**2CS503 Design and Analysis of Algorithms**

**Tutorial 4: Divide and Conquer**

1. Which of the following algorithms is NOT a divide & conquer algorithm?
**(A)** Euclidean algorithm to compute the greatest common divisor
**(B)** Heap Sort
**(C)** Cooley-Tukey fast Fourier transform
**(D)** Quick Sort

2. Consider the following C code:

```
int main()
{
int x, y, m, n;
scanf ("%d %d", &x, &y);
/* x > 0 and y > 0 */
m = x; n = y;
while (m != n)
{
if(m>n)
m = m - n;
else
n = n - m;
}
printf("%d", n);
}
```

**(Try to find the complexity of the above code)**
What does the program compute?
**(A)** x + y using repeated subtraction
**(B)** x mod y using repeated subtraction
**(C)** the greatest common divisor of x and y
**(D)** the least common multiple of x and y

3. Consider the polynomial $(x)=a_0+a_1x+a_2x^2+a_3x^3$ where $a_i$ != 0, for all i. The minimum number of multiplications needed to evaluate p on an input x is:
**(A)** 3
**(B)** 4
**(C)** 6
**(D)** 9

4. Maximum Subarray Sum problem is to find the subarray with maximum sum. Given an array {12, -13, -5, 25, -20, 30, 10}, find the maximum subarray sum. The naive solution for this problem is to calculate sum of all subarrays starting with every element and return the maximum of all. We can solve this using Divide and Conquer, what will be the worst case time complexity using Divide and Conquer.
(A) O(n)
(B) O(nlogn)
(C) O(logn)
(D) O(n^2)

5. Consider a situation where you don't have function to calculate power (pow() function in C) and you need to calculate x^n where x can be any number and n is a positive integer. What can be the best possible time complexity of your power function?
**(A)** O(n)
**(B)** O(nLogn)
**(C)** O(LogLogn)
**(D)** O(Logn)

6. Let P be a QuickSort Program to sort numbers in ascending order using the first element as pivot. Let t1 and t2 be the number of comparisons made by P for the inputs {1, 2, 3, 4, 5} and {4, 1, 5, 3, 2} respectively. Which one of the following holds?
(A) t1 = 5
(B) t1 < t2
(C) t1 > t2
(D) t1 = t2

7. The usual $\Theta(n^2)$ implementation of Insertion Sort to sort an array uses linear search to identify the position where an element is to be inserted into the already sorted part of the array. If, instead, we use binary search to identify the position, the worst case running time will
(A) remain $\Theta(n^2)$
(B) become $\Theta(n (\log n)^2)$
(C) become $\Theta(n \log n)$
(D) become $\Theta(n)$

8. Randomized quicksort is an extension of quicksort where the pivot is chosen randomly. What is the worst case complexity of sorting n numbers using randomized quicksort?
(A) O(n)
(B) O(n Log n)
(C) $O(n^2)$
(D) O(n!)

9. Write an Algorithm for Binary Search. Analyse it for Best case, Worst case and Average Case with suitable example using divide and conquer algorithm.

10. Write and Algorithm for Quick sort. Analyse it using divide and conquer for all cases Best case, Worst case and Average case.

11. Write an algorithm for Merge Sort and Analyse it using divide and conquer.

12. Can we improve the time complexity of multiplying large integers using Divide and Conquer? Prove your answer with suitable example of multiplying 981 and 1234. (Hint: Karatsuba Algorithm)

13. You have an array of n elements. Suppose you implement quicksort by always choosing the central element of the array as the pivot. Then the tightest upper bound for the worst case performance is _____.