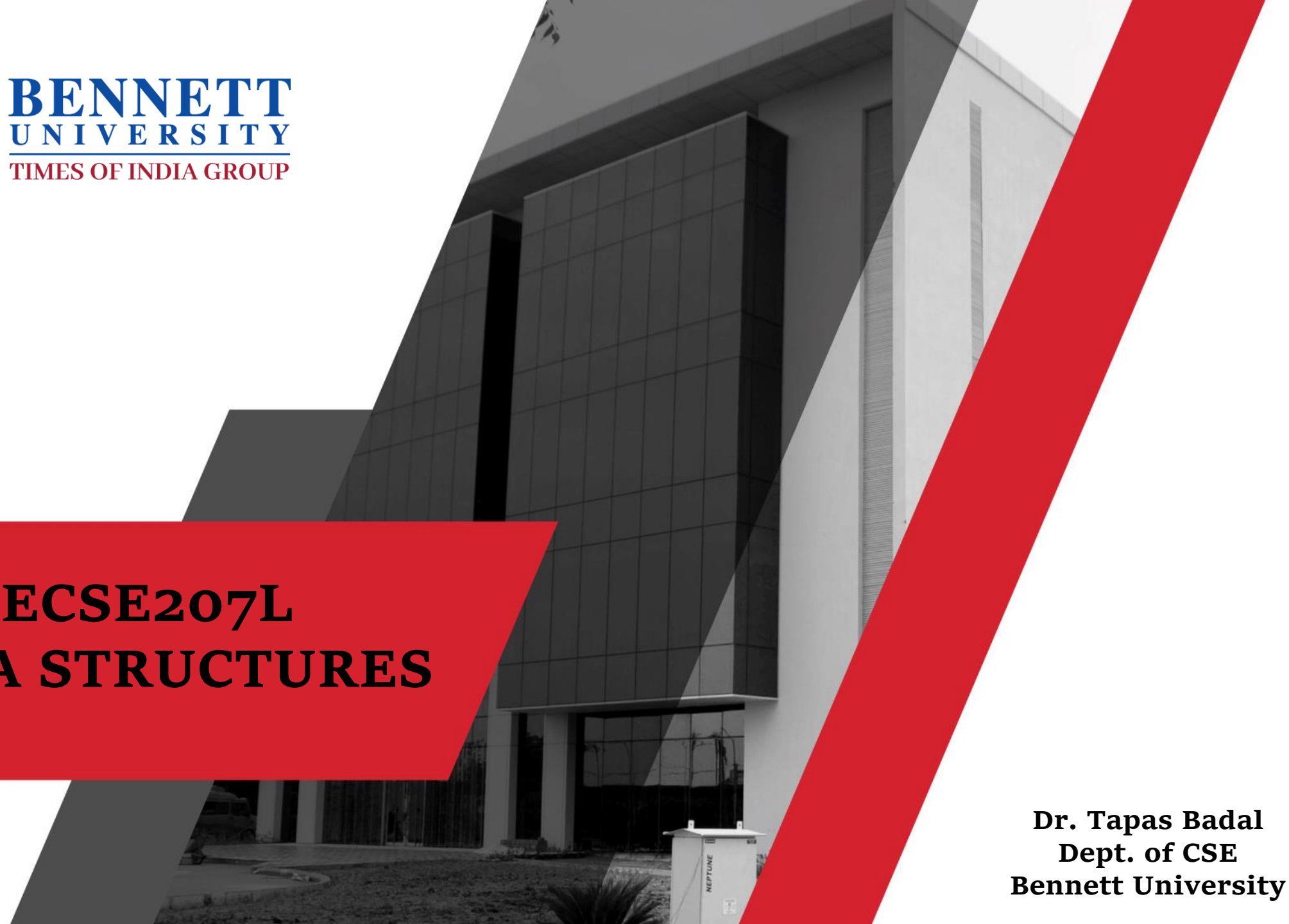
BENNETT UNIVERSITY
TIMES OF INDIA GROUP



ECSE207L
DATA STRUCTURES

Dr. Tapas Badal
Dept. of CSE
Bennett University

Tutorial 2

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

1. **What are differences between performance and complexity?**

Performance of a program or an algorithm is measured using various factors – like Time, Space, and Network, etc.

Time - How much time the program takes to execute?

Space - How much memory or space the program is using while it's executed (also the physical space)

Network - If there are any network related activities the program is doing, which may affect the performance).

Complexity is used to measure performance. Or **Complexity is a measure of how the Resources (in this case Time) changes as the size of the problem gets larger**."

Performance: how much time/memory/disk/... is actually used when a program is run. This depends on the machine, compiler, etc. as well as the code.

Complexity: how do the resource requirements of a program or algorithm scale, i.e., what happens as the size of the problem being solved gets larger.

Complexity affects performance but not the other way around.

O(1)

O(1) represents an algorithm that takes the same amount of time to execute regardless of the number of inputs. So, 1 item takes 1 second, 10 items take 1 second, 100 items take 1 second and so on. Therefore, performance is not affected by the size of the input data.

O(N)

O(N) represents an algorithm where the size of the input data impacts the execution time. The performance of the algorithm is directly proportional to the number of inputs. So, 1 item takes 1 second, 10 items take 10 seconds, 100 items take 100 seconds and so on.

O(log N)

O(log N) represents an algorithm where the number of computations grows linearly as input data grows exponentially. So 1 item takes 1 second, 10 items take 2 seconds, 100 items take 3 seconds and so on.

O(2^N)

O(2^N) represents an algorithm where execution time is doubled for each additional input. So, 1 item takes 2 seconds, 2 items take 4 seconds, 3 items take 8 seconds and so on.

O(N!)

O(N!) represents a factorial algorithm that must perform N! calculations. So, 1 item takes 1 second, 2 items take 2 seconds, 3 items take 6 seconds and so on. An example of this algorithm is one that recursively calculates Fibonacci numbers.

O(N2)

O(N2) represents an algorithm that is directly proportional to the square of the sizes of the inputs and must performs N2calculations (by definition). Bubblesort is a good example of this algorithm.

O(N log N)

(N log N) represents an algorithm that will in increase in execution time proportionate to the number of the input times the logarithm of the number of the input. Mergesort and quicksort are good examples of this algorithm.

1. Find the time complexity of given expression?
   a. 3n^2 + 5n + 6
   b. 10n + 2 nlogn + 4logn
   c. 6logn + n
   d. 3nlogn + 2n
   e. 8log n + 4 loglogn
   f. 3n + 2(logn)^2 + 4logn

O(1)

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

```
void printFirstElementOfArray(int arr[])
{
printf("First element of array = %d",arr[0]);
}
```

O(n)

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

```
void printAllElementOfArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }
}
```

$O(n^2)$

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

```
void printAllPossibleOrderedPairs(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%d = %d\n", arr[i], arr[j]);
        }
     }
}
```

$O(2^n)$

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

```
int f (int num)
{
    if (num <= 1) return num;
    return f(num - 2) + f(num - 1);
}
```

```
void printAllItemsTwice(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }

    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }
}
```

This is O(2n), which we just call O(n).

```
void printFirstItemThenFirstHalfThenSayHi100Times(int arr[], int size)
{
   printf("First element of array = %d\n",arr[0]);

   for (int i = 0; i < size/2; i++)
   {
      printf("%d\n", arr[i]);
   }

   for (int i = 0; i < 100; i++)
   {
      printf("Hi\n");
   }
}
```

This is $O(1 + n/2 + 100)$, which we just call $O(n)$.

```
void printAllNumbersThenAllPairSums(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%d\n", arr[i] + arr[j]);
        }
    }
}
```

Here our runtime is $O(n + n^2)$, which we just call $O(n^2)$.

O(n^3 + 50n^2 + 10000) is O(n3)

O((n + 30) * (n + 5)) is O(n2)

```
bool arrayContainsElement(int arr[], int size, int element)
{
    for (int i = 0; i < size; i++)
    {
        if (arr[i] == element) return true;
    }
    return false;
}
```

Here we might have 100 items in our array, but the first item might be the that element, in this case we would return in just 1 iteration of our loop. In general we'd say this is O(n) runtime and the "worst case" part would be implied. But to be more specific we could say this is worst case O(n) and best case O(1) runtime. For some algorithms we can also make rigorous statements about the "average case" runtime.

```
int sum =0;
        for(int i = 0; i < n; i++)
                {
                        for(int j = 0; j <  n; j++)
                        {
                                sum++;
                        }
                }
```

```
int sum =0;
for(int i = 0; i < n; i+=2)
        {
                for(int j = 0; j <  n; j+=10)
                {
                        sum++;
                }
        }
```

```
int sum =0;
for(int i = 0; i < n; i++)
        {
                for(int j = 0; j <=  i; j++)
                {
                        sum++;
                }
        }
```

```
int sum =0;
        for(int i = 0; i < n; i++)
        {
                for(int j = 0; j <  m; j++)
                {
                        sum++;
                }
        }
```

```
int sum =0;
        for(int i = 1; i < n; i+=2)
        {
                for(int j = 1; j <  n; j*=2)
                {
                        sum++;
                }
        }
        for(int k = n; k >= 1; k--)
        {
                sum++;
        }
```

```
int sum =0;
        for(int i = 1; i <= n; i++)
        {
                for(int j = 1; j <=  i; j++)
                {
                        for(int k = 1; k <= 133; k++)
                        {
                        Sum++;
                        }
                }
        }
```

(n + 1)^3 is O(n^3)

f(n) = n Σ i=1 i is O(n^2).

Write formula for computing memory address in a 3D tensor.

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

THANKYOU

@csebennett

cse_bennett