Question 1 [10 Marks]

Given a string (array of characters, Str[1…n]), complete the recursive function given below **(in pseudocode form)** that returns the reversed string. *Note that you are NOT required to print the reversed string and instead you are required to reverse the string in the memory*. Also, write the asymptotic time complexity of your algorithm. For example, the string "stony" is required to be converted to "ynots" in the memory. Note that this is just an example and your solution should work for a general case Str[1…n].

```
reverse(str, 1, n)   {1st Recursive function call; 1 is the 1st index, n is the last
index, i.e., n is the size of the string}

reverse(str, first, last)
BEGIN
      IF first ≥ last
            Return
      ELSE
            Swap(str[first], str[last])
            reverse(str, first+1, last-1)
      END IF
END
```

Asymptotic Time Complexity:
$T(n) = T(n-2) + 1$
$\Theta(n)$

Question 2 [5x3=15 Marks]

Provide a worst-case asymptotic time complexity of the following algorithms/ codes by using a suitable asymptotic notation considering a nearest function. Assume that there are no errors/ bugs in the algorithms/ codes. Also, provide the justification for your answer in the last column.

	Algorithm/ code	**Time Complexity**	**Justification**

a)	`for (int i = 1; i<=n*n; i++)`
	`{`
	`      for (int j = i; j <n; j++)`
	`           cout << j % n;`
	`}`
		$\Theta(n^2)$	Outer loop executes $n \times n = n^2$ times.
			Inner loop executes i times, maximum n times only for 1st n iterations of the outer loop.

b)	`ch = 'a';`
	`// Assume that a user enters the`
	`// character 'z' after k attempts`
	`while ( ch != 'z' )`
	`{`
	`      for (int i =n; i >=1; i=i/2)`
	`      {`
	`            cout << ch;`
	`      }`
	`      cin >> ch;`
	`}`
		$\Theta(k\log n)$	While loop executes maximum of k times
			For loop runs from n down to 1 with a decreasing multiplicative factor of 2. So, it executes logn times.

c)	`// Assume that the number of nodes`
	`// in the binary tree is n`
	`void printInorder(struct Node* node)`
		$\Theta(n)$	In inorder traversal, we traverse each node exactly one

	`{`
	`    if (node == NULL)`
	`        return;`
	`    printInorder(node->left);`
	`    cout << node->data << "  ";`
	`    printInorder(node->right);`
	`}`
			time.

d)	`for (int i=1; i<N; i++)`
	`{`
	`      int a = 0, b = 0;`
	`      for (k = 0; k < N; k++)`
	`            a = a + rand();`
	`      for (j = 0; j < M; j++)`
	`            b = b + rand();`
	`}`
		$\Theta(N\times(N+M))$	Outer for loop executes N times.
			Inner for loops execute N+M times.

e)	`Power(a,k)  {calculate a^k,k≥0}`
	`BEGIN`
	`    IF k = 0 THEN`
	`         Return 1`
	`    ELSE`
	`         Return a × Power(a, k - 1)`
	`    END IF`
	`END`
		$\Theta(k)$	$T(n) = T(n-1) + 1$
			leads to linear complexity.

Question 3 [4+6+5=15 Marks]

Solve the following recurrence using the three methods discussed in the class.

$T(n) = 4T(n/2) + n^2$

a) Can the Master theorem (for solving recurrence) be applied to the above recurrence relation: Why or why not? If yes, then write the time complexity for this recurrence using Master Theorem.

$a = 4$, $b = 2$, $f(n)=n^2$

Master theorem is applicable as it follows the general form required for the application of Master Theorem.

$n^{\log_b a} = n^{\log_2 4} = n^2 = f(n)$, $k = 0$, so the 2nd case of Master Theorem applies.

So $T(n)=\Theta(n^2 \lg n)$

b) Solve the recurrence relation using iteration method. Show all steps.

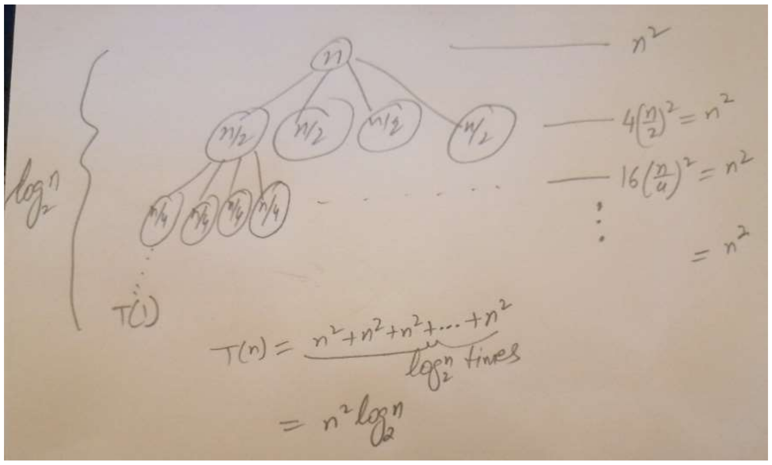
$$\begin{aligned}
T(n) &= 4T(n/2) + n^2 \\
&= 4(4T(n/2^2) + (n/2)^2) + n^2 \\
&= 4^2T(n/2^2) + (n/2)^2) + n^2 + n^2 \\
&= 4^2(4T(n/2^3) + (n/4)^2) + 2n^2 \\
&= 4^3T(n/2^3) + n^2 + n^2 + n^2 \\
&= 4^3T(n/2^3) + 3n^2
\end{aligned}$$

… after k steps, we have

$$\begin{aligned}
T(n) &= 4^kT(n/2^k) + kn^2 \\
&= n^2T(1) + n^2\log_2 n \qquad \text{because } 2^k=n
\end{aligned}$$

$= n^2\log_2 n$ because $T(1)=1$

c) Solve the recurrence using recursion tree method.



n
n/2
n/4
T(1)
$\log_2^n$
n^2
$4(\frac{n}{2})^2 = n^2$
$16(\frac{n}{4})^2 = n^2$
$= n^2$
$T(n) = \underbrace{n^2+n^2+n^2+\dots+n^2}_{\log_2^n \text{ times}}$
$= n^2\log_2^n$

Question 4 [10 Marks]

Given a sorted array A of n *distinct* integers, some of which may be negative, give an efficient algorithm (in pseudocode form) to find an index i such that $1 \leq i \leq n$ and A[i]=i provided such an index exists. If there are many such indices, the algorithm can return any one of them. Your algorithm must run in O(lg n) time. There will be no credit for providing a worse solution (e.g., linear or worse). Note that all distinct elements mean that no element in the data is repeated.

Suppose that we have this Array A={-1,0,1,2,4,11,13,15,20,25}

Index	1	2	3	4	5	6	7	8	9	10
Element	-1	0	1	2	4	11	13	15	20	25

We are given that the array is already sorted. It means that the element at index i-1 is less than the element at index i. Similarly, the element at index i+1 is greater than the element at index i. So, this is a binary search problem.

```
Index-Search(A, b, e)
BEGIN
        IF (b > e)
                Return -1     {if no answer, return -1}
        END IF
        m = floor(e + b) / 2
        IF A[m] = m
                Return m
        END IF
        IF A[m] > m
                Return Index-Search(A,b,m-1)
        ELSE
                Return Index-Search(A,m+1,e)
        END IF
END
```
Time complexity: same as that of binary search, O(lg n).