Sagar
Date:
Page:

Searching/ Sorting Techniques	Time Complexity			Space complexity		
	Best	Worst	Avg case	Best	Worst	Avg
Linear Search	$\Omega(1)$	$O(N)$	$\Theta(N)$	-	-	$\Theta(1)$
Binary Search	$\Omega(1)$	$O(\log_2 N)$	$\Theta(\log_2 N)$	-	-	$\Theta(1)$
Bubble Sort	$\Omega(N)$	$O(N^2)$	$\Theta(N^2)$	-	-	$\Theta(1)$
Selection Sort (No best case and worst case.)			$\Theta(N^2)$	-	-	$\Theta(1)$
Insertion Sort	$\Omega(N)$	$O(N^2)$	$\Theta(N^2)$			$\Theta(1)$
Merge Sort			$\Theta(N \log_2 N)$			$O(N)$
Quick Sort	$\Omega(N \log_2 N)$ Not sorted		$O(N^2)$ Sorted			

$N + 1 + \log_2 n + 1 = O(N)$

Merge Sort

Space complexity

↓

Auxiliary space (used)

Sorting Techniques

Bubble Sort
Selection
Insertion
} Brute Force

Divide & Conquer
- Quick sort
- Merge Sort
- Heap sort

} Tree based sorting

Counting Sort
Radix Sort
} Linear sorting
Stable sorting technique

Bubble Sort

First comp

100 70 90 60 40 50

i: ① → 70 90 60 40 50 (100)

comp

→ 70 60 40 50 (90) 100

→ 60 40 50 (70) 90 100

→ 40 50 (60) 70 90 100

→ 40 (50) 60 70 90 100

Total no - 6
no of iteration 5 [No of comparision in 1st step
no of steps - 5

Sagar
Date:
Page:

~~Alg~~ ALGORITHM Bubble Sort (A[], N)
BEGIN:

```
FOR i = 1 TO N-1 DO
    FOR j = 1 TO N-i DO
        IF (A[j] > A[j+1]) THEN
            Exchange (A[j], A[j+1])
END;
```

Space complexity - $\theta(1)$

Exchange statement

i =	1	2	---	N-1
	N-1	N-2		
	(N-1)+3(N-1)	i		
	=4(N-1)	4(N-2)		4.1

$= 4(1+2+3 \text{ -- } -(N-1)$

$= 4\Sigma(N-1) = \frac{4^2 N(N+1)}{2}$

$2N^2 - 2N$

$N^2 > N = 2N^2$

$= N^2$

$O(N^2)$

Best case

Inputting no weight all sorted	10	50	60	65	(70) → (N-1)
	10	50	60	(65)	700
	1				
	!			↓	

Total cas (N-1)

$\Omega(N)$

Sagar
Date:
Page:

Improved Bubble Sort

Best Case

```
ALGORITHM BubbleSort(A[], N)
BEGIN:
    Flag = 1
    FOR i=1 TO N-1 and Flag == 1 DO
        Flag = 0
        FOR j=1 TO N-i DO
            IF (A[j] > A[j+1]) THEN
            Exchange (A[j], A[j+1])
            Flag = 1
END;
```

Selection Sort

				min	
100	70	90	60	40	50
(40)	70	90	60	100	50
	(50)	90	60	100	min 70
		(60)	90	100	70
			(70)	min 90 100	
					(90) 100

```
ALGORITHM SelectionSort(A[], N)
BEGIN:
    FOR i=1 TO N-1 DO
        min = i
        FOR j=i+1 TO N DO
            IF A[j] < A[min] THEN
                Min = j
        Exchange (A[min], A[i])
END;
```

Eg min 10 70 60 40 50

Sagar
Date:
Page:

i=1 i=2 - - - i=N-1

4+2(N-1) 4+2(N-2) 4+2

$(n-1)4 + 2\left(\frac{n(n-1)}{2}\right)$

$= 4(n) + n^2 - n = 3n + n^2 - 4 = n^2 > n$

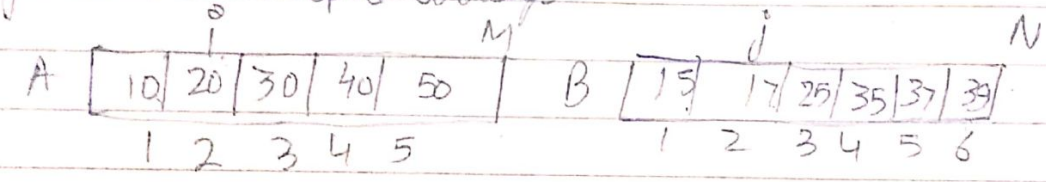
$= N^2$

$O(N^2)$ Worst case

$\Omega(N^2)$

Best case ~~$(N+1) = \Omega(N)$~~

Merge the given sorted array such that the resulting array is also sorted and it contains all elements of both the input arrays?



10	15	17	20	25	30	35	37	39	40	50

MergeArray(A[], M, B[], N)

C[M+N]

i=1

j=1

k=1

WHILE i<=M and j<=N DO

IF A[i] < B[j] THEN

C[k] = A[i]

k++

i++

AUB

IF A[i] == B[j]

C[k] = A[i]

i++

k++

j++

Sagar
Date:
Page:

```
        ELSE
            C[K]=B[j]
            R++
            j++
    WHILE i<=M DO
        C[R]=A[i]
        K++
        i++
    WHILE j<=N DO
        C[R]=B[j]
        R++
        j++
    RETURN C
END;
```

Time Complexity - O(m+n)

Space Complexity - O(m+n)

A=[10,20,30,40,50], B=[9,17,20,33,37,39

1) AUB

```
ALGORITHM UNION(A[],M,B[],N)
C[M+N]
    i=1
~~A∩B~~ j=1
    R=1
WHILE i<=M and j<=N DO
        IF A[i] == B[j]
            C[R]=A[i]
            i++
            j++
            R++
```

```
ELSEIF A[i]<B[j] THEN
        C[R]=A[i]
          R++
          i++
    ELSE
        C[R]=B[j]
          R++
          j++
```

Same

A∩B

```
ALGORITHM INTERSECTION(A[],M,B[],N
C[M+N]
      i=1
      j=1
      R=1
WHILE i<=M and j<=N Do
        IF A[i]==B[j]
            C[R]=A[i]
              i++
              j++
              R++
        ELSE
            IF A[i]>B[j] THEN
                j=i++
            ELSE
                j++
```

Q A-B

Only elements of A not in B

```
WHILE i<=M AND j<=N DO
IF A[i] == B[j] THEN
        i++ , j++
ELSE
    IF A[i] <B[j] THEN
        C[R]=A[i]
        R++
        i++
    ELSE
        j++
WHILE i<=M DO
    C[R] = A[i]
    R++
    i++
WHILE j<=N DO
```

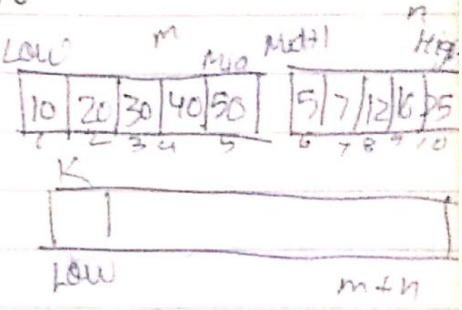
Q B-A

```
WHILE i<=M AND j<=N DO
IF A[i] == B[j] THEN
        i++
        j++
ELSE
        IF B[j] <A[i] THEN
            C[R]= B[j]
            R++
            i++
        ELSE i++
```

Sagar
Date:
Page:

```
WHILE (i <=N) DO
    C[R] = B[j]
        R++
        j++
```

```
ALGORITHM MERGE (A[], Low, Mid, High)
BEGIN:
        i=Low , j=Mid+1    k=Low
        WHILE i<=Mid AND j<=High DO
            IF A[i]<A[j] THEN
                C[K]=A[i]
                R++, i++
            ELSE
                C[K]=A[j]
                K++, j++
WHILE i<=mid DO
        C[R] = A[i]
            R++, i++
    WHILE j<=High DO
            C[R]=A[j]
            R++, j++
FOR i = Low TO High DO
        A[i] = C[i]
```



$$C\left(\frac{N}{2}+\frac{N}{2}\right)+N$$

$$C(N+N) = C+N$$

$$O(N)$$

Space complexity $= O(N)$

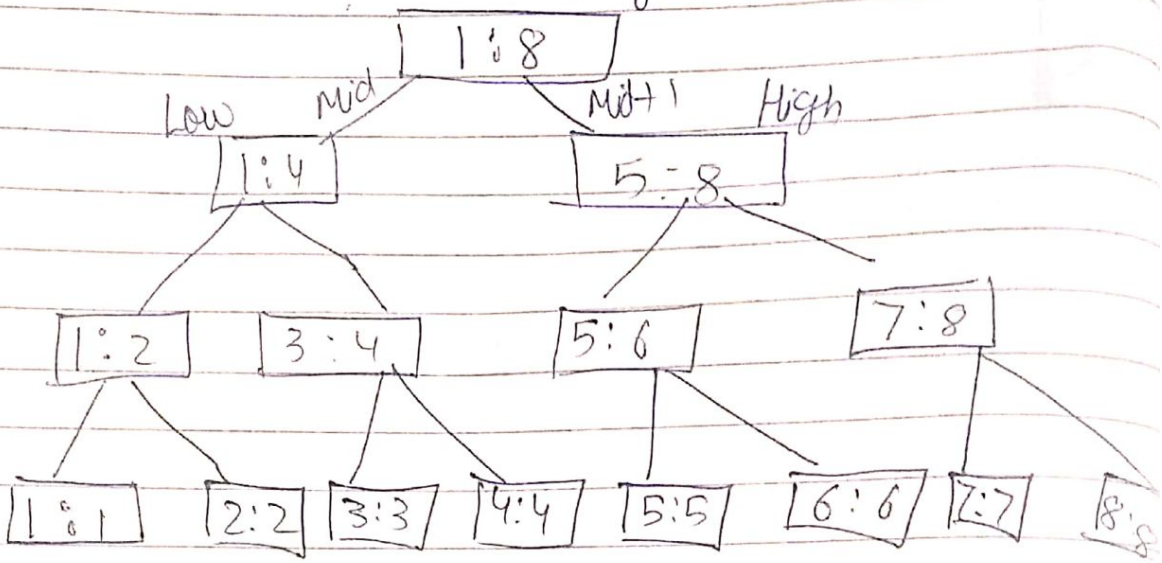
$$\left(\frac{N}{2}+\frac{N}{2}\right)+3 = O(N$$

Sagar
Date:
Page:

Solving bigger problem is tough than smaller one

Divide & Conquer

Merge Sort



ALGORITHM MergeSort ~~Merge Sort~~ (A[], Low, High) Let T(N)

BEGIN:

IF Low < High THEN

MID = $\left\lfloor \frac{Low + High}{2} \right\rfloor$ 1

MergeSort (A[], Low, Mid) $T(\frac{N}{2})$

MergeSort (A[], Mid+1, High) $T(\frac{N}{2})$

Merge (A[], Low, Mid, High) $O(N)$

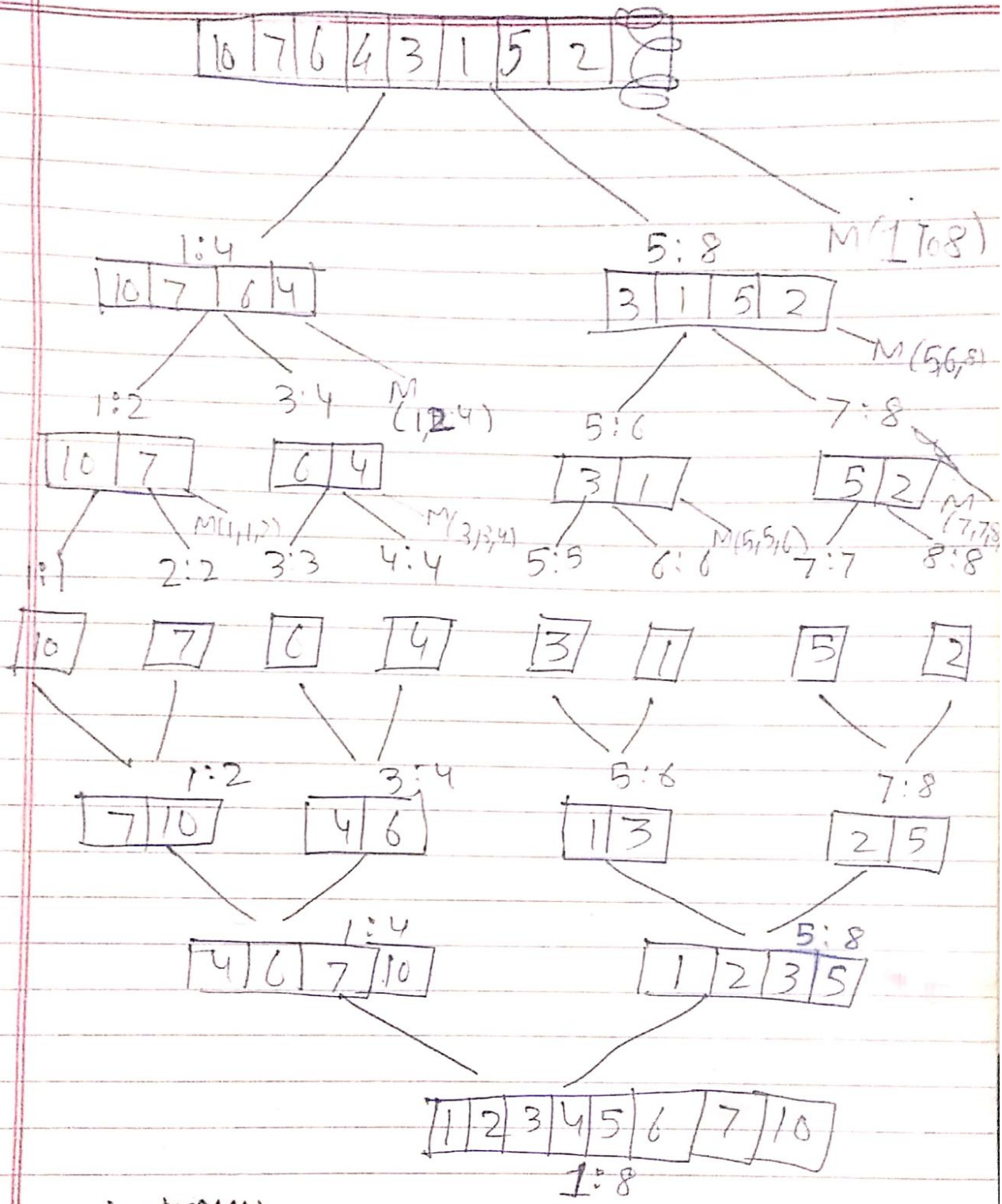
END;

$$T(N) = 1 + T(\tfrac{N}{2}) + T(\tfrac{N}{2}) + O(N)$$

$$T(N) = 2T(\tfrac{N}{2}) + CN + 1$$

$$T(N) = 2T(\tfrac{N}{2}) + O(N)$$

$$T(N) = 2T(\tfrac{N}{2}) + C'(N)$$



As recursive process

$$\Rightarrow T\left(\frac{N}{2}\right) = 2T\left(\frac{N}{4}\right) + C'\left(\frac{N}{2}\right)$$

$$\Rightarrow T(N) = 2\left(2T\left(\frac{N}{4}\right) + C'\left(\frac{N}{2}\right)\right) + C'(N)$$

$$T(N) = 2^2 T\left(\frac{N}{2^2}\right) + C'N + C'(N)$$

$$= 2^2 T\left(\frac{N}{2^2}\right) + 2C'N = 4\left[2T\left(\frac{N}{2^3}\right) + C'\left(\frac{N}{2^2}\right)\right] + 2C'(N)$$

Sagar
Date:
Page:

$$= 2^3 T\left(\frac{N}{2^3}\right) + 3c'(N)$$

In K^{th} element we have 1 elements

$$= 2^K T\left(\frac{N}{2^K}\right) + K c'(N)$$

$$= N\,T(1) + (\log_2 N)\,c'(N) \qquad \Rightarrow \frac{N}{2^K} = 1$$

$N = 2^K$

$\log_2 N = K$

Effort to sort 1 element

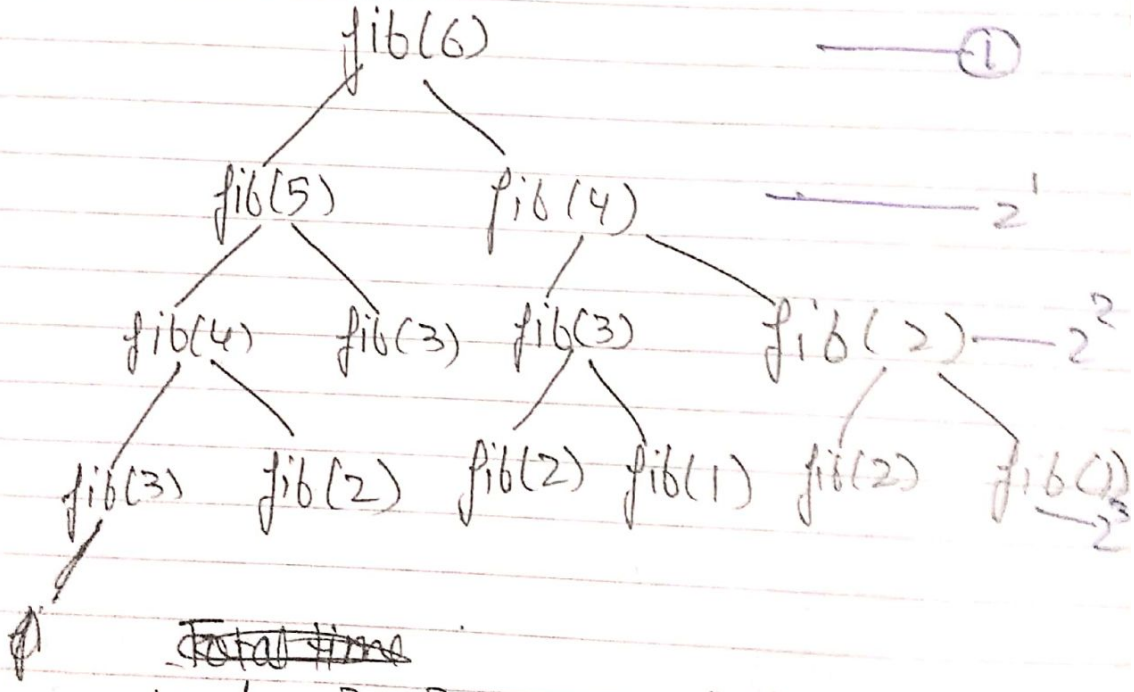
$$= N \cdot (1) + (\log_2 N)\,O(N)\,c'$$

Time complexity of merge sort on an array of size 1

$$N + c' N \log_2 N = O(N \log_2 N)$$

```
ALGORITM Fibonacci(N)
BEGIN:
    IF N==1 THEN
        RETURN 0
    ELSE
        IF N==2 THEN
            RETURN 1
        ELSE
            RETURN Fibo(N-1)+Fibo(N-2)
END;
```

Fib(n



~~Total time~~

$1+2^1+2^2+2^3+---+2^{n-3}$

GP ⇒:

$$\frac{1(2^{n-2}-1)}{2-1} = (2^{n-2}-1)$$

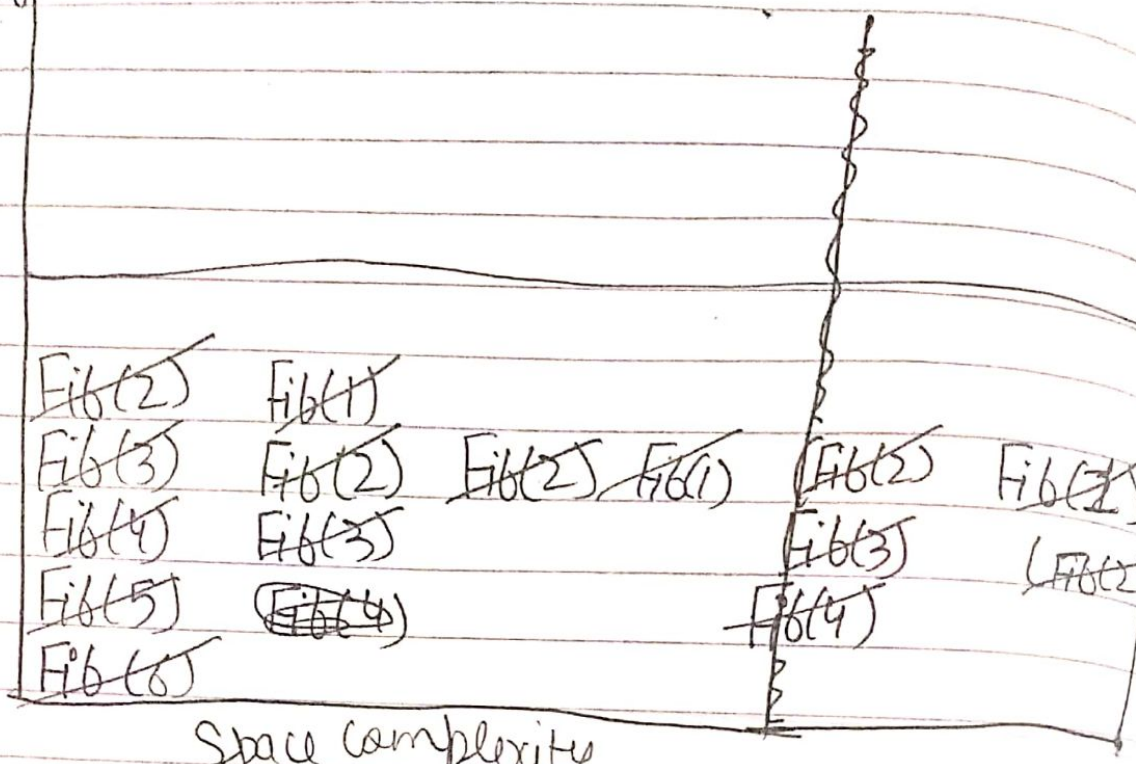
Total time of insertion deletion of activation Record

$(2^{n-2}-1)+2c = (2^{n-1}-2)c = \mathcal{O}(2^n)$ → Time complexity

Sagar

Date:
Page:

Stack flow



Space complexity

5·C

(n-1) C

$\Theta(n)$

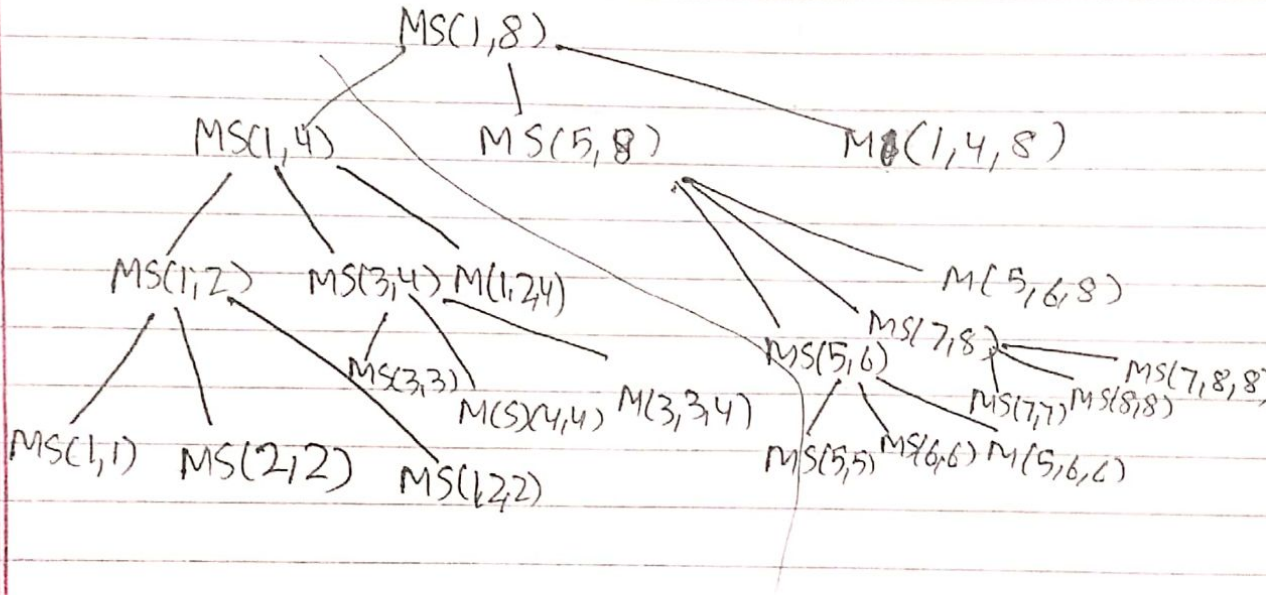
Sagar
Date:
Page:

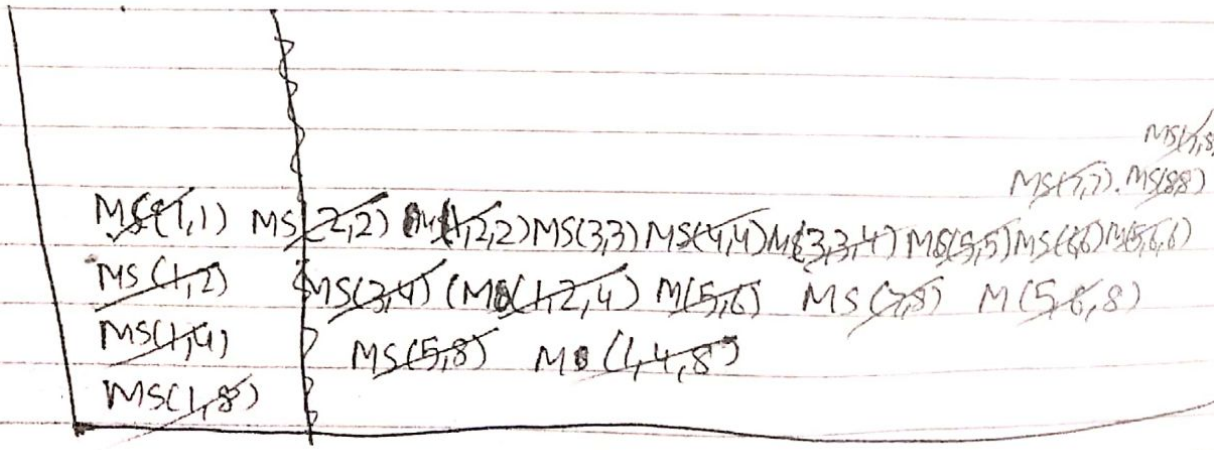
ALGORITHM Merge Sort. (A[ ], Low, High)
BEGIN:

IF Low < High THEN

$$Mid = \left\lfloor \frac{Low + High}{2} \right\rfloor$$

MergeSort (A, Low, Mid)
MergeSort (A, Mid+1, High)
Merge (A, Low, Mid, High)





Space complexity

Sagar
Date:
Page:

$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$

4	3+1
5	4+1
6	5+1
7	6+1

$\log_2 64 = \log_2 2^6 = 6$

$N\text{elument} = (\log_2 N + 1)\, c$
$= \Theta(N)$

Assignment

Merge Sort (1, 16)

Partition

Sagar
Date:
Page:

1	2	3	4	5	6	7	8	9	10
12	9	15	6	20	8	10	60	7	∞

Pivot = 12

i = 1 , j = 10

i < j

1 < 10

i = 3 , j = 9

A[3] ↔ A[9]

1	2	3	4	5	6	7	8	9	10
12	9	7	6	20	8	10	60	15	∞

i < j

A[5] ↔ A[7]

1	2	3	4	5	6	7	8	9	10
12	9	7	6	10	8	20	60	15	∞

i > j

A[1] → A[6]

1	2	3	4	5	6	7	8	9	10
8	9	7	6	10	12	20	60	15	∞

```
ALGORITHM Partition (A[], Low, High)
BEGIN:
    i = Low
    j = High+1
    Pivot = A[Low]
    DO
        DO
            i++
            WHILE (A[i] < Pivot)
        DO
            j--
            WHILE (A[j] > Pivot)
        IF i < j THEN
            Exchange (A[i], A[j])
```

Sagar
Date :
Page :

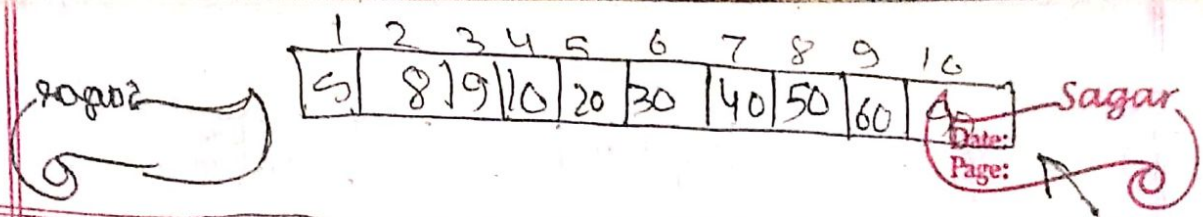
```
WHILE (i<j)
Exchange (A[Low], A[j])
RETURN j
END;
```

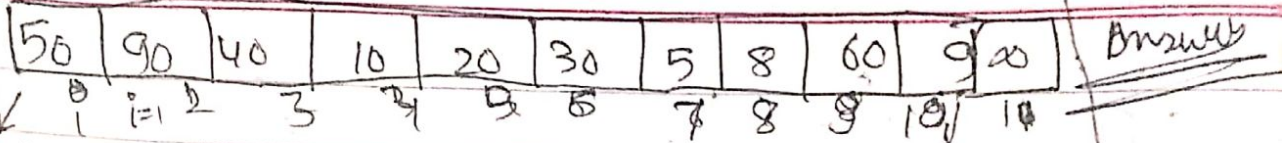
Space complexity = 3 $O(1)$

Time complexity $= 3 + N + \frac{N}{2} \times 3 + 3 + 1$

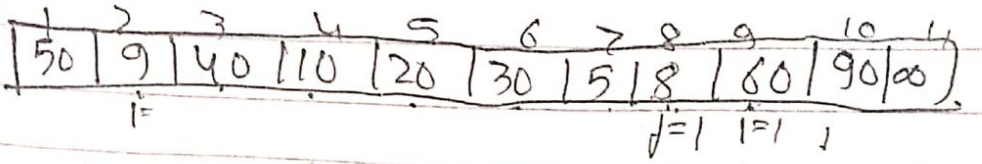
- N: Total time it will move
- $\frac{N}{2} \times 3$: Exchange func in loop
- 3: Exchange
- $+1$

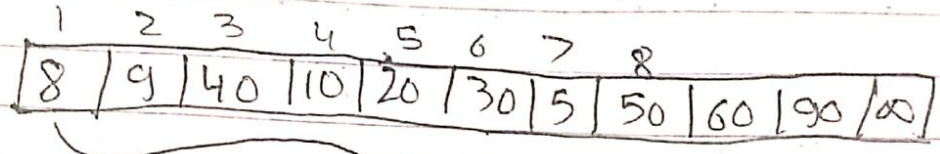
$= \frac{5N}{2} + 7 = O(N)$

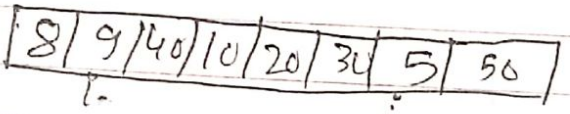


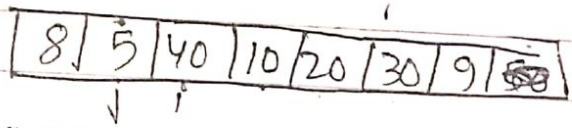


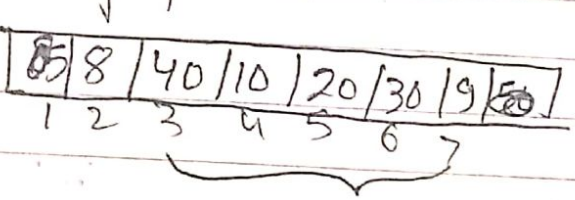
Pivot



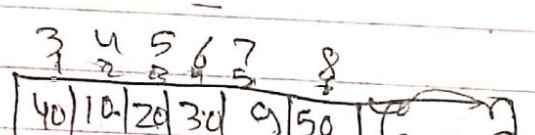


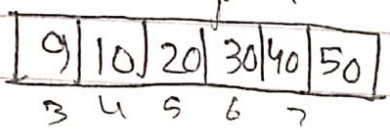






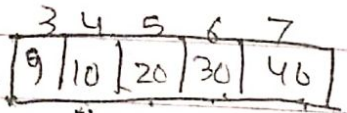
Pivot=10

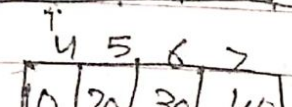


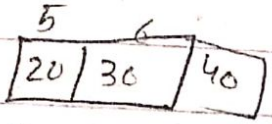


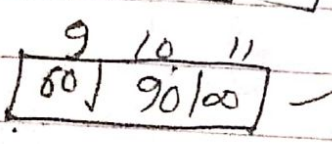
9 fixed

a[j]=Pivot







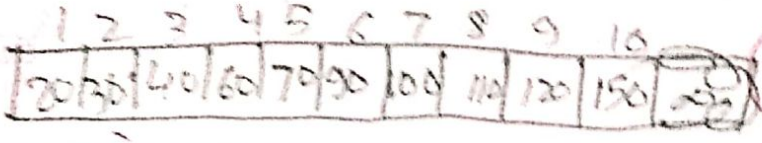


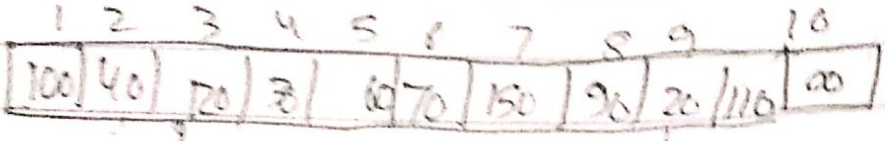
Scanned with CamScanner

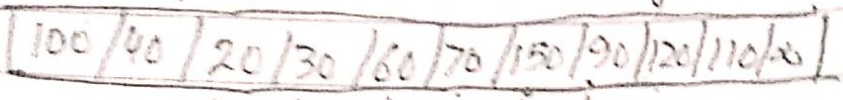
Sagar

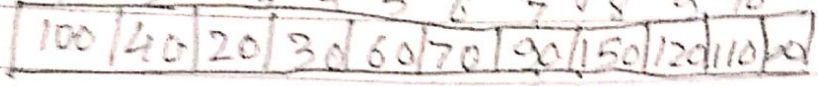
Date:
Page:

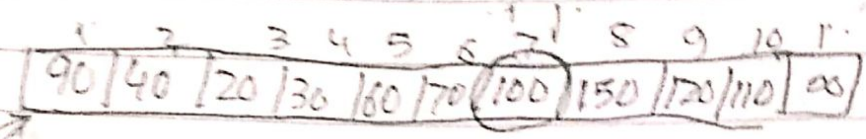
Quick

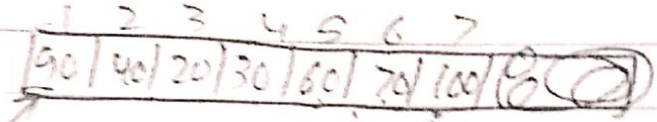


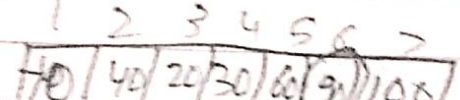


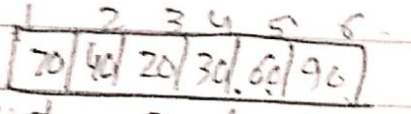


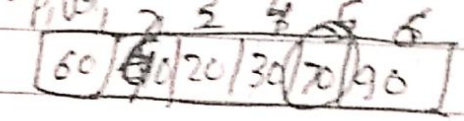


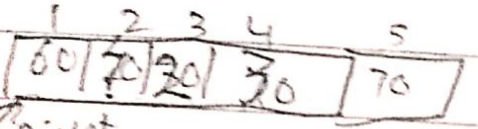


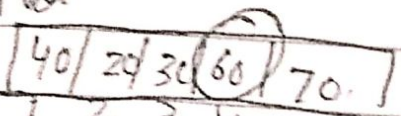


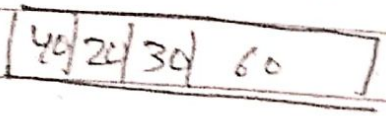


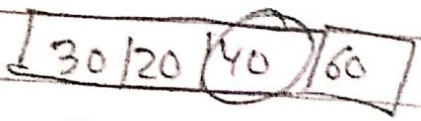


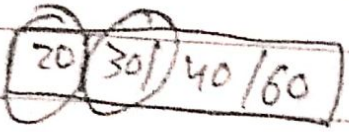


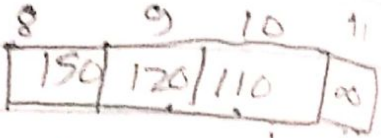


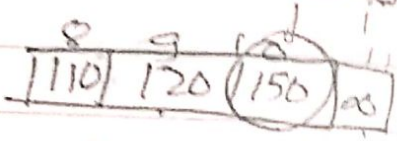


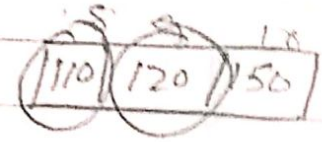












j = Hight + 1 → in Partition

Sagar
Date:
Page:

Quick Sort (A[], Low, High)

ALGORITHM Quick Sort (A[], Low, High)
BEGIN:
IF Low < High THEN (if we have zero or one element in array no partition can be performed)
j = Partition (A, Low, High)
Quick Sort (A, Low, j-1)
Quick Sort (A, j+1, High)
END;

Best Case

$$T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{2}\right) + \Theta(N)$$

$$T(N) = 2T\left(\frac{N}{2}\right) + cN$$

$$T\left(\frac{N}{2}\right) = 2\left[2\left(T\left(\frac{N}{4}\right) + \frac{cN}{2}\right)\right] + cN$$

$$= T\left(\frac{N}{2}\right) = 4T\left(\frac{N}{4}\right) + 2cN$$

$$T\left(\frac{N}{8}\right) = 4\left[2T\left(\frac{N}{8}\right) + \frac{cN}{2^2}\right] + 2cN$$

$$T(N) = 8T\left(\frac{N}{8}\right) + 3cN$$

$$T(N) = 2^3 T\left(\frac{N}{2^3}\right) + 3cN$$

$$T(N) = 2^R T\left(\frac{N}{2^R}\right) + RcN$$

Time complexity of Quick sort on an array of size 1

$$T(N) = N(1) + \log_2 N\, cN \qquad \frac{N}{2^R} = 1 \Rightarrow R = \log_2 N$$

$$T(N) = N + N\log_2 N$$

$$T(N) = \Theta(N \log_2 N)$$

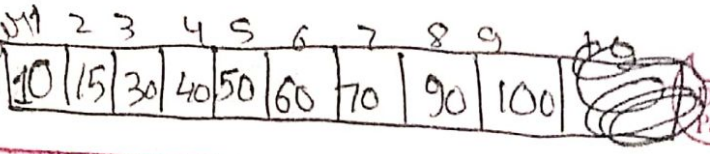
as $N\log_2 N > N$

$\therefore N\log_2 N = \mathcal{U}(N \log_2 N)$

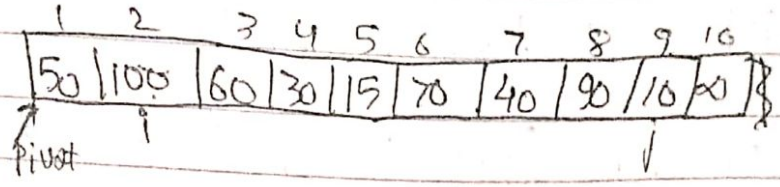
Sagar

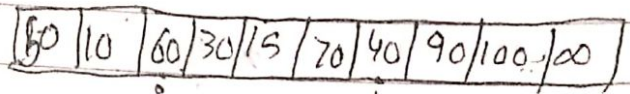
Date:
Page:

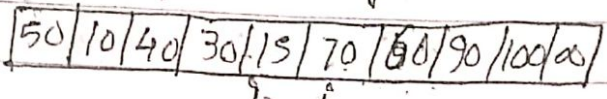
Answer

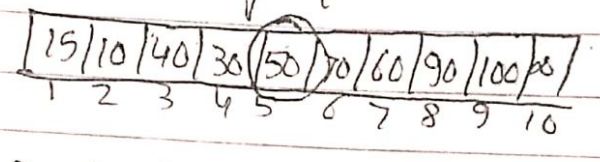


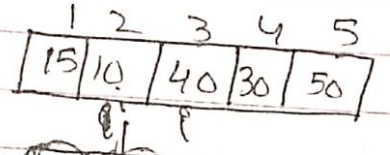
Bubble

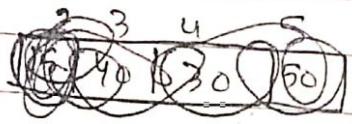


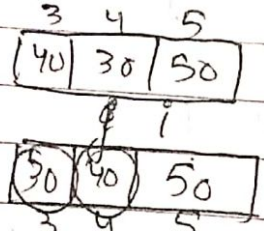


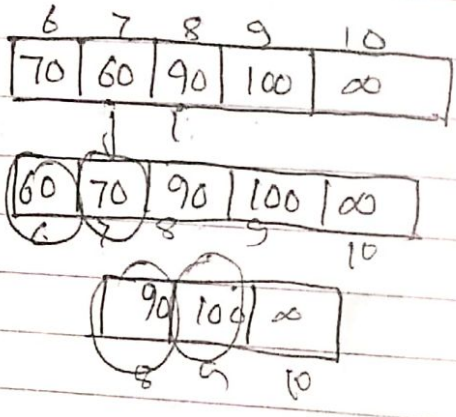


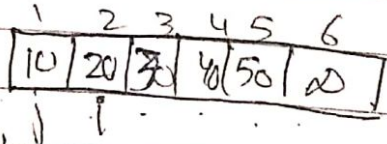


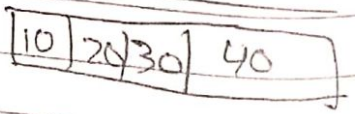


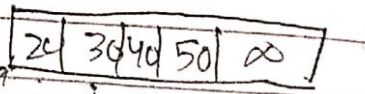


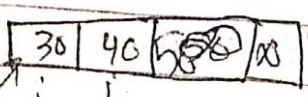


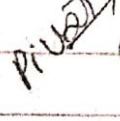












Sagar
Date:
Page:

6 Quick sort worst case occur when all elements are already sorted.

~~Burst time~~ Worst Case

$T(N) = T(0) + T(N-1) + \Theta(N)$

$= 1 + T(N-1) + cN$

$= T(N-1) + cN + 1$

$= T(N-1) + \Theta(N)$

$T(N) = T(N-1) + c'N$

$= T(N-2) + c'(N-1) + c'N$

$T(N) = T(N-3) + c'(N-2) + c'(N-1) + c'N$

$T(N) = T(N-R) + c'(N-R) + \dots + c'(N-(R-1))$.

$T(N) = T(2) + c'3 + \dots + c'(N-1) + c'(N)$

$= T(1) + c'2 + \dots + c'(N-1) + c'(N)$

$= 1 + c'(2 + 3 + \dots + N)$.

$= 1 + c'\left(\frac{N(N+1)}{2} - 1\right)$

$= 1 + c'\left(\frac{N^2}{2} + \frac{N}{2} - 1\right)$

$= \frac{c'N^2}{2} + \frac{c'N}{2} - c' + 1$

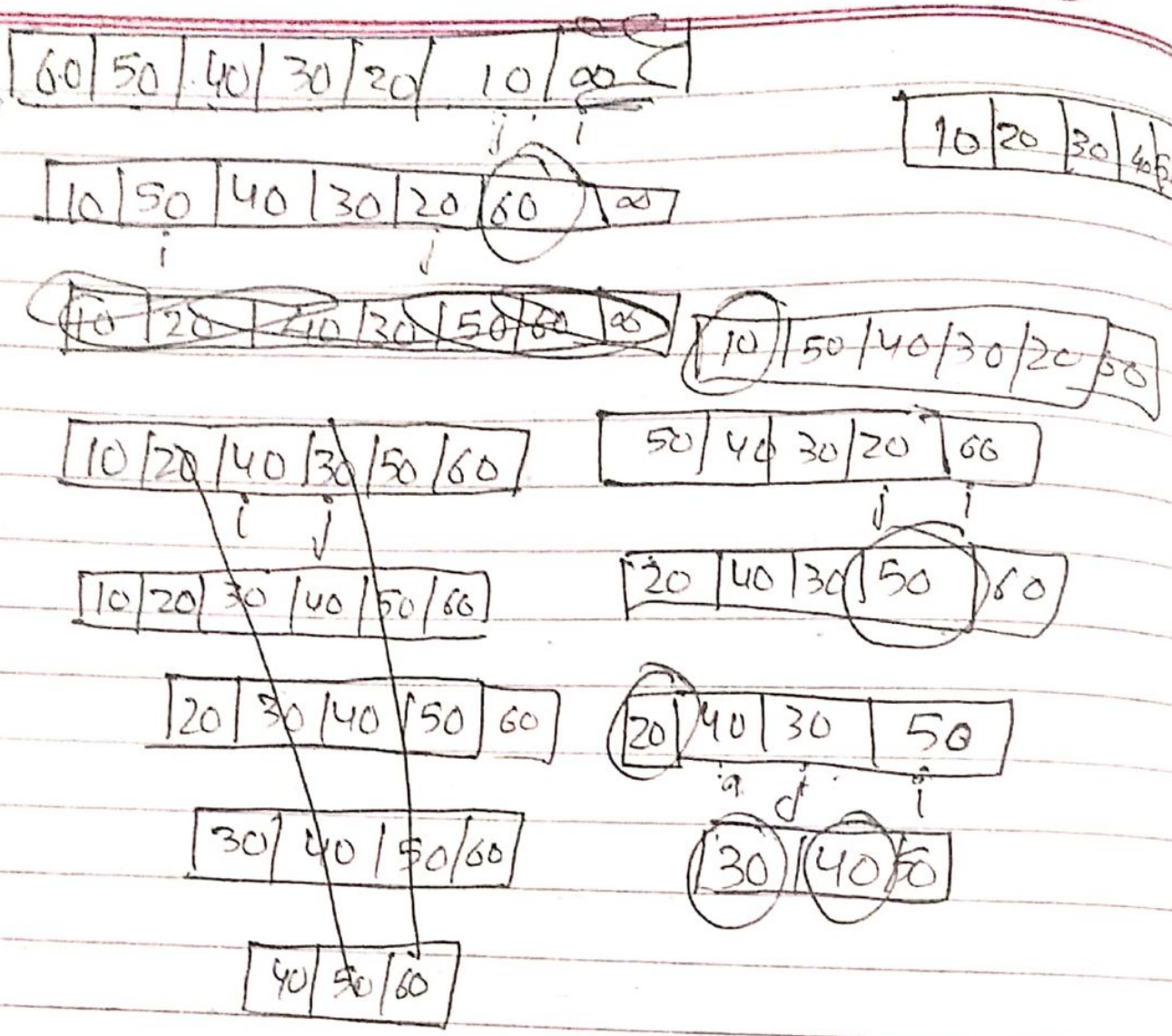
$= c'N^2$

$= O(N^2)$

pivot = i

sagar

Date :
Page :

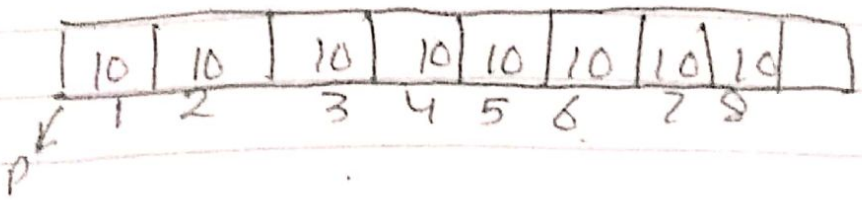


$$T(N) = T(N-1) + 0 + C(N)$$

$$= \Theta(N^2)$$

Sagar
Date:
Page:

* Abstract Data Type
* Time Space Trade off



$$T(N) = 2T\left(\frac{N}{2}\right) + \Theta(N)$$

Best case

$$= \Omega(N \log_2 N)$$

If nos are sorted and reversed then it is worst case because each time first element get fixed. If nos are equal then best case

$$= T\left(\frac{N}{3}\right) + T\left(\frac{2N}{3}\right) + \Theta(N)$$

$$= T\left(\frac{N}{4}\right) + T\left(\frac{3N}{4}\right) + \Theta(N)$$

$$= T\left(\frac{N}{5}\right) + T\left(\frac{4N}{5}\right) + \Theta(N)$$

⋮

$$= T\left(\frac{N}{9}\right) + T\left(\frac{8N}{9}\right) + \Theta(N)$$

As a result of partition if there are same nos on right hand side & same are on LHS than worst case would be reached

Sagar
Date:
Page:

```
Partition (A[], Low, High
    i = Low
    j = High + 1
    Mid = ⌊(Low + High)/2⌋
    Exchange (A[Low], A[Mid])
        Pivot = A[Low]
        Do
            Do
                i++
            WHILE(A[i] < Pivot)
            Do
                j--
            WHILE(A[j] > Pivot)
            IF i<j THEN
                A[i] <-> A[j]
        WHILE i<j
```

Randomized Quick Sort

```
r = Random(Low : High)
Exchange(A[Low], A[r])
```

In the normal working of Quick Sort, When the input nos are sorted or reverse sorted the partition leaves in only one half of the elements leaving other part to have zero elements. This way the recurrence of the Quick Sort becomes $T(N) = T(0) + T(N-1) + O(N)$
To avoid this 2 methods can be implayed to ensure two partitions on each time partition is called.

Sagar
Date:
Page:

1) Method 1 - Before Partition Mid Index as Mid = $\lfloor \frac{low+high}{2} \rfloor$ and exchange 1st element with mid index element

2) Select a random index in between low and high and exchange 1st element with the number at randomly selected index. before each partition.

The above 2 methods creates 2 partitions ~~in~~ rather than 1 in most of the cases.

r = rand() % (High - Low + 1) + Low

r = random (Low, High)

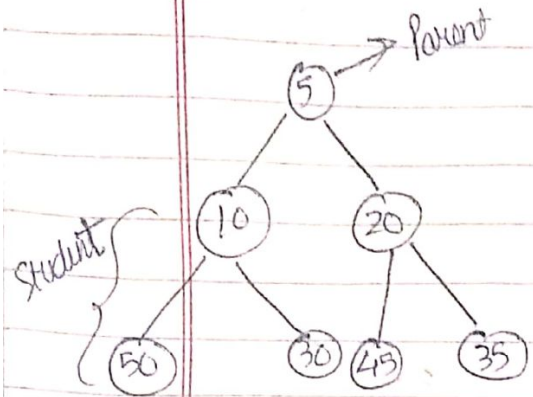
Exchange (A[Low], A[r])

Time Space complexity ⇒ If we improve time then expected space de

Merge Sort	Space
Tm nlogN	O(N)

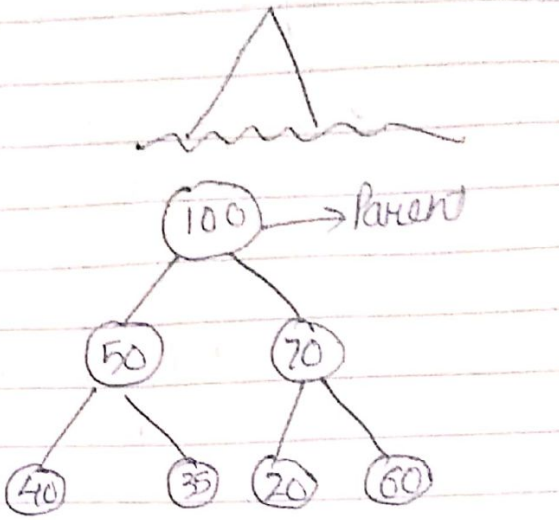
Sagar
Date:
Page:

Heap Sort



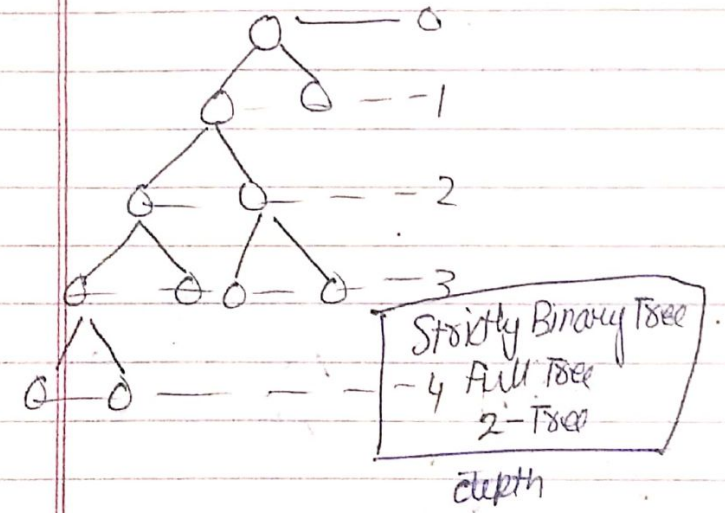
Min Heap

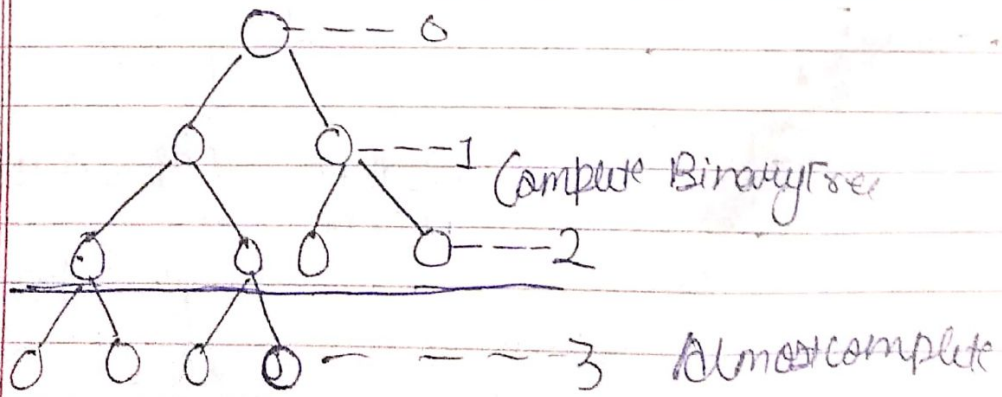
Parent less info than children



Max Heap

Parent more info than children





Heap is a Data Structure that satisfies 2 properties.

1) Shape of Heap should be almost complete Binary Tree.

2) Heap nodes follow specified order

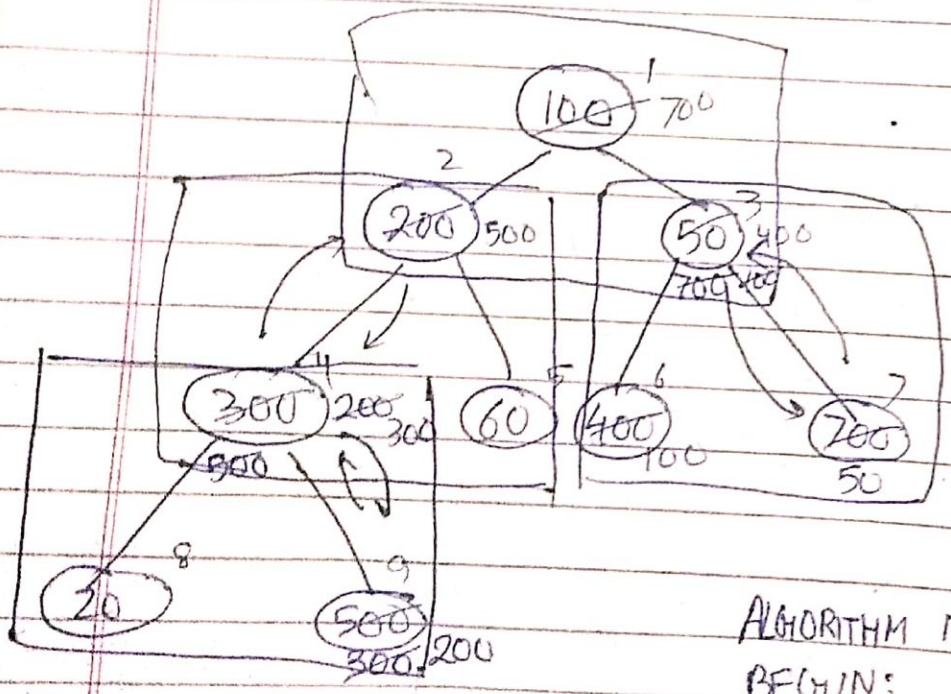
a) Max Heap - Parent nodes contain ~~#~~ larger info than child node

b) Min Heap - Parent node contains smaller info than child nodes

a) Max heap - Root node contains largest info

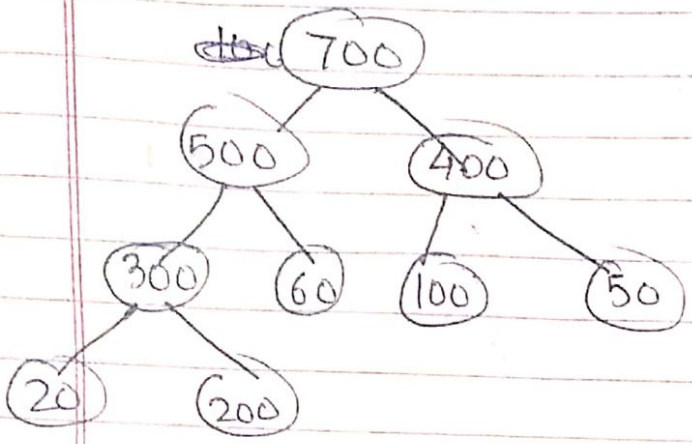
b) Min heap - Root node contains smallest info

A	100	200	50	300	60	400	700	20	500
	1	2	3	4	5	6	7	8	9

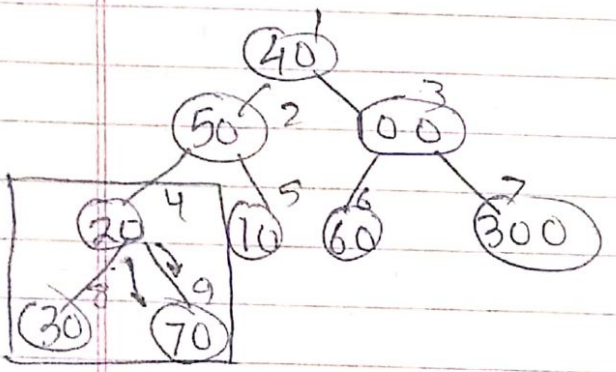


Adjust(A,4,9)
Adjust(A,3,9)
Adjust(A,2,9)

ALGORITHM Max Heapify(A[1,N])
BEGIN:
FOR i=⌊N/2⌋ TO 1 STEP -1 DO
Adjust(A,i,N)
END;

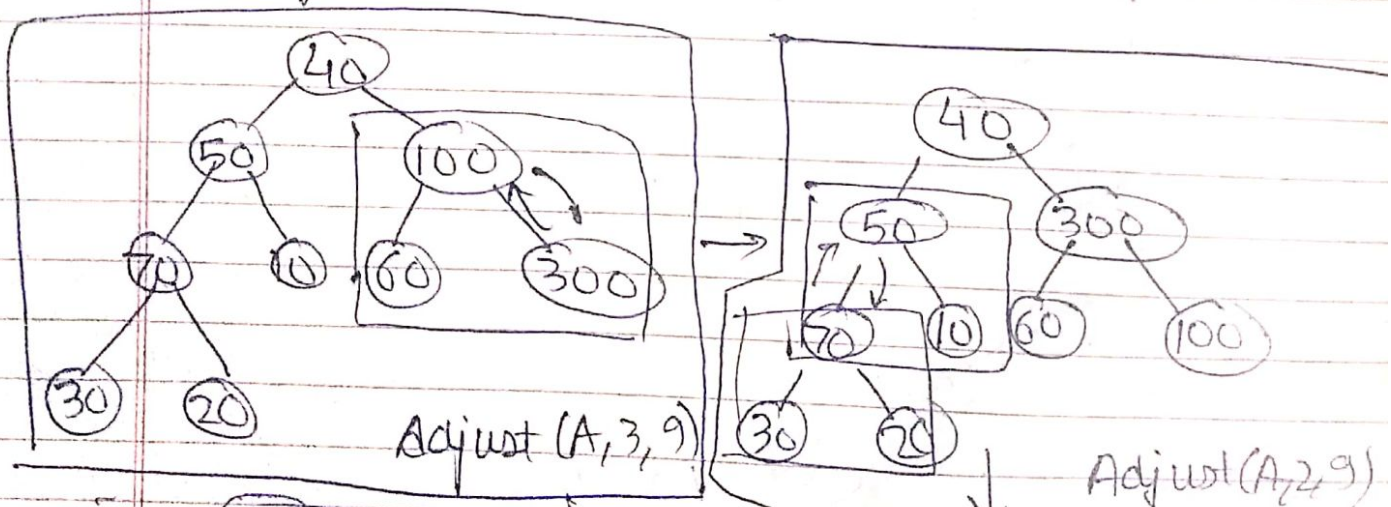


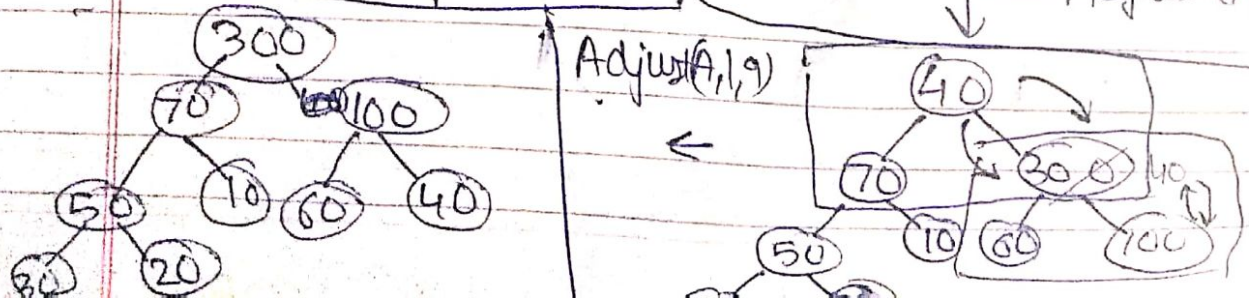
40	50	100	20	10	60	300	30	70



Adj

Adjust (A, 4, 9)





→ first find complexity of Adjust

Sagar
Date:
Page:

ALGORITHM HeapSort (A[ ], N)
BEGIN:
MaxHeapify(A, N) $\theta(N) = 2N$

FOR j = N TO 2 STEP -1 DO (N-1)
Exchang(A[1], A[j]) - 3
Adjust(A, 1, j-1) $-(\log_2 N \times 7)$

END.

$= 2N + (N-1)(3+7)$
$2N + 3N - 3 + 7N\log_2 N$
$= N\log_2 N$

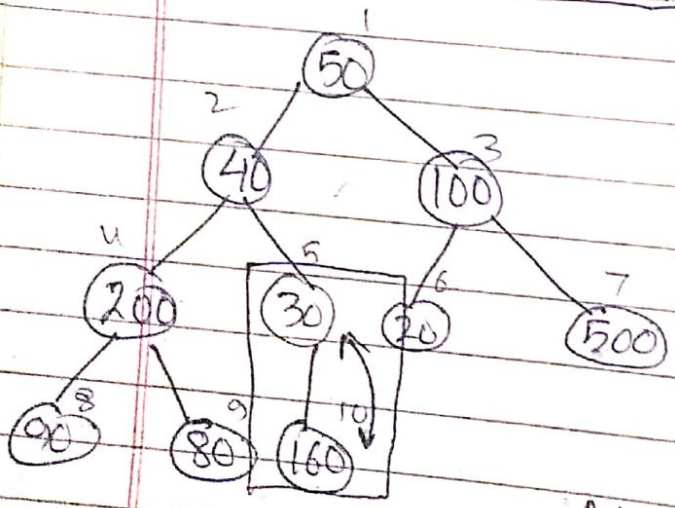
Time complexity
$= O(N\log_2 N)$

so that it

do not have to Exchange [A[1]], [A[1]] element

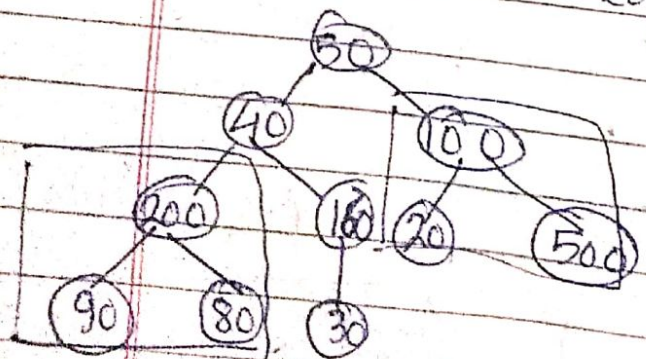
Perform HeapSort

1	2	3	4	5	6	7	8	9	10
50	40	100	200	30	20	500	90	80	160



$\frac{1+10}{2} = 5$

Adjust(A, 5, 10)

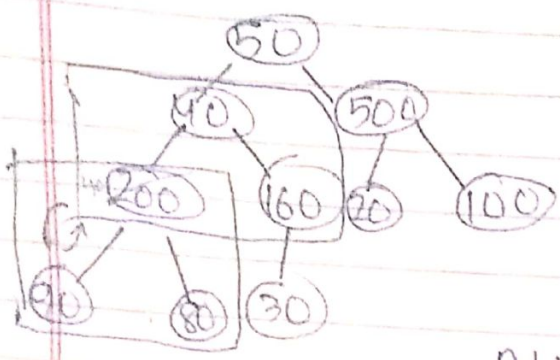


Adjust(A, 4, 10)

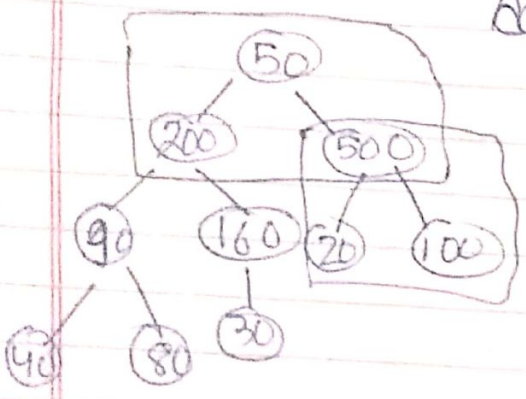
Heap formation
By using top

Sagar
Date:
Page:

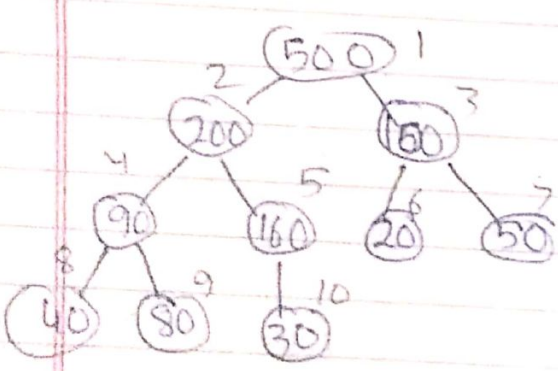
Adjust (A, 3, 10)



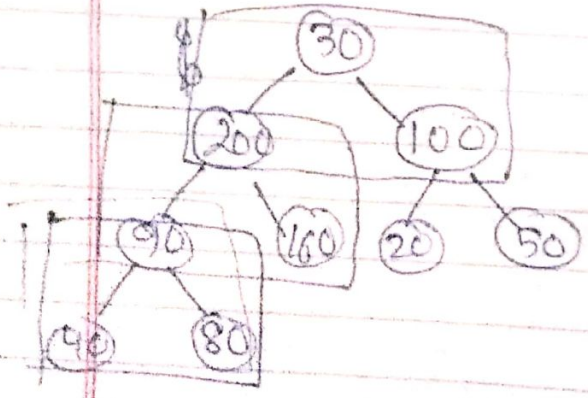
Adjust (A, 2, 10)



Adjust (A, 1, 10)



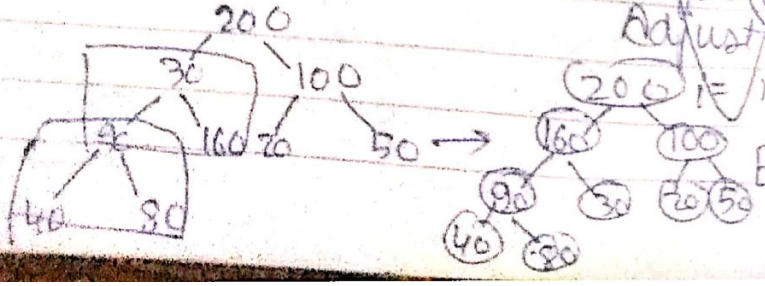
Exchange index 1 and 10



$\left\lfloor \frac{1+9}{2} \right\rfloor = 4$

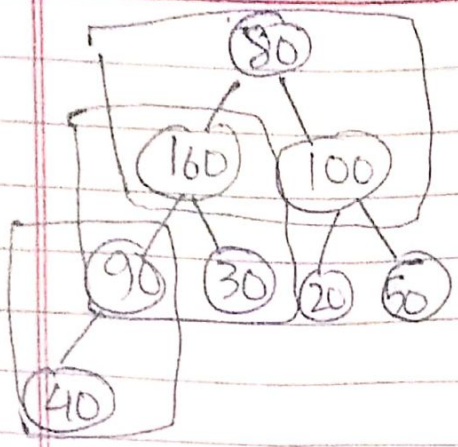
Adjust (A, 1, 9)
Adjust (A, 3, 9)
Adjust (A, 2, 9)
Adjust (A, 1, 9)

Adjust(A1, 9)

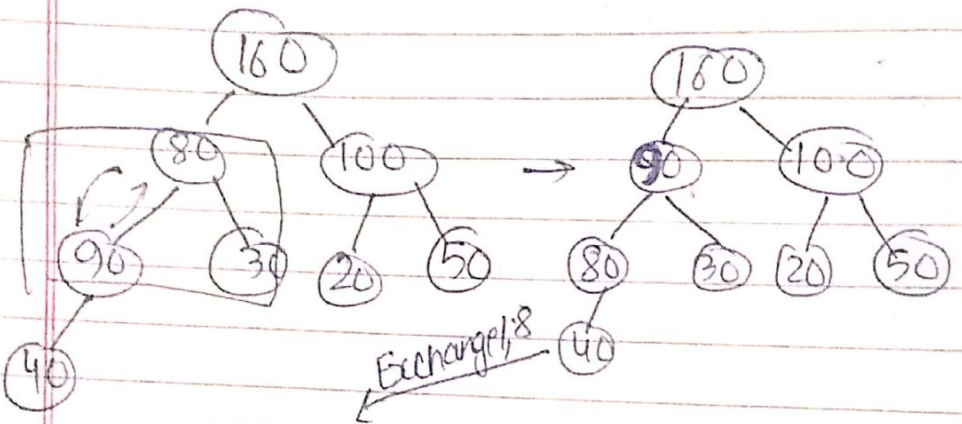


Exchange 1 and 2

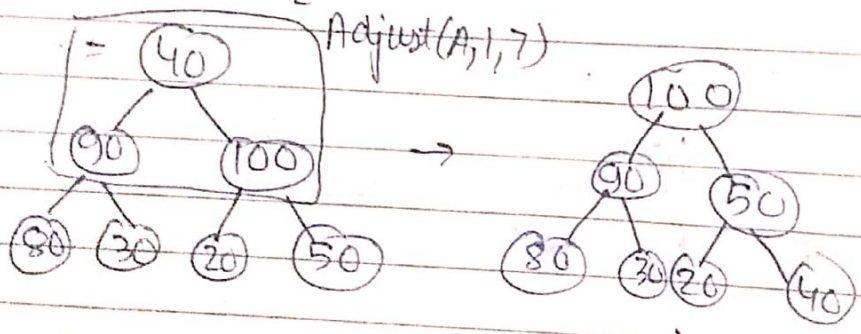
Sagar
Date:
Page:



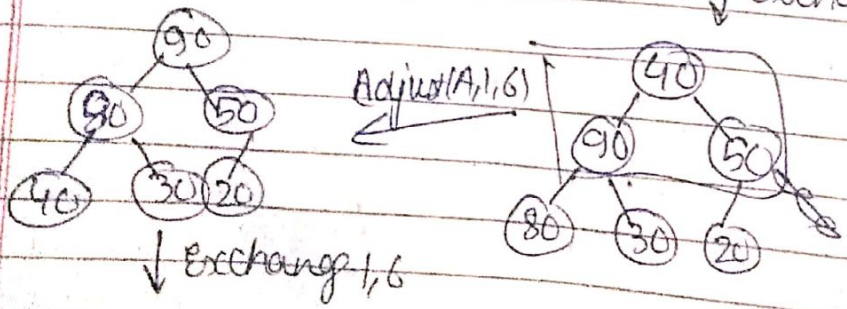
Adjust(A,1,8)



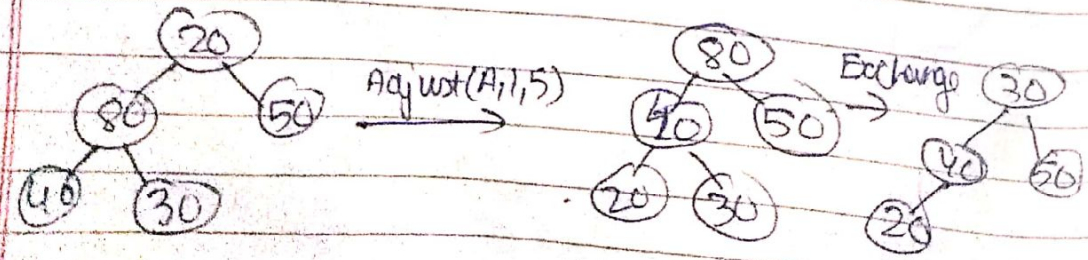
Exchange(1,8)

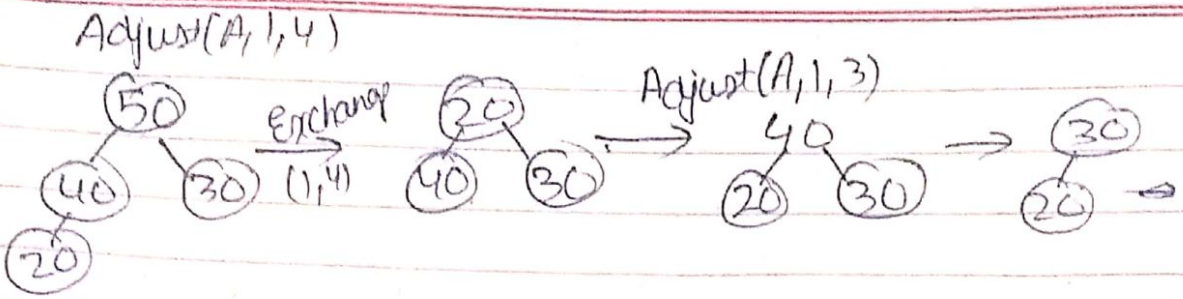


Exchange(1,7)



Exchange 1,6





20	30	40	50	80	90	100	160	200	500

```
ALGORITHM Adjust (A[ ], i, N)
BEGIN:
WHILE 2*i <= N DO
    j = 2*i   - Left child                    (j+1) Right child
    If (j+1 <= N) THEN
        IF (A[j+1] > A[j])
            j = j+1
        IF (A[j] > A[i])
            Exchange (A[j], A[i])
        ELSE
            BREAK
    i = j
END;
```

Space Complexity → $O(1)$ (j and Temp.)

Time Complexity

$N = 1+2^1+2^2+2^3+---+2^{\alpha}$

$N = \frac{1 \cdot (2^{\alpha+1}-1)}{2-1} \Rightarrow N+1 = 2^{\alpha+1}$

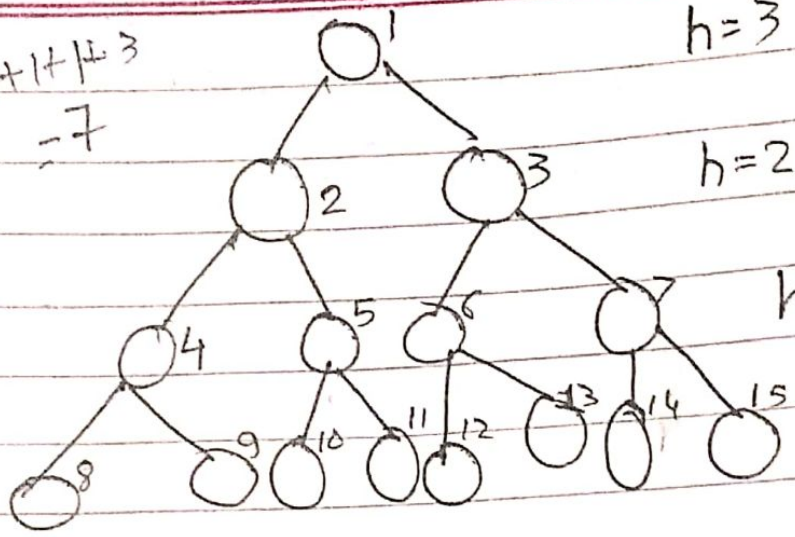
~~$\log_2 N = \log_2 2^{\alpha+1}$~~ $\log_2(N+1) = \alpha+1$

$\Rightarrow \alpha = \log_2(N+1) - 1 \sim \log_2 N$

Height of tree = $\log_2 N$ = max distance of root node from leaf node.

$\left\lceil \frac{N}{2^{h+1}} \right\rceil$

Sagar
Date:
Page:



$1+1+1+1+3 = 7$

$h=3 \quad \left\lceil \frac{15}{2^{3+1}} \right\rceil = 1$

$h=2 \quad \left\lceil \frac{15}{2^{2+1}} \right\rceil = 2$

$h=1 \quad \left\lceil \frac{15}{2^{1+1}} \right\rceil = 4$

$h=4 \quad \left\lceil \frac{15}{2^{0+1}} \right\rceil = 8$

Max Heapify (A[], N)

FOR $i = \left\lfloor \frac{N}{2} \right\rfloor$ TO 1 STEP-1

Adjust (A, i, N)

$$\sum_{h=0}^{\log_2 N} \left\lceil \frac{N}{2^{h+1}} \right\rceil O(h)$$

Sum of this series $= 2N$

$= O(N)$

Sagar
Date:
Page:

	Merge Sort	Quick Sort	Heap Sort
Time	$O(N\log_2 N)$	$O(N^2), O(N\log_2 N)$	$O(N\log_2 N)$
Space	$O(N)$	$O(\log_2 N)$	$O(1)$

$$\log_2 N\left\lceil \frac{N}{2^{h+1}} \right\rceil O(h) = \sum_{h=1}^{\log_2 N} \frac{N}{2^{h+1}} O(h)$$

$$= CN \sum_{h=1}^{N} \frac{h}{2^{h+1}} = CN\left[\frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \cdots \infty\right] = 2$$

* Multiplier of $N\log_2 N$ is such in Quick sort i.e. 1 than in merge sort i.e. 2 & then in heap sort i.e. 7

∴ Quick sort is best among quick, merge, heap sort

Sagar
Date:
Page:

	Time	Space	Exchange / non Exchange
Bubble			Exchange
Selection			Exchange
Insertion			Non Exchange
Quick			Exchange
Merge			Non Exchange
Heap			Exchange
Counting			Non Exchange.

External - All data element not required simultaneously in the RAM. Only a part of data is required which is currently used.

Internal - All data element ~~is~~ read simultaneously

Stable / Exchange

Sagar
Date:
Page:

Stable/Non Stable	Storage RAM/Secondary (Disk)
Non Stable	Internal
Non Stable	Internal
Stable	External
Non Stable	Internal
Stable	External
Non Stable	Internal
Stable	Internal

ALGORITHM RadixSort(A[], N, d)
BEGIN:
FOR i=1 Tod DO
Sort A[] at Radix i using
CountingSort
END;

Radaix sort

	Last	1st last	3rd last
496	300	300	199
300	690	210	210
690	210	511	300
484	511	632	396
396	632	842	484
210	842	844	496
844	484	484	511
511	844	690	632
632	496	496	690
199	396	396	842
842	199	199	844