ADA

Ch-1

☆ **Algorithm** :- An algorithm is a steps of operations to solve a problem performing calculation, data processing, and automated reasoning tasks.

☆ **Algorithm design techniques** :-

→ Divide & conquer
→ Greedy approch
→ Dynamic programming
→ Branch and bound
→ Backtracking
→ Randomized algorithm

☆ **Analysis measures** :-

→ Space complexity
→ Time complexity
→ Input size
→ Computing best, Average, worst case
→ Computing order of growth of algorithm

☆ **Properties** :- Input, Output, Definiteness, Finiteness, Effectiveness

☆ **Approaches to analyze an algorithm**

priori (Theoretical)	posteriori (Empiri
1. Algorithm	Program
2. Independent of language	Dependent on language
3. Hardware independent	Hardware dependent.
4. Time & space f^n	Watch time & bytes

Ch-2

* Best, average, worst case

* Worst: We calculate upper bound.
→ Max. no. of operations to be executed.
→ E.g. For linear search, the worst case happens when the element to be searched is not present in the array.

* Average: We take all posibilities i/ps and calculate computing time for all of the inputs.
→ Eg. For ~~liear~~ linear search, Assume all cases are uniformely distributed. (including the case x not being present in array)
→ So we sum all the cases and divide the sum by (n+1)

* Best : We calculate lower bound.
→ Min no. of operations to be executed.
→ Eg. For linear search, x is present at the first location

* Amortized analysis
1) Aggregate 2) Accounting 3) Potential.

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \dots < 2^n < 3^n < \dots < n^n$

Page No.
Date: / /

✱ Asymptotic notations:

O	Big-oh	Upper bound
Ω	Big-omega	lower bound
Θ	Theta	Average bound

1) Big-oh (O)

→ The f^n $f(n) = O(g(n))$ if $\exists$ +ve constant c & n_0 such that $f(n) \leq c * g(n)$ $\forall n \geq n_0$

e.g. $f(n) = 2n+3$

$2n+3 \leq$ ___ so here only 1 term with coefficient

↑ Here 2 terms

$2n+3 \leq 10n$, $2n+3 \leq 5n$; $n \geq 1$

$f(n)$ c $g(n)$

$f(n) = O(n)$

$2n+3 \leq 0.5n^2$; $n \geq 1$

$f(n)$ c $g(n)$

$\therefore f(n) = O(n^2)$

$1 < \log n < \sqrt{n} < n < n\log n < \dots < n^2$

lower bound
Big Omega (Ω)

Average bound
Ω, Θ, O

upper bound
↑ Big-oh (O)

2) Big omega (Ω)

→ The f^n $f(n) = \Omega(g(n))$ if $\exists$ +ve constants c & n_0 such that $f(n) \geq c * g(n)$ $\forall n \geq n_0$

$f(n) = 2n + 3$

$2n + 3 \geq n \times 1$; $\forall n \geq 1$

$f(n)$ g c

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

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

~~f(n)~~ આગળના page ના lower bound

3) Theta notation (θ)

→ The f^n $f(n) = \theta(g(n))$ if $\exists$ +ve constants ~~c~~ c_1, c_2 & n_0 such that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

e.g. $f(n) = 2n + 3$

$1 \times n \leq 2n + 3 \leq 5 \times n$

c_1 $g(n)$ $f(n)$ c_2 $g(n)$

$f(n) = \theta(n)$

Date: / /

☆ Loop invariant and the correctness of algorithm.

→ We use loop invariants to help us understand why an algorithm is correct. We must show 3 things about a loop invariant.

1) Initialization: It is true prior to the first iteration of the loop.
2) Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration.
3) Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

☆ Algorithms:

⇒ Bubble sort:

⇒ How it works:

1. The bubble sort starts with the very first index and makes it a bubble element. Then it compares the bubble element, which is currently our first index element, with the next element. If the bubble element is greater and the second element is smaller; then both of them will swap.

After swapping, the second element will become the bubble element. Now we will compare the second element with the third as we did in earlier step and swap them if required. The same process is followed until the last element.

2. We will follow the same process for the rest of iterations. After each of the iteration, we will notice that the largest element present in the unsorted array has reached the last index.

Ex: 32, 51, 27, 85, 66
Pass 1: 32, 27, 51, 85, 66
Pass 2: 27, 32, 51, 85, 66
Pass 3: 27, 32, 51, 66, 85
Pass 4: 27, 32, 51, 66, 85 sorted

* Time coplexity
Best = $O(n)$
Average = $O(n^2)$
Worst = $O(n^2)$

Algorithm : Bubble sort

	cost	times
for i ← 1 to n do	c_1	$n+1$
for i ← 1 to n-1 do	c_2	$\sum_{i=1}^{n} (n+1-i)$
if T[i] > T[j]	c_3	$\sum_{i=1}^{n} (n-i)$
T[i] ↔ T[j]	c_4	$\sum_{i=1}^{n} (n-i)$
end		
end		

Analysis

$$T(n) = c_1(n+1) + c_2 \sum_{i=1}^{n} (n+1-i) + c_3 \sum_{i=1}^{n} (n-i) + c_4 \sum_{i=1}^{n} (n-i)$$

Best case
if i = 1

$$T(n) = c_1 n + c_1 + c_2 n + c_3 n - c_3 + c_4 n - c_4$$
$$= n(c_1 + c_2 + c_3 + c_4) + (c_3 + c_4 - c_1)$$
$$= an - b$$
$$T(n) = \theta(n)$$

Page No.
Date: / /

Worst case:

i:n

$$T(n) = c_1 n + c_1 + c_2 n + c_2 \Phi - c_2\left(\frac{n(n+1)}{2}\right) + c_3 n$$

$$- c_3\left(\frac{n(n+1)}{2}\right) + c_4 n + c_4\left(\frac{n(n+1)}{2}\right)$$

$$= \left[-\frac{c_2}{n^2} - \frac{c_3}{n^2} - \frac{c_4}{n^2}\right] + \left[-\frac{c_2}{n} - \frac{c_3}{n} - \frac{c_4}{n}\right] + c_1 + c_2 + c_3 + c_4$$

$$= an^2 + bn + c$$

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

Average Same as worst case $T(n) = \Theta(n^2)$

$\Rightarrow$ Insertion sort:

→ Insertion sort works by inserting an element into its appropriate position during each iteration.
→ Insertion sort works by comparing an element to all its previous elements until an appropriate position is found.
→ Whenever an appropriate position is found, the element is inserted there by shifting down remaining elements.

Algorithm Procedure insert $(T[1,\dots n])$	cost	times
for $i \leftarrow 2$ to n do	c_1	n
$x \leftarrow T[i]$	c_2	$n-1$
$j \leftarrow i-1$	c_3	$n-1$
while $j>0$ and $x < T[j]$ do	c_4	$\sum_{i=2}^{n} T_j$
$T[j+1] \leftarrow T[j]$	c_5	$\sum_{i=2}^{n} T_j - 1$
$j \leftarrow j-1$	c_6	$\sum_{i=2}^{n} T_j - 1$
end		
$T[j+1] \leftarrow x$	c_7	$n-1$
end		

Analysis $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\sum_{i=2}^{n} T_j\right) + (c_5 + c_6)\sum_{j=2}^{n}(T_j - 1) + c_7(n-1)$

where $T_j = \sum_{i=2}^{n} T_j = \sum_{i=2}^{n} n-i$

Best case:

Take $j=1$

$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_5(n-1) + c_7(n-1)$

$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$

$= an - b$

$= O(n)$

Worst case: $j = n$

$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \frac{n}{2} + c_5 \frac{n}{2} + c_6 \frac{n}{2}$

$+ c_4 \frac{n^2}{2} + c_5 \frac{n^2}{2} + c_6 \frac{n^2}{2} + c_7 n - c_2 - c_3 - c_7.$

$$= n^2\left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) + n\left(c_1 + c_2 + c_3 + c_7 + \frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) - (c_2 + c_3 + c_7)$$

$$= an^2 + bn + c$$

$$T(n) = O(n^2)$$

Average case: same as worst case $T(n) = O(n^2)$

Ex: 32, 51, 27, 85, 66, 23, 13, 57

Pass 1: 32, 27, 51, 66, 23, 13, 57, 85

Pass 2: 27, 33, 51, 23, 13, 57, 66, 85

Pass 3: 27, 33, 23, 13, 51, 57, 66, 85

Pass 4: 27, 23, 13, 33, 51, 57, 66, 85

Pass 5: 23, 13, 27, 33, 51, 57, 66, 85

Pass 6: 13, 23, 27, 33, 51, 57, 66, 85

Pass 7: 13, 23, 27, 33, 51, 57, 66, 85

Ex: 3 | 9 6 1 2
(3: sorted; 9 6 1 2: unsorted)

3 9 | 6 1 2
(3 9: sorted)

3 6 9 | 1 2

1 3 6 9 | 2

1 2 3 6 9

⇒ ✱ Selection sort:

→ It works by repeatedly selecting elements
→ Algorithm finds the smallest element in the array first & exchanges it with the time element in the first position

• Algorithm

```
procedure select (T[i...n])                    cost
    for i ← 1 to n-1 do                        c1      n
        minj ← i ;                             c2      n-1
        minx ← T[i]                            c2      n-1
    for j ← i+1 to n do                        c3      Σ_{i=1}^{n-1} (i+1)
        if T[j] < minx then minj ← j           c4      Σ_{i=1}^{n-1} i
                            minx ← T[j]        c5      Σ_{i=1}^{n-1} i
    T[minj] ← T[i]                             c6      n-1
    T[i] ← minx                                c7      n-1
```

Analysis

$$T(n) = c_1 n + c_2(n-1) + c_3\left(\sum_{i=1}^{n-1}(i+1)\right) + c_4\left(\sum_{i=1}^{n-1}(i)\right) + c_5\left(\sum_{i=1}^{n-1} i\right) + c_6(n-1) + c_7(n-1)$$

$$= c_1 n + c_2 n + c_6 n + c_7 n + c_3 \frac{n}{2} + c_4 \frac{n}{2} + c_5 + c_5 \frac{n^2}{2} + c_4 \frac{n^2}{2} + c_5 \frac{n^2}{2} - c_2 - c_6 - c_7$$

$$= n\left(c_1 + c_2 + c_6 + c_7 + \frac{c_3}{2} + \frac{c_4}{2} + \frac{c_5}{2}\right) + n^2\left(\frac{c_3}{2} + \frac{c_4}{2} + \frac{c_5}{2}\right) - (c_2 + c_6 + c_7)$$

$$= an^2 + bn + c$$

Page No.
Date: / /

Best = Average = worst = $O(n^2)$

Ex: 3 69 6 1 2
1 9 6 3 2
1 2 6 3 9
1 2 3 6 9 sorted

⇒ Shell sort:

→ Shell sort works by comparing elements that are distant rather than adjacent elements in an array.
→ Shell sort uses a sequence $h_1, h_2 \ldots h_t$ called the increament sequence. Any increament sequence is fine as long as $h_1 = 1$ and some other choices are better than others.
→ Shell sort makes multiple passes through a list and ~~sorted~~ sorts a no. of equally sized sets using the insertion sort.

Ex:- 54, 26, 93, 17, 77, 31, 44, 55, 20

	Row by row	column by column
54 26 93	26 54 93	17 31 55
17 77 31 →	17 31 77 →	20 44 77
44 55 20	20 44 55	26 54 93

sorted array 17 20 26 31 44 54 55 77 93

Heap sort बाकि है.

Page No.
Date: / /

5. Sorting in linear time

⇒ Bucket sort:

→ It runs in linear time when the i/p is drawn from uniform distribution.

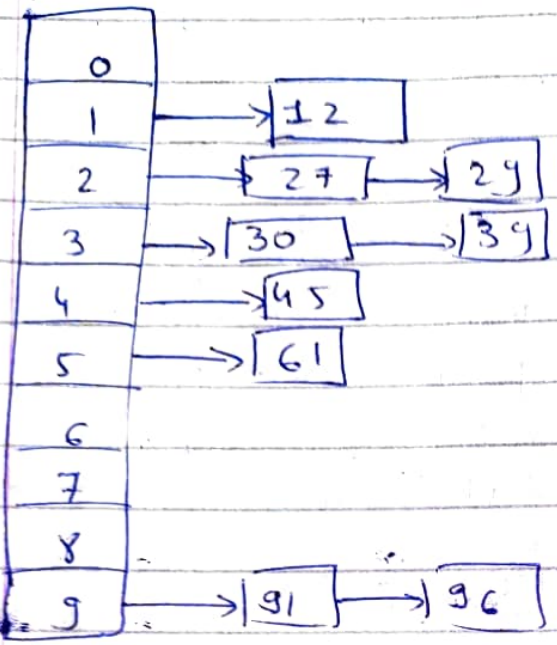
→ Like counting sort assumes that the i/p consists of integers in a small rang, bucket sort assumes that the i/p generated by a random process that distributes elements uniformly over the interval $[0, 1)$

Like counting, Bubble sort is fast because it assumes something about the input.

→ Assumption: the key in the range $[0, N)$

Algo. BUCKET-SORT(A)
n ← length[A]
for i ← 1 to n
    do insert A[i] into list B[⌊n A[i]⌋]
for i ← 0 to n-1
    do sort list B[i] with insertion sort
Concatenate the lists B[0], B[1], ..., B[n-1] together in order.

Ex:- 45, 96, 29, 30, 27, 12, 39, 61, 91



After sorting 12 27 29 30 39 45 61 91 96

⟹ Radix sort:

→ The op^n of radix sort on a list of seven 3-digit no. The leftmost column is the i/p. The remaining columns show the list after successive sorts on increasingly significant digit positions

→ Shading indicates the digit position sorted on to produce each list from the previous one

→ The code for radix sort is straightforward The following procedure assumes that each element in the n-element array A has d digits, where digit 1 is the lowest-order digit and digit d is the highest order digit.

→ Given n ~~input~~ d-digit numbers in which each digit can take on up to k possible values, RADIXSORT correctly sorts these numbers in $\Theta(d(n+k))$ time.

Algo. RADIX SORT (A, d)
for i ← 1 to d
do use a stable sort to sort array A on digit i

Ex. 363, 729, 329, 873, 691, 521, 435, 297

3	6	3		6	9	1		5	2	1	
7	2	9		5	2	1		7	2	9	
3	2	9		3	6	3		3	2	9	
8	7	3		8	7	3		4	3	5	
6	9	1		4	3	5		3	6	3	
5	2	1		2	9	7		8	7	3	
4	3	5		7	2	9		6	9	1	
2	9	7		3	2	9		7	2	9	
		sort column 1			sort column 2				sort column 3		sorted

⇒ Counting sort:

→ It assumes that each of the n i/p elements is an integer in the range 0 to k, for some integer k.
→ When $k = O(n)$, the sort runs in $O(n)$ time.
→ The basic idea of counting sort is to determine, for each i/p element x, the no. of elements less than x.

-> This information can be used to placed ~~directly~~ element x directly into it's position in the o/p array

Algo COUNTING SORT (A,B,k)

```
for i<-0 to k
    do c[i]<-0
for j<-1 to length[A]
    do c[A[j]] <- c[A[j]] + 1
// c[i] now contains the no. of elements equal to i
for i<-1 to k
    do c[i] <-c[i]+ c[i-1]
// c[i] now contains the no. of elements less than or equal to i
for j<- length[A] down to 1
    do B[c[A[j]]] <- A[j]
       c[A[j]] <-c[A[j]] - 1
```

Ex Given array

step-1

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

↑ max

step-2 Determine size of array C as max. value in Array = 6

1	2	3	4	5	6
0	0	0	0	0	0

step-3 Update array C with occurences of each value of array A

1	2	3	4	5	6
2	0	2	3	0	1

step-4 In array C from index 2 to n add value with previous element.

2+0 0+2 4+3 7+0 7+1

1	2	3	4	5	6
2	2	4	7	7	8

as it is

Create o/p array B[1 ... 8]
start positioning elements of Array A to B shown as follows.

step-5 Array A

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

Array C

1	2	3	4	5	6
2	2	4	7	7	8

Array B

1	2	3	4	5	6	7	8
1	1	3	3	4	4	4	6

☆ Heap sort

→ It is a binary tree with the following properties.

1. Complete binary tree; that is each level of the tree is completely filled, except possibly the bottom level. At this level it is filled from left to right
2. It satisfies the heap order property; the data item stored in each node is greater than or equal to the data item stored in its children node

→ Time complexity

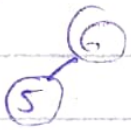
$T(n) = O(n \log n)$ (Best, worst & average)

Max Heap

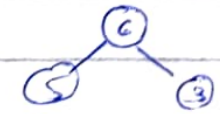
Heap array 6 5 3 1 8 7 24

→ 6 | 5 3 1 8 7 24

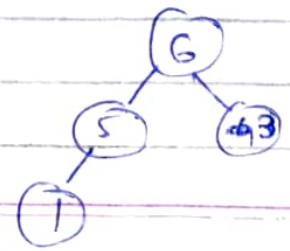
| 3 1 8 7 24



→ | 1 8 7 2 4

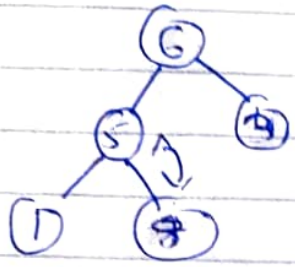


| 8 7 2 4

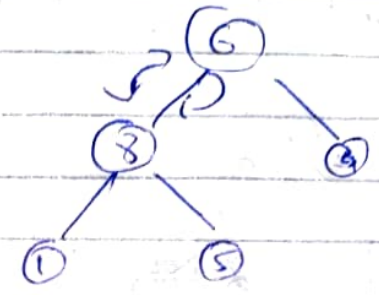


Page No.
Date: / /

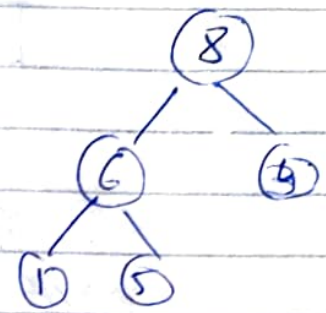
□ 7 2 4



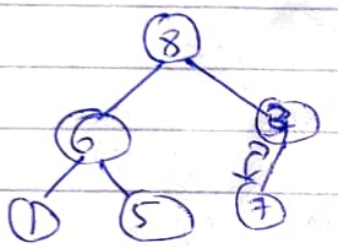
□ 7 2 4



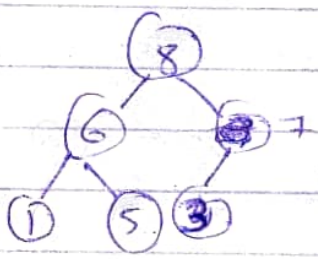
□ 7 < 4



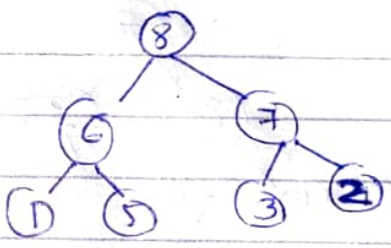
□ 2 4



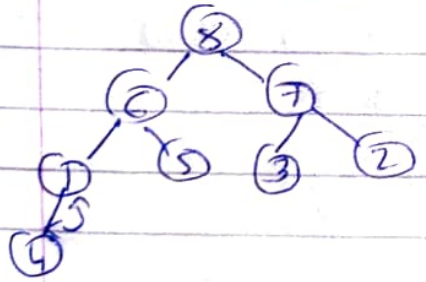
□ 2 4



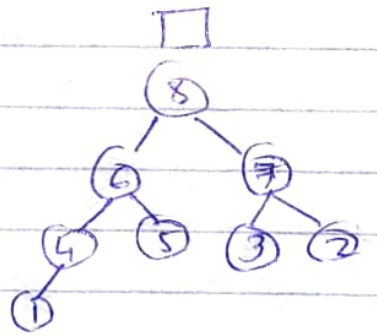
□ 4



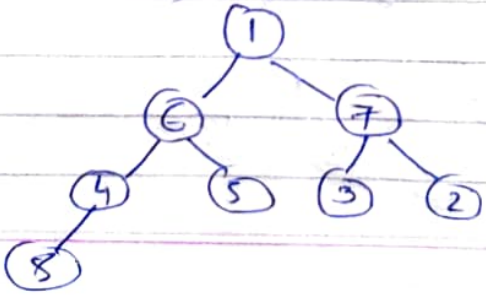
□



□



select first node 8 & swap with last node.

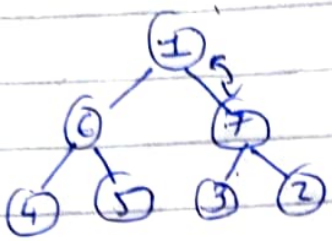


1	6	7	4	5	3	2	8

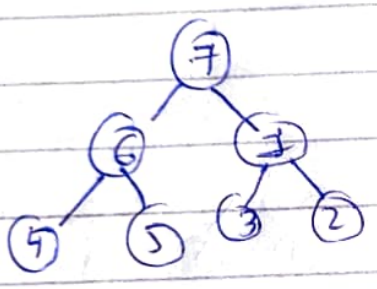
Page No.
Date: / /

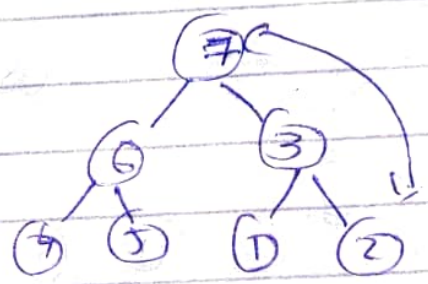
Decrease heap size by one

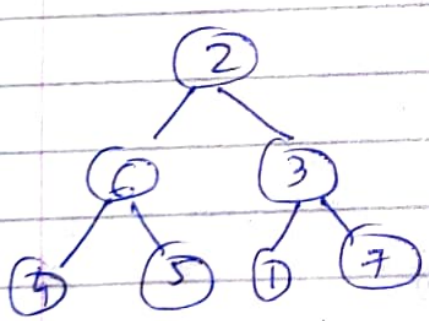
1	6	7	4	5	3	2	8



Heapify new array

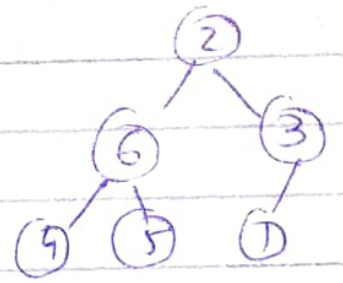




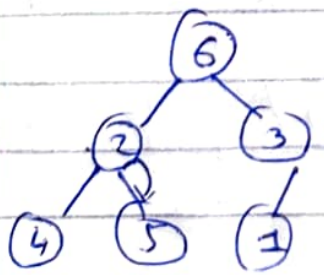


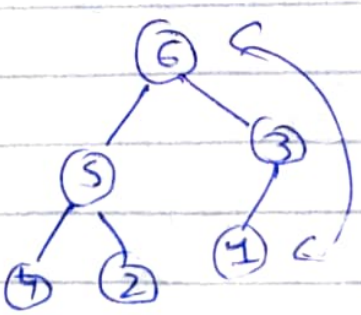
decrease array

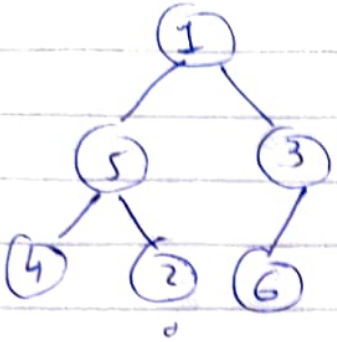
2 6 3 4 5 1 7 8



Heapify array

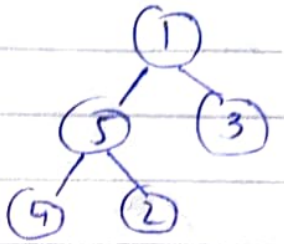




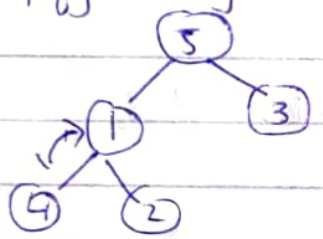


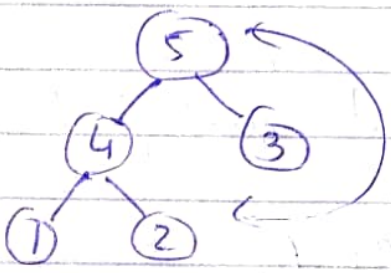
decrease array

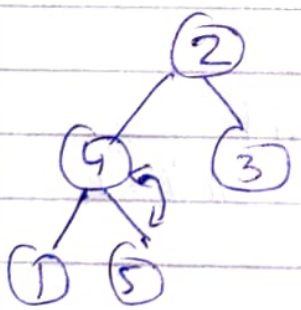
1 5 3 4 2 | 6 7 8

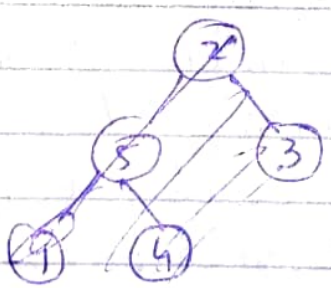


Heapify array.



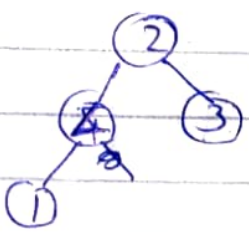




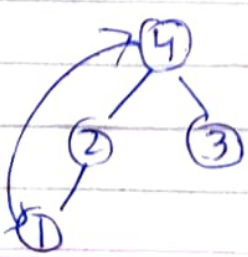


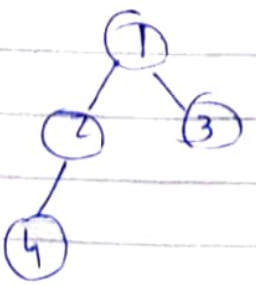
decrease array

2 4 3 1 | 5 6 7 8



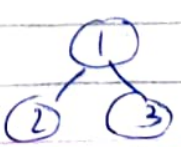
Heapify array





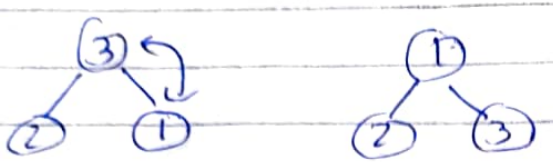
decrease array

1 2 3 | 4 5 6 7 8

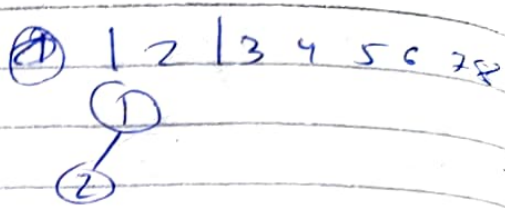


Page No.
Date: / /

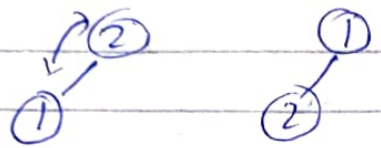
Heapify array



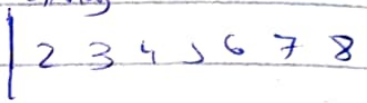
decrease array



Heapify



decrease array



sorted array 1 2 3 4 5 6 7 8

Algorithm HEAPSORT (A)

BUILD-MAX-HEAP (A)
for i ← length [A] downto 2
do exchange A[1] ↔ A[i]
heap-size [A] ← heap-size [A] - 1
MAX-HEAPIFY (A, 1)

Ch-3 Divide & conquer.

→ Many algoriths are recursive in nature to solve a given de problem recursively dealing with sub-problems

→ ~~Generally~~ In this approch, a problem is divided into smaller problems, then the smaller problems are solved independently, and finally the sol's of smaller problems are combined into a solution for the large problems.

→ Generally, divide-and-conquer algorithms have three parts -

* Divide the problem into no. of sub-problems that are smaller instances of the same problem.
* Conquer the subproblem by solving them recursively. If they are small enough, solve the sub-problems as base cases.
* Combine the solutions to the sub-problems into the sol^n for the original problem.

* Application of divide & conquer approach

→ Finding the max. & min of sequance of numbers
→ Strassen's matrix multiplication
→ Merge sort
→ Binary search

★ Recurrence relation by substitution method

★ $T(n) = \begin{cases} 1 & ; n=0 \\ T(n-1)+1 & ; n>0 \end{cases}$

$T(n) = T(n-1) + 1$ — (1)

Find, $T(n-1)$, $T(n-2)$

$T(n-1) = T(n-1-1) + 1$

$T(n-1) = T(n-2) + 1$ — (2)

$T(n-2) = T(n-3) + 1$ — (3)

Now

$T(n) = T(n-2) + 1 + 1$ (Put $T(n-1)$)

$= T(n-2) + 2$

$= T(n-3) + 1 + 2$ (Put $T(n-2)$)

$T(n) = T(n-3) + 3$

{ continue for k times

$T(n) = T(n-k) + k$

Assume $n-k = 0$

$n = k$

$T(n) = T(n-n) + n$

$T(n) = T(0) + n$

$T(n) = 1 + n$

$O(n) = \Theta(n)$

Page No.
Date: / /

$$\star\ T(n) = \begin{cases} 1 & ; n=0 \\ T(n-1)+n & ; n>0 \end{cases}$$

$$T(n) = T(n-1)+n \quad \text{—(1)}$$

Find $T(n-1)$, $T(n-2)$

$$T(n-1) = T(n-2)+n-1 \quad \text{—(2)}$$
$$T(n-2) = T(n-3)+n-2 \quad \text{—(3)}$$

Put eqn (2) & (3)

$$T(n) = T(n-2)+(n-1)+n$$
$$= T(n-2)+2n-1$$
$$= T(n-3)+n-2+2n-1$$
$$= T(n-3)+3n-3$$

$$T(n) = T(n-3)+(n-2)+(n-1)+n$$

$$\Rightarrow$$

$$T(n) = T(n-k)+(n-(k-1))+(n-(k-2))+\ldots+(n-1)+n$$

Assume $n-k=0 \Rightarrow n=k$

$$T(n) = T(n-n)+(n-n+1)+(n-n+2)+\ldots+(n-1)+n$$

$$= T(0)+1+2+\ldots+(n-1)+n$$
$$= 1+1+2+3+\ldots n$$
$$= 1+\frac{n(n+1)}{2}$$

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

Page No.
Date: / /

$$\star\ T(n) = \begin{cases} 1 & ; n=0 \\ T(n-1) + \log n & ; n>0 \end{cases}$$

$T(n) = T(n-1) + \log n$ — (1)

Find $T(n-1)$, $T(n-2)$

$T(n-1) = T(n-2) + \log(n-1)$ — (2)

$T(n-2) = T(n-3) + \log(n-2)$ — (3)

Put (2) & (3)

$T(n) = T(n-2) + \log(n-1) + \log n$

$T(n) = T(n-3) + \log(n-2) + \log(n-1) + \log n$

{upto k

$T(n) = T(n-k) + \log(n-(k-1)) + \log(n-(k-2)) + \dots + \log(n-1) + \log n$

Assume $n-k = 0 \Rightarrow n = k$

$T(n) = T(n-n) + \log(n-n+1) + \log(n-n+2) + \dots + \log(n-1) + \log n$

$= T(0) + \log 1 + \log 2 + \dots + \log n$

$= 1 + \log 1 + \log 2 + \dots + \log n$

$T(n) = 1 + \log n!$

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

$a + ar + ar^2 + ar^3 + \dots + ar^k = \frac{a(r^{k+1} - 1)}{r - 1}$

Page No.
Date: / /

short cut

$T(n) = T(n-1) + 1$	$O(n)$	☆
$T(n) = T(n-1) + n$	$O(n^2)$	
$T(n) = T(n-1) + \log n$	$O(n \log n)$	☆
$T(n) = T(n-1) + n^2$	$O(n^3)$	☆
$T(n) = T(n-2) + 1$	$O(n)$	☆
$T(n) = T(n-100) + n$	$O(n^2)$	
$T(n) = 2T(n-1) + 1$	$O(2^n)$	☆

$$T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1) + 1 & n > 0 \end{cases}$$

$T(n) = 2T(n-1) + 1$

$T(n-1) = 2T(n-2) + 1$

$T(n) = 2T(n-1) + 1$

$T(n-1) = 2T(n-2) + 1$

$T(n-2) = 2T(n-3) + 1$

$T(n) = 2[2T(n-2) + 1] + 1$

$= 2^2 T(n-2) + 2 + 1 \quad -(2)$

$= 2^2 [2T(n-3) + 1] + 2 + 1$

$T(n) = 2^3 T(n-3) + 2^2 + 2 + 1$

} upto k

$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \dots + 2^2 + 2 + 1$

Assume $n - k = 0 \Rightarrow n = k$

$$T(n) = 2^n T(n-n) + 2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2 + 1$$

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

$$= 2^n + \frac{1\left(2^{(n-1)+1} - 1\right)}{2-1} \quad = 1 + 2 + 2^2 + \cdots + 2^{n-1} + 2^n$$

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

$$= 2^n + 2^n - 1$$

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

$$= 2^n (1+1) - 1$$

$$= 2^1 \cdot 2^n - 1$$

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

$$T(n) = O(2^n)$$

$$\cdot\ \Theta(2^n)$$

★ Master theorem for decreasing f^n's

$$T(n) = a\,T(n-b) + f(n)$$

[Ex: $T(n-1) + n$] or ~~$O(n+f(n))$~~

where $a > 0$, $b > 0$ & $f(n) = O(n^k)$ where $k \geq 0$

if $a = 1$ $\quad O(n^{k+1})$ or $O(n + f(n))$

if $a > 1$ $\quad O(n^k\, a^n)$, $O(n^k\, a^{n/b})$

if $a < 1$ $\quad O(n^k) = O(f(n))$

Page No.
Date: / /

* Recurrence relation dividing functions

* $T(n) = \begin{cases} 1 & n=1 \\ T(n/2)+1 & n>1 \end{cases}$

$T(n) = T(n/2) + 1$ — (1)

$T(n/2) = T\left(\frac{n}{2^2}\right) + 1$

$T(n) = T\left(\frac{n}{2^2}\right) + 1 + 1$

$T(n) = T\left(n/2^2\right) + 2$ — (2) $\left[T\left(n/2^2\right) = T\left(\frac{n}{2^3}\right) + 1\right]$

$T(n) = T\left(n/2^3\right) + 3$ — (3)

} upto k

$T(n) = T\left(n/2^k\right) + k$

assume $\frac{n}{2^k} = 1 \Rightarrow 2^k = n$ & $k = \log_2 n$

So, $T(n) = 1 + \log n$

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

Page No.
Date: / /

$$T(n) = \begin{cases} 1 & n = 1 \\ T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$$

$$T(n) = T\left(n/2\right) + n \quad \text{— (1)}$$

$$T(n) = \left[T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$$

$$T(n) = T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n \quad \text{— (3)}$$

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

$$T(n) = T\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}} + \cdots + \frac{n}{2^2} + \frac{n}{2} + n$$

Assume $\frac{n}{2^k} = 1 \Rightarrow n = 2^k = k = \log_2 n$

$$T(n) = T(1) + \log_2 n - 1 \; n \left[\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2}\right]$$

approx 1

$$= 1 + n[1 + 1]$$

$$= 1 + 2n$$

$$= O(n)$$

☆ Master theorem for dividing f^n.

(1) $\log_b a$ $\quad T(n) = aT\left(n/b\right) + f(n)$

(2) k $\quad a \geq 1$, $b > 1$ $\quad f(n) = \theta(n^k \log^p n)$

case 1 if $\log_b a > k$ then $\theta(n^{\log_b a})$

case 2 if $\log_b a = k$ $\rightarrow \theta(n^k \log^{p+1} n)$

- if $p > -1$
- if $p = -1$ $\quad \theta(n^k \log\log n)$
- if $p < -1$ $\quad \theta(n^k)$

case 3 if $\log_b a < k$ $\quad$ if $p \geq 0$ $\theta(n^k \log^p n)$; if $p < 0$ $\theta(n^k)$

---

1 if $f(n)$ is in $O(n^{\log_b a})$ $\left[f(n) < n^{\log_b a}\right]$ then $T(n) = \theta(n^{\log_b a})$

2 $f(n)$ is in $\theta(n^{\log_b a})$ $\left[f(n) = n^{\log_b a}\right]$ then $T(n) = \theta(n^{\log_b a} \log n)$

3 $f(n)$ is in $\Omega(n^{\log_b a})$ $\left[f(n) \geq n^{\log_b a}\right]$ then $T(n) = \theta(f(n))$

$$\boxed{T(n) = aT\left(n/b\right) + f(n)} \star$$

Page No.
Date / /

Exc: $T(n) = 2T\left(\frac{n}{2}\right) + 1$

$a = 2, \quad b = 2, \quad f(n) = \Theta(1)_{(k)}{}_{(p)}$

$= \Theta(n^0 \log^0 n)$

$k = 0, \quad p = 0$

$\log_b a = \log_2 2 = 1, \quad k = 0$

$\log_b a > k$ (case 1)

~~$\Theta($~~ $\Theta(n^{\log_b a})$

$\Theta(n^{\log_2 2})$

$\Theta(n)$

Exc $T(n) = 4T\left(\frac{n}{2}\right) + n$ [easy →]

$a = 4, \quad b = 2, \quad f(n) = \Theta(n)$

$= \Theta(n^1 \log^0 n)$

$\log_2 4 = 2, \quad k = 1, \quad p = 0$

$\log_2 4 > k$

$\Theta(n^{\log_2 4}) = \Theta(n^2)$

Page No.
Date: / /

Ex $T(n) = 8T\left(\frac{n}{2}\right) + n^{(k=1)}$

$\log_2 8 = 3 > k = 1$

$\Theta(n^{\log_2 8}) = \Theta(n^3)$

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

$\log_2 4 = 2 \quad k = 2$

$\log_2 4 = k$

Ex= $T(n) = 4T\left(\frac{n}{2}\right) + n$

$a = 4$
$b = 2$
$f(n) = n$

$n^{\log_b a} = n^{\log_2 4} = n^2$

Here $f(n) < n^2$
$n < n^2$ (case 1)

$T(n) = \Theta(n^{\log_b a})$
$= \Theta(n^2)$

Recurrence relation for Root of n:

Ex: $$T(n) = \begin{cases} 1 & ; n = 2 \\ T\sqrt{n} + 1 & ; n > 2 \end{cases}$$

$$T(n) = T\sqrt{n} + 1 = T\left(n^{1/2} + 1\right) \quad -(1)$$

$$T(n) = T\left(n^{\frac{1}{2^2}}\right) + 2 \quad -(2)$$

$$T(n) = T\left(n^{1/2^3}\right) + 3 \quad -(3)$$

$$T(n) = T\left(n^{1/2^k}\right) + k \quad -(4)$$

Assume $n = 2^m$

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

Assume $T\left(2^{\frac{m}{2^k}}\right) = T(2)$

$$\Rightarrow \frac{m}{2^k} = 1$$

$$\Rightarrow m = 2^k \Rightarrow k = \log_2 m$$

$$n = 2^m \qquad m = \log_2 n$$

$$k = \log\log_2 n$$

$$\theta\left(\log\log_2 n\right)$$

☆ Recurrence tree method.

→ It is pictorial representation of an iteration method which is in the form of a tree where at each level nodes are expanded

→ In general, we consider the second term in recurrence as root.

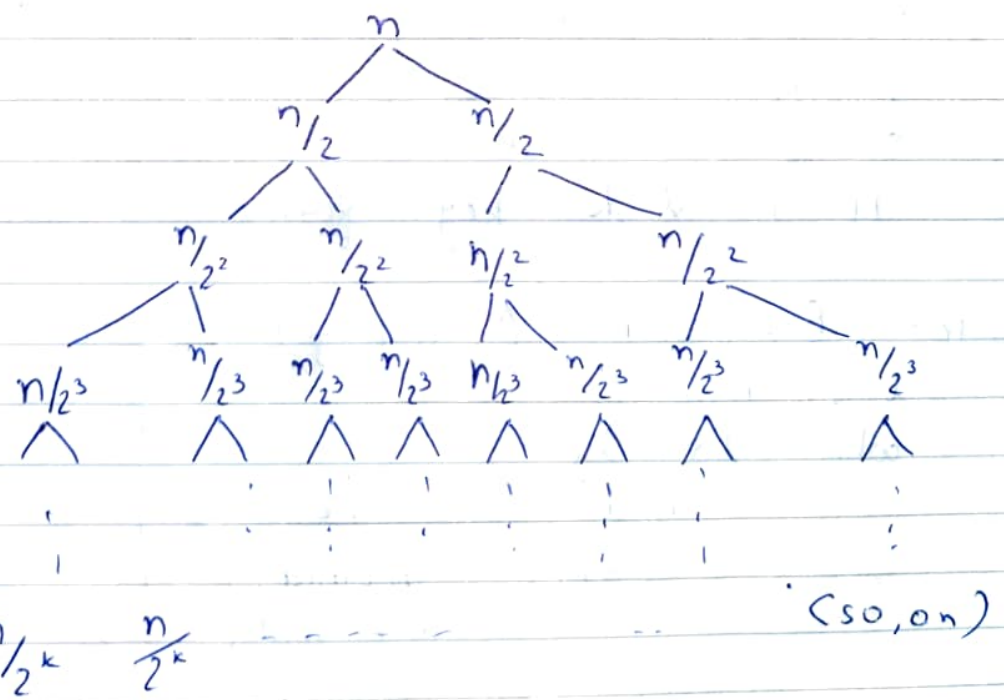
→ Useful in divide & conquer

→ Sometimes difficult to come up with a good guess. In recursion tree, each root & child represents the cost of a single subproblem.

→ We sum the costs within each of the levels of the tree to obtain a set of pre-level costs and then sum all pre-level costs to determine the total cost of all levels of the recursion.

→ A recursion tree is best used to generate a good guess, which can be verified by the substitution method.

Ex: $$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$



from given in problem:

$$n/2^k = 1$$

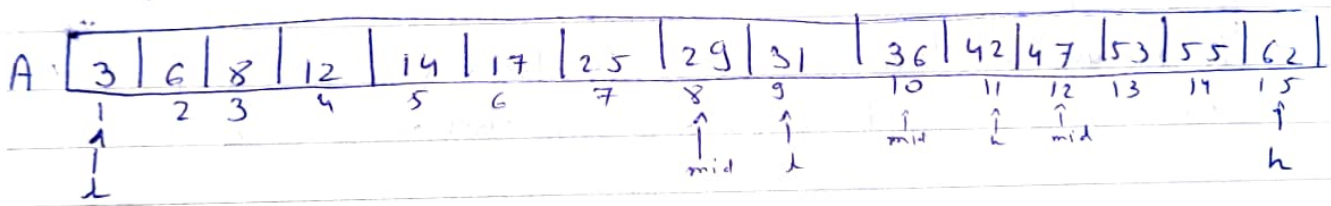
$$n = 2^k$$

$$k = \log n$$

$$\therefore T(n) = kn$$

$$= n\log n$$

★ Binary search:



Iterative method

key = 42

l	h	mid = (l+h)/2
1	15	(1+15)/2 = 8
9	15	12
9	11	10
11	11	l = h key found

Algo.

```
int BinSearch (A, n, key)
    l = 1, h = n;
    while (l <= h) {
        mid = (l+h)/2
        if (key == A[mid])
            return mid;
        if (key < A[mid])
            h = mid - 1
        else
            l = mid + 1
    }
    return 0;
}
```

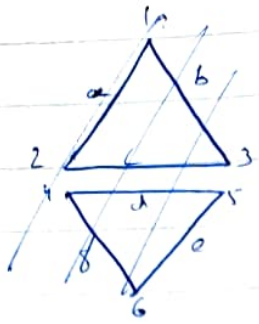
Recursive method

Algo

```
Algorithm RBinSearch(l, h, key)
{
    if (l == h)
    {
        if (A[l] == key)
            return l;
        else return 0;
    }
    else
    {
        mid = (l+h)/2;
        if (key == A[mid])
            return mid;
        if (key < A[mid])
            return RBinSearch(l, mid-1, key);
        else
            return RBinSearch(mid+1, h, key);
    }
}
```

→ $$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + 1 & n > 1 \end{cases}$$

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

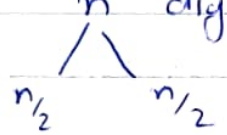


$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

✱ Multiply large integers

→ If n is the no. of digits, then addition & subtraction algorithms runs in $O(n)$ time.

→ But the standard algorithm for multiplication runs in $O(n^2)$ time, which is quite costly.



A [n-1 ... 0]

→ $c = a * b$

$c = c_2 10^n + c_1 10^{n/2} + c_0$

n : length of number.

$c_2 = a_1 * b_1$

$c_0 = a_0 * b_0$

$c_1 = (a_1 + a_0)(b_1 + b_0) - (c_2 + c_0)$

Ex: t_3 9 8 1 X 1 2 3 4
add 0 here

0 9 8 1 X 1 2 3 4
a_1 a_0 b_1 b_0

$c_2 = a_1 \times b_1$
$= 09 \times 12$
$= 108$

$c_0 = a_0 \times b_0$
$= 81 \times 34$
$= 2754$

$c_1 = (a_1 + a_0)(b_1 + b_0) - (c_2 + c_0)$

$= (9+81)(12+34) - (108 + 2754)$
$= (90)(4) - 2862$
$= 4140 - 2862$
$c_1 = 1278$

$c = c_2 10^{n} + c_1 10^{n/2} + c_0$

$= (108)10^4 + (1278)10^2 + 2754$
$= 10800000 + 127800 + 2754$
$= 12105 54$

★ Strassen's Matrix multiplication.

$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$ $= C = \begin{bmatrix} & \\ & \end{bmatrix}$

2 × 2 should be same 2 × 2 — 2 × 2

↑ size of answer matrix

```
for (i=0; i<n; i++)
{   for (j=0; j<n; j++)
    {   c[i,j] = 0;
        for (k=0; k<n; k++)
        {   c[i,j] += A[i,k] * B[k,j];
        }
    }
}
```

Time complexity $O(n^3)$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$P = (A_{11} + A_{22})(B_{11} + B_{22})$
$Q = B_{11}(A_{21} + A_{22})$
$R = A_{11}(B_{12} - B_{22})$
$S = A_{22}(B_{21} - B_{11})$
$T = B_{22}(A_{12} + A_{11})$
$U = (B_{11} + B_{12})(A_{21} - A_{11})$
$V = (B_{21} + B_{22})(A_{12} - A_{22})$

$C_{11} = P + S - T + V$
$C_{12} = R + T$
$C_{21} = Q + S$
$C_{22} = P + R - Q + U$

Ex- $A = \begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 6 & 7 \\ 3 & 8 \end{bmatrix}$

$A_{11} = 1 \quad A_{12} = 3 \quad A_{21} = 7 \quad A_{22} = 5$
$B_{11} = 6 \quad B_{12} = 7 \quad B_{21} = 3 \quad B_{22} = 8$

$P = (A_{11} + A_{22})(B_{11} + B_{22}) = (1+5)(6+8) = 6 \times 14 = 84$
$Q = B_{11}(A_{21} + A_{22}) = 6(7+5) = 72$
$R = A_{11}(B_{12} - B_{22}) = 1(7-8) = -1$
$S = A_{22}(B_{21} - B_{11}) = 5(3-6) = -15$
$T = B_{22}(A_{12} + A_{11}) = 8(1+3) = 32$
$U = (B_{11} + B_{12})(A_{21} - A_{11}) = (13)(6) = 78$
$V = (B_{21} + B_{22})(A_{12} - A_{22}) = (11)(-2) = -22$

$C_{11} = P + S - T + V$

$= 84 + (-15) - 32 + (-22)$

$= 15$

$C_{12} = R + T$

$= -1 + 32$

$= 31$

$C_{21} = Q + S$

$= 72 - 15$

$= 57$

$C_{22} = P + R - Q + V$

$= 84 + (-1) - (72) + 78$

$= 89$

$$C = \begin{bmatrix} 15 & 31 \\ 57 & 89 \end{bmatrix}$$

Min-Max Algo.

→ Write an algo. to find min & max element from given array. Draw min-max tree from the given array and also write it's complexity.

22, 13, -5, -8, 15, 60, 17, 31, 47

Sol"

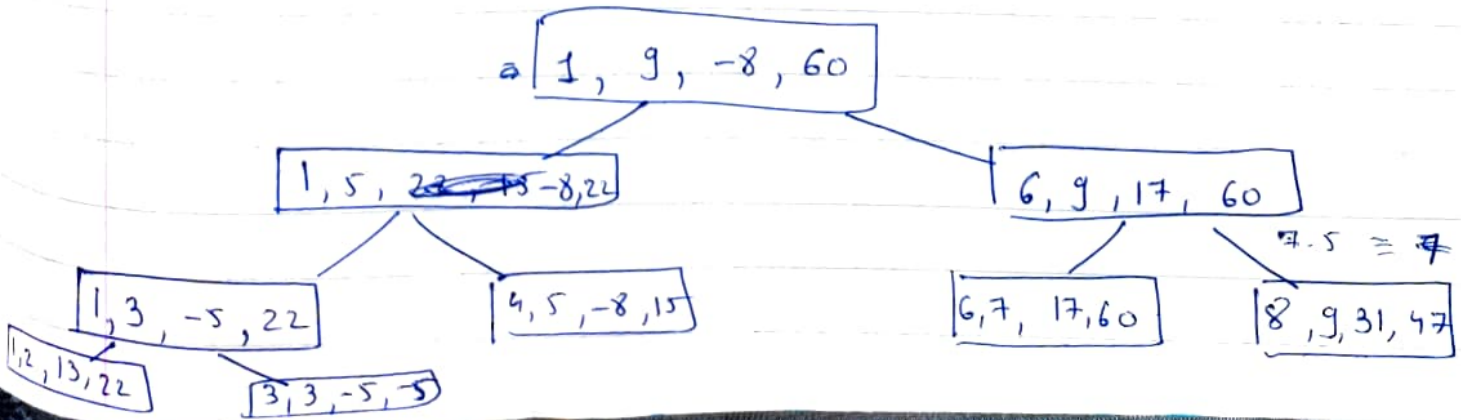
Index	1	2	3	4	5	6	7	8	9
Array	22	13	-5	-8	15	60	17	31	47

low, high, min, max

low, high → Index; min, max → Array

Complexity: $\log_2 n$

By using mid = $\frac{low + high}{2}$



Complexity $\log_2 n$

★ Mearge sort

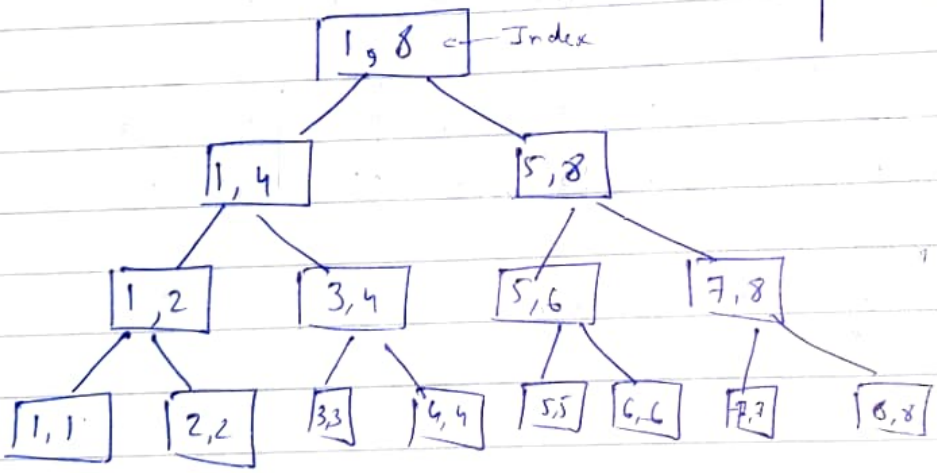
	1	2	3	4	5	6	7	8
A	9	3	7	5	6	4	8	2

Time complexity
$O(n \log n)$

```
Algorithm MergeSort (l, h)
{
    if (l < h)
    {
        mid = (l + h) / 2;
        MergeSort (l, mid);
        MergeSort (mid+1, h);
        Merge (l, mid, h);
    }
}
```

Pross
Large size list
Linked list
External sorting
Stable

Con.
Extra space
No small problem
Recursive



9 3 7 5 6 4 8 2
3 9 5 7 4 6 2 8
3 5 7 9 2 4 6 8
2 3 4 5 6 7 8 9

Time complexity $O(n \log n)$

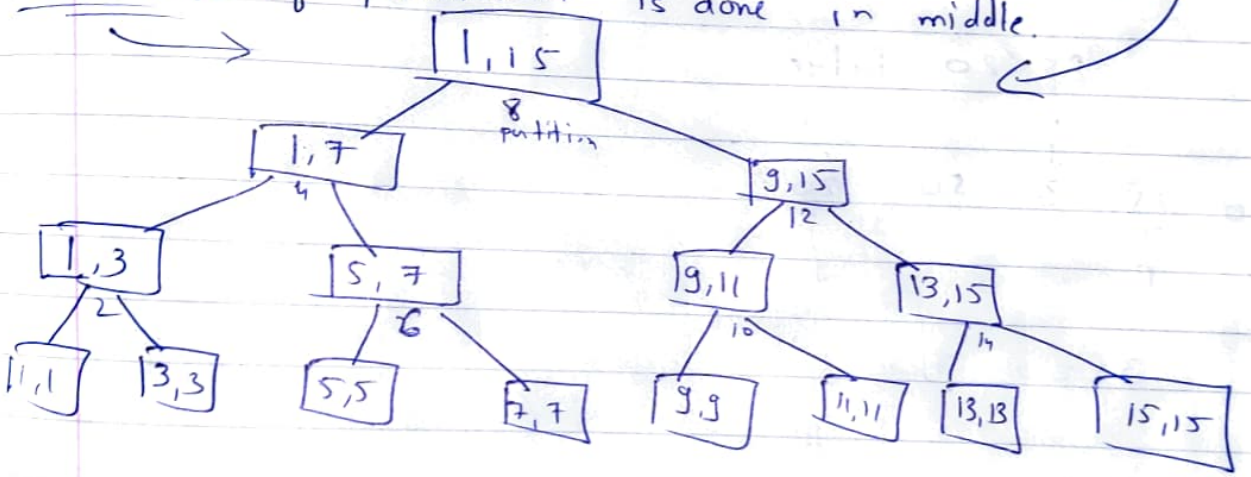
✱ Quick sort

```
QUICKSORT (l, h)
{
    if (l < h)
    {
        j = partition (l, h)
        Quick Sort (l, j);
        Quick Sort (j+1, h);
    }
}
```

Time complexity
$O(n \log n)$

1 ... 15

Best case if partition is done in middle.



worst case → If Array is already sorted!
Time complexity $O(n^2)$

Ex: 35 / 50 / 15 25 80 20 90 45

i, p

j

Ex: 5 3 8 2 4 6 2 7

P i n

1) 10 25 3 50 20

i,P j

10 < 20 True

10 25 3 50 20

P j

2) 10 25 3 50 20

i,P j

25 < 20 False

i

3) 10 25 3 50 20

P j

3 < 20 True

10 3 25 50 20

i P j

i

4) 10 3 25 50 20

i j

P

50 < 20 False

10 3 20 50 25

< 20 P > 20

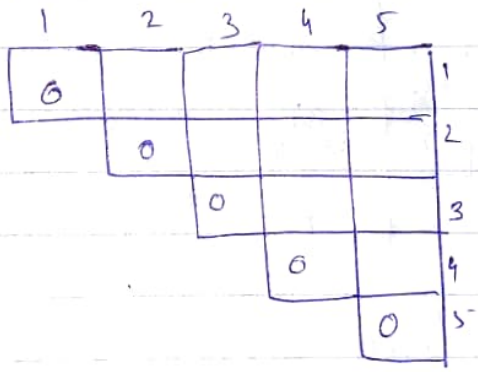
10	3	10<3	false	50	25	50<25	False
p,i	j			p,i	j		

3 10 | 25 , 50

sorted array 3 10 20 25 50

Matrix chain multiplication

Ex: We are given the sequence [4, 10, 3, 12, 20, 7]
The matrices have size 4x10, 10x3, 3x12, 12x20, 20x7.
We need to compute M[i,j] 0 ≤ i, j ≤ 5. We know
M[i,i]=0 for all i



n = 6

no. of ~~bote~~ blocks = 5

step-1 Draw the matrix

step-2 Assign the product numbering.

P_0	P_1	P_2	P_3	P_4	P_5
4	10	3	12	20	7

step-3 Check the value of i,j
if i=j then result value = 0

step-4 if $i < 7$ then

do {

$m[i,j] = \min[i,k] + \min[k+1, j] + P_{i-1} \cdot P_j \cdot P_k$ }

Now for, $i = 1, j = 2$

assume $k = 1$ $(i \le k < j)$

$m[1,2] = \min[1,1] + \min[1+1, 2] + P_0 \cdot P_2 \cdot P_1$

$= 0 + 0 + (4 \times 3 \times 10)$

$= 120$

$m(1,2) = m_1 \times m_2 = (4 \times 10) \times (10 \times 3) = 4 \times 10 \times 3 = 120$

$m(2,3) = 10 \times 3 \times 12 = 360$

$m(3,4) = 3 \times 12 \times 20 = 720$

$m(4,5) = 12 \times 20 \times 7 = 1680$

1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

$m(1,3) \neq m_1 \cdot m_2 \cdot m_3 +$

$M(1,3) = \min \begin{cases} M[1,2] + M[3,3] + P_0 P_2 P_3 = 120 + 0 + 4 \cdot 3 \cdot 12 = 264 \\ M[1,1] + M[2,3] + P_0 P_1 P_3 = 0 + 360 + 4 \cdot 10 \cdot 12 = 840 \end{cases}$

take min value.

$M[1,3] = 264$

$M(2,4) = \min \begin{cases} M[2,3] + M(4,4] + P_1 P_3 P_4 = 360 + 0 + 10 \cdot 12 \cdot 20 = 2760 \\ M[2,2] + M[3,4] + P_1 P_2 P_4 = 720 + 10 \cdot 3 \cdot 20 = 1320 \end{cases}$

$M[2,4] = 1320$

$M[3,5] = 1140$

1	2	3	4	5	
0	120	264			1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

$M[1,4] = M_1 M_2 M_3 M_4$

$$M[1,4] = \min \begin{cases} M[1,3] + M[4,4] + p_0 p_3 p_4 = 1224 \\ M[1,2] + M[3,4] + p_0 p_2 p_4 = 1080 \\ M[1,1] + M[2,4] + p_0 p_1 p_4 = 2120 \end{cases}$$

$M[1,4] = 1080$

$M[2,5] = 1350$

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

$$M[1,5] = \min \begin{cases} M[1,4] + M[5,5] + p_0 p_4 p_5 = 1080 + 0 + 4 \cdot 20 \cdot 7 = 1544 \\ M[1,3] + M[4,5] + p_0 p_3 p_5 = 264 + 1680 + 4 \cdot 12 \cdot 7 = 2016 \\ M[1,2] + M[3,5] + p_0 p_2 p_5 = 120 + 1140 + 4 \cdot 3 \cdot 7 = 1344 \\ M[1,1] + M[2,5] + p_0 p_1 p_5 = 0 + 1350 + 4 \cdot 10 \cdot 7 = 1630 \end{cases}$$

$M[1,5] = 1344$

1	2	3	4	5	
0	120	264	1080	1344	1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

Exponential:

$X = a^n$

$$a^n = \begin{cases} a & \text{if } n=1 \\ (a^{n/2})^2 & \text{if } n \text{ is even} \\ (a \cdot a^{n-1}) & \text{if } n \text{ is odd} \end{cases}$$

$X = a^{27}$

$= a \cdot a^{26}$

$= a \cdot (a^{13})^2$

$= a \cdot (a \cdot a^{12})^2$

$= a\,(a \cdot (a^6)^2)^2$

$= a\,(a \cdot (a^3)^2)^2)^2$

$= a\,(a\,(a \cdot a^2)^2)^2)^2$

Algo

```
f^n expo (a, n) {
    if n=1 return a
    if n is even => return [expo (a, n/2)]
    if n is odd then return a * expo(a, n-1)
```

Time complexity

$$T(n) = \begin{cases} 0 & n=1 \\ T(n/2) + 1 & n = \text{even} \\ T(n-1) + 1 & n = \text{odd} \end{cases}$$