Name - Yogesh Agarwala
EE19B130

Assignment -2

(1) Solution :

$4n \log n + 2n \longrightarrow O(n \log n)$

$3n + 100 \log n \longrightarrow O(n)$

$n^2 + 10n \longrightarrow O(n^2)$

$2^{10} \longrightarrow O(1)$

$4n \longrightarrow O(n)$

$n^3 \longrightarrow O(n^3)$

$2^n \longrightarrow O(2^n)$

$n \log n \longrightarrow O(n \log n)$

$2^{\log n} = n^{\log 2}$

→ Case 1: base of log is 2
$= O(n)$

→ Case 2: base of log > 2
$= O(n^{\log 2})$

Case 1: base of log is 2

Now,
$O(1) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

$\boxed{2^{10}} < \boxed{2^{\log n} = 3n + 100 \log n = 4n} < \boxed{n \log n = 4n \log n + 2n} < \boxed{n^2 + 10n} < \boxed{n^3} < \boxed{2^n}$

though
$2^{\log n} < 3n + 100 \log n < 4n$
as n approaches ∞
but their asymptotic growth rate is
same.

Case 2: base of log > 2

Now,
$O(1) < O(n^{\log 2}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

$\boxed{2^{10}} < \boxed{2^{\log n}} < \boxed{3n + 100 \log n = 4n} < \boxed{n \log n = 4n \log n + 2n} < \boxed{n^2 + 10n} < \boxed{n^3} < \boxed{2^n}$

(2)

- **Method 1** : Formal definition

$$f(n) = O(g(n))$$

if there exist constants $c, n_o > 0$

Such that

$$0 \leq f(n) \leq c g(n) \quad \text{for all } n \geq n_o$$

$$f(n) = \Omega(g(n))$$

if there exist constants $c, n_o > 0$

Such that

$$0 \leq c g(n) \leq f(n) \quad \text{for all } n \geq n_o$$

$$f = \Theta(g(n))$$

if there exist constants $c_1, c_2 \& n_o > 0$

Such that

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{for all } n \geq n_o$$

- **Method 2** : Limit

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & , \quad f(n) = O(g(n)) \\ \infty & , \quad f(n) = \Omega(g(n)) \\ \text{otherwise}, \to \begin{cases} f = O(g(n)) \\ = \Omega(g(n)) \\ = \Theta(g(n)) \end{cases} \end{cases}$$

(a) $f(n) = n - 100$ $g(n) = n - 200$

Method 1 :

$f(n) \leq c\, g(n)$ $f(n) \geq c\, g(n)$

$n - 100 \leq c\,(n - 200)$ $n - 100 \geq c\,(n - 200)$

Let $c = 2$ Let $c = 1$

      $\& n_0 = 300$ $n_0 = 1$

$\Rightarrow$ for all $n > n_0$ $\Rightarrow$ for all $n > n_0$

       $f(n) \leq c\, g(n)$ $f(n) \geq c\, g(n)$

              $\downarrow$ $\downarrow$

      $f(n) = O(g(n))$ $f(n) = \Omega(g(n))$

                     $\downarrow$

                  $f(n) = \Theta(g(n))$

So

for $f(n) = n - 100$

     $g(n) = n - 200$ $\Big\} \longrightarrow$

$\boxed{\begin{array}{l} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \\ f(n) = \Theta(g(n)) \end{array}}$

method 2 :

$\displaystyle \lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{n - 100}{n - 200} = 1$

$\therefore \begin{cases} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \\ f(n) = \Theta(g(n)) \end{cases}$

(b) $f(n) = 100n + \log n$ $\qquad$ $g(n) = n + (\log n)^2$

method 1:

$f(n) \leq cg(n)$

$100n + \log n \leq c(n + (\log n)^2)$

Let
$c = 100$
$n_0 = 2$

$\Rightarrow$ for all $n \geq n_0$

$\qquad f(n) \leq cg(n)$

$\downarrow$

$f(n) = O(g(n))$

$f(n) \geq cg(n)$

$100n + \log n \geq c(n + \log^2 n)$

Let $c = 1$
$\qquad n_0 = 1$

Satisfies
$100n + \log n \geq c(n + \log^2 n)$
for all $n \geq n_0$

$\downarrow$

$f(n) = \Omega(g(n))$

$f(n) = \Theta(g(n))$

So for
$f(n) = 100n + \log n$
$g(n) = n + (\log n)^2$
$\Bigg\} \longrightarrow$
$\boxed{\begin{array}{l} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \\ f(n) = \Theta(g(n)) \end{array}}$

method 2:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{100n + \log n}{n + \log^2 n} = \frac{100 + \log n / n}{1 + \log^2 n / n} =$$

$$= 100 \qquad \longrightarrow \boxed{\begin{array}{l} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \\ f(n) = \Theta(g(n)) \end{array}}$$

© $f(n) = \log(2n)$

$g(n) = \log(3n)$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{\log(2n)}{\log(3n)}$$

$$= \lim_{n \to \infty} \frac{\left(\frac{1}{2n}\right)(2)}{\left(\frac{1}{3n}\right)(3)}$$

$$= 1$$

$\therefore$ $f(n) = O(g(n))$

$f(n) = \Omega(g(n))$

$f(n)) = \Theta(g(n))$

Alternate method :

$$f(n) = \log(2n) = \log 2 + \log n$$

$$g(n) = \log(3n) = \log 3 + \log n$$

So both $f(n)$ and $g(n)$ have same asymptotic growth rate of $O(\log n)$

$\therefore$ $f(n) = \Theta(g(n))$

which also mean

$f(n) = O(g(n))$

& $f(n)^{\circ} = \Omega(g(n))$

(d) $f(n) = n^{1.01}$

$g(n) = n \log^2 n$

$$\lim_{n \to \infty} \frac{n^{1.01}}{n \log^2 n} = \frac{n^{0.01}}{\log^2 n}$$

$$= \frac{0.01 \, n^{-0.99}}{2 \log n \left(\frac{1}{n}\right)} \quad \left(\begin{array}{c}\text{using}\\ \text{L'hopital rule}\end{array}\right)$$

$$= \frac{0.01}{2} \cdot \frac{n^{0.01}}{\log n}$$

$$= \frac{0.01^2}{2} \cdot \frac{n^{-0.99}}{1/n}$$

$$= \frac{0.01^2}{2} \, n^{0.01}$$

$$= \infty$$

$$\therefore \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

$$\Rightarrow \boxed{f(n) = \Omega(g(n))}$$

(3) Describe an efficient algorithm for finding the ten largest elements in a sequence of size n. What is the running time of your algorithm.

<u>Pseudocode</u> :

<u>Part 1</u> : Sorting the array using mergesort

```
def merge (A, B):
        C = [ ]
        i = 0
        j = 0
        n = len(A) + len(B)
        for k in range (0, n):
            if A[i] < B[j]:
                C[k] = A[i]
                i++
            else:
                C[k] = B[j]
                j++
        return C

def merge sort (arr):
        n = len (arr)
        if (n == 1):
            return arr

        else: mid = n//2
              A = mergesort (arr[:mid])
              B = mergesort (arr[mid:])

              merged array = merge (A, B)
              return merged array.
```

Part 2: now from the sorted array, insert the 10 largest element into a new array

```
def tenlargest (sortedarray):

    n = len (sortedarray)
    arr = [] // output array

    for i in range (n-1, n-11, -1):

        arr[n-i-1] = sortedarray[i]

    return arr
```

Part 3: main function:

```
inputarray = [2, 3, 1, 7, 8, 23, 11, 9, 1, 3, 5, 12]
sorted array = mergesort (input array)
final array = tenlargest (sorted array)
print (final array)
```

Output:
[23, 12, 11, 9, 8, 7, 5, 3, 3, 2]

Time complexity:

$$\text{Time complexity} = \text{mergesort} + \text{tenlargest}$$
$$= O(n\log n) + O(1)$$
$$= O(n\log n)$$

(4) Use the divide and conquer integer multiplication algorithm to multiply the two binary integers

10011011 and 10111010

Divide each of the two binary number in two halves

$$x = 2^{n/2} x_L + x_R$$
$$y = 2^{n/2} y_L + y_R$$

So
$$xy = \left(2^{n/2} x_L + x_R\right)\left(2^{n/2} y_L + y_R\right)$$

$$= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

$$= 2^n y_L x_L + 2^{n/2}\left[(x_L + x_R)(y_L + y_R)' - x_L y_L - x_R y_R\right] + x_R y_R$$

So we have three subproblem

$$a = x_L y_L$$
$$b = x_R y_R$$
$$c = (x_L + x_R)(y_L + y_R)$$

$$xy = 2^n a + 2^{n/2}(c - a - b) + b$$

then compute
   a, b, c recursively

<u>Example:</u>

in our question, we need to multiply 10011011 & 10111010

$$x = 10011011 = 2^4(1001) + 1011$$

$$y = 10111010 = 2^4(1011) + 1010$$

$$xy = 2^8(x_L y_L) + 2^4\left[(x_L + x_R)(y_L \pm y_R) - x_L y_L - x_R y_R\right] + x_R y_R$$

$$a = x_L y_L = 1001 \times 1011$$

$$b = x_R y_R = 1011 \times 1010$$

$$c = (x_L + x_R)(y_L + y_R) = (1001 + 1011) \times (1011 + 1010)$$

$$= 10100 \times 10101$$

Now

a, b, c will be computed recursively

$$\Rightarrow \quad a = 1100011$$

$$b = 1101110$$

$$c = 0110100100$$

$$xy = 2^8(a) + 2^4(c - a - b) + b$$

$$\therefore xy = 2^8(1100011) + 2^4(1101001\,1) + 1101110$$

$$= (1110000100111110)_2$$

$$= (28830)_2$$

$$1001,1011 \;*\; 1011,1010$$

$a_1 = x_L y_L$
$= 1001 * 1011$

$b_1 = x_R y_R$
$= 1011 * 1010$

$c_1 = (x_L + x_R) * (y_L + y_R)$
$= (1001 + 1011) * \left(\begin{matrix} 1011 + \\ 1010 \end{matrix} \right)$

$= 10100 * 10101$

$a_2 = 10 * 10$

$b_2 = 01 * 11$

$c_2 = (10 + 01) * (10 + 11)$

$a_2 =$

b_2

c_2

a_2

b_2

c_2

a_3 b_3 c_3 a_3 b_3 c_3 a_3 b_3 c_3 a_3 b_3 c_3

a_3 b_3 c_3

Time complexity:

$$T(n) = 3T(n/2) + O(n)$$

using master theorem we get

$$T(n) = O(n^{\log_2 3})$$

$$= O(n^{1.59})$$

Implementation in Python ⟶ P.T.O

Q4. Use the divide and conquer integer multiplication algorithm to multiply the two binary integers 10011011 and 10111010.

In [8]:
```python
# function that multiplies two bit strings X and Y and returns
# the product in decimal format

from math import floor, ceil

def karatsuba(x, y):

    """converting int to strings, for easy access to digits"""
    sx = str(x)
    sy = str(y)
    n = max(len(sx), len(sy))

    """base case of recursion"""
    if len(sx) == 1 and len(sy) == 1:
        return x*y


    else:
        """split the digit sequences about the middle"""
        m = ceil(n/2)
        a = int(x // (10**m))
        b = int(x % (10**m))
        c = int(y // (10**m))
        d = int(y % (10**m))


        """recursively calculate the 3 products"""
        ac = karatsuba(int(a), int(c))
        bd = karatsuba(int(b), int(d))
        adbc = karatsuba(int(a)+int(b), int(c)+int(d)) - ac - bd

        """this little trick, writing n as 2*m takes care of both
        even and odd n"""
        return (2**(2*m))*ac + (2**m)*adbc + bd
```

In [11]:
```python
# Python program to convert decimal to binary

def decimalToBinary(n):
    return bin(n).replace("0b", "")
```

In [10]:
```python
# program to take inputs from the user and then print the result

x = int(input("Enter x: "))
y = int(input("Enter y: "))

product = karatsuba(x,y)
product_in_binary = decimalToBinary(product)

print ("x*y in binary = ",product_in_binary)
print ("x*y in decimal = ",product)
```

```
Enter x: 10011011
Enter y: 10111010
x*y in binary =  111000010011110
x*y in decimal =  28830
```

(5) **Algorithm :**

max element (arr, i, j) :

$\qquad n = (i + j)/2$

$\qquad$ if $arr[n-1] \leq arr[n] \geq arr[n+1]$:
$\qquad\qquad$ return n

$\qquad$ elif $arr[n-1] > arr[n]$:
$\qquad\qquad$ return $(max \, element \, (arr, i, n-1))$

$\qquad$ elif $arr[n] < arr[n+1]$
$\qquad\qquad$ return $(max \, element \, (arr, n+1, j))$

\# example unimodal array.
$arr = [1, 2, 4, 7, 3, 0]$

print $(max element \, (arr, 0, len(arr)-1))$

Explanation :
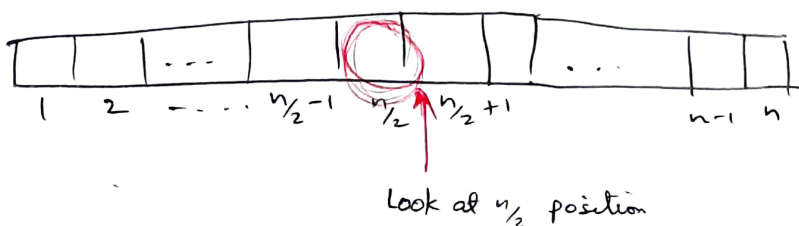
Look at $n/2$ position

If $arr[n/2] < arr[n/2 - 1]$ then only look at
$\qquad$ left half i.e $1 \cdots n/2 - 1$ for finding max element.

elif $arr[n/2] < arr[n/2 + 1]$ then only look at
$\qquad$ right half i.e $n/2 + 1 \cdots n$ for finding max element

elif $arr[n/2 - 1] \leq arr[n/2] \, \& \geq arr[n/2 + 1]$
$\qquad$ then $n/2$ is the max element position

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

$$T(n) = \left(\overline{T(n/4) + c} \right) + c$$

$$T(n) = \left(\overline{T(n/8) + c} \right) + c + c$$

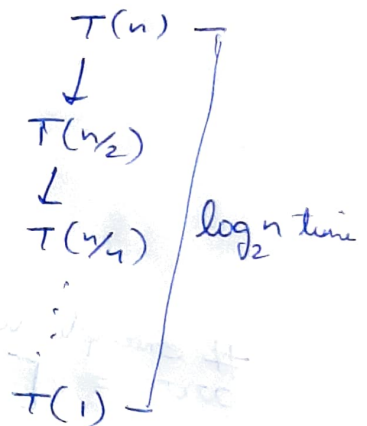$$T(n) = T\left(\frac{n}{2^k} \right) + ck$$

where

$$k = \log_2 n$$

$$\therefore T(n) = T\left(\frac{n}{2^{\log_2 n}} \right) + c\log_2 n$$

$$= T\left(\frac{n}{n} \right) + c\log_2 n$$

$$= T(1) + c\log_2 n$$

$$\therefore T(n) = \Theta(\log n)$$

$T(n)$ ⌐

↓

$T(n/2)$

↓

$T(n/4)$ $\Big|$ $\log_2 n$ time

⋮

$T(1)$

Pseudocode :

```
def merge (A, B):

        C = [ ]  // output array
        i = 0
        j = 0
        n = len (A) + len (B)

        for k in range(0, n):
                if A[i] < B[j] :
                        C[k] = A[i]
                        i++

                else :
                        C[k] = B[j]
                        j++


        return C


def mergesort (arr):

        n = len (arr)
        if (n == 1):
                return arr


        else :  mid = n//2
                A = mergesort (arr[:mid])
                B = mergesort (arr[mid:])

                mergedarray = merge (A, B)
                return mergedarray.
```

```
def removeduplicates (sortedarray):
    arr = [] // output array with no duplicate
    i = 0
    j = 0
    n = len(sortedarray)
    while (i < n):
        //copying value to output array from sorted array.
        arr[j] = sorted array[i]
        while (sortedarray[i] == sortedarray[i+1])
            i = i+1
            //when duplicates is found then
            don't copy to output arra.
        i = i+1
        j = j+1

    return arr // this is the final array with
                              no duplicates
```

Part 3: Main function (taking input & printing result)

inputarray = [2,3,1, 3, 6, 2,1,3]
sortedarray = mergesort (inputarray)
finalarray = removeduplicates (sorted array)

Concept used :

To remove all duplicates in O(n logn), we
first sort the array using mergesort & then
remove duplicates by traversing the sorted array

Time complexity = mergesort + linear traversal of
                                                                      array

$$= O(n \log n) + O(n)$$

$$= O(n \log n)$$