**COMSATS Institute of Information Technology, Islamabad Campus**

**Department of Computer Science**

**Design and Analysis of Algorithms – CSC301**

**BCS – IVA**



**Quiz #1 Marks: 20**

**Mapped to CLO1**

**Q1: a) Describe tight bound using time complexity of any standard algorithm.**

**Tight Bound Big Theta:**

**Definition:**

Tight bound refers to big theta i.e., the notation which represents the average value or range within the actual time of execution of an algorithm.

**Explanation:**

Tightly bound means a function is bounded above and below by some functions in a range.

Statistically we say that f(n)= Θ(g(n)) when there exists constants **c1 and c2 that c1.g(n)<=f(n)<=c2.g(n)** for all sufficiently large value of n. where n is a positive integer and g is a tight bound for function f.

**Example:**

The best-case running time of insertion sort is **a\*n + b.**

Here the theta bound i.e., **g(n)= n, a\*g(n)<=f(n)<=c\*g(n)** for all large values of **n.**

Hence the best case of complexity can be represented as Θ(n)

**b) Provide an example of a program whose time complexity will be Ω (n2).**

**for i <— 1 to length[A]**

** some code**

**for j <— 1 to length[A]**

** some code**

**for k <— 1 to length[A]**

**some code**

**c) Find the time complexities of following codes.**

|  |  |  |
| --- | --- | --- |
| **statements** | **Cost** | **Times** |
| *int i=0;* | **C1** | **1** |
| *int j=0;* | **C2** | **1** |
| *int k=0;* | **C3** | **1** |
| *for (i = n / 2; i <= n; i++) {* | **C4** | **n/2+1** |
| *for (j = 2; j <= n; j = j \* 2) {* | **C5** | **Log n/2** |
| *k = k + n / 2;*  *}}* | **C6** | **Log n/2-1** |

|  |
| --- |
| Total Time complexity = O **(n log n)** |

|  |  |  |
| --- | --- | --- |
| **statements** | **Cost** | **Times** |
| *int i=0;* | **C1** | **1** |
| *int j=0;* | **C2** | **1** |
| *for (i = 0; i < N; i++) {* | **C3** | **n** |
| *a = a + rand();}* | **C4** | **n-1** |
| *for (j = 0; j < M; j++) {* | **C5** | **n** |
| *b = b + rand();}* | **C6** | **n-1** |

|  |
| --- |
| Total Time complexity =O **(n)** |

**d) Fill out the table below to provide a comparison between insertion and selection sort.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Space complexity** | **Best case time complexity (TC)** | **Average case TC** | **Worst case TC** | **Number of comparisons** | **Number of data swaps** |
| **Insertion Sort** | O **(1)** | O **(n)** | O **(n^2)** | O **(n^2)** | O **(n^2)** | O **(n^2)** |
| **Selection Sort** | O **(1)** | O **(n^2)** | O **(n^2)** | O **(n^2)** | O **(n^2)** | O **(n)** |

**e) Explain time complexities of inner loops of insertion and selection sort.**

* **Insertion Sort:**

for j <— 2 to length[A]

. (some code)

i=j —1 (Statement a)

while i > 0 and A[i]> key

(some code)

**Time Complexity:**

**Reason:**

While loop is an inner loop used for comparisons, the variable i used in this loop is directly dependent on the value of outer loop variable j (Statement a), so the loop will run as many times as j.

* **Selection Sort:**

for j=1 to n-1

(some code)

for i=j+1 to n

**Time Complexity:**

**Reason:**

The inner loop in selection sort is used to find the minimum in array, it will run (n-j+1) times and as it depends on outer for loop variable. It will run one time extra in order to check the condition hence the condition becomes false and it will come out of the inner loop.