Usman Institute of Technology

Affiliated with NED University of Engineering and Technology

Running Time of an Algorithm

1. Calculate running time of the following code:

```
for i in range(10)
                print("Hello");
```

Solution:

```
1. for i in range(10)                    10 + 1
2.                print("Hello");        1 * 10
```

The line #1 will run 10 times where + 1 is one extra time for checking the last condition. The cost of line #2 is 1 but it is part of a loop, so its cost will be multiplied to the cost of the loop. The total running time is $T(n) = 11 + 10$.

2. Calculate running time of the following code:

```
for i in range(n)
                print("Hello");
```

Solution:

```
1. for i in range(n)                     n + 1
2.                print("Hello");        1 * n
```

The line #1 will run n times where + 1 is one extra time for checking the last condition. The cost of line #2 is 1 but it is part of a loop, so its cost will be multiplied to the cost of the loop. The total running time is $T(n) = (n + 1) + (1 * n)$.

3. Calculate running time of the following code:

```
for i in range(n)
        for j in range(10)
                print("Hello");
```

Solution:

```
1. for i in range(n)                    n + 1
2.       for j in range(10)             (10 + 1) * n
3.               print("Hello");        1 * 10 * n
```

The line #1 will run n times where + 1 is one extra time for checking the last condition. The cost of line #2 is 10 but it is part of a loop, so its cost will be multiplied to the cost of the loop (see line #1). The line #3 is part of two loops, therefore, its cost will be multiplied to cost of both loops. The total running time will be in form of $T(n) = a \cdot n + b$, where a and b are constants.

4. Calculate running time of the following code:

```
for i in range(n)
        for j in range(n)
                print("Hello");
```

Solution:

```
for i in range(n)                       n + 1
        for j in range(n)               n * (n+1)
                print("Hello")          1 * (n*n)

T(n) =                                  2(n*n) + 2n + 1
```

Regardless of the coefficients, the equation must be in the form of $an^2 + bn + c$ where a, b and c are constants.

5. Calculate running time of the following code:

```
1       for i in range(n):              n + 1
2               x = 1                   1 * n
3               while x<=n:             (t + 1) * n
4                       x = x * 2       1 * t * n
5                       print("Hello");  1 * t * n
```

We assume that total time taken by line #3 is 't'. In order to find the T(n) we have to solve this for t. If we observe the value of x, then it will be:

$$1, 2, 4, 8, \ldots, n$$
$$2^0, 2^1, 2^2, 2^3, \ldots, 2^k$$

If we can find k, then we can conclude that this loop will run k times (or precisely k+1 times). We know, $2^x = N$, therefore:

$$2^k = N$$

Taking $\log_2$ both sides,

$$\lg 2^k = \lg N$$
$$k \lg 2 = \lg N$$
$$k = \lg N$$

We can use this information to find T(n):

```
1        for i in range(n):                    n + 1
2                x = 1                         1 * n
3                while x<=n:                   (lgN + 1) * n
4                        x = x * 2             1 * lgN * n
5                        print("Hello");       1 * lgN * n
```

$$T(n) = (n+1) + (1*n) + (\lg N + 1) * n + (1 * \lg N * n) + (1 * \lg N * n)$$

You can solve this for the rest. It must be in the form of $T(n) = a \cdot n \lg N + b \cdot n + c$, where a, b, and c are constants.

6. Calculate running time of following code:

```
for(i=1; i<= n; i=i*2)
        for(j=1; j<=i; j++)
                print("Hello");
```

Solution: For outer loop, we know the i increases in the power of 2. The inner loop is dependent on outer loop and in order to calculate we have to sum the total number of iterations taken by inner loop.

```
for(i=1; i<= n; i=i*2) - 1, 2,4,8,16 .... x
        for(j=1; j<=i; j++) - 1 + 2 + 4 + 8 + 16 + .... + x
                print("Hello");
```

First, we need to identify the limit for the outer loop as the same will be used for inner loop as well. We know:

$$2^i \leq n$$

and

$$i = \log n$$

In order to calculate the total steps taken by inner loop:

$$= 1 + 2 + 4 + 8 + \ldots + \log n$$
$$= 2^0 + 2^1 + 2^2 + \ldots + 2^{\log n}$$

We can solve this by using geometric series:

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$$

Hence,

$$1 + 2 + 4 + \ldots + \log n = \sum_{k=0}^{\log n} 2^k = \frac{2^{\log n + 1} - 1}{2 - 1} = 2^{\log n + 1}$$

We know $2^{\log n+1} = (n+1)^{\log 2} = (n+1)$ (using logarithmic)

Now,

```
for(i=1; i<= n; i=i*2) - log(n)
        for(j=1; j<=i; j++) - (n+1)
                print("Hello");
```

and, $T(n) = \log n + n + 1$

7. Devise an algorithm that finds the sum of all the integers in a list.
   (a) Give pseudo-code of the algorithm.
   (b) Calculate running time ($T(n)$) of the algorithm.

Solution:

```
sum=0, i=0
while (i <= n)
        sum = sum + i
        i = i+1
```

There is only one loop so $T(n)$ should be look like $T(n) = n + c$ where c is a constant.

8. Find Running Time for Binary Search algorithm (iterative solution).

Solution:

```
1. BinarySearch (data, e)
2.        l = 0                                    1
3.        r = len(data) -1                         1
4.        while l<r:                               t
5.                mid = (l + r) / 2                1 * t
6.                if data[mid] == e:               1 * t
7.                        return mid               1 * t
8.                if e < data[mid]:                1 * t
9.                        r = mid - 1:             1 * t
10.               else:
11.                       l = mid + 1
12.       return -1                                1
```

We know, the $T(n)$ would be in form of $T(n) = a \cdot t + b$, where a and b are constants. So, we have to find the t. In order to find the t, we have to understand that how many time this *while* loop will run.

To understand this, assume that initial value of l and r are 0 and n, respectively. We are finding the mid point by dividing 2, so next value of l and r could be 0 and n/2 or n/2+1 and n, respectively. In any of these cases, the data from which we have to find the element is reduce to $n/2$. In next iteration, the data will be reduced to half again i.e. $\frac{n/2}{2} = \frac{n}{4}$. This will continue until we reach to just one element. This can be describes as the following:

$$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, 1 \tag{1}$$

OR

$$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, \frac{n}{n} \tag{2}$$

OR

$$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, \frac{n}{2^x} \tag{3}$$

If we can find x, then we can determine how many times the while loop will run. From Equation 2 and Equation 3, we know:

$$2^x = n$$

By taking $\log_2$ both sides,

$$x = \lg n$$