seconds. How long will it take for a data set with  $n = 5$ ?
2. For an  $O(n^k)$  algorithm, where  $k$  is a positive integer, an instance of size  $m$  takes 32 seconds to run. Suppose you run an instance of size  $2m$  and find that it takes 512 seconds to run. What is the value of  $k$ ?
3. Assume that an  $O(\log_2 n)$  algorithm runs for 10 milliseconds when the input size ( $n$ ) is 32. What input size makes the algorithm run for 14 milliseconds?
4. Determine the running time for the following function in terms of the variable  $n$  ( $n > 0$ ). The answer should simply be in terms of Big-O. Justify your answer.

```
int function6(int A[], int B[], int n) {  
    int i=0,j=0;  
    while (i < n) {  
        while (j < n && A[i] > B[j])  
            j++;  
        i++;  
    }  
    return j;  
}
```

5. Determine the running time for the following function in terms of the variable  $n$  ( $n > 0$ ). The answer should simply be in terms of Big-O. Justify your answer.

```
int function9(int n) {  
    int i, j;  
    for (i=0; i<n; i++)  
        for (j=0; j<n; j++)  
            if (j == 1)  
                break;  
    return j;  
}
```



6. Determine the running time for the following function in terms of the variable  $n$  ( $n > 0$ ). The answer should simply be in terms of Big-O. Justify your answer.

```
void function8(int n) {  
    while (n > 0) {  
        System.out.printf("%d\n", n);  
        n = n/2;  
    }  
}
```

