02.05.2024

Algorithm UniqueElements(A[0..n-1])
// Determines whether all the elements in a given array are distinct
// Input: An array A[0...n-1]
// Output - Return "true" if all the elements in A are distinct
// and "false" otherwise

```
for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        if A[i] = A[j] return false
return true
```

~~return true~~

Eg:-

	0	1	2	3	4
A	5	3	2	5	1

n = 5

I i = 0 to 3

(1.) j = 0+1 = 1 to 4
A[0] = A[1]
5 = 3
False

(2.) j = 1+1 = 2
A[0] = A[2]
5 = 2
False

(3.) j = 2+1 = 3
A[0] = A[3]
5 = 5
True

→ Loop stops when "true" is returned.

- Best case:- when first 2 elements are same.
- Worst case:- when all elements are distinct.

$$\text{Worst}(n) = \sum_{i=0}^{n-2} \cdot \sum_{j=i+1}^{n-1} 1 \rightarrow \text{no of basic operation}$$

$$\sum_{i=0}^{n-2} = n-2-0+1 = n-1$$

$$\sum_{j=i+1}^{n-1} = n-1-(i+1)+1 = n-1-i-1+1 = n-i-1$$

*Matrix Multiplication

General Plan for Analyzing the Time Efficiency of a Recursive Algorithm

- Decide on a parameter (s) ~~indicated~~ indicating an input size. → value of n
- Identify the algorithm's basic operation.
- Check whether the no. of the times the basic operation is executed can vary on diff. inputs of same size, it can be best case, worst case, or avg case, must be investigated separately.
- Set up a recurrence relation, with an appropriate initial condition, for the no. of times the basic operation is executed.
- Solve the recurrence, or, atleast ascertion the order of growth of its solution.

Eg:- $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

① $5! = 5 \times 4!$ ← value of $4! \leftarrow 5 \times 24 = 120$

② $4! = 4 \times 3!$ ← value of $3! \leftarrow 4 \times 6 = 24$

③ $3! = 3 \times 2!$ ← value of $2! \leftarrow 3 \times 2 = 6$

④ $2! = 2 \times 1!$ ← value of $1! \leftarrow 2 \times 1 = 2$

⑤ $1! = 1$

} Backward Recurrence Relation

Algorithm F(n)

// Computes n! recursively

// Input: A non-negative integer n

// Output: The value of n!

if n=0 return 1

else return F(n-1)*n.

→ no. of times multiplication takes place in F(n).

Let M(n) → No. of times recursion takes place

F(n) → function.

F(n) = F(n-1) * n

M(n) = M(n-1) + 1 → basic operation

↓

No. of times multiplication takes place in F(n-1).

initial condition = ~~n=0~~. stopping condition [same]

n=0 → return 1

M(0)=0. → [Executed only once but not called again].

F(0)=1 . [Executed once to get 0!=1.]

M(n) = M(n-1) + 1

M(n-1) = M(n-2) + 1

M(n-2) = M(n-3) + 1

Generally, M(n) = M(n-k) + k $\in$ ~~$\Theta(n)$~~ $\Omega(n)$.

k → any variable considered for 'for' loop

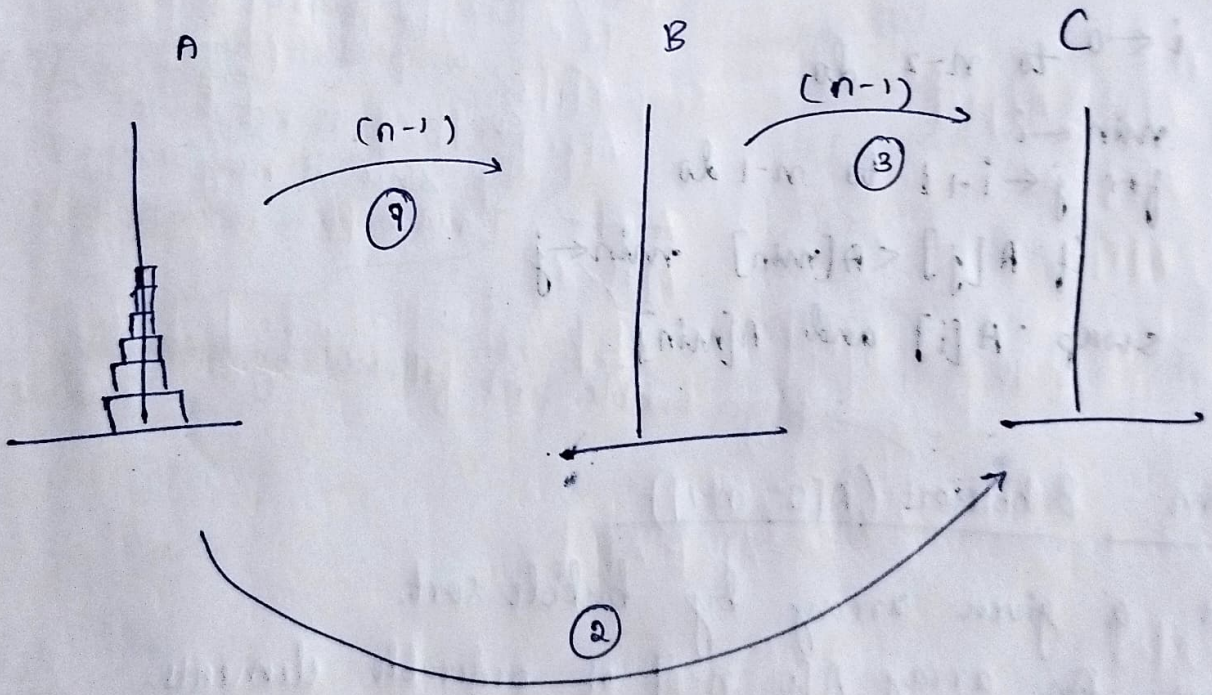
* Recurrence Tree → no. of times recurrence function executes.

M(n)
|
M(n-1)
|
M(n-2)
|
.
.
.
M(0)

Say n=20, M(20)=21
n=5, M(5)=6

* Tree height → less → Best case.
→ more → Worst case

Tower of Hanoi



$M(n) = 2M(n-1)+1$ → for 5 recursions.

Best case:- when $n=5$, $M(5)=31 \in O(n)$.

→ $O(2^n)$.

02.05.2024

Module 2

Brute Force Technique

- A straight forward approach towards a problem.

~~Types~~

Algorithm SelectionSort(A[0..n-1])

// Sorts a given array by selection sort.

// Input:- An array A[0...n-1] of orderable elements.

// Output:- Array A[0...n-1] of sorted in nondecreasing order.

```
for i ← 0 to n-2 do
    min ← i
    for j ← i+1 to n-1 do
        if A[j] < A[min]  min ← j
    swap A[i] and A[min]
```

Only 1 basic operation i.e, comparison (happens in every step). Swapping is not basic step since it may or may not happen in every step.

→ bigger bubble first followed by smaller bubbles

Algorithm BubbleSort(A[0..n-1])

// Sorts a given array by bubble sort.

// Input:- An array A[0...n-1] of orderable elements.

// Output:- Array [0..n-1] sorted in nondecreasing order.

```
for i ← 0 to n-2 do
    for j ← 0 to n-2-i do
        if A[j+1] < A[j] swap A[j] and A[j+1].
```

Eg:-

Selection Sort

A | 2 | 7 | 4 | 1 | 5 | 3 |

(1) i = 0
min ← 0
j = i + 1 = 0 + 1 = 1
A[j] < A[min]
A[1] < A[0]
~~7 < 2~~ 7 < 2
False
~~min = j = 1~~ min = 0.

(2) i = 0
min ← 0
j = 1 + 1 = 2
A[2] < A[0]
4 < 2
False.
min ← 0

(3). i = 0.
min ← 0.
j = 2 + 1 = 3
A[3] < A[0]
1 < 2
True
~~A[0] A 1~~
~~A[3] = 2~~
min = 3.

(4) i = 0
min ← 3
j = 3 + 1 = 4
A[4] < A[3]
5 < 1
False

(5.). i = 0
min ← 3
j = 4 + 1 = 5
A[5] < A[3]
3 < 1
False

Swap A[i] & A[min]
A[0] & A[3]
A[0] = 1
A[3] = 2.

06.05.2024

Bubble Sort

n → no of elements
i → no of pass
j → for elements

	0	1	2	3	4	5	6
A	89	45	68	90	29	34	17

Pass : 1

(1) i = 0
j = 0
A[j+1] < A[j]
A[1] < A[0]
45 < 89
True
Swap A[j] & A[j+1]
∴ A[0] = 45
A[1] = 89

45	89	68	90	29	34	17

(2) i = 0
j = 1
A[j+1] < A[j]
A[2] < A[1]
68 < 89
True
Swap A[j] & A[j+1]
~~A[j] =~~
A[1] = 68
A[2] = 89

45	68	89	90	29	34	17

(3) i = 0
j = 2
A[3] < A[2]
90 < 89
False.
j = j+1 = 2+1 = 3

(4) i = 0
j = 3
A[4] < A[3]
29 < 90
True
∴ A[3] = 29
A[4] = 90

A

45	68	89	29	90	34	17

(5) i = 0
j = 4
A[5] < A[4]
34 < 90 ⇒ True

A

45	68	89	29	34	90	17

(6) · i = 0
j = 5
A[6] < A[5]
17 < 90 ·
True ·
Swap A[6] & A[5]
A[5] = 17
A[6] = 90 ·

	0	1	2	3	4	5	6
A	45	68	89	29	34	17	90

Pass : 2

(1) · i = 1
j = 0 ·
A[1] < A[0]
68 < 45
False
j = j+1 = 0+1 = 1 ·

(2) · i = 1
j = 1
A[2] < A[1]
89 < 68 ~~< 89~~
False
j++ = 1+1 = 2 ·

(3) · i = 1
j = 2
A[3] < A[2]
29 < 89
True ·
A[2] = 29
A[3] = 89 ·

(4) · i = 1
j = 3 ·
A[4] < A[3]
34 < 89
True
A[3] = 34
A[4] = 89 ·

(5) · i = 1
j = 4
A[5] < A[4]
17 < 89
True
A[4] = 17
A[5] = 89 ·

A	45	68	29	34	17	89	90

Pass : 3

(1) i = 2
j = 0
A[1] < A[0]
68 < 45
false
j++ = 1 ·

(2) · i = 2
j = 1
A[2] < A[1]
29 < 68
True ·
A[1] = 29
A[2] = 68

(3) · i = 2
j = 2
A[3] < A[2]
34 < 68
True
A[2] = 34
A[3] = 68

i=2
j=3
A[4] < A[3]
17 < 68
True
A[3] = 17
A[4] = 68

∴ A

45	29	34	17	68	89	90

Pass: 4

(1) i=3
j=0
A[1] < A[0]
29 < 45
True
A[0] = 29
A[1] = 45

(2) i=3
j=1
A[2] < A[1]
34 < 45
True
A[1] = 34
A[2] = 45

(3) i=3
j=2
A[3] < A[2]
17 < 45
True
A[2] = 17
A[3] = 45

⇒A

29	34	17	45	68	89	90

Pass: 5

(1) i=4
j=0
A[1] < A[0]
34 < 29
False
j++

(2) i=4
j=1
A[2] < A[1]
17 < 34
True
A[1] = 17
A[2] = 34

∴ A

29	17	34	45	68	89	90

Pass: 6

(1) i=5
j=0
A[1] < A[0]
17 < 29
True
A[0] = 17
A[1] = 29

⇒A

17	29	34	45	68	89	90

For Analysis of Selection Sort

$$= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (n-1) - (i+1) + 1$$

$$= \sum_{i=0}^{n-2} n-1-i-i+1$$

$$= \sum_{i=0}^{n-2} (n-i-1)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \quad -(*)$$

Consider, $\sum_{i=0}^{n-2} i = \sum_{i=0}^{n-2} \frac{n(n+1)}{2}$ [To remove Σ, replace n by $n-2$]

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

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

Consider, $\sum_{i=0}^{n-2} (n-1) = (n-1) \sum_{i=0}^{n-2} 1$

$$= (n-1)\{(n-2)-0+1\}$$

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

Substituting in (*)

$$= (n-1)(n-1) - \frac{(n-2)(n-1)}{2} = (n-1)\left[(n-1) - \frac{(n-2)}{2}\right]$$

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

Order of Growth = $O(n^2)$ or $O(n^2)$

↓ highest degree.

Bubble Sort

$$= \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1$$

$$= \sum_{i=0}^{n-2} (n-2-i) - 0 + 1 \quad = \sum_{i=0}^{n-2} (n-i-1)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \qquad -(1).$$

Consider $\sum_{i=0}^{n-2} i = \frac{n(n+1)}{2}$.

Replace n by $n-2$

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

$$\sum_{i=0}^{n-2} i = \frac{(n-2)(n-1)}{2} \qquad -(2)$$

Consider $\sum_{i=0}^{n-2} (n-1) = (n-1) \sum_{i=0}^{n-2} 1$

$$= (n-1)\{(n-2) - 0 + 1\}$$

$$= (n-1)(n-1) \qquad -(3)$$

Substituting (2) & (3) in (1)

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

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

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

$= \frac{n^2-n}{2}$

Order of growth = $O(n^2)$ or $\Theta(n^2)$.

String Matching

* text is a string.
* pattern is a substring

If length of string is m and length of pattern is n, then $n \leq m$

07.05.2024

```
Algorithm BruteForceStringMatch (T[0..n-1], P[0..m-1])
// Implements brute-force string matching
// Input:- An array T[0..n-1] of n characters representing a
           text and an array P[0..m-1] of m characters
           representing a pattern.
// Output:- The index of the first character in the text that
            starts a matching substring or -1 if search is
            unsuccessful.
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[i+j] do
        j = j+1
    if j = m return i
return -1
```

Order of Growth = $O(n^2)$ or $\Theta(n^2)$

Highest degree

Bubble Sort

$$= \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1$$

$$= \sum_{i=0}^{n-2} (n-2-i) - 0 + 1 \quad = \sum_{i=0}^{n-2} (n-i-1)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \qquad -(1).$$

Consider $\sum_{i=0}^{n-2} i = \frac{n(n+1)}{2}$

Replace n by $n-2$

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

$$\sum_{i=0}^{n-2} i = \frac{(n-2)(n-1)}{2} \qquad -(2)$$

Consider $\sum_{i=0}^{n-2} (n-1) = (n-1) \sum_{i=0}^{n-2} 1$

$$= (n-1)\{(n-2) - 0 + 1\}$$

$$= (n-1)(n-1) \qquad -(3)$$

Substituting (2) & (3) in (1)

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

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

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

$= \frac{n^2-n}{2}$

Order of growth = $O(n^2)$ or $\Theta(n^2)$.

String Matching

* text is a string.
* pattern is a substring

If length of string is m and length of pattern is n, then $n \leq m$

07.05.2024

```
Algorithm BruteForceStringMatch (T[0..n-1], P[0..m-1])
// Implements brute-force string matching
// Input:- An array T[0..n-1] of n characters representing a
           text and an array P[0..m-1] of m characters
           representing a pattern.
// Output:- The index of the first character in the text that
            starts a matching substring or -1 if search is
            unsuccessful.
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[i+j] do
        j = j+1
    if j = m return i
return -1
```

j → represents pattern
i → represents text

Eg:- ~~N0~~ N O B O D Y _ N O T I C E D _ H I M

N O T

N O T

N O T.

~~N O T~~

N O T

N O T

N O T

N O T

N O T

Outer loop → (n-m+1) comparisons
Inner loop → m comparisons

Worst case = m x (n-m+1)

Avg. case = n+m

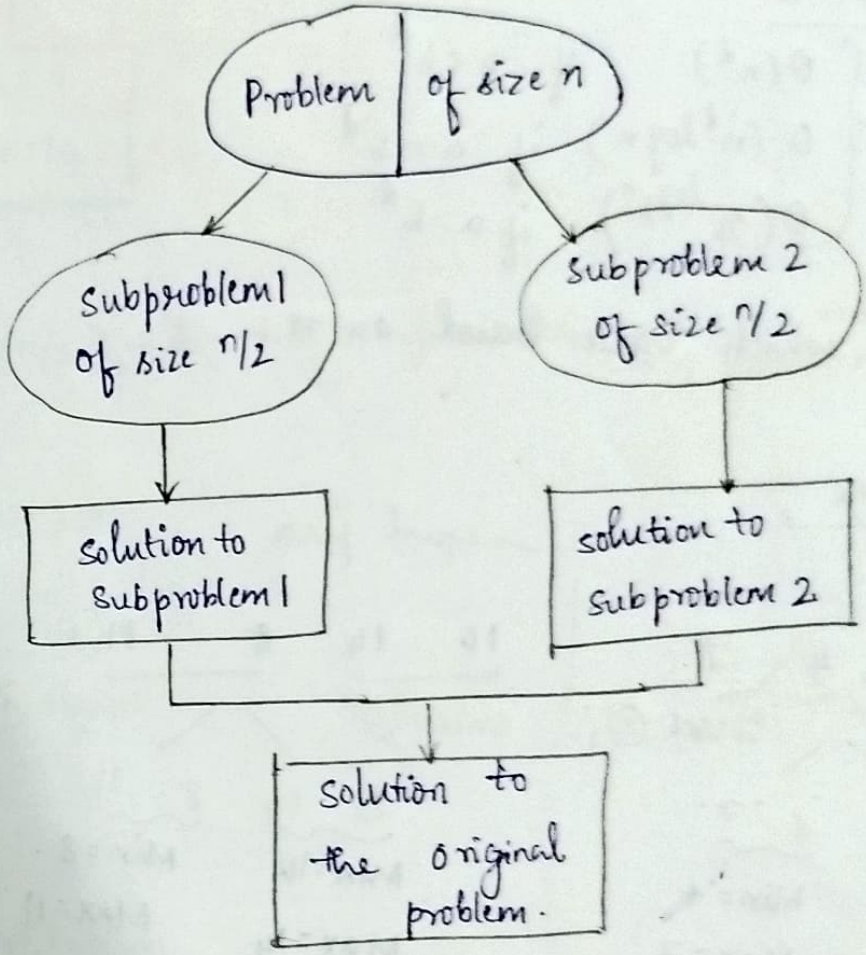
~~Best case~~

Divide-and-Conquer

- A major problem divided to ~~equal no.~~ subtopics (equal no. of groups)
  - ↓
  - ~~Eac~~ Soln. for each ~~sub~~ sub-problem
  - ↓
  - Merge ↓ soln. of each
  - ↓
  - one ~~so~~ final solution for the major problem.

(Q) Example of a problem with 2 diff. solutions (one using Brute and the other using Divide-and-Conquer).

Typical case of Divide-and-Conquer Technique



11.05.2024

Coin Counterfeit Problem

$$T(n) = aT(n/b) + f(n)$$

$f(n)$ → func. which gives solution to the problem.

~~a~~ → $T(n)$ → time taken to solve a given problem of n inputs

n → total no. of coins taken in the problem.

a, b → constants.

* Order of growth dependent on n.

* If n increases, order of growth increases.
Similarly if n decreases, order of growth decreases.

※ ※

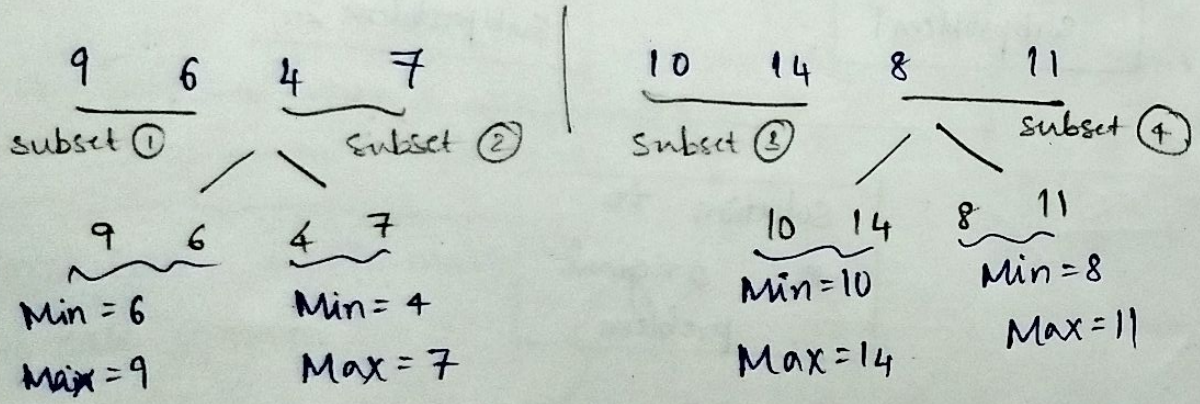
Master Theorem

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

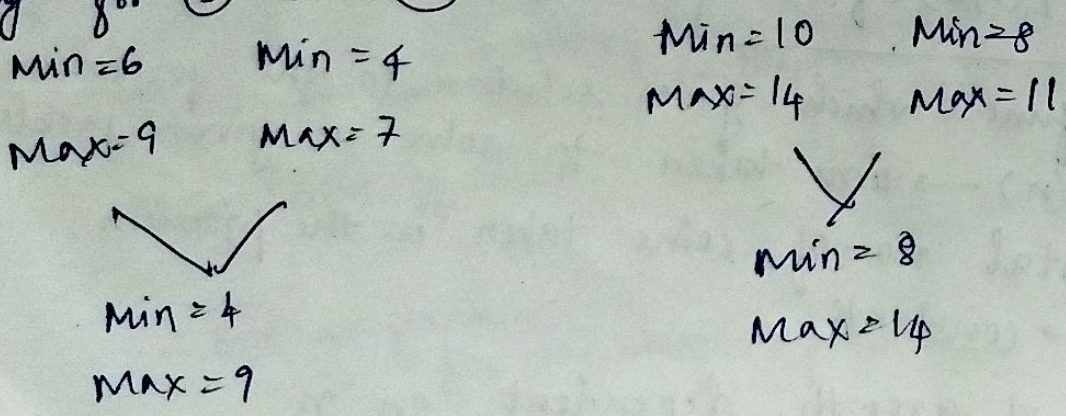
* d → value which varies based on n.

Eg:- Min. Max.

→ equal parts.



- Compare ① min & ② min ⇒ ✱ result = min
  Compare ① max & ② min ⇒ result = max.
- Similarly ~~9 and~~ for ③ and ④

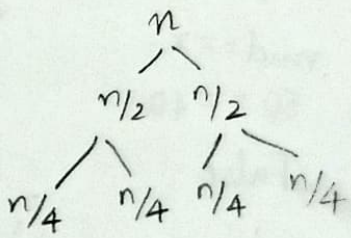


- Min = 4, Max = 9 | Min = 8, Max = 14

Compare

Min = 4
Max = 14

* $T(n) = 2T(n/2) + 2$ → 2 comparisons → 1 for min, 1 for max
  - 2 equal parts of n



* $T(n/2) = 2T(n/4) + 2$

⋮

$T(1)$

Recurrence Relation

Binary Search

```
mid = (i+1)/2
    if (a[mid] == x)
        return (mid)
    else if (a[mid] > x)
        Binarysearch (a, i, mid-1, x)
    else
        Binarysearch (a, mid+1, j, x)
```

i → start
j → end

Eg:- 10 20 30 40 50 60

i → ↑ mid ← j

x = 40.

(1) mid == x

30 == 40

False

(2) a[mid] > x

30 > 40

False

~~(3)~~ 10 20 30 | 40 50 60

i j

40 50 60. => (n/2)/2 = n/4.

↓ ↓ ↓

i mid j

mid == x.

(1). 50 == 40.

False

(2). a[mid] > x.

50 > 40

True

i = 40

j = mid - 1 = 40.

mid = 40.

(3) 40 == 40

True.

Recurrence Relation

only 1 subset is enough for comparison.

$T(n) = 1 \cdot T(n/2) + c$

where c → c no. of comparisons

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

$T(n/2) = T(n/4) + c$

$T(n/4) = T(n/8) + c$

.

.

.

$T(1)$.

$\Rightarrow T(n) = T(n/4) + 2c = T(n/8) + 3c = T(n/16) + 4c$

$= T(n/2^2) + 2c = T(n/2^3) + 3c = T(n/2^4) + 4c.$

$= T(n/2^k) + kc$

If $2^k = n$

$T(n) = T(n/n) + kc$

$$\boxed{T(n) = T(1) + kc} \quad -(1)$$

Consider $2^k = n$

Take $\log_2$ on both sides.

$k \log_2 2 = \log_2 n$ ($\log_2 2 \to 1$)

$$\boxed{k = \log_2 n} \quad -(2)$$

Substituting k in (1)

$$\boxed{T(n) = T(1) + \log n \cdot c}$$ → To express the whole equation in terms of n

$$\boxed{T(n) = \log_2 n.}$$ → Recurrence Relation.

Merge Sort

Algorithm Mergesort (A[0 .... n-1])

// Sorts array A[0... n-1] by recursive mergesort

// I/p: An array A[0... n-1] of ordered elements.

// O/p:- An array A[0.. n-1] sorted in ascending order.

~~if A>1~~

if n>1

Copy A[0.. $\lceil n/2 \rceil - 1$] to B[0... $\lceil n/2 \rceil - 1$]

Copy A[$\lceil n/2 \rceil$. n-1] to C[0.... $\lceil n/2 \rceil - 1$]

Mergesort (B[0.... $\lceil n/2 \rceil - 1$]).

Mergesort (C[0... $\lceil n/2 \rceil - 1$]).

Merge (B,C,A).

13-05-2024

Algorithm Merge (B[0.,p-1], C[0..q-1], A[0.. p+q-1])

// Merges 2 sorted arrays into one sorted array.

// Input- Arrays, B[0...p-1] and C[0..q-1] Both sorted.

// Output:- Sorted array A[0.. p+q-1] of the elements of B and C.

i←0; j←0; k←0.

while i<p and j<q do

if B[i] ≤ C[j]

A[k]←B[i]; i=i+1.

else $A[k] \leftarrow C[j]$; $j = j+1$

$k \leftarrow k+1$

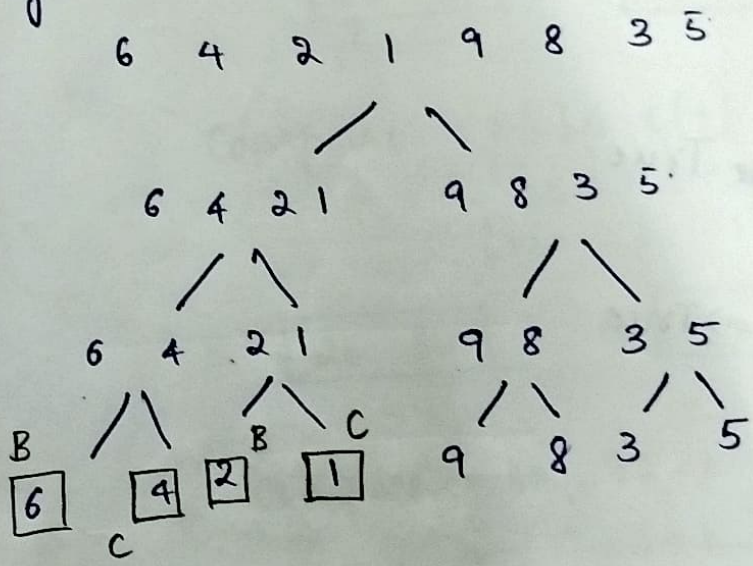
if $i = p$

copy $C[j \ldots q-1]$ to $A[k \ldots p+q-1]$.

else copy $B[i \ldots p-1]$ to $A[k \ldots p+q-1]$.

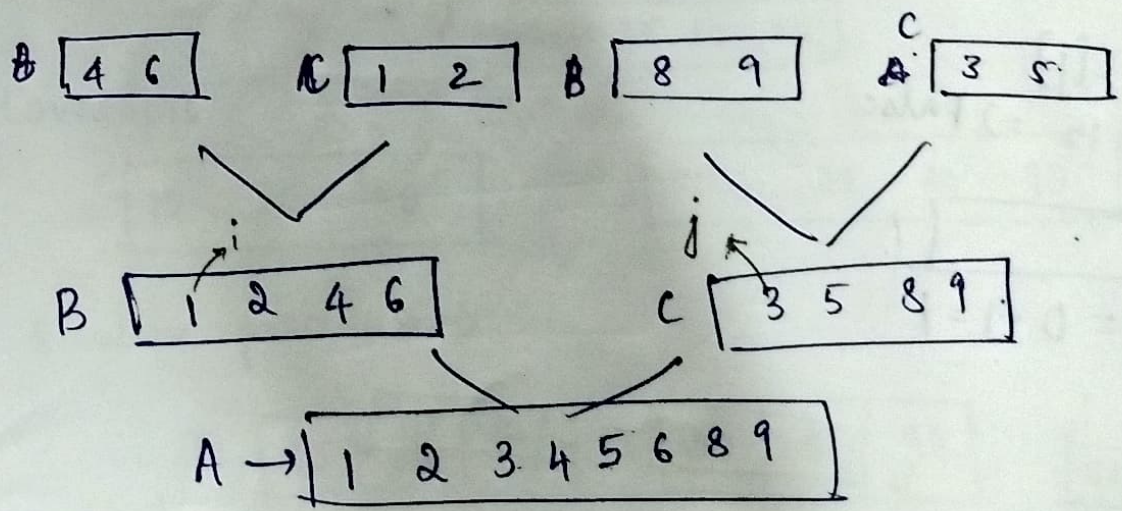
* Merge sort stops the division when there is a single element in the array. → merge called.
* Merge works by conquering the solution.

Eg:-

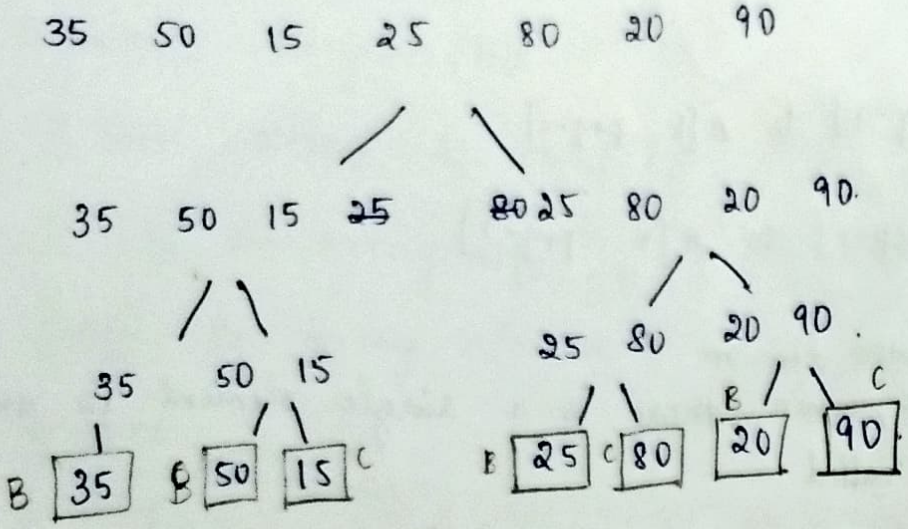


[Divide until there is 1 element].

- Assume $6 \in B$, $4 \in C$.
  Compare $6 \leq 4$ => False



(1). 35 50 15 25 80 20 90.



Compare ~~25, 80,~~ ~~25~~, 50, 15

50 <= 15 => False

35 | 15 50 |

Compare 25, 80 => 25 ≤ 80 => ~~False~~ True.

| 25 80 |

Compare 20, 90 => 20 ≤ 90 => True

| 20 90 |

=>

∴ B | 35 | C | 15 50 | B | 25 80 | C | 20 90 |

i j

Compare B[i] ≤ C[j]

35 ≤ 15 => False

∴ A | 15 |

j = j+1 = 0+1 = 1

i ~~i=0~~ i

B | 35 | C | 15 50 | B | 25 80 | C | 20 90 |

Compare $B[i] \leq C[j]$

$35 \leq 50 \Rightarrow$ True $\Rightarrow$ A [15 35

$i = i+1 = 0+1 = 1$

$i = p \Rightarrow$ Copy remaining elements of C to A.

A [15 35 50.

$\Rightarrow$.

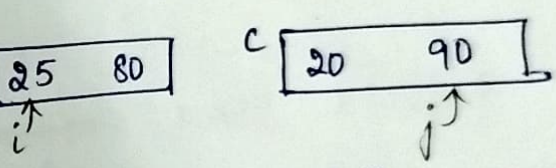
Consider B [25 80] C [20 90]

Compare $B[i] \leq C[j]$

$25 \leq 20 \Rightarrow$ False $\Rightarrow$ A [20

$j = j+1 = 0+1 = 1$

B [25 80] C [20 90]

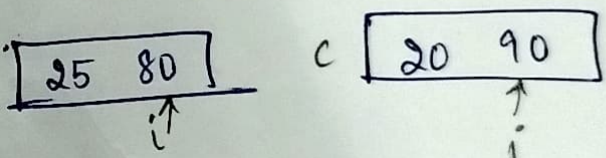


Compare $B[i] \leq C[j]$

$25 \leq 90 \Rightarrow$ True $\Rightarrow$ A [20 25

$i = i+1 = 0+1 = 1$

B [25 80] C [20 90]



Compare 80, 90 $\Rightarrow$ $80 \leq 90 \Rightarrow$ True $\Rightarrow$ A [20 25 80]

~~$i = i+1 = 1+1$~~

$i = p \Rightarrow$ True.

Copy all remaining elements of C.

A [20 25 80 90]

Consider;

[15 35 50.]

[20 25 80 90.]

Similar procedure $\Rightarrow$

A [15 | 20 | 25 | 35 | 50 | 80 | 90]

Algorithm Partition $A[l..r]$

// Partition

Algorithm Quick Sort ($A[l..r]$)

// sorts a subarray of quicksort

// Input - Subarray of array $A[0..n-1]$ defined by its left and right indices l and r.

// Output - Sub array $A[l...r]$ sort

* The very first element in the array → considered as pivot element.
* 2 indices → say i, j
* i → next element to pivot → always increment ← left to right
* j → points to last element → always decrement ← right to left
* Arrange such that the elements < pivot ⇒ left
  elements > pivot ⇒ right

Po

Algorithm HoarePartition (A[l r])

// Partitions a subarray by Hoare Partition (A[l...r])

// Input: Subarray of array A[0 n-1], defined by its left & right indices l and r (l<r).

// Output: Partition of A[l...r], with the split position returned as this function is value.

```
p ← A[l]
i ← l ; j ← r+1
repeat
    repeat i ← i+1 until A[i] ≥ p
    repeat j ← j
```

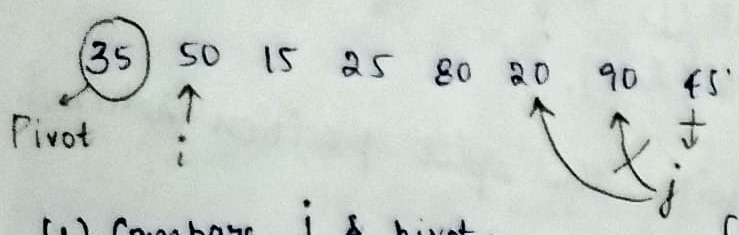
Quick Sort

Eg:- 35 50 15 25 80 20 90 45

$A[i] \geq p \Rightarrow$ stop

$A[j] \leq p \Rightarrow$ Stop

Pass I



(1) Compare i & pivot.

$50 \geq 35 \Rightarrow$ ~~stop~~ True

(i → no more incremented)

(2) Compare j & pivot

$35 \leq 45 \Rightarrow$ True

(decrement j)

(3) Compare j & pivot

$35 \leq 90 \Rightarrow$ True

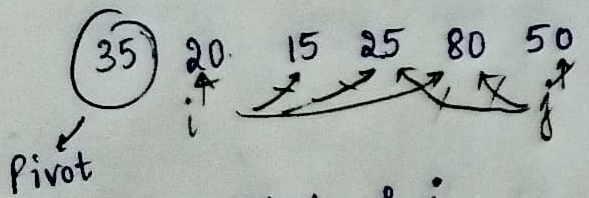
decrement j (j--)

(4) Compare j & pivot

$35 \leq 20 \Rightarrow$ False

(stop j decrement)

If at this point, i & j do not have a cross-over, then swap element pointed by j and i.



(1) Compare pivot & i

$35 \leq 20 \Rightarrow$ False

increment i

i++.

(2) Compare pivot & i

$35 \leq 15 \Rightarrow$ False

i++

(3) $35 \leq 25 \Rightarrow$ False

i++

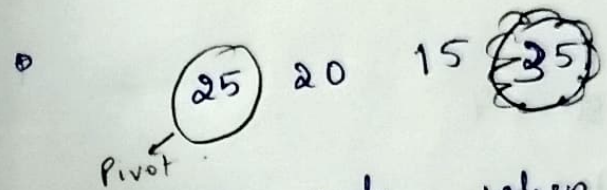
(4) $35 \leq 80 \Rightarrow$ True

(5) Compare j & pivot
$35 \leq 50$ => True
j--

(6.) Compare 80 & 35
$35 \leq 80$ => True
j--

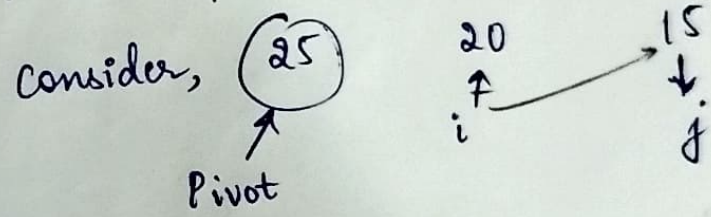
(7) Compare 25 & 35
$35 \leq 25$ => False
(stop j decrement)

(8) Swap j & pivot
~~(whenever~~ (If i & j cross-over)



* Pass I ends when there is a swap of j & pivot.
* ~~Pivot element~~ 35 here → acts as middle of partition.
* i must points to element next to pivot

Consider,



(1). $20 \geq 25$ => False
i++

(2) $15 \geq 25$ => False
Stop

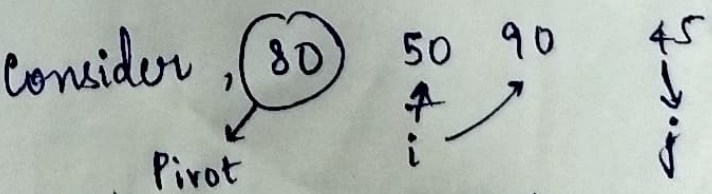
(3). Compare j & pivot
& $15 \leq 35$ => True.
(stop)

(4) Swap pivot & j

15	20	25

=> Sorted

Consider,



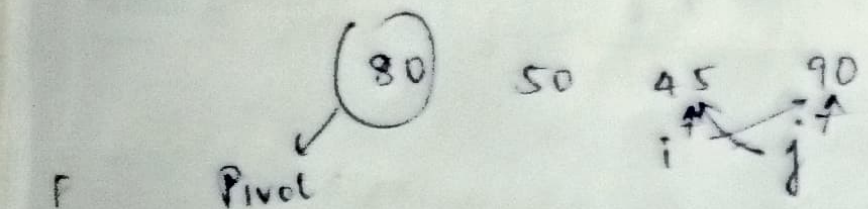
(1). Compare i & pivot
$50 \geq 80$ => False
i++

(2). $90 \geq 80$ => True.
~~i++~~ Stop

(3) Compare j & pivot

$45 \leq 80 \Rightarrow$ True.

~~j--~~ Stop



(4) ~~Compare~~ Swap i & j
(since no crossover)

(5.) ~~Com~~ $45 \geq 80 \Rightarrow F$

i++

(6.) $90 \geq 80 \Rightarrow T$

Stop

(7) $90 \leq 80 \Rightarrow F$

j--

(8) $45 \leq 80 \Rightarrow T$

Stop.

(9) swap j & pivot

45	50	80	90

Sorted array:-

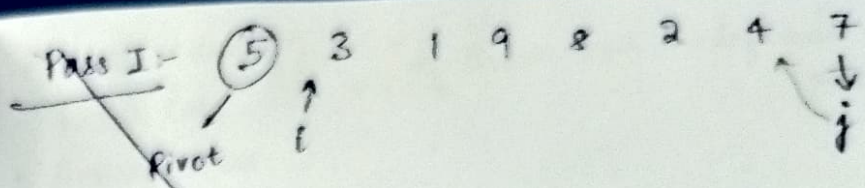
15 20 25 35 45 50 80 90.

Apply quick sort on the following set of arrays:-

~~5 3~~ 5 3 1 9 8 2 4 7

~~Pass I:-~~

Pass I :-

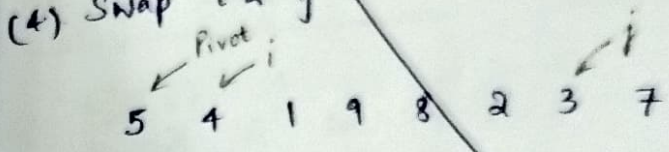


(1) $3 \leq 5 \Rightarrow T$
stop

(2) $5 \leq 7 \Rightarrow T$
j--

(3) $5 \leq 4 \Rightarrow F$
stop

(4) Swap i & j



(1) $4 \leq 5 \Rightarrow T$
Stop i

(2) $3 \leq 5$

- Best case
- Worst case
- Average case

} → of Quick sort → depends on pivot element

Average case:-

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

Apply MergeSort & Quicksort for the following:-

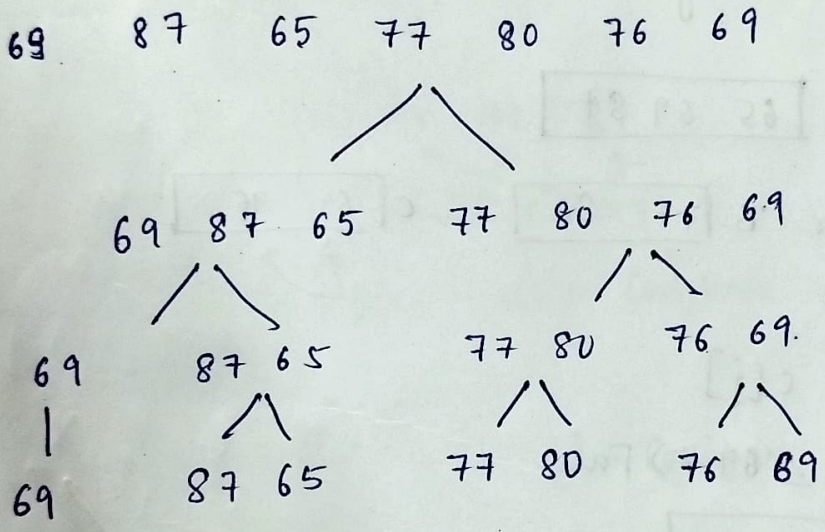
69 87 65 77 80 76 69

E X A M P L E

A→65 B C D E F G H
E→7 I J K L M N
O P Q R S T U
V X Y Z

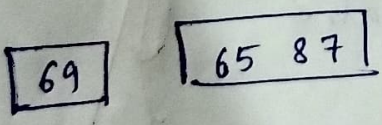
- Compare ASCII values.

Merge Sort

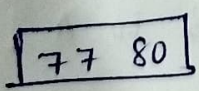


Assume 87 ∈ B, 65 ∈ C.

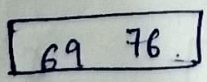
Compare 87 & 65 => ~~65~~ $87 \leq 65$ => False.

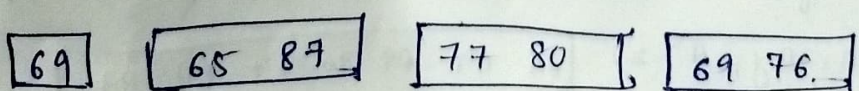


Compare $77 \leq 80$ => True



Similarly, 76, 69.





B [69] C [65 87] B [77 80] C [76 69 76]

i ↑ j ↑

Compare i & j => 69 ≤ 65 => False

A [65]

j++

Compare 69 & 87 => 69 ≤ 87 => True

A [65 69]

i++

i = p.

∴ Copy remaining elements of C to A.

A [65 69 87]

Similarly for B [77 80] C [69 76]

i ↑ j ↑

Compare B[i] ≤ C[j]

77 ≤ 69 => False

A [69]

j++

Compare 77 & 76 => 77 ≤ 76 => False

A [69 76]

Copy remaining elements of B to A.

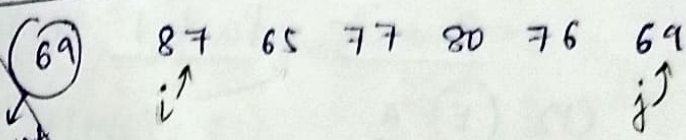
A [69 76 77 80]

Similarly for B [65 69 87] A [69 76 77 80]

Sorted Array, A => [65 69 69 76 77 80 87]

Accordingly => Array is A E E L M P X.

Quick Sort

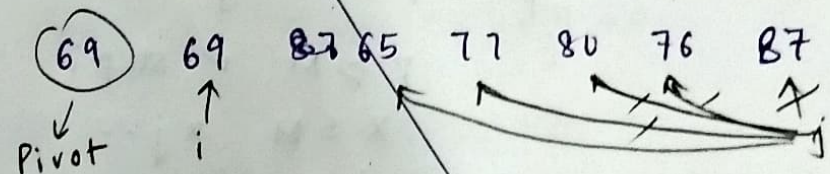


Pass I:-

(1) Compare i & pivot
69 ≤ 87 => ~~False~~ True
~~i++~~ Stop

(2). Compare j & pivot
69 ≤ 69 => True
Stop.

(3). Swap i & j



(4). Compare 69 ≤ 69 => True
Stop.

(5). Compare 69 ≤ 87 => True
j--

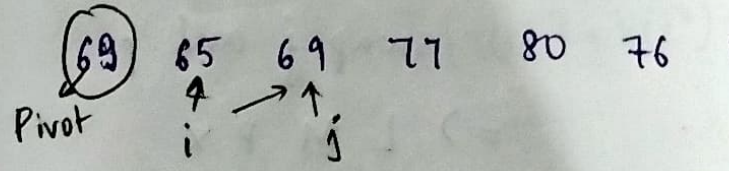
(6). Compare 69 ≤ 76 => True
j--

(7). Compare ~~80~~ 69 ≤ 80 => True
j--

(8). Compare 69 ≤ 77 => True
j--

(9). Compare 69 ≤ 65 => False
Stop.

(10). Swap i & j



(11). Compare 65 ≥ 69 => False
i++

(12). Compare 69 ≥ 69 => True
Stop.

(13). Compare 69 ≥ 69 => True
Stop.

Quick Sort

E X A M P L E

P ↑ i ↑ j

Pass I

(1) X ≥ E ✓ stop i

E ≤ E ✓ stop j

swap i & j

(2) E E A M P L E

P ↑ i ↑ j

E ≥ E ✓ stop i

X ≤ E × j--

L ≤ E × j--

P ≤ E × j--

M ≤ E × j--

A ≤ E ✓ stop j

Swap i and j

(3) E A E M P L X

P ↑ i ↑ j

E ≥ A × j++

E ≥ E ✓ stop i

E ≤ E ✓ stop j

swap j with p.

E A (E) M P L X

↑ p.

~~Pass II~~ Part 1

(1) (F) A

P ↑ swap j and p

(2) A E

~~(3) Pass III~~ Part 2

M P L X

P ↑ i ↑ j ↑

p ≥ M ✓ stop i

X ≤ M × j--

L ≤ M ✓ stop j

swap i & j

M L P X

p. ↑ i ↑ j ↑

L ≥ M × i++

P ≥ M ✓ stop i

P ≤ M × j--

L ≤ M ✓ stop i

swap j and p.

=> L M P X

Sorted => A E E L M P X

16.05.2024

Merge sort Time Complexity

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

$T(n/2) = 2T(n/4) + n/2 \rightarrow (1)$ [∵ only half of the array is considered]

$T(n/4) = 2T(n/8) + n/4 \rightarrow (2)$

$T(n/8) = 2T(n/16) + n/8 \rightarrow (3)$

General :-

$$T(n) = 2\left[2T(n/4) + n/2\right] + n$$

$$= 4T(n/4) + \frac{2n}{2} + n$$

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

$$= 4\left[2T(n/8) + n/4\right] + 2n$$

$$= 8T(n/8) + \frac{4n}{4} + 2n$$

$$= 8\left[T(n/8)\right] + 3n$$

$$\boxed{T(n) = 2^k\left[T\left(n/2^k\right)\right] + kn} \quad \text{---(4)}$$

If $n = 2^k$

$$T(n) = 2^k[T(1)] + k(2^k) = \underline{2^k(k+1)} \quad -(4)$$

Consider $n = 2^k$

$$\log_2 n = k \log_2 2$$

$$\boxed{k = \log_2 n} \rightarrow (5).$$

Substituting (5) in (4).

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

↳ For best, worst, avg. case

Quick Sort Time Complexity

Avg /Best

$$T(n) = \begin{cases} 1 & , \text{if } n=1 \\ 2T(n/2) & , \text{if } n>1 \end{cases}$$

Assume $n/2^k = 1$

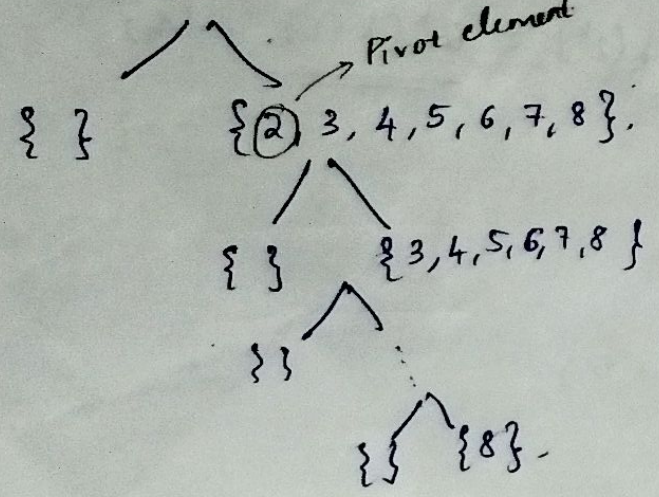
$n = 2^k$

∴ Avg, & Best Case of Quick Sort $\approx$ Avg., Best, Worst Case of Merge Sort

~~For Worst Case:-~~ For Worst Case :-

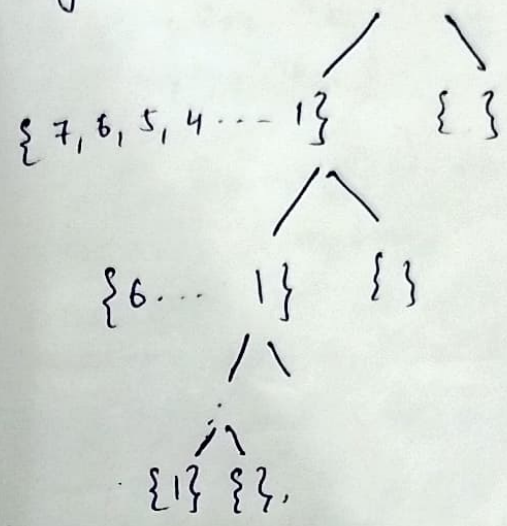
*If array is in ascending order

2 3 4 5 6 7 8



* If the array is in descending order

Eg:- 7 6 5 4 3 2 1



Time Complexity

$T(n) = [n(n+1)/2] - 1$ → in every level, one set is empty

$= \frac{n^2+n}{2}$ → Sum of n natural numbers

Considering higher order growth,

Time Complexity = n^2

Multiplication of large numbers

* Using divide and conquer

* Consider a and b.

* ↓2 digits ↓

a_0, a_1 b_0, b_1.

$c = a \times b$.

(Let) ↓3 digits

c_0, c_1, c_2

$c_0 = a_0 b_0$, ~~$c_1 = a_0 b_1 + b_0 a_1$~~.

$c = c_0 + 10c_1 + 100c_2 = 100c_2 + 10c_1 + c = c_2 10^2 + c_1 10^1 + c_0 10^0$.

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

$c_2 = a_1 b_1$

20.05.2024

$a = a_1 10^{n/2} + a_0$

$b = b_1 10^{n/2} + b_0$

$c = (a_1 10^{n/2} + a_0) + (b_1 10^{n/2} + b_0)$.

$10^n (a_1 b_1) + 10^{n/2} (a_1 b_0 + a_0 b_1) + a_0 b_0$

$c_0 = a_0 b_0$

$c_1 = (a_1 b_0 + a_0 b_1)$

$c_2 = a_1 b_1$

$M(n) = 3M(n/2)$ for $n > 1$.

$M(n/2) = 3M(n/4)$

$\vdots \qquad \vdots$

$M(1) = 1$.

Consider $n = 2^k$

$M(2^k) = 3M(2^{k-1})$

$M(2^{k-1}) = 3M(2^{k-2})$

$\log n = k \log_2^2$

$\boxed{k = \log n}$

$\Rightarrow T(n) = \log n$.

Strassen's Matrix

$$a=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad b=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, \quad c=\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$a=\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}, \quad b=\begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}, \quad c=\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}$$

$$c=\begin{bmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{bmatrix}$$

where $m_1=(a_{11}+a_{22})*(b_{11}+b_{22})$

$m_2=b_{11}(a_{21}+a_{22})$

$m_3=a_{11}(b_{12}-b_{22})$

$m_4=a_{22}(b_{21}-b_{11})$

$m_5=b_{22}(a_{11}+a_{12})$

$m_6=(b_{11}+b_{12})*(a_{21}-a_{11})$

$m_7=(b_{21}+b_{22})*(a_{12}-a_{22})$

(1). $A=\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B=\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Soln:- $m_1=(1+4)*(1+4)=25$

$m_2=1*(3+4)=7$

$m_3=1*(2-4)=-2$

$m_4=4*(3-1)=8$

$m_5=4*(1+2)=12$

$m_6=(1+2)*(3-1)=6$

$m_7=(3+4)*(2-4)=-14$

$$c=\begin{bmatrix} 25+8-12-14 & -2+12 \\ 7+8 & 25-2-7+6 \end{bmatrix} = \begin{bmatrix} 33-26 & 10 \\ 15 & 22 \end{bmatrix}$$

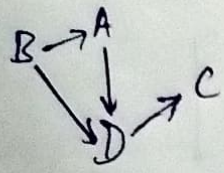
$$c=\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

21.05.2024

Topological Sort

* DAG - Directed Acyclic Graph
  └→ must ~~be~~ always be considered for topological sort

Eg:



* Order obtained after topological sort → always sequential

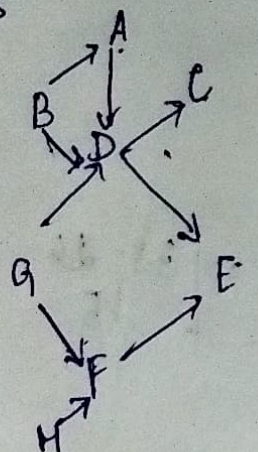
Eg: Here order = B A D C

[Order = D C is also possible but it does not give a solution to the problem.]

* Solution to the problem ⟶ must contain all elements in sequential order.
  ↓
  Must touch all vertices in ~~the~~ the graph.

* Uses 2 methods ⟶ DFS
  └→ Decrease and Conquer.

(1). DFS



→ Choose a starting point for the graph ~~whose incoming edge = 0~~. (Either B or G). ← any vertex

→ DFS(B)

→ Push dead ~~enter~~ end vertex to stack.

→ Backtrack.

→ Keep track of all the visited nodes

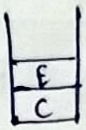
(i). B → A → D → C

→ Push C to stack

→ Backtrack

(ii). B → A → D → E

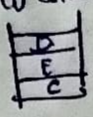
→ Push E to stack



→ Backtrack.

(iii). B → A → D.

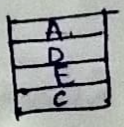
→ Push D to stack



→ Backtrack.

(iv). B → A

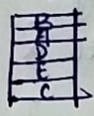
→ Push A to stack



→ Backtrack.

(v). B

→ Push B to stack



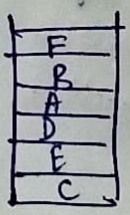
~~All the po~~ * G and F, H nodes cannot be visited from here.

* Choose a new node.
Call DFS(G).

(i). G → F

[E not included since its already visited].

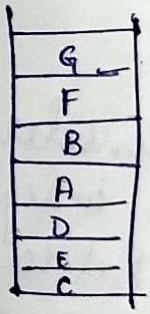
→ Push F to stack



→ Backtrack.

(ii). G

→ Push to stack.



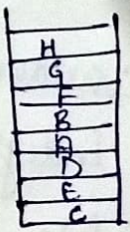
→ Backtrack.

Call DFS(H).

(iii). Choose H as new node.

H

→ Push H to stack



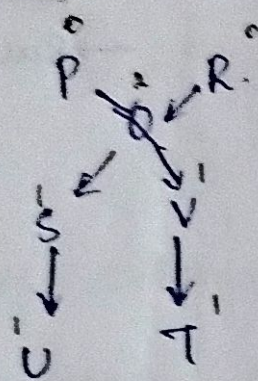
→ No more elements left to visit.

Order = H G F B A D E C

* The dependencies of the node must be before the node in the order sequence.

Time complexity, $\boxed{T(n) = O(V+E)}$ → Time taken to visit all the vertices and edge

(2). Decrease and Conquer.



o ← No. of incoming edges.

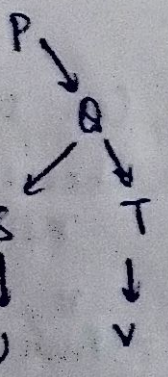
* Starting point — vertex with zero incoming edge.

Topological order = R P Q S U T V

Consider R

* If the vertex has indegree = 0.

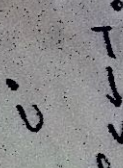
Delete the vertex & its edge



→ Consider S

Indegree = 0

Delete S



→ Consider V

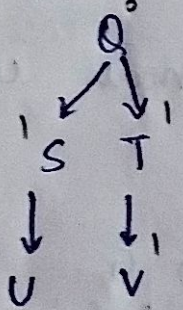
indegree = 0

Delete V

→ Consider P.

indegree = 0

Delete P.



→ Consider U

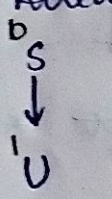
indegree = 0

Delete U

→ Consider Q.

indegree = 0

Delete Q.



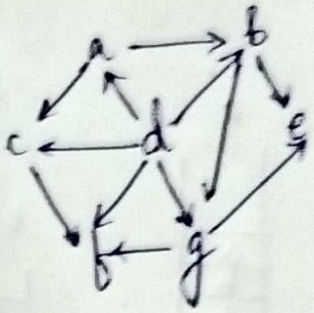
→ Consider T

indegree = 0

Delete T

V.

Apply DFS based algorithm to solve the topological sorting problem for the following di-graph.

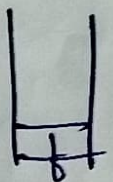


Order = f c e g b a d

Order = d a b g e c f

(i) d → c → f

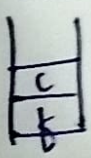
→ Push f to stack



→ Backtrack.

(ii) d → c

→ Push c to stack



→ Backtrack

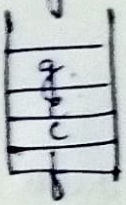
(iii) d → a → b → g → e

→ Push e to stack

→ Backtrack

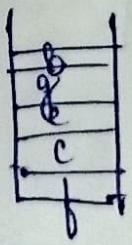
(iv) d → a → b → g

→ Push g to stack



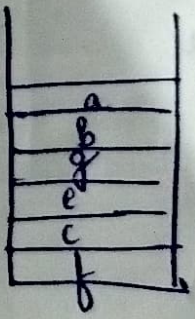
(v) d → a → b

→ Push b to stack



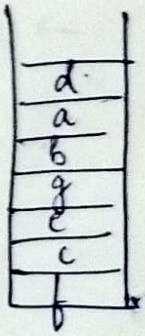
(vi) d → a

→ Push a to stack.

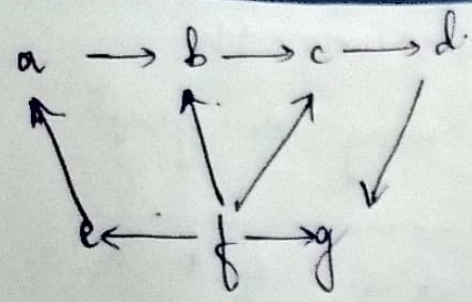


(vii) d

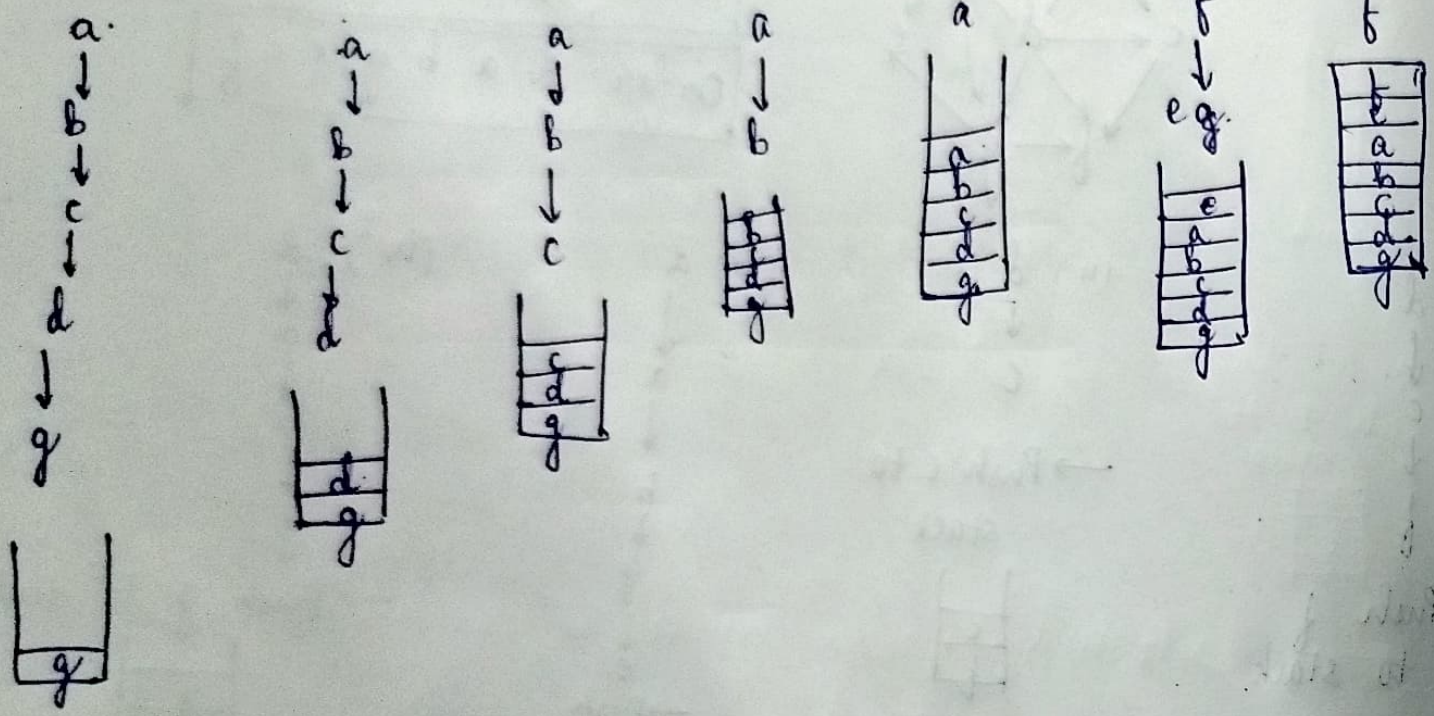
→ Push d to stack.



→ Pop

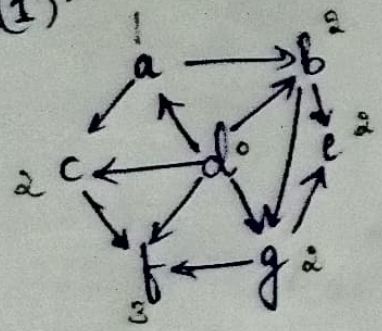


order = f e a b c d g

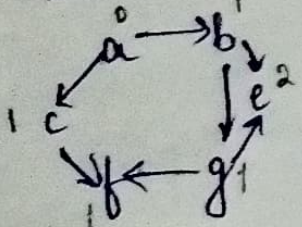


Decrease and Conquer

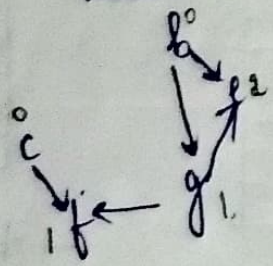
(1)



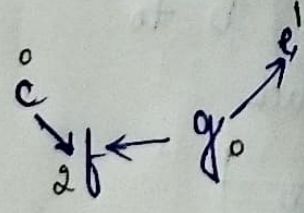
→ Consider d
indegree = 0
Delete d.



→ Consider a
indegree = 0.
Delete a.

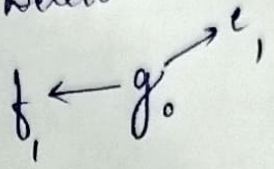


→ Consider b
indegree = 0.
Delete b

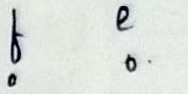


21-07-

→ Consider c
indegree = 0.
Delete c



→ Consider g
indegree = 0
Delete g.

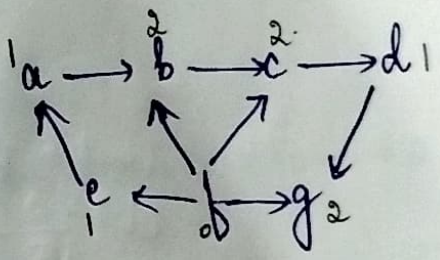


→ Consider e
indegree = 0.
Delete e.

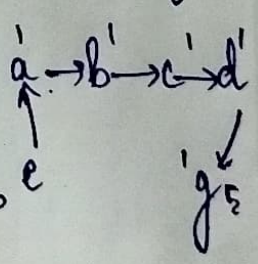
→ Consider f
indegree = 0
Delete f

Order = b d a b c g e f

(2.)

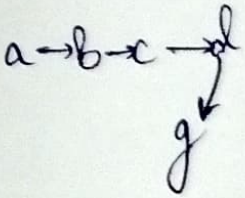


→ Consider f
indegree = 0.
Delete f

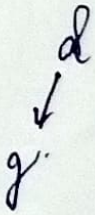


→ Consider b
indegree = 0
Delete b.
c → d
g

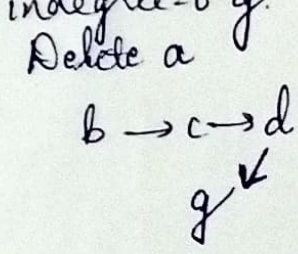
→ Consider e
indegree = 0
Delete e



→ Consider & c
indegree = 0
Delete & c



→ Consider a
b → c → d
Delete a



→ Consider d
indegree = 0
Delete d

→ consider g
indegree = 0
Delete g

Ge g°

~~Order = d a b c~~

Order = f e a b c d g

21.05.2024

Module 3

Greedy Method

→ looks for → minimum → cost / risk → maximum profit

Coin change

Standard denominations = 1, 2, 5, 10, 20, 50, 100, 200, 500.

Eg: (i) value = 49.

1	2	5	10	20	50	100	200	500

value = 20 + 20 + 5 + 2 + 2 = 49

(ii) value = 87

value = 50 + 20 + 10 + 5 + 2.

Algorithm Greedy(a, n) (a → cost, n → profit)

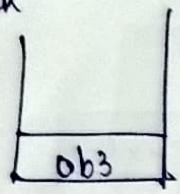
```
{
  solution = ϕ
  for( )
  {
    x: select(a)
    if(x is feasible)
    then
      solution = union(x)
  }
  return solution
}
```

Knapsack

objects	ob1	ob2	ob3
profit	25	24	15
weight	18	15	10

(i).

Considering capacity of sack = profit = 20.

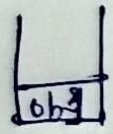


20 - 15 = 5

profit ob2 = $\frac{5}{24} \times 15 = 3.25$

Total profit = 15 + 3.25 = 18.25

(ii). Considering capacity of sack = weight = 20. (next highest).



20 - 18 = 2

ob2 = $\frac{2}{15} \times 24 = 3.2$

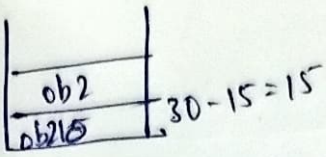
Profit = 25 + 3.2 = 28.2

(1).

objects	ob1	ob2	ob3	ob4
Profit	30	28	23	21
weight	10	15	5	2

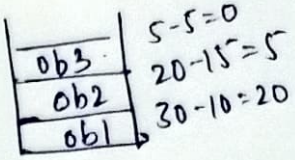
Capacity = 30

Capa



~~P=15+15~~

P = 28+28 = 56



Total = 30+28+23

Profit = 81