Assignment No. 2.

1. Print elements of an array (forward and backward)

Ans. int [] arr = new arr[5];
for (int i = 0; i < arr.length ; i++) {
    s.o.p(arr[i]);
}
// forward direction

for (int j = arr.length - 1 ; j >= 0 ; j--) {
    s.o.p(arr[j]);
}
// backward direction

e.g. Ex(1) arr = [1, 2, 3, 4] // length = 4

① i = 0, 0 < 4 → arr[0] = 1
② i = 1, 1 < 4 → arr[1] = 2
③ i = 2, 2 < 4 → arr[2] = 3
④ i = 3, 3 < 4 → arr[3] = 4
⑤ i = 4, 4 < 4 → break;

① j = 4-1 = 3, 3 >= 0 → arr[3] = 4
② j = 4-2 = 2, 2 >= 0 → arr[2] = 3
③ j = 4-3 = 1, 1 >= 0 → arr[1] = 2
④ j = 4-4 = 0, 0 >= 0 → arr[0] = 1
⑤ j = 4-5 = 0, -1 >= 0 → break;

* // Using recursion

```
main () {
    fun(arr, 0);
}
void fun(int[] arr, int index) { if (index < arr.length) return;
    s.o.p(arr[index]);
    fun(arr, index + 1);
}
```

e.g. arr = [1, 2, 3, 4]

fun(arr, 0) → arr[0] = 1
↳ fun(arr, 1) → arr[1] = 2
↳ fun(arr, 2) → arr[2] = 3
↳ fun(arr, 3) → arr[3] = 4
↳ fun(arr, 4) → break;

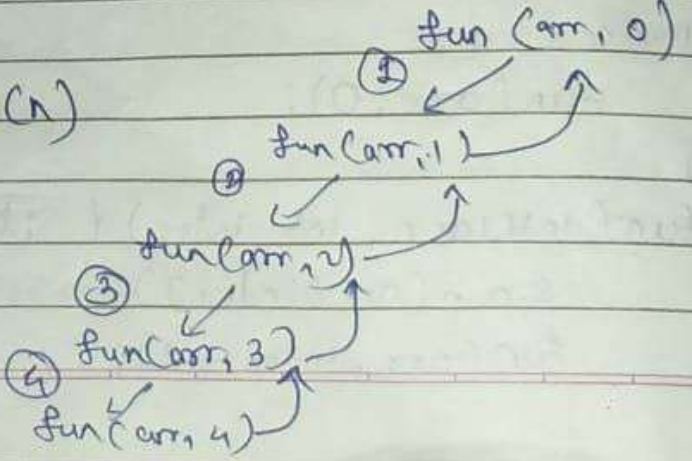
* // elements in reverse

```
main() {
    fun(arr, arr.length-1);
}
void fun(arr, index) {
    if(index < 0) return;
    System.out.print(arr[index]);
    fun(arr, index-1);
}
```

e.g. arr = [1, 2, 3, 4]

fun(arr, 3) → arr[3] = 4
↳ fun(arr, 2) → arr[2] = 3
↳ fun(arr, 1) → arr[1] = 2
↳ fun(arr, 0) → arr[0] = 1
↳ fun(arr, -1) → ~~arr[-1]~~ → break;

Time Complexity = O(n)



2. Find the sum of digits of a number.

```
main(){
    s.o.p(sum(num));
}
int sum(num){  if(num<=0) return 0;
    num = num/10;   rem = num%10;
    return rem + sum(num/10);
}
```

* eg. N = 1234

① sum(1234) → num ≠ 0 → 1234 % 10 = 4 + sum(123)

② sum(123) → num ≠ 0 → 123 % 10 + sum(12) = 3 + sum(12)

③ sum(12) → num ≠ 0 → 12 % 10 + sum(1) = 9 + sum(1)

④ sum(1) → num ≠ 0 → 1 % 10 + sum(0) = 10 + sum(0)

⑤ sum(0) → num <= 0 → return;

s.o.p(sum) = 10

Time complexity = O(log n)

3. Find the product of digit of a number.

→

```
main(){
    so.p(product(num));
}
product(int num){
    if (product <= 1) return 1;
    int rem = num % 10;
    return (rem * product (num / 10));
}
```

eg. N = 231

① product(231)
↳ 231 > 1
↳ rem = 231 % 10 = 1 → (1) rem * product(23);

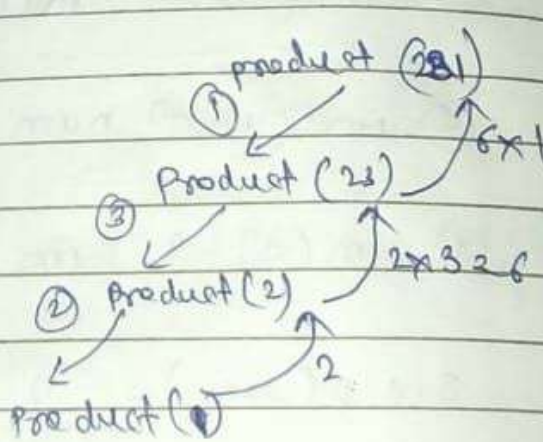
② product(23)
↳ 23 > 1
↳ rem = 23 % 10 = 3 → (1+3) rem * product(2);

③ product(2)
↳ 2 > 1
↳ rem = 2 % 10 = 2 → (1+3+2) rem * product(0);

④ product(0)
↳ 0 <= 1
↳ return 1;

∴ product = 1 * 3 * 2 = 6

∴ Time complexity = O(log n)



4. Count number of digits in a number

```
main() {
    S.O.P (count(num));
}
count (int num) {
    if (num <=0) return 0;
    return 1 + count (num/10);
}
```

eg. 9876

① count (9876) → num > 0 → 1 + count (987)

② count (987) → num > 0 → 1 + 1 + count (98)

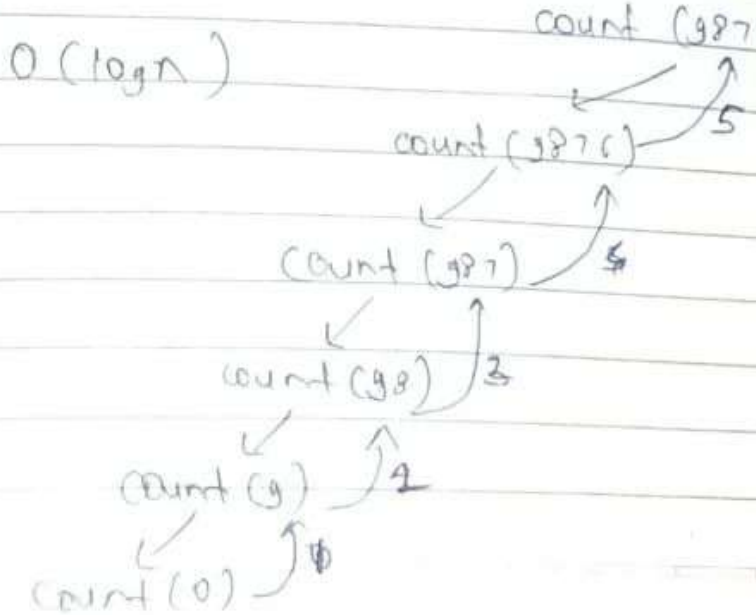
③ count (98) → num > 0 → 2 + 1 + count (9)

④ count (9) → num > 0 → 3 + 1 + count (0)

⑤ count (0) → return 0;

∴ count = 5

Time complexity = O (log n)



5. Find the maximum element in an array.

→

```
main(){
    s.o.p(max(nums, index));  //([2,5,9,1,6], 0)
}
int max(int[] nums, int index){
    if(index == nums.length) return 0;
    return Math.max(nums[index], nums(index+1, nums));
}
```

eg. [2, 5, 9, 1, 6]

① max(nums, 0) → 0 ≠ 5 → Math.max(2, max(nums, 1)); (9)

② max(nums, 1) → 1 ≠ 5 → math.max(5, max(nums, 2)); (9)

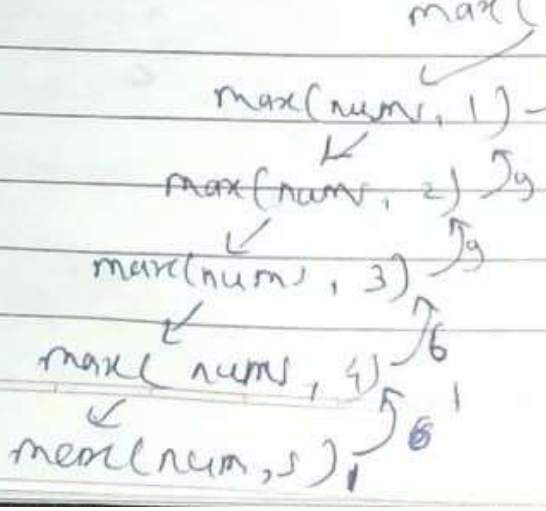
③ max(nums, 2) → 2 ≠ 5 → math.max(9, max(nums, 3)); (8)

④ max(nums, 3) → 3 ≠ 5 → Math.max(1, max(nums, 4)); (8)

⑤ max(nums, 4) → 4 ≠ 5 → math.max(6, max(nums, 5)); (1)

⑥ max(nums, 5) → 5 = 5 → return 1;

∴ max = 9 , Time complexity = O(n),



(Recursion Tree)

6. check if an array is sorted (strictly increasing)

```
→ main(){
    isSorted(nums, 1);
  }
boolean isSorted(int[] nums, index){
    if(nums[index-1] >= nums[index]) return false;
    else if(index == nums.length) return true;
    return isSorted(nums, index+1);
  }
```

① isSorted(nums=[1,2,3,4], 1)
↳ (nums[0] < num[1])
↳ isSorted(num, 2)

② isSorted(nums, 2)
↳ (nums[1] < nums[2])
↳ isSorted(nums, 3)

③ isSorted(nums, 3)
↳ num[2] < num[3]
↳ isSorted(nums, 4)

④ isSorted(nums, 4)
↳ ~~return~~ index = 4 = nums.length;
↳ true;

true

true

true

∴ true, array is sorted.

Time complexity = O(n)

7. Check if a number is prime

e.g. 13 → 13 × 1 = 13
1 × 13 = 13

```
int counter = 0;
for (int i = 1; i <= sqrt(n) ; i++){
    if (n % i == 0 && n % (n/i) {
        counter ++ ;
    }
    if (counter > 1) {
        break ;
    }
}
if (counter == 1){
    S.o.p("Prime");
}
```

time complexity = $O(\sqrt{n})$

e.g. 36 → 36 | 1 × 36, 2 × 18, 3 × 12, 4 × 9, 6 × 6

for (i=1; i<=6; i++){

① i=1
↳ 36 % 1 == 0 && 36 % 36 == 0
↳ counter = 1;

② i=2
↳ 36 % 2 == 0 && 36 % 18 == 0
↳ counter = 2;

break;

counter = 2, so not Prime

e.g. 29 → 1 × 29

for (i=1; i<=5; i++){

① i=1
↳ 29 % i == 0 && 29 % 29 == 0 ✓
↳ counter = 1;

↳ for, i = 2, 3, 4, 5
↳ 29 % i ≠ 0

counter = 1, so, it is Prime

8. Find the first index of an element in an array.

→

```
main(){
    firstOcc(arr, 0);
    if(index == arr.length) return -1;
}
key = 7;
int firstOcc(arr, index){
    if(index == arr.length) return -1;
    if(arr[index] == key)
        return index;

    return firstOcc(arr, index+1);
}
```

① firstOcc([4,2,7,7,2], 0){
↳ arr[index] ≠ key
firstOcc(arr, 1)

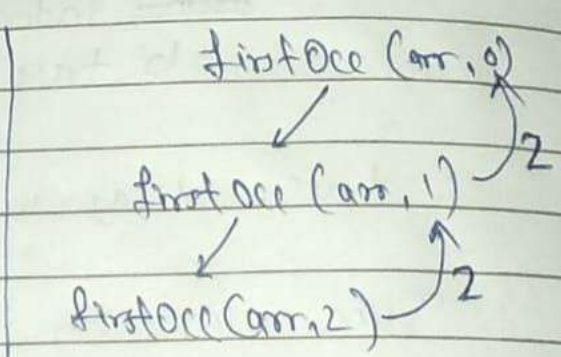
② firstOcc(arr, 1)
↳ arr[1] ≠ key;
firstOcc(arr, 2)

③ firstOcc(arr, 2)
↳ arr[2] = key;
return 2; //index

index

index

∴ Time complexity = O(n);



9. Find the last index of an element in an array.

→

```
main() {
    lastOcc(arr, arr.length-1);
}
int lastOcc(arr, index) {
    if (index == -1) return -1;
    else if (arr[index] == key)
        return index;

    return lastOcc(arr, index-1);
}
```

e.g. [4, 2, 7, 7, 9]

① lastOcc(arr, 4)
↳ arr[4] ≠ 7
↳ index = 4-1 = 3 // lastOcc(arr, 3)

② lastOcc(arr, 3)
↳ arr[3] = 7
↳ return 3;

∴ Time complexity = O(n)

lastOcc(arr, arr.length-1)
↓ ↑ 3
lastOcc(arr, 3)

b. Reverse a number using Recursion.

→

```
main(){
    S.o.p(reverseNum(num));
}
int reverseNum(num){
    if(num<=0) return 0;
    return ((num%10)*10 + reverseNum(num/10));
}
```

e.g. N = 1234

① reverseNum(1234){
↳ Num > 0
↳ 4 (4×10) + reverse(123)

② reverseNum(123)
↳ num > 0
↳ 43 0 + reverseNum(12)

③ reverseNum(12)
↳ num > 0
↳ 4320 + reverseNum(1)

④ reverseNum(1)
↳ num > 0
↳ 4321 + reverseNum(0)

⑤ return 0; // reverseNum = 4321

∴ Time Complexity = O(log n)

11. Count how many times a digit appears in a number

→

```
main () {
    int count = 0;
    S.O.P ( countNum ( 717237 ) );
}
void countNum ( int num ) {
    if ( num <= 0 ) return 0 ; rem = num % 10;
    num = num / 10;

    if ( rem == digit ) {
        count ++;
    }
    countNum ( num );
    return count;
}
```

① countNum (717237)
↳ rem = 7, count = 1, num = 71723

② countNum (71723)
↳ rem = 7, num = 7172

③ countNum (7172)
↳ rem = 2, num = 717

④ countNum (717)
↳ rem = 7, count = 2, num = 71

⑤ countNum (71)
↳ rem = 1, num = 7

⑥ count (7)
↳ rem = 7, count = 3

Time complexity = $O(\log n)$

return ((num % 10)

12. check if a number is palindrome.

→ main() {
int reverse = reverseNum (num);
s.o.p (is Palindrom (num, reverseNum));
}
int rverseNum (num) {
if (num <= 0) return 0;
return ((num % 10) * 10 + reverseNum (num/10));
}
boolean is Palindrom (num, reverseNum) {
if (num == reverseNum) {
return true;
} else {
return false;
}
}

- e.g. N = 121

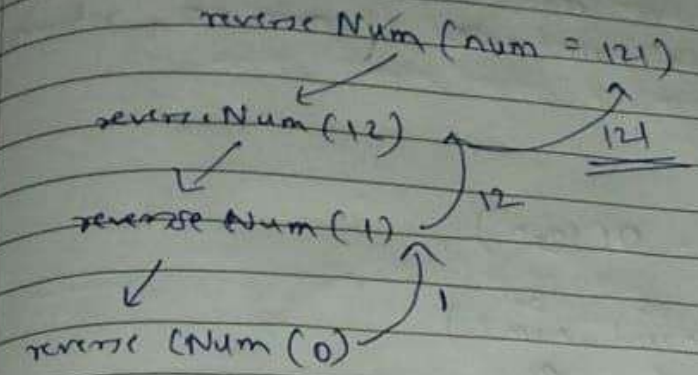
① reverse (121) {
↳ 1x10 + reverse (12)

② reverse (12)
↳ 120 + reverse (1)

③ reverse (1)
↳ 120 + 1 ; reverse = 121

↳ Palindrome (121, 121)
↳ 121 = 121
↳ Therefore, 121 is palindrome.

Time Complexity = O(log n)



13 Find GCD of two numbers using recursion.

```
main(){
    S.o.p(gcd(num1, num2));
}
int gcd(num1, num2){
    if(num1==0 || num2==0) return Math.max(num1,num2);
    max = math.max(num1, num2);
    min = Math.min(num1, num2);

    return gcd(max%min, min);
}
```

e.g. A = 24, B = 36

① gcd(24, 36)
↳ max = 36, min = 24
↳ gcd(36%24, 24) = gcd(12, 24)

② gcd(12, 24)
↳ max = 24, min = 12
↳ gcd(24%12, 12) = gcd(0, 12)

③ gcd(0, 12)
↳ max = 12, min = 0
↳ num1 == 0 || num == 0, return max

∴ gcd = 12

∴ Time complexity = O(log n)

gcd(num1, num2)
24 36
↓ ↑ 12
gcd(12, 24)
↓ ↑ 12
gcd(0, 12)

(Recursion Tree)

14 Print all numbers from 1 to N divisible by 3.

```
main (){
    divisibleByThree(num, 1);
}
void divisibleByThree(num, index){
    if(3 * index > num)
        return;
    s.o.p(3 * index);
    divisibleByThree(num, index+1);
}
```

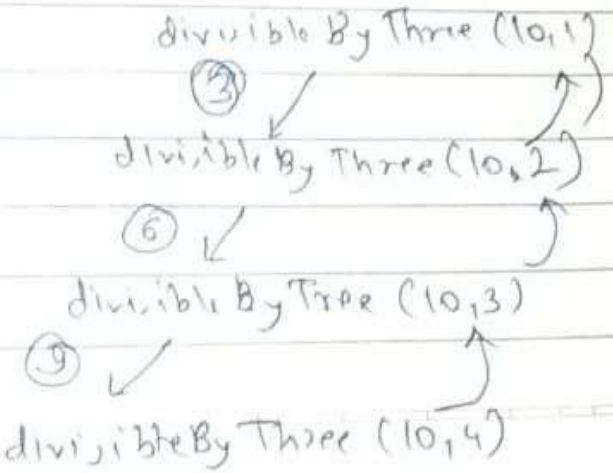
① divisibleByThree(10, 1)
↳ 3×1 < num
↳ s.o.p(3); divisibleByThree(10, 2)

② divisibleByThree(10, 2)
↳ 3×2 < num
↳ s.o.p(6); divisibleByThree(10, 3)

③ divisibleByThree(10, 3)
↳ 3×3 < num
↳ s.o.p(9); divisibleByThree(10, 4)

④ divisibleByThree(10, 4)
↳ 3×4 > num → return;

∴ Time Complexity = $O(\frac{n}{3})$



15. Find power of a number using recursion.

→

```
main () {
    S.o.p(pow(int A, int B));
}
int pow (num1, num2) {
    if (num2 <= 0) return 1;
    return num1 * pow(num1, num2-1);
}
```

Eg. pow (2, 4)

↳ 4 > 0

↳ return 2 * pow (2,3) ← 8 × 2 = 16

pow (2,3)

↳ 3 > 0

↳ return 2 * pow (2,2) ← 4 × 2 = 8

pow (2,2)

↳ 2 > 0

↳ return 2 * pow (2,1) ← 2 × 2 = 4

pow (2,1)

↳ 1 > 0

↳ return 2 * pow (2,0) ← 2 × 1 = 2

return 1;

power = 16, Time complexity = O(n1m2);