CSCI 532 – Algorithm Design
Assignment 0

Name: Badarika Namineni CWID: 50233279

Question 1: Come up with a real-world problem in which only the best solution will do. Then come up with one in which a solution that is "approximately" the best is good enough.

Solution:

The best solution is Biometrics used by Government. For example, US Consulate collects Biometrics for the authenticate the person for dispensing Visa.

An approximate solution for finding a flight fare from source to destination should be good. Even if you don't find the best fare, you would still reach the desired destination.

Question 2: Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size, insertion sort runs in 8n2 steps, while merge sort runs in 64nlg(n) steps. For which values of n does insertion sort beat merge sort?

Solution:

Insertion sort running time is less than Merge sort running time.
So, 8n2 < 64nlog(n)
To evaluate the expression, we expand the above expression in below way:

8*n*n < 8*8*n*log(n)

By canceling 8 & n on both sides, we eventually get the below expression

n < 8*log(n)

Now, lets start substituting n with real numbers.

N = 1	1 < 8*log1	= 1 < 0		= False
N = 2	2 < 8*log2	= 2 < 8*1	= 2 < 8	= True
N = 4	4 < 8*log4	= 4 < 8*2	= 4 < 16	= True
N = 10	10 < 8*log10	=10 < 8*3.3	=10 < 26.4	= True
N = 16	16 < 8*log16	=16 < 8*4	=16 < 32	= True
N = 20	20 < 8*log20	=20 < 8*4.3	=20 < 34.4	= True
N = 24	24 < 8*log24	=24 < 8*4.5	=24 < 26	= True
N = 32	32 < 8*log32	=32 < 8*5	=32 < 40	= True
N = 42	42 < 8*log42	=42 < 8*5.3	=42 < 42.4	= True
N = 43	43 < 8*log43	=43 < 8*5.4	=43 < 43.2	= True
N = 44	44 < 8*log44	=44 < 8*5.4	=44 < 43.2	= False

After N = 43, Merge sort beats Insertion sort.
The range that insertion beats merge sorts is **2<=n<=43.**

Question 3: Using Figure 2.2 as a model, illustrate the operation of Insertion-Sort on the array A = (31, 41, 59, 26, 41, 58).

Solution:

According to the figure, the given array has to be sorted in an ascending order.

Step 1: Compare 1st 2 numbers 31 and 41 in the array. 31 < 41 and there is no movement in the position of the array.

1	2	3	4	5	6
31	41	59	26	41	58

Step 2: Compare element 3 (59) with the previous 2 elements (31, 41) in the array. Since 31 and 41 are smaller than 59, no movement in the position of the array

1	2	3	4	5	6
31	41	59	26	41	58

Step 3: Now, compare element 4 (26) with 31, 41 and 59.

1	2	3	4	5	6
31	41	59	26	41	58

Since 26 is smaller than 31, 41, 59 swap the position and move it to position 1

1	2	3	4	5	6
26	31	41	59	41	58

Step 4: Compare element 5 (41) with the 26, 31, 41, 59.

1	2	3	4	5	6
26	31	41	59	41	58

Swap 41 and 59 and move the element to the position 4

1	2	3	4	5	6
26	31	41	41	59	58

Step 5: Compare 58 with the previous elements and swap if it is smaller.

1	2	3	4	5	6
26	31	41	41	59	58

Since, 58 is smaller than 59 and greater than 41, swap both the elements and re-arrange to get the final sorted array.

1	2	3	4	5	6
26	31	41	41	58	59

Question 4: Express the function n3/1000 − 100n2 − 100n + 3 in terms of Θ-notation.

Solution:

We need to consider the fastest growth rate in the function for the Θ-notation. N3 is fastest here, we write it as **Θ(n3).**

Question 5: Explain why the statement, “The running time of algorithm A is at least O(n2),” is meaningless

Solution:

Big O is used to describe the worst-case scenario. The algorithm run time f(n) should be smaller than c*g(n) which is the time of the upper bound.

F(n) <= c*g(n)   [C is constant and holds real number and g(n) is upper bound Function]

In this case, the algorithm run time will not exceed upper bound time. The statement “The running time of algorithm A is at least O(n2)” is contradicting with the proven statement.

For example, if f(n) is the run time of Algorithm A. Then f(n) >= O(n^2)
Because f(n) >= O(n^2), there's no information about upper bound of f(n)

Assume g(n) = O(n^2)
The statement becomes f(n) >= g(n), but g(n) could be anything that is "smaller" than n^2 .So, there's no conclusion about lower bound of f(n) too.

Question 6: Prove that the running time of an algorithm is Θ(g(n)) if and only if its worst-case running time is O(g(n)) and its best-case running time is Ω(g(n)).

Solution:

The worst-case running time is O(g(n)). The best-case running time is Ω(g(n))

Let’s assume the algorithm running time is f(n). `
If f(n)=Θ(g(n)), then

$0 \le c1g(n) \le f(n) \le c2g(n)$ [n≥n0, c1, c2 are constants]

As $0 \leq f(n) \leq c2g(n)$ for $n \geq n0$ and $f(n)=O(g(n))$, $f(n)$ is upper bounded by $O(n)$ which is the worst-case running time of the algorithm.

As $0 \leq c1g(n) \leq f(n)$ for $n \geq n0$ and $f(n)=\Omega(g(n))$, $f(n)$ is lower bounded by $\Omega(n)$ which is the best-case running time of the algorithm.