
AIUB
SOLUTION

Properties of Algorithm

- Finiteness : The algorithm must always terminate after a finite number of steps.
- Definiteness : Each steps must be precisely defined.
- Input : An algorithm has zero or more inputs.
- Output : An algorithm has one or more outputs which have a specified relation to the inputs.
- Effectiveness : All operations to be performed must be sufficiently.

Tools and techniques to design an Algorithm

- Recursion : Inorder, pre order, post oder tree traversal, DSF of graphs
- Devide and Conquer : Binary search, quick sort, merge sort
- Greedy method : Knapsack problem, graph coloring, job scheduling
- Dynamic programming : Random number generator, Matrix chain multiplication

What is recursion and its properties?

The process in which a function calls itself is called recursion.

Properties –

- Must have a base value where it terminates
- The recursive call should progress in such way that every time a recursive call comes closer to the base criteria.

What is divide and conquer, give an example

In algorithmic method, the design is to take an input, break the input into minor pieces, decide the problem on each small pieces and then merge the solution is called divide and conquer.

On what the performance of an algorithm depends?

Efficiency of an algorithm means how fast it can produce the correct result for the given problem. Besides it depends upon its time and space complexity. The complexity of an algorithm is a function that provides the running time and space for data.

Types of analysis –

- Worst case :
  1) Provides an upper bound on running time.
  2) An absolute guarantee that the algorithm would not run longer, no matter what the inputs are.

- Average case:
  1) Provides a prediction about the running time.
  2) Assumes the input is random.

- Best case :
  1) Provides a lower bound on running time
  2) Input is the one for which the algorithm runs faster.

Master Method

* ~~1 অথবা 1 এর চেয়ে ছোট হলে~~ $n \log n$
* ~~0 হলে~~ $\log n$
* ~~1 এর চেয়ে বড় হলে তাই হবে।~~

$T(n) = 2T(n/2) + \Theta(n)$ → $f(n)$

Case 2: $a = 2$, $b = 2$

$$n^{\log_b a} = n^{\log_2 2} = n^1$$

$\therefore O(n \log n)$

$T(n) = 9T(n/3) + n$ → $f(n)$

Case 1: $a = 9$, $b = 3$

$$n^{\log_b a} = n^{\log_3 9} = n^2$$

$\therefore O(n^2)$

$T(n) = 3T(n/4) + n \log n$ → $f(n)$

Case 3: $a = 3$, $b = 4$

$$n^{\log_b a} = n^{\log_4 3} = n^{0.79}$$

$\therefore O(n \log n)$

$T(n) = T(n/2) + 1$ → $f(n)$

Case 3: $a = 1$, $b = 2$

$$n^{\log_b a} = n^{\log_2 1} = n^0$$

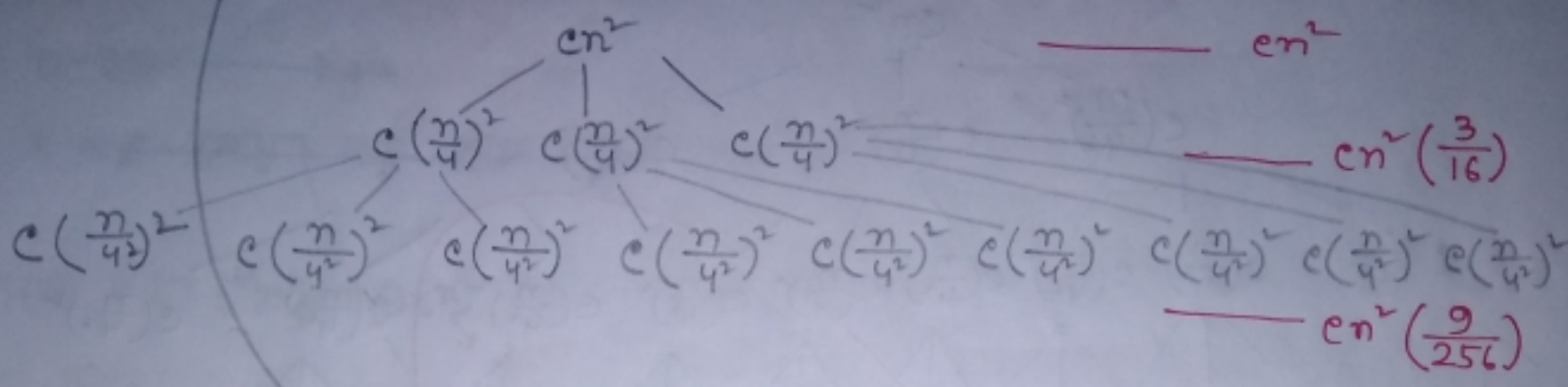
$\therefore O(\log n)$

Case-1: Root level cost is ~~same~~ greater than $f(n)$ → $\Theta(n^{\log_b a})$

Case-2: Root level cost is same as $f(n)$ → $\Theta(n^{\log_b a} \log n)$

Case-3: Root level cost is less than $f(n)$ → $\Theta(f(n))$

** $T(n) = 3T(n/4) + cn^2$



cost of level = $(\frac{3}{16})^i cn^2$

depth of the tree $\Rightarrow \frac{n}{4^1} = 1$

$\Rightarrow n = 4^i$

$\Rightarrow i = \log_4 n$

cost of root is max

Total cost = $cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \cdots + (\frac{3}{16})^{n-1} cn^2 + \Theta(3^{\log_4 n})$

cn^2 ↓ bigger

lesser

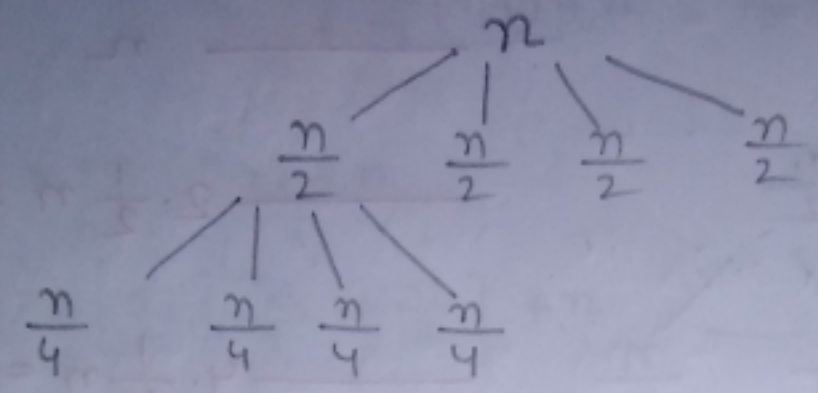
$\therefore O(n^2)$

no of leaves

$3^i = 3^{\log_4 n}$

$= n^{\log_4 3}$

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



$n \times 4\frac{1}{2} = 2n$

$n \times 16 \times \frac{1}{4} = 4n$

cost of level : $(2)^i n$

depth of the tree : $\frac{n}{2^i} = 1$

$\Rightarrow n = 2^i$

$\Rightarrow i = \log_2 n$

cost of leaves is max

No of leaves : $4^i = 4^{\log_2 n}$

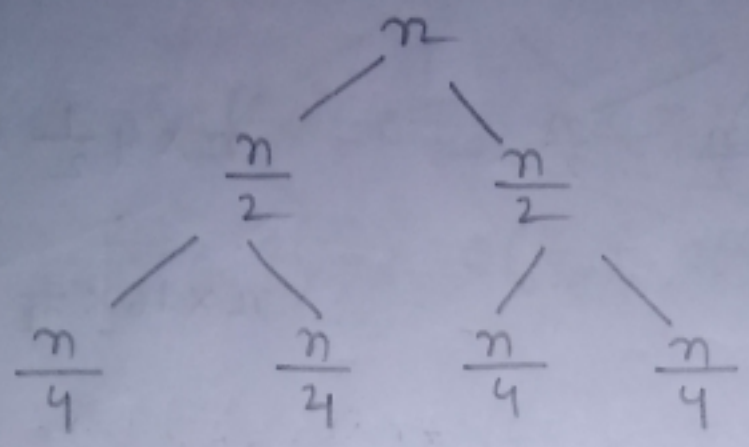
$= n^{\log_2 4}$ [formula]

$= n^2$

Total cost : $n + 2n + 4n + \cdots + O(n^2)$

$\therefore O(n^2)$

** $2T(n/2) + \theta(n)$



n

$2 \cdot \frac{1}{2} n = n$

$4 \cdot \frac{1}{4} n = n$

cost of level : $(1)^i n$

depth : $(\frac{n}{2^i}) = 1$

$\Rightarrow n = 2^i$

$\Rightarrow i = \log_2 n$

level = depth + 1

$= \log_2 n + 1$

Total cost : cost of each level * No of level

$= n * (\log_2 n + 1)$

$= n \log_2 n + n$

$\therefore O(n \log_2 n)$

** Exceptional **

cost of each node is same

Backward substitution method

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

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

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

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

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

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

$$= 2^3 T\left(\frac{n}{8}\right) + n + 2n$$

$$= 2^3 T\left(\frac{n}{8}\right) + 3n \quad = 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

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

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

$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$$

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

$$= n\underbrace{T\left(\frac{n}{n}\right)} + \underbrace{n \log n}$$

$$= n + n\log n$$

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

let $n = 2^k$

$\Rightarrow k = \log_2 n = \log n$

Time complexity of merge and quick sort

* $T(n) = \begin{cases} 2 & n = 1 \\ 2T(n/2) + 2n + 3 & \text{if } n > 1 \end{cases}$

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

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

$= 2\left[2T\left(\frac{n}{4}\right) + n + 3\right] + 2n + 3$

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + n + 3$

$= 2^2 T\left(\frac{n}{4}\right) + 2n + 6 + 2n + 3$

$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{2} + 3$

$= 2^2 T\left(\frac{n}{4}\right) + 4n + 9$

$= 2^2\left[2T\left(\frac{n}{8}\right) + \frac{n}{2} + 3\right] + 4n + 9$

$= 2^3 T\left(\frac{n}{2^3}\right) + 2n + 12 + 4n + 9$

$= 2^3 T\left(\frac{n}{2^3}\right) + 6n + 21 = 2^3 T\left(\frac{n}{2^3}\right) + 2 \times 3n + 3\left[2^2 + 2^1 + 2^0\right]$

$T(n) = 2^K T\left(\frac{n}{2^K}\right) + 2Kn + \underbrace{3\sum_{j=0}^{K-1} 2^j}_{= 3(2^K - 1)}$

$= nT\left(\frac{n^K}{n^2}\right) + 2Kn + 3(n-1)$

$n = 2^K$

$\Rightarrow K = \log_2 n$

$= \log n$

$= n + 2n\log n + 3n - 3$

$T(1) = 2$

$= 2n + 2n\log n + 3n - 3$

$= 5n + 2n\log n - 3$

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

** $T(n) = \begin{cases} 1 & n=1 \\ 8T(n/2) + n^2 & , n>1 \end{cases}$

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

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

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

$= 8^2 T(\frac{n}{4}) + 3n^2$

$= 8^2\left[8T(\frac{n}{8}) + \frac{n^2}{16}\right] + 3n^2$

$= 8^3 T(\frac{n}{8}) + 8n^2 + 3n^2$

$= 8^3 T(n/8) + 11n^2$

$= (2^3)^3 T\left(\frac{n}{2^3}\right) + 11n^2$

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

$= n^3 T(\frac{n}{n})$

$= n^3$

$\therefore O(n^3) = T(n)$

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

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

$T(\frac{n}{2^2}) = 8T(\frac{n}{8}) + \frac{n^2}{16}$

$n = 2^K$

$\Rightarrow K = \log n$

Knapsack problem

Item	Weight	Benifit	B/w	x
1	12	60	5	1
2	25	100	4	8/25
3	15	90	6	1

capacity = 35

35 - 15 = 20

20 - 12 = 8

8 - 8 = 0

$\sum x w = 1 \times 12 + \frac{8}{25} \times 25 + 1 \times 15$

$= 12 + 8 + 15$

$= 35$

$\sum x b = 1 \times 60 + \frac{8}{25} \times 100 + 1 \times 90$

$= 60 + 32 + 90$

$= 182$

quick sort

No → swap

Three steps-

I) Find pivot that divede the array into two halves

II) quick sort left half

III) quick sort right half

* right elements > pivot
* left elements < pivot

I) 12, 5, 4, 3, 14, 16, 2, 9, 8, 15
pivot ↑ ; Left ↓ ; right ↓ ←

II) 12, 5, 4, 3, 14, 16, 2, 9, 8, 15
P ; L ; R

III) 8, 5, 4, 3, 14, 16, 2, 9, 12, 15
P R

IV) 8, 5, 4, 3, 14, 16, 2, 9, 12, 15
L R

V) 8, 5, 4, 3, 12, 16, 2, 9, 14, 15
P L R

VI) 8, 5, 4, 3, 12, 16, 2, 9, 14, 15
P L R

VII) 8, 5, 4, 3, 9, 16, 2, 12, 14, 15
P L R

VIII) 8, 5, 4, 3, 9, 16, 2, 12, 14, 15
P L R

IX) 8, 5, 4, 3, 9, 12, 2, 16, 14, 15
P L R

X) 8, 5, 4, 3, 9, 12, 2, 16, 14, 15
P L R

XI) 8, 5, 4, 3, 9, 12, 16, 14, 15
P LR

* Pivot থেকে right বড়
right shift ←
or
swap pivot and right

* Pivot থেকে left ছোট
left shift →
or
swap pivot and left

xii) 8, 5, 4, 3, 9, 12, 16, 14, 15
P P
L R L R

xiii) 8, 5, 4, 3, 9, 12, 15, 14, 16
P P
L R L R

xiv) 3, 5, 4, 8, 9, 12, 15, 14, 16
P P
L R L R

xv) 3, 5, 4, 8, 9, 12, 15, 14, 16
P P
LR LR

xvi) 3, 5, 4, 8, 9, 12, 15, 14, 16
P P
L R L R

xvii) 3, 5, 4, 8, 9, 12, 14, 15, 16
P P
L R L R

xviii) 3, 5, 4, 8, 9, 12, 14, 15, 16
P P
LR LR

xix) 3, 5, 4, 8, 9, 12, 14, 15, 16
P
L R

xx) 3, 4, 5, 8, 9, 12, 14, 15, 16
P
L R

xxi) 3, 4, 5, 8, 9, 12, 14, 15, 16
P
LR

xxii) 3, 4, 5, 8, 9, 12, 14, 15, 16

Best case : $O(n \log n)$

Worst case : $O(n^2)$

Average case : $O(n \log n)$

Merge Sort

Three steps:

I) Devide

II) Sort

III) Merge

0	1	2	3	4	5	6	7	8
3,	41,	52,	26,	38,	57,	9,	49,	40

$\frac{0+8}{2} = 4$

$\frac{4}{2} = 2$

0	1	2	3	4				
3,	41,	52,	26,	38	57,	9,	49,	40

3,	41,	52,	26,	38	57	9	49	40

3	41	52	26	38	57	9	49	40

3	41	52	26	38	9	57	40	49

3	41	52	26	38	9	40	49	57

3	26	38	41	52	9	40	49	57

3	9	26	38	40	41	49	52	57

Best C :

Worst C : $O(n \log n)$

Average C :

Counting sort :

10, 7, 12, 4, 9, 13

Max = 13
Min = 4
So range is between 4-13

index	4	5	6	7	8	9	10	11	12	13

create index from 4 to 13

count	1	0	0	1	0	1	1	0	1	1

the array will hold the count of each number

sum count	1	1	1	2	2	3	4	4	5	6

compare index + sum count

✓ 4	5	6	✓ 7	8	✓ 9	✓ 10	11	✓ 12	✓ 13
1	1	1	2	2	3	4	4	5	6

sorted array	4	7	9	10	12	13

* repeated numbers *

3, 4, 10, 8, 2, 4

Max - 10
Min - 2
range 2-10

index	2	3	4	5	6	7	8	9	10

count	1	0	2	0	0	0	1	1	1

sum count	1	1	3	3	3	3	4	5	6
	0	0	2	2	2	2	3	4	5

** now for index 2 reduce sum count by 1 **

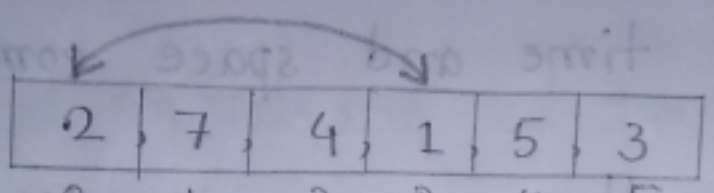
sorted array	2	4	4	8	9	10
	2	4	4	8	9	10

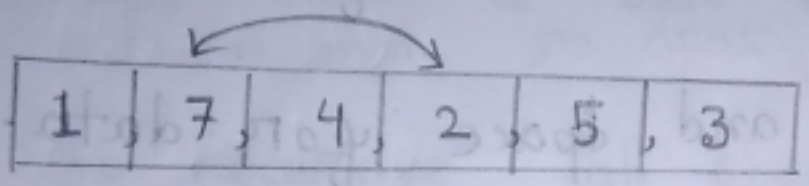
Selection sort

I) Selection

II) Swapping

III) Counter shift





1 , 2, 4 7 , 5 , 3

1 2 3 7 5 4

1 2 3 4 5 7

Best case

Worse case } $O(n^2)$

Average case

Insertion sort

7, 8, 5, 2, 4, 6, 3

left বড় হলে swap

7, 5, 8, 2, 4, 6, 3

5, 7, 8, 2, 4, 6, 3

5, 7, 2, 8, 4, 6, 3

5, 2, 7, 8, 4, 6, 3

2, 5, 7, 8, 4, 6, 3

2, 5, 7, 4, 8, 6, 3

2, 5, 4, 7, 8, 6, 3

2, 4, 5, 7, 8, 6, 3

2, 4, 5, 7, 6, 8, 3

2, 4, 5, 6, 7, 8, 3

2, 4, 5, 6, 7, 3, 8

2, 4, 5, 6, 3, 7, 8

2, 4, 5, 3, 6, 7, 8

2, 4, 3, 5, 6, 7, 8

2, 3, 4, 5, 6, 7, 8

sorted

Best case : $O(n)$ sorted

Worst case : $O(n^2)$ unsorted

Pseudocode

Bubble sort

```
for i = 1 to n-1
  for j=1 to n-1
if (A(j) > A(j+1))
    temp = A(j)
    A(j) = A(j+1)
    A(j+1) = temp
  End if
 End for
End for
```

Selection sort

```
for i=1 to n-1
do:
     smallsub = 1
  for j=i+1 to n-1
  if A(j) < A(smallsub)
    smallsub=j
    End if
  End for
Temp = A(i)
 A(i) = A(smallsub)
A(smallsub) = temp
End for
```

Counting sort

```
counting sort (A,B,k)
for i=1 to k
do:
    c[i]=0
for j=1 to n
do:
  c[A(j)] = c[A[j]+1]
for i=2 to k
do:
 c[i] = c[i] + c[i-1]
for j=n  down to 1
do:
    B[c[A[j]]] = A[j]
C[A[j]] = c[A[j]]-1
```

insertion sort

```
for i=2 to n
    j=i
Dowhile (j>1) and (A(j) < A(j-1))
 Temp = A(j)
A(j) = A(j-1)
A(j-1) = temp
J = J-1
 End do
End for
```

Binary search

```
low = 0
high = n-1
while (low <= high)
{ mid = (low + high)/2
 if (A[mid] > value)
   high = mid - 1
 else if (A [mid] < value)
      low = mid + 1
 else
      return mid
```

Merge sort

```
Merge_sort (array A, int p, int r)
{ if (p<r)
  { q = (p+r)/2
   mergesort (A, p, q)
   mergesort (A, q+1, r)
   merge (A, p, q, r) }
Merge (array A, int p, int q, int r)
{ array B
   i = K = P
   j = q+1
while (i<=q and j<=r)
  { if (A[i] <= A[j])
      B[k++] = A[i++]
  else B[k++] = A[j++] }
while (i<=q)
   B[k++] = A[i++]
while (j<=r)
  B[k++] = A[j++]
for (i=p to r)
 do A[i] = B[i]
```

quick sort

```
quick sort (A, p, r)
  if (p<r)
q = partition (A, p, r)
quicksort (A, p, q-1)
quicksort (A, q+1, r)

partition (A, p, r)
   x = A[r]
  i = p-1
for j = p to r-1
   if A[j] ≤ x
        i = i+1
   A[i] = A[j]
  A[i+1] = A[r]
  return i+1
```