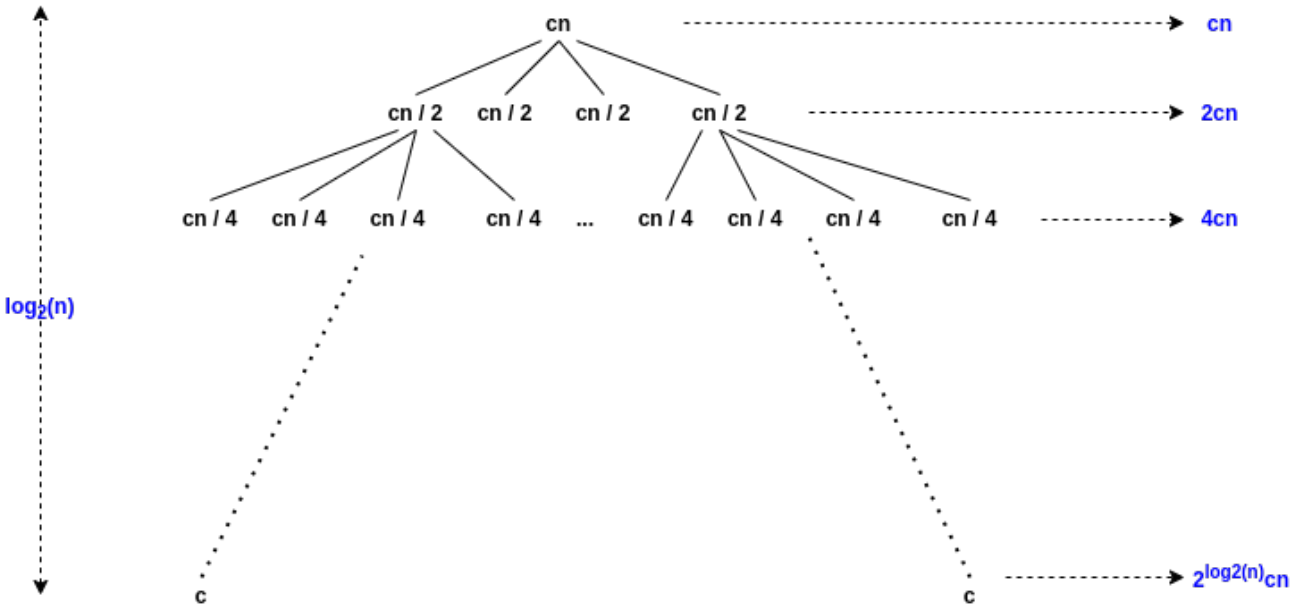
Question 1

The recursion tree for the recurrence $T(n) = 4T(n/2) + cn$ is as follows:



$T(n) = cn + 2cn + ... + 2^{log_2 n}.cn$

$\implies T(n) = \sum_{i=0}^{log_2 n} 2^i.c.n$

$\implies T(n) = (2^{log_2 n+1} - 1).cn$

$\implies T(n) = (2n - 1).cn$

$\implies T(n) = \Theta(n^2)$

Verifying bound by Substitution Method:

We will guess an upper bound of kn^2.

Base Case: $n = 1$

$T(1) = c \leq kn^2$. This is true as long as $k \geq c$.

Inductive Step: Let us assume this to be true for $n/2$.

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

$T(n/2) \leq k(n/2)^2$

$\implies 4T(n/4) + c(n/2) \leq k(n/2)^2$ $\rightarrow$ (i)

We need to show it to be true for n.

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

$\implies T(n) = 4\{4T(n/4) + c(n/2)\} + cn$

$\implies T(n) = 16T(n/4) + 3cn$ $\rightarrow$ (ii)

From (i),

$4T(n/4) \leq k(n/2)^2 - cn/2$

$\implies 16T(n/4) + 3cn \leq kn^2 + cn$ $\rightarrow$ (iii)

Using (iii) in (ii),

$\implies T(n) \leq kn^2 + cn$

$\implies T(n) = \Theta(n^2)$

Hence Proved.

Question 2

Given an array A of n distinct integers sorted in ascending order.

To decide whether $A[i] = i$, binary search can be used as follows ($SEARCH(A, 1, n)$):

```
function SEARCH(array, start, end)
    middle <- (start + end) / 2
    if start == end AND array[middle] != middle
        return False
    end if
    if array[middle] == middle
        return True
    else if array[middle] > middle
        SEARCH(array, start, middle)
    else if array[middle] < middle
        SEARCH(array, middle, end)
    end if
end function
```

Every time that the function $SEARCH$ is called, the length of the part of the array in which the element i is to be searched is halved. Thus, the running time of the algorithm is $O(logn)$.

Question 3

When the element $A[i]$ is larger than its children, calling $MAX - HEAPIFY(A, i)$ has **no effect**. After carrying out the comparisons in $MAX - HEAPIFY$, it is found that $A[i]$ is the largest, and thus, the procedure just returns.

Question 4

Let A and B be the sorted sequences with m and n integers respectively.

Let i and j be the indexes of medians of A and B.

Suppose $A[i] < B[j]$.

If $i + j \geq k$, then the subsequence following the median in B can be deleted. This is so because, in such a case, the kth smallest element in the sequence obtained by merging A and B will lie before the medians of A or B.

However, if $i + j < k$, then the subsequence preceding the median in A can be deleted. This is so because, in such a case, the kth smallest element in the sequence obtained by merging A and B will lie after the median of B and each of the elements in the subsequence preceding the median in A are anyways smaller than the median of B (based on the supposition in (i)).

Each time this recursive algorithm is called, the subsequence preceding or following one of the medians is eliminated. Thus, the run time of this algorithm is $O(log(max(m, n)))$.

Question 5

The pivot element in each partition in the $QUICKSORT$ algorithm is smaller than all other elements in the sequence. Thus, the partition will produce two subsequences in which one contains only one element (pivot element in each partition) and the other contains the remaining elements.

Thus, the run time of each recurrence is $T(n) = T(n-1) + n$.

$\implies T(n) = n + (n-1) + ... + T(1)$

$\implies T(n) = \Theta(n^2)$

Question 6

Given a set S of n 2-dimensional points in the plane.

If $n \leq 1$, the maximal set is just S itself.

Otherwise, sort S in the lexicographic order, i.e. primarily based on the x-coordinates, and based on the y-coordinates in the event of a tie between the x-coordinates.

Next, we recursively solve the maximal set problem for the set of points which are lexicographically smaller than the median and also for the set of points which are lexicographically larger than the median.

Let these solutions be $M1$ and $M2$ respectively. The maximal set of points $M2$ are also the maximal points for S. However, some of the maximal points in $M1$ might be dominated by

the lexicographically smallest point in $M2$. All such points can be eliminated from M1, and the union of the resulting set with $M2$ will be the maximal set of S.

In the pseudo code below, it is assumed that the time complexity of LEXICOGRAPHIC-SORT is linear.

```
INPUT: A - Array of n points in the plane

OUTPUT: M - Array of maximal points

function MAXIMAL-SET(S)
    n <- Size of A
    if n <= 1
        return S
    end if

    SS <- LEXICOGRAPHIC-SORT(S)
    Let P be the median of SS with index p

    M1 <- MAXIMAL-SET(SS[1 : p-1])
    M2 <- MAXIMAL-SET(SS[p : n])

    MM2 <- LEXICOGRAPHIC-SORT(M2)
    q <- MM2[1]

    For each point r in M1, do
        if x, y coordinates of r <= x, y coordinates of q
            Eliminate r from M1
    return M1 U M2
```

There are two recursive calls within the procedure. The rest of the steps in the procedure can be completed in linear time. Thus, the running time for the MAXIMAL-SET algorithm can be described as follows:

$$T(n) = \begin{cases} c & if n \leq 1 \\ 2T(\frac{n}{2}) + cn & if n \geq 2 \end{cases}$$

Thus, the time complexity of the algorithm is $O(nlogn)$.

Question 7

Convert each of the integers to base n. Each number will have atmost $\log_n n^2 = 2$ digits. Perform radix sort on these numbers. Since each number will have atmost 2 digits, the radix

sort will only need 2 passes. For each pass, there are n possible values which can be taken on, i.e. $0, 1, 2, ...n-1$. We can use counting sort to sort each digit, which has a linear time complexity. Thus, counting sort is applied twice for such a radix sort, resulting in a time complexity of $O(2(n))$, i.e. $O(n)$.

Question 8

Consider the element a_{n1}, i.e. the first element of the last row in the matrix.

If $x = a_{n1}$, the search is complete. If $x < a_{n1}$ eliminate the last row. If $x > a_{n1}$, eliminate the first column.

```
INPUT: A - Matrix, m - Number of rows, n - Number of columns, x - key

OUTPUT: True, if x is in matrix; False otherwise.

function MATRIX-SEARCH(A, m, n, x)
    if A is non-empty
        element <- A[m][1]
        if element == x
            return True
        else if element < x
            A <- REMOVE-LAST-ROW(A)
            MATRIX-SEARCH(A, m-1, n, x)
        else if element > x
            A <- REMOVE-FIRST-COLUMN(A)
            MATRIX-SEARCH(A, m, n-1, x)
        end if
    end if
    return False
end function
```