Name — Ayan Sadhukhan
Registration No - 19BEC1448

Assignment 1

Arrays

1. Problem Statement

Array Manipulation - Starting with a 1-indexed array of zeroes and a list of operations, for each operation add a value to each of the array element between two given indices, inclusive. Once all operations have been performed, return the maximum value in your array.

For example, the length of the array of zeroes $n = 10$, The list of queries is as follows:-

a	b	k
1	5	3
4	8	7
6	9	1

Add the values of k between the indices a and b inclusive.

index -> 1 2 3 4 5 6 7 8 9 10

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[3, 3, 3, 3, 3, 0, 0, 0, 0, 0]

[3, 3, 3, 10, 10, 7, 7, 7, 0, 0]

[3, 3, 3, 10, 10, 8, 8, 8, 1, 0]

The largest value is 10 after all operations are performed.

2. Efficient Data Structure for the problem:

Arrays is used as the efficient data structure for this problem statement.

3. Design technique Used for the Algorithm

Normal array is used for solving the problem. A for loop is used to ~~fix~~ develop an algorithm to find the output of the problem statement.

4. Pseudocode

```
main()
    int N, K, p, q, sum, i, j, max → 0, u → 0.
    input N
    input K
    int a[N+1]
  for(i → 0 to K)
      input p
      input q
      a[p] += a[p] + sum
      if((q+1) <= N):
          a[q+1] = a[q+1] - sum
      endif
  end for
  for(i → 1 to N)
      u → u + a[i]
      if (max < u)
          max = u
      endif
  end for
  output max
end main
```

5. Time Complexity

Time complexity of the above pseudocode is ~~O(n)~~ $O(n)$.

Stacks

1. Problem Statement :-

Skyline Real Estate Developers is planning to demolish a number of old, unoccupied buildings and construct a shopping mall in their place. ~~You~~ The task is to find the ~~longest~~ solid area in which the mall can be constructed.

There are a number of buildings in a certain two dimensional landscape. Each building has a height, given by h[i] where i ∈ [1,n] If you join k adjacent buildings, they will form a solid rectangle

of area $k \times \min(h[i], h[i+1], \ldots h[i+k-1])$

For example heights array

$h = [3, 2, 3]$.

A rectangle of height $k = 2$ and length $k = 3$ can be constructed within the boundaries. The area formed is $h, k = 2, 3 = 6$.

2. Efficient Data Structure Used for the Problem:

Stacks data structure is used for the given problem.

3. Design technique used for the Algorithm

Divide and Conquer Approach is used for this problem. Considering the full histogram [start, end]. Find the position of the minimum value in [start, end]. Let it be n.

If the optimal subpart contains the n^{th} bar. Then its area would be:

Height [n] x (end - start + 1)

If the optimal subpart doesn't contain the u^{th} bar then, either it will be in [start, u-1] or in [u+1, end] to which is divide and conquer.

Area (start, end) = max (Height [u] x (end - start + 1). Area (start, u-1). Area (u+1, end); u is the position of the smallest bar in [start, end].

4. Pseudocode :-

```
int s[]
Histogram ( int height[])
    height.push(0)
    int sum = 0, i = 0
    while ( i < height.size())
        if(s.empty() || height[i] > height[s.back()])
            s.push(i)
            i = i+1
        end if
        else
            int t ← s.back()
            s.pop()
```

```
        sum = max(sum, height[t] * (s.empty() ?
                                    i : i - s.back() - 1));

    end else

end while

return sum

end Histogram

main()
    int i, j, k, base = 0, n
    while (input(n) == 1)
        height.assign(n, 0)
        for(i → 0 to n)
            input(height[i])
        end for
        print(Histogram(height))
    end while

end main.
```

5. Time Complexity

$O(n \log n)$

Queues

1. Problem Statement :-

You are given Q queries. Each query consists of a single number N. You can perform any of the 2 operations on N in each move:

(i) If we take 2 integers a and b. where $N = a \times b (a \neq 1, b \neq 1)$, then we can change $N = \max(a, b)$

(ii) Decrease the value of N by 1.

Determine the minimum number of moves required to reduce the value of N to 0.

2. Efficient Data structure for this problem :-

Queue data structure is used for this problem statement.

3. Design technique used for the algorithm :-

We use queues to solve this problem by implementing the

unique operation of queue is FIFO.

4. Pseudocode

```
main ( )
    int dis[10000]
    int n, que
    input (que)
    while (que --)
        input(n)
        if (n = 0)
            output 0
            continue
        endif
        queue q
        q.push(n)
        dis[n] = 1
        while (!q.empty ())
            int now = q. front()
            q.pop()
            if ( dis[now-1] = = 0)
                dis[now-1] = dis[now] + 1
                if (now-1 = =0
                    break
                q.push (now-1)
```

```
else
for (i = 2 ; i*i <= now, i++)
    if(now%i==0)
        int fac = max(i, now/i)
        if (dis[fac]==0)
            dis[fac] = dis[now] +1
            q.push (fac);
        endif
    end if
end for
end while
end while
output ( dis[0] -1 )
end main
```

5. Time complexity

$O(n^2)$

Linked List

1. Problem Statement:

A linked list is said to contain a cycle if any node is visited more than once while traversing the list. The function has one parameter, a pointer to a node object named head that points to the head of a linked list. The function returns a boolean denoting whether or not there is a cycle in the list, if there is a cycle return true otherwise false.

2. Data Structure used for this Problem:

Linked List

3. Design technique for the algorithm:

There are 3 scenarios to consider:

(i) the list is empty (ie head is null)

(ii) the list does not contain a cycle, so one can traverse the list and terminate once there are no more nodes.

(iii) The list contains a cycle so attempting to traverse will keep in a loop.

We must traverse the list using two pointers that we will refer to as slow and fast. Our slow pointer moves forward 1 node at a time, and our fast pointer moves forward 2 nodes at a time. If at any point in time these pointers refer to the same object, then there is a loop, otherwise the loop does not contain a loop.

4: Pseudocode

```
bool hascycle (Node *head)
    Node * fast = head
    Node *slow = head
    while( fast != NULL and slow != NULL
            and fast -> next )
        fast = fast -> next -> next
        slow = slow -> next
        if ( fast = slow)
```

return 1
end if
end while
return 0
end hascycle

5. Time Complexity:

O(n)

Tree

1. Problem Statement

We define a tree as a binary tree if it has the following ordering requirements:-

(i) data value of every node in a node's left subtree is less than the data value of that node.

(ii) The data value of every node in a node's right subtree is greater than the data value of that node.

Given the root node of a binary tree can we determine its binary search tree.

2. Data Structure Used

Linked List is used for representing the trees.

3. Design Technique for the Algorithm

A recursive function is used to implement the algorithm.

4. Pseudocode

```
bool check(Node * root, int minval,
    int maxval)
  if (root == NULL)
    return true
  endif
  if (root -> data < minval or
      root -> data > maxval)
    return false
  end if
  return ( check(root -> left, minval,
                 root -> data -1)
           and check(root -> right,
```

```
        root -> data + 1, maxVal)
    :
    end check

bool check2( Node * root)
    :   return . check ( root, 0, 10000)
    :
end check 2 .
```

6. Time Complexity

O(N)

Graph

1. Problem Statement

We are given a tree (a simple connected graph with no cycles) Find the maximum number of edges you can remove from the tree to get a forest such that each connected component of the forest contains an even number of nodes.

For example a tree with 4 nodes can be cut at most 1 time to create an even forest.

2. Data Structure Used

We use a linked list to represent the graph and the tree in the solution.

3. Design Technique for the Algorithm

What we need is a subtree that has even number of vertices and as we get it we can remove the edge that connects it to the remaining of the tree so that we are left with a subtree of even vertices and the remaining tree with vertices N. We are left with the same problem again with remaining tree having even number of vertices because. N is even so even - even = even. So we repeat this algorithm until the remaining tree cannot be decomposed. To maximise the number of subtree we have to remove the subtree which cannot be decomposed in the above manner.

4. Pseudocode

```
dfs (int node)
    visit[node] = true
    int num_verten = 0
    for( i → 0 to al[node].size )
        if( ! visit [al[node][i]] )
            int num = dfs (al[node][i])
            if num % 2 = 0
                ans ++
            else
                num_verten = numverten + num
        end if
    end for
    return num_verten + 1
end dfs
```

5. Time Complexity

$O(N)$

Divide and Conquer

1. Problem Statement

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod obeying the simple rules:-

(i) Only one disk can be moved at a time.

(ii) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack ie disk can be moved if it is the uppermost disk on the stack.

(iii) No disk may be placed on top of a smaller disk.

2. Data Structure Used

Normal number of characters are used.

3. Design Technique

Divide and Conquer algorithm along with a recursive function.

4. Pseudocode

```
towerofhanoi (int n, char fromrod, char torod,
                                    char auxrod)
    if n = 1
    output (Move disk 1 from rod', fromrod,
                        'to rod', torod)
    end if
  towerofhanoi (n-1, fromrod, auxrod, torod)
  output ('move disk', n, 'from rod',
                fromrod, 'to rod', torod)
  towerofhanoi (n-1, auxrod, torod, fromrod)
end towerofhanoi

main
    n = 4
    towerofhanoi (n, A, C, B)
    return 0
end main
```

5. Time Complexity

~~O(n!)~~ O(nlogn)

~~Brut~~

Brute Force

1. Problem Statement

Given an array of n elements. find the given element in the array.

2. Data Structure Used

Array is used for the problem

3. Design Technique

Linear Search which is a Brute Force method.

4. Pseudocode

```
Search (int arr[], int n, int x)
    int i
    for(i -> 0 to n)
        if (arr[i] = x)
            return i
        endif
    end for
    return -1
end search
```

main

```
int arr[] = input from user
int x = 10
n = sizeof of arr
result = search(arr, n, x)
```

end main

5. Time Complexity

O(N)

Dynamic Programming

1. Problem Statement

n^{th} Catalan Number - Catalan numbers are a sequence of natural numbers that occurs in many counting problems like:

- The first few catalan numbers for n = 0, 1, 2, 3, are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ...

2. Data Structure Used

Array to store the output for each n^{th} number.

3. Design Technique

A recursive function along with storing them in an array of numbers is used.

4. Pseudocode

```
catalan ( int n )
:
:   int catalan[n+1].
    catalan[0] = catalan[1] = 1
```

```
for (i → 2 to n)
    catalan[i] = 0
    for (i → 0 to i-)
        catalan[i] = catalan[i] +
                     catalan[j] * catalan[i-j-1]
end for
return catalan[n]

end catalan

main
    for(i=0 to 10)
        output(catalan[i])
end main
```

5. Time Complexity

$O(N^2)$