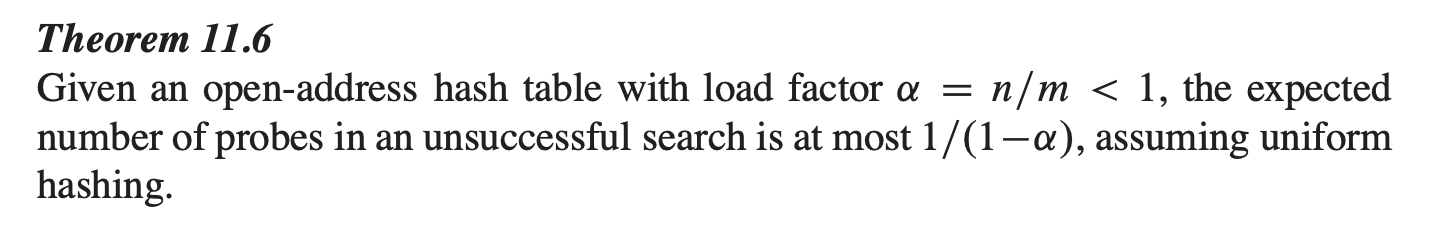
Time complexity of building max heap is O(n)

A red-black tree with 𝑛 internal nodes has height at most 2 lg(𝑛 + 1).

A picture containing animal

Description automatically generated



等比数列求和公式：A picture containing object, clock

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A close up of text on a white background

Description automatically generated

Longest Common Subsequence with DP:

Bottom up:

If (A[i] == B[j]) {

LCS[I, j] = 1 + LCS[i – 1,j – 1]

} else {

LCS[i, j] = max(LCS[I – 1, j], LCS[I, j - 1]);

}

def printLCSSubStr(X: str, Y: str,

                   m: int, n: int):

    # Create a table to store lengths of

    # longest common suffixes of substrings.

    # Note that LCSuff[i][j] contains length

    # of longest common suffix of X[0..i-1] and

    # Y[0..j-1]. The first row and first

    # column entries have no logical meaning,

    # they are used only for simplicity of program

    LCSuff = [[0 for i in range(n + 1)]

                 for j in range(m + 1)]

    # To store length of the

    # longest common substring

    length = 0

    # To store the index of the cell

    # which contains the maximum value.

    # This cell's index helps in building

    # up the longest common substring

    # from right to left.

    row, col = 0, 0

    # Following steps build LCSuff[m+1][n+1]

    # in bottom up fashion.

    for i in range(m + 1):

        for j in range(n + 1):

            if i == 0 or j == 0:

                LCSuff[i][j] = 0

            elif X[i - 1] == Y[j - 1]:

                LCSuff[i][j] = LCSuff[i - 1][j - 1] + 1

                if length < LCSuff[i][j]:

                    length = LCSuff[i][j]

                    row = i

                    col = j

            else:

                LCSuff[i][j] = 0

    # if true, then no common substring exists

    if length == 0:

        print("No Common Substring")

        return

    # allocate space for the longest

    # common substring

    resultStr = ['0'] \* length

    # traverse up diagonally form the

    # (row, col) cell until LCSuff[row][col] != 0

    while LCSuff[row][col] != 0:

        length -= 1

        resultStr[length] = X[row - 1] # or Y[col-1]

        # move diagonally up to previous cell

        row -= 1

        col -= 1

    # required longest common substring

    print(''.join(resultStr))