WILLMAN

[DAA] Assignment – 1

Priyansh Bakhtani
20190802079

Q.(1) Sort the functions –

Sol $4\lg(\lg n) < 4\log n < n^{1/2}\lg^4(n) < 5n < n^4 < (\lg n)^{5\lg n}$

$< (\lg)^{\lg n} < n^{n^{1/5}} < 5^n < 5^{5n} < (n/4)^{n/4} < (n)^{n/4} <$

$4^{n^4} < 4^{4^n} < 5^{5^n}$

Q.(2) solve the following recurrences using masters theorem.

(i) $T(n) = 2T(n/4) + 1$

Sol comparing $2T(n/4)$ with $aT(n/b)$

$(n)^{\log_b a} = n^{\log_4 2} = n^{1/2}$

comparing $n^{1/2}$ with 1 $\therefore n^{1/2} > 1$

$T(n) = \theta(n^{1/2})$ Ans

(ii) $T(n) = 2T(n/4) + n^{1/2}$

Sol comparing $2T(n/4)$ with $aT(n/b)$

$(n)^{\log_b a} = (n)^{\log_4 2} = n^{1/2}$

on comparing $n^{1/2} = n^{1/2}$

$\therefore T(n) = \theta(n^{1/2}\log n)$ Ans

③ $T(n) = 7T(n/3) + n^2$

Sol Comparing $7T(n/3)$ with $aT(n/b)$

$$n^{\log_b^a} = (n)^{\log_3^7} \approx (n)^{1.77}$$

On comparing $(n)^{1.77} < (n)^2$

$\therefore T(n) = \Theta(n^2)$ Ans

④ $T(n) = 7T(n/2) + n^2$

Sol Comparing $7T(n/2)$ with $aT(n/b)$

$$n^{\log_b^a} = (n)^{\log_2^7} \approx (n)^{2.805}$$

On comparing with $f(n)$ $n^{\log_2^7} > n^2$

$\therefore T(n) = \Theta(n^{\log_2^7})$ Ans

⑤ $T(n) = T(n-3) + n^2$

Ans Masters theorem can't be applied since $T(n-3)$ is not in the form ~~of~~ $aT(n/b)$

⑥ $T(n) = 2T(n/2) + n/\log n$

Ans Masters theorem can't be applied since $f(n)$ is not a polynomial $(n/\log n)$.

④ $T(n) = T(n-2) + n/\log n$

Ans Master's theorem can't be applied since $T(n-2)$ is not in the form of $aT(n/b)$ also $f(n) = n/\log n$ which is not a polynomial.

Q⑤ Using substitution method show the recurrence equation for $T(n) = 2T(n/2) + \Theta(n)$. Compute the upper bound and lower bound separately with steps involved.

Sol $T(n) = 2T(n/2) + \Theta(n)$ — ①

$T(n/2) = 2T(n/2^2) + cn/2$ → subs. in ①

$T(n) = 2\left[2T\left(n/2^2\right) + cn/2\right] + cn$

$T(n) = 4T\left(\frac{n}{2^2}\right) + cn + cn$ — ②

after solving for k steps.

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

assuming $\frac{n}{2^k} = 1$ for getting $T(1)$

$n = 2^k$ and $k = \log n$

$$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^k}\right) + \log_2 n\,(cn)$$

$T(n) = nT(1) + \log_2 n\ (cn)$

assuming $T(1) = 0$

$T(n) = cn \log n$

$\boxed{T(n) = \Theta(n \log n)}$

# Upper bound for $T(n) = 2T(n/2) + \Theta(n)$

$T(n) \leq 2T(n/2) + cn$ — (1) for some (+)ve const c

Guess:- $T(n) \leq dn \log n$ for some (+)ve const d

$T(n/2) \leq \frac{dn}{2} \log(n/2)$ substituting in (1)

$T(n) \leq 2\left[\frac{dn}{2} \log(n/2)\right] + cn$

$\leq \frac{2dn}{2} \log(n/2) + cn$

$\leq dn \log n - dn \log_2 2 + cn$

$\leq dn \log n - n(d - c)$

if $(d - c) > 0$

$d > c$

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

Q Lower bound for $T(n) = 2T(n/2) + \Theta(n)$

$T(n) \geq 2T(n/2) + cn$ — ① for some positive const. c

Guess :- $T(n) \geq dn\log n$ for some const. d.

$T(n/2) \geq \frac{dn}{2}\log(\frac{n}{2})$ substituting in ①

$T(n) \geq 2[\frac{dn}{2}\log(\frac{n}{2})] + cn$

$\geq dn\log n - dn + cn$

$\geq \underbrace{dn\log n}_{\text{desired}} - \underbrace{n(d-c)}_{\text{residual}}$

If $d - c \leq 0 \implies c \geq d$ $\quad T(n) = O(n\log n)$

Q ④ Write a short note on Merge sort with an example (with working) write pseudo code for recursive & iterative method. Explain time complexity briefly.

Sol Merge sort is a divide and conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge function() is used for merging two halves. The merge (arr, l, m, r) is key process that assumes that assumes arr[l..m] and arr[m..r] are sorted and merges the two sub-sorted arrays into one.

* Working of merge sort :-

To understand merge sort we take an unsorted array as following →

| 10 | 14 | 28 | 11 | 7 | 16 | 30 | 50 | 25 | 18 |
|---|---|---|---|---|---|---|---|---|---|

we know that merge sort first divides the whole array iteratively into equal halves unless individual element array is achieved. we see that an array of 10 elements is divided into 2 arrays of size 5.

| 10 | 14 | 28 | 11 | 7 | 16 | 30 | 50 | 25 | 18 |
|---|---|---|---|---|---|---|---|---|---|

This does not change the sequence of appearance of items in the original. Now we divide these 2 arrays into halves.

| 10 | 14 | 28 | 11 | 7 | 16 | 30 | 50 | 25 | 18 |
|---|---|---|---|---|---|---|---|---|---|

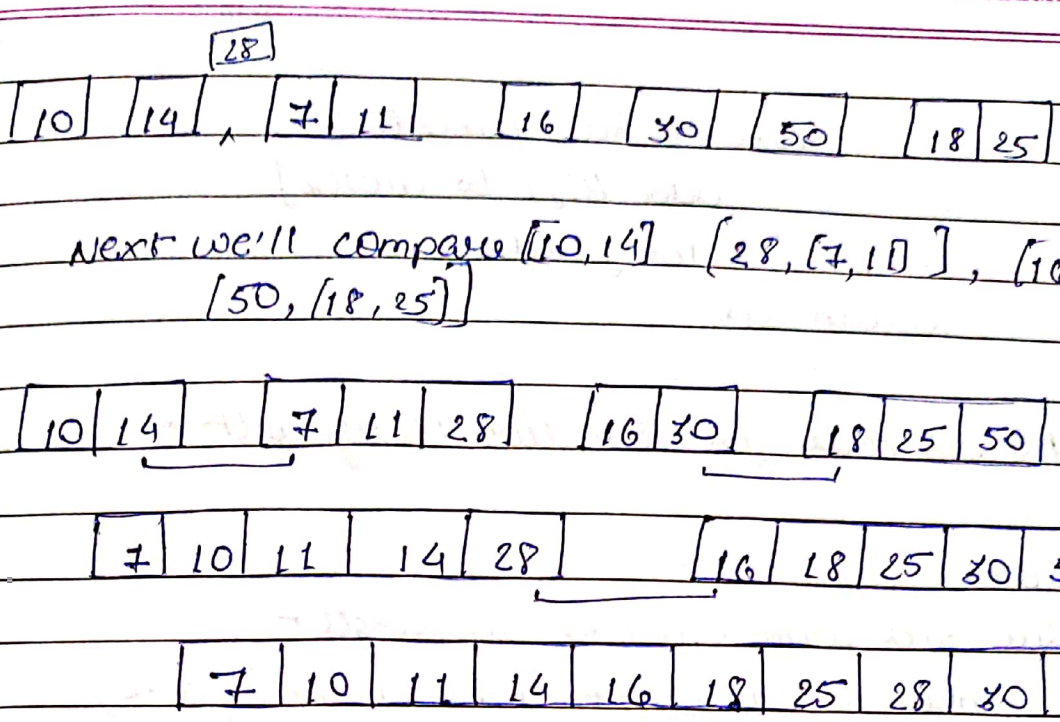
we further divide these arrays till we achieve atomic value which can be no more divided.

| 10 | 14 | 28 | 11 | 7 | 16 | 30 | 50 | 25 | 18 |
|---|---|---|---|---|---|---|---|---|---|

| 10 | 14 | 28 | 11 | 7 | 16 | 30 | 50 | 25 | 18 |
|---|---|---|---|---|---|---|---|---|---|

Now, we combine them exactly in the order as they were broken down.

The last broken list was 11, 7 and 25, 18 arranging & merging them :




Next we'll compare [10,14] [28,[7,10]], [16,30], [50,[18,25]]

* Pseudo code for recursive of merge sort.

```
If length of array greater than 1
    mid = array length // 2
    L = 1st sorted array [: mid]
    R = 2nd array [mid :]
    Merge sort (L, R)
i, j, k = 0
    while (L and R have elements)
    if L[0] > R[0]
    | add L[i=0] to ~~k++~~] arr[k]
      increment i
    else
      add R[j] to arr[k]
      inc. j
    k += 1
while L has elements
    add L[i] to arr[k]
    inc i and k
```

WILLMAN

while R has elements:
    add R[j] to arr[k]
    inc j and k
Return arr

8. Pseudo code for iterative Mergesort -

size = 1
while size become equal to arr length -
    mid = min of (left+size-1) and (arr length - 1)
    right = (2* size + left - 1, (arr len - 1)) if
        2 * size + left - 1 > (len(arr) - 1)

    n1 = mid - left + 1
    n2 = right - mid
    L, L = [0] * n1, [0] * n2
    for i in range 0, n1
        L[i] is arr[left + i]
    for in range 0, n2
        R[i] is arr[mid + 1 + i]

    i, j, k = 0, 0, left
    while n1 and n2 have elements
        if L[i] > R[j]
            add R[j] to arr[k]
            inc j
        else :
            add L[i] to arr[k]
            inc. i
    k += 1

while n1 has elements
add L[i] to arr[k]
inc i and k

while n2 has elements
add R[j] to arr[k]
inc j & k

Left = Left + size * 2
size = 2 x size
return arr.

Time Complexity of Merge sort
Time complexity can be expressed as following recurrence relation:
$T(n) = 2T(n/2) + \Theta(n)$

and the solution of the recurrence is $\Theta(n \log n)$

Time complexity of Merge sort is $\Theta(n \log n)$ in all 3 case (worst, average, and best)

Q5) Write pseudo code and explain with example different partition algorithms of quick sort. Explain briefly the time complexity of quick sort.

Sol^n Pseudo code & exp. of different partition algorithms.

```
Partition (A, p, q)        # A[p --- q]
    x ← A[p]
    i ← p[index]
    for j ← p+1 to q
        if A[j] ≤ x
            then i = i+1
            exchange A[i] ↔ A[j]
        exchange A[p] ↔ A[i]
    return i

Quick sort (A, p, q)
    if p<q
        then q ← partition (A, p, q)
        Quick sort (A, p, r-1)
        Quick sort (A, r+1, q)

Quick sort (A, l, n)        # invoking func.
```

Ex arr = [10, 80, 30, 90, 40, 50, 70]

1) define low and high to be 0 and 6

2) and arr[h] i.e last element to be pivot

define i and j to be 0 and employ loop for i and j

~~In fi~~

3) In first iteration i and j are both 0
∴ no swapping, increment i and j

4) when i=1, j=1
compare arr[j] with pivot
arr[j] > pivot
increment j

5) i=1, j=2 on comparing arr[j] with pivot
arr[j] < pivot
swap both
arr = [10, 30, 80, 90, 40, 50, 70]
inc. i, j.

6) i=2, j=3 arr[j] > pivot
increment j, no swapping.

7) i=2, j=4 arr[j] < pivot
swap both & inc. i, j
arr = [10, 30, 40, 90, 80, 50, 70]

⑧ now j=5 arr[j] ≤ pivot swap both
i=3 and increase i,j

arr: [10, 30, 40, 50, 80, 90, 70]

j=6 Break and swap arr[i] with arr[pivot]
i=4

arr: [10, 30, 40, 50, 70, 90, 80]

⑨ now again set i,j=0 and pivot value arr[-1] = 80
and repeat the ~~#~~ iteration. (Using recursion).

Till we achieve arr = [10, 30, 40, 50, 70, 80, 90]

# HOARE's Partition Algorithm (Pseudo code)

```
x = A[p]
i = p-1
j = q+1
while True
    repeat
        j = j-1
    until A[j] ≤ x
    repeat
        i = i+1
    until A[i] ≥ x
    if i < j
        exchange A[i] with A[j]
    else
        return j
exchange a[0] with a[j]
```

Ex)

100 250 25 800 65 360 3 480
pivot → i, j ←, j

100 3 25 800 65 360 250 480
i, j

100 3 25 65 800 360 250 480
p ←→ j, i

65 3 25 | 100 | 800 360 250 480
≤ 100 ≥ 100

Date : / / Page
WILLMAN

* Time complexity →

$T(n) = T(k) + T(n-k-1) + \Theta(n)$

# Worst case → $T(n) = T(n-1) + \Theta(n)$

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

# Best case → $T(n) = 2T(n/2) + \Theta(n)$

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

# Average case →

If we case where partition puts $O(n/9)$ elements in one set & $O(9n/10)$ in another set.

$T(n) = T(n/9) + T(9n/10) + \Theta(n)$

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