# DESIGN AND ANALYSIS OF ALGORITHMS – CSCI 612

**NAME:** AADITHYA GOPALAKRISHNA BHARADWAJ

**ZID:** Z1862641

**ASSIGNEMENT NUMBER, SEMESTER:** Assignment 2, SPRING 2019

==========

**1.) Express the function n^3/1000 – 100n^2 – 100n + 3 in terms of Θ–notation.**

**Answer.)** n^3/1000 - 100n^2 - 100n + 3 is **Θ (n^3).**

**Explanation:**

In order to express the function in terms of Θ notation, identify the term, which is having the highest power and ignore the coefficients of the term. So, the highest term in this equation is n^3/1000, ignoring the constants we get n^3.

==========

**2.) Express the function 25n^2 + 100nlgn + 20n – 5 in terms of Θ–notation.**

**Answer.)** 25n^2 + 100nlgn + 20n – 5 is **Θ (n^2).**

**Explanation:**

In order to express the function in terms of Θ notation, identify the term, which is having the highest power and ignore the coefficients of the term. So, the highest term in this equation is 25n^2, ignoring the constants we get n^2. Because the order growth is:

Constant < logn < n < nlogn < n^2 < n^3 < $2^n$

==========

**3.) Some functions are listed in the following table. Assume an algorithm to solve problem takes f(n) microseconds. Time t is a parameter of time interval. For each function f(n) and time t, determine the largest size n of a problem that can be solved in time t. Fill the values in the table. For example, an algorithm to solve a problem takes $f(n)=n^2$ microseconds, then the largest size nmax of the problem that can be solved in 1 minute will have:**

- ➔ **$n^2 \leq 60 \times 1000000$ (60 seconds per minute, 10^6 microseconds per second)**
- ➔ **$n \leq 7745.97$ -> nmax = 7745**

**Answer.)**

| Time t / *f(n)* | 1 Second | 1 Minute | 1 Hour |
|---|---|---|---|
| **lg(n)** | $2^{10\text{^}6}$ | $2^{6*10\text{^}7}$ | $2^{36*10\text{^}8}$ |
| **n** | $10^6$ | $6*10^7$ | $36*10^8$ |
| **$n^2$** | 1000 | 7745 | 60000 |
| **$n^3$** | 100 | 391 | 1532 |
| **$2^n$** | 19 | 25 | 31 |
| **n!** | 9 | 11 | 12 |

**Findings:**

From the table we can say that time complexity varies with the length of the input. As the length of the input increases so does the time. So, the length of input is directly proportional to that of time complexity. Hence, we can conclude from the table that logarithmic value of 'n' takes the most time. Hence, we can safely say that: $O(\lg(n)) > O(n) > O(n^2) > O(n^3) > O(2^n) > O(n!)$

==========

**4.) Given a sorted array of integers A and another integer x, describe an efficient algorithm to determine whether or not there exist two elements in A whose sum is exactly x. Analyze your algorithm regarding the best case and worst case.**

**Answer.)**

As per question since the array is already sorted i.e. ($a_1 < a_2 < a_3 < \ldots\ldots < a_n$), we can use the following algorithm to identify if there are two elements in Array A whose sum is exactly x:

**Input:** list S sorted in ascending order, x

**Output:** true if there exist two elements in S whose sum is exactly x, false otherwise

*SUM - S (A,x)*

1.) i ← 1, j ← n

2.) while i <= j do

3.) if S[i] + S[j] = x then
return true

4.) if S[i] +S[ j] < x then

i ← i+1

else

j ← j −1

5.) return false

**Explanation:** For 1<= u < v <=n, call (u, v) an x pair if $a_u + a_v = x$. Here the number of x pairs can be many. The initial value of i = 1 and that of j = n. Sum S contains a pair of (i, j) and either i increases or j decreases until it finds an x pair or i = j.

**The worst-case performance is *O*(log n) and it compares n times (so we can say to determine x the worst case is $\Theta(n)$ ) and the best-case performance is Ω(1).**