Q1. Given below are three algorithms $arrayMin_1$, $arrayMin_2$ and $arrayMin_3$ for computing the minimum value from an unordered array of numbers. Analyze each of the algorithms to compute its asymptotic run time as a function of the input array size. Provide the detail of the analysis in each case. (5+10+10=25 points)

Hint: (i) For algorithm $arrayMin_1$ use simple counting of primitive operations, for algorithms $arrayMin_2$ and $arrayMin_3$ write appropriate recurrence equations and solve them to get a closed form solution.

(ii) Use Iterative Substitution method or Master method (where appropriate) for finding solution to the recurrence equation.

Master method is given below for your reference.

For recurrence equation of the type

$$T(n) = \begin{cases} 
  c & \text{if } n < d \\
  aT(n/b) + f(n) & \text{if } n \geq d
\end{cases}$$

the solution to the recurrence equation using Master method is as follows:

1. $T(n)$ is $\Theta(n^{\log_b a})$ if $f(n)$ is $O(n^{\log_b a - \varepsilon})$
2. $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$ if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$
3. $T(n)$ is $\Theta(f(n))$ if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$

(A) **Algorithm $arrayMin_1(A, n)$**

- **Input**: array $A$ of $n$ integers
- **Output**: minimum element of $A$

```
currentMin ← A[0] --------- 1
for i ← 1 to n - 1 do --------- n
    if A[i] < currentMin then -- n
        currentMin ← A[i] -- n
return currentMin --------- 1
```

Ans: Total Ops : 3n+2.

So run time is $\Theta(n)$
Algorithm *arrayMin2*(A, n)

**Input** array A of n integers

**Output** minimum element of A

if **n=1**

    return A[0]

**Min1** ← A[0]

{Let A[1.. n-1], be the array A containing all elements except A[0].}

**Min2** ← *arrayMin2*(A[1.. n-1], n-1)

if **Min1** ≤ **Min2** then

    return **Min1**

else

    return **Min2**

Ans: T(n) = 2 for n = 1

T(n) = T(n-1) + 3 for n > 1

We have to find a close form solution to the recurrence equation:

T(n) = T(n-1) + 3 = T(n-2) + 2*3 = T(n-3) + 3*3 = ….. = T(n-i) + i*3

Let us choose ‘i’ such that n-i = 1. That mean i = (n-1)

Then T(n) = T(1) + (n-1)*3 = 2 + 3n – 3 = 3n - 1

Thus T(n) is Θ(n).
Algorithm `arrayMin3(A, n)`

Input: array A of n integers
Output: minimum element of A

if n=1
    return A[0]

m ← ⌊n/2⌋
{Let A[0 .. m-1] be the array A containing first m elements,
and A[m .. n-1] be the array A containing last n-m elements.}

Min1 ← `arrayMin3(A[0 .. m-1], m)`
Min2 ← `arrayMin3(A[m .. n-1], n-m)`
if Min1 < Min2 then
    return Min1
else
    return Min2

Ans:

T(n) = 2  for n = 1
T(n) = 2T(n/2) + 3 for n > 1

We have to find a close form solution to the recurrence equation:
T(n) = 2T(n/2) + 3
We will apply Master method here.
a = 2, b = 2
So \(\log_b a = 1\)
and \(n^{\log_b a} = n\)
f(n) = 3. That means f(n) is \(O(1)\) or \(O(n^0)\) or \(O(n^{\log_b a - \epsilon})\)
Using case 1 of Master method we get \(T(n) = \Theta(n^{\log_b a}) = \Theta(n)\)
Q2. Multi-way search trees are ordered trees with the following properties.

- Each internal node is a \( d \)-node with \( d \geq 2 \)
- Each \( d \)-node \( v \) of the tree has \( d \) children \( v_1, \ldots, v_d \) and stores \((d-1)\) items with keys \( k_1, \ldots, k_{d-1} \) such that \( k_1 \leq \ldots \leq k_{d-1} \).
- The key \( k \) of each item stored in the tree rooted at \( v_1 \) is less than or equal to \( k_1 \).
- The key \( k \) of each item stored in the tree rooted at \( v_d \) is greater than or equal to \( k_{d-1} \).

We have learnt about (2,4) trees that are multi-way trees with additional properties to maintain balance. These properties are:

(a) every node has at most four children
(b) all external nodes have same depth.

Maintaining these properties requires some effort after performing insertions and deletions in a this tree.

Let us define (2,3) trees that are similar to (2,4) trees in all respect except that every node has at most three children. That means:

(a) every node has at most two keys and three children
(b) all external nodes have same depth.

Create such a (2,3) tree by inserting keys 30, 40, 24, 58, 48, 26, 11 in to an empty tree.

Delete keys 30, 21, 48, 58 from the resulting tree.

Show the balanced (2,3) tree resulting after every key insertion and deletion. (20 points)
Q3. Perform the required modification operation on the trees given below.
(A) Insert key 4 into the following Red-Black tree. (10 points)

If the insertion results in violation of any red-black tree property then remedy the problem and show each intermediate tree generated during the remedy process. To distinguish between Red and Black nodes label each node as R or B in your drawings.
(B) Prove that the height of an AVL tree $T$ storing $n$ items is $O(\log n)$. (15 points)

(Definition of AVL Tree: An AVL tree is a binary search tree, and at every internal node of the tree the height of its left and right sub-trees differ by at most one.)

Ans: (See book; Theorem 3.2.)

Let $n(h)$ be the minimum number of nodes in an AVL tree of height $h$.

- Number of nodes in an AVL tree of height 1 is 1. So $n(1) = 1$
- Number of nodes in an AVL tree of height 2 is 2. So $n(2) = 2$

From the definition of AVL tree, at every internal node of the tree the height of its left and right sub-trees differ by at most one.

So $n(h) = 1 + n(h-1) + n(h-2)$

Or $n(h) > 2n(h-2)$

By recursive substitution:

- $n(h) > 2n(h-2)$
- $> 2^2 n(h-2.2)$
- $> 2^3 n(h-2.3)$
- $> \ldots$
- $> 2^i n(h-2.i)$

Let us choose $i$ such that $(h-2.i) = 2$. Then $i = (h-2)/2$.

So $n(h) > 2^{(h-2)/2} n(2)$

- $> 2^{(h-2)/2} \cdot 2$
- $> 2^{h/2}$

or $\log(n(h)) > h/2$

or $2\log(n(h)) > h$

or $h < 2\log(n(h))$

Thus an AVL tree storing $n$ items has height at most $2\log n$ or in other words height is $O(\log n)$. 

- 8 -
Q4. Write in pseudo code an algorithm for merging two ordered arrays of equal size in such a way that the resulting array after the merge remains ordered. Analyze the algorithm to compute its run time as a function of the input array size $n$. (15 points)

Ans:

Algorithm merge (A, B, n)

Input: Ordered arrays A and B of size n.
Output: An ordered array C containing all the elements of A and B.

Let C be an array of size 2n.

i ← 0
j ← 0
k ← 0

while (i < n) and (j < n) do
    if (A[i] ≤ B[j])
        C[k] = A[i]
        Increment i
    else
        C[k] = B[j]
        Increment j
    Increment k

While (i < n) do
    C[k] = A[i]
    Increment k
    Increment i

While (j < n) do
    C[k] = B[j]
    Increment k
    Increment j

Return

Analysis:

$1^{st}$ while statement and $2^{nd}$ while statement guarantee that all the elements of the array A are copied to C only once. So total O(n) operations.

$1^{st}$ while statement and $3^{rd}$ while statement guarantee that all the elements of the array B are copied to C only once. So total O(n) operations.

We have taken into consideration all the array elements copy operations in the three while loops.

So total cost is O(n) + O(n) i.e. O(n).

So the runtime of the algorithm is O(n).
Q5. Compute $\Theta$ notation for the number of times the statement “$x -\rightarrow x+1$” is executed as a function of $n$. (5+5+5=15 points)

(a) for $i \leftarrow 1$ to $n$
    for $j \leftarrow 1$ to $i$
        for $k \leftarrow 1$ to $j$
            $x \leftarrow x + 1$

Ans: $\Theta(n^3)$

(b) $i \leftarrow n$
    while ($i \geq 1$)
        $x \leftarrow x + 1$
        $i \leftarrow i / 2$

Ans: $\Theta(\log n)$

(c) $i \leftarrow 2$
    while ($i < n$)
        $i \leftarrow i^i$
        $x \leftarrow x + 1$

Ans: $\Theta(\log \log n)$