Algorithm : Algorithm is a Step by Step Solution of a particular problem.

OR.

An algorithm is a well defined computational procedure that take some values as an input and produce some value as a output.

Every algorithm must satisfy the following condition -

① Input : Every algorithm contain finite no. of input.

② Output : Every algorithm contains atleast one output.

③ Finiteness : Algorithm terminates after finite no. of steps.

Main issues related to algorithm :-

- How to design an algorithm.
- How to validate the algorithm.
- How to analyse the algorithm.

How to design an algorithm

Appropriate planning is required which designing an algorithm because proper planning makes an algorithm. Sufficient and accurate.

There are different approach or techniques to design an algorithm.-

① Incremental:

In this approach, array index increase in incremental approach.

eg) Insertion sort and linear search.

② Divide and Conquer Approach:

A divide and Conquer ~~problem~~ algorithm first divide the problem into sub-problem and find out the solution for each sub-problem and then merge the solution of subproblem.

eg) Binary search, quick sort, merge sort, heap sort.

③ Dynamic approach:

Dynamic approach are also based on divide and Conquer approach. In dynamic approach, problems can be dependent to each other and dynamic programming find out the optimal sol^n. Optimal sol^n means best (max. or min.).

eg) LCS (Longest common subsequence), shortest path, matrix chain Multiplication

④ Greedy approach:

It is similar to dynamic approach but the difference is that greedy approach takes best decision at a time. and dynamic programming take all possible solution.

eg) MST (min. spanning tree)
Huffman problem

⑤ **Back-Tracking:**

Back Tracking is a mathematical approach to try out various sequence of decision until we find out one solution.

eg) Graph Colouring problem, n-queen problem, hamiltonian circuit.

⑥ **Branch and Bound:**

Branch and bound represent the tree. Edge represent the branch & nodes indicates the bounded value. and bounded value can be two types-

upper bound - indicates max.

lower bound - indicates min.

eg) TSP (travelling Sales man problem), Knapsack problem

⑦ **Approximation:**

Approximation algorithm are used which problem cannot be solve in polynomial time.

eg) Halting problem, TSP, vertex cover.

⑧ **Randomized:**

Randomized approach works for large amount of data. Randomized algorithm select random number at any time during the process.

eg) hiring problem and randomized quick sort.

How to Validate the algorithm

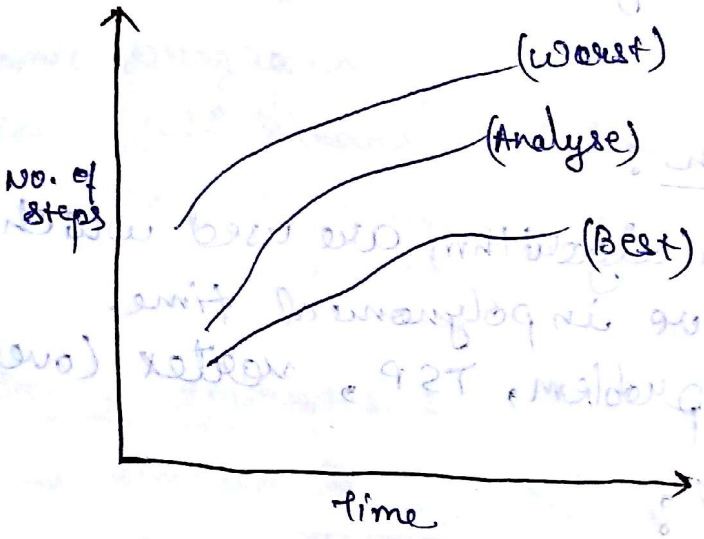
It is necessary to show that it computes the correct answer for all possible legal input in a finite no. of steps.

How to analyse the algorithm

Analyses of algorithm means to determine how much computing time and storage required. The performance of the algorithm depends on two main factor.

i) time
ii) Space

there are different case to analyse the algorithm



i) **Worst Case**: It means the max. no. of steps taken on size 'n'

eg) Searching the last element

ii) **Best Case**: It means the min. no. of steps taken on size 'n'

eg) Searching the first element.

iii) Average Case: It means the average no. of steps taken on size 'n'.

eg) Searching the middle element.

Characteristics of the Algorithm

① Correctness: Algorithm should be correct.

A common technique ~~shou~~ is used to prove the correctness by Induction method.

② Efficiency: Efficiency is measured by time and space.

③ Simplicity: Algorithm should be simple to understand.

Insertion Sort :

① for J ← 2 to length (A) (steps 1 to 7)
② Key ← A[j]
③ i ← j − 1
④ while i > 0 and A[i] > Key (steps 4 to 6)
⑤ do A[i+1] ← A[i]
⑥ i ← i − 1
⑦ A[i+1] ← Key

Q:

6	3	5	7	2	4

using Insertion sort.

Sol:

Step ①

for $j = 2$ to 6.

Key ← 3

i ← 1

A[2] ← A[1]

6	6	5	7	2	4

i ← 0

A[1] ← Key(3)

3	6	5	7	2	4

Step ②

for $j = 3$

Key ← 5

i ← 2.

A[3] ← A[2]

3	6	6	7	2	4

i ← 1

~~condition false~~ condition false.

A[2] ← Key(5)

3	5	6	7	2	4

Step 3

for j = 4.

Key ← 7

i ← 3

Condition false.

3	5	6	7	2	4

Step 4

for j = 5

Key ← 2

i ← 4

Condition true

A[5] ← A[4]

3	5	6	7	7	4

i ← 3.

Condition true

A[4] ← A[3]

3	5	6	6	7	4

i ← 2.

Condition true

A[3] ← A[2]

3	5	5	6	7	4

i ← 1

Condition true

A[2] ← A[1]

3	3	5	6	7	4

i ← 0

Condition false

A[1] ← Key (2)

2	3	5	6	7	4

Step 5

for j = 6

Key ← 4

i ← 5

condition true

A[6] ← A[5]

2	3	5	6	7	7

i ← 4

condition true

A[5] ← A[4]

2	3	5	6	6	7

i ← 3

condition true

A[4] ← A[3]

2	3	5	5	6	7

i ← 2.

Condition false.

A[3] ← key(4)

2	3	4	5	6	7

Analysis of Insertion Sort

~~1) Best case:~~

~~*Algo: ① for J ← 2 to length(A)~~

~~② Key ← A[j]~~

	cost	time
① for J ← 2 to length(A)	C_1	n
② Key ← A[j]	C_2	$n-1$
③ i = j - 1	C_3	$n-1$
④ while i > 0 & A[i] > Key	C_4	$\sum_{j=2}^{n} t_j$
⑤ A[i+1] ← A[i]	C_5	$\sum_{j=2}^{n} t_j - 1$
⑥ i ← i - 1	C_6	"
⑦ A[i+1] ← Key	C_7	$n-1$

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

For best Case :

When data is already sorted.

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

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

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

$$T(n) = an - b$$

$$\Rightarrow \boxed{T(n) = \Omega(n)}$$

For Worst Case :

When data is in descending order.

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

$$T(n) = c_1(n) + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2} - 1\right) + c_7(n-1)$$

$$T(n) = nc_1 + (n-1)c_2 + (n-1)c_3 + \left(\frac{n^2+n-2}{2}\right)c_4 + \left(\frac{n^2-n-2}{2}\right)c_5 + \left(\frac{n^2-n-2}{2}\right)c_6 + c_7(n-1)$$

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

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

$$T(n) = an^2 + bn + c$$

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

for Average Case

Same as worst Case.

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

Merge Sort :

It depends on divide and Conquer approach

Algorithm:

MergeSort (A, P, R)

① if P < R then
② $q \leftarrow \frac{\lfloor P+R \rfloor}{2}$
③ MergeSort (A, P, q)
④ MergeSort (A, q+1, R)
⑤ Merge (A, P, q, R)

P is the index of 1st element
R is the index of last element

to divide

(for the merging)

① $n_1 \leftarrow q - p + 1$

② $n_2 \leftarrow r - q$

③ create array
$L[1, \ldots, n_1+1]$
and $R[1, \ldots, n_2, n_2+1]$

④ for $i \leftarrow 1$ to n_1

~~⑤ do $i \leftarrow 1$ to n_1~~ ⑤ do $L[i] \leftarrow A(p+i-1)$

⑥ for $j \leftarrow 1$ to n_2

⑦ do $R[j] \leftarrow A[q+j]$

⑧ $L[n_1+1] \leftarrow \infty$

⑨ $R[n_2+1] \leftarrow \infty$

⑩ $i \leftarrow 1$

⑪ $j \leftarrow 1$

⑫ for $k \leftarrow p$ to r

⑬ do if $L[i] \leq R[j]$

⑭ then $A[k] \leftarrow L[i]$

⑮ $i \leftarrow i+1$

⑯ else $A[k] \leftarrow R[j]$

⑰ $j \leftarrow j+1$

eg)

5	2	4	7	1	3	2	6

$\Rightarrow p = 1$, $r = 8$

$q = \lfloor \frac{1+8}{2} \rfloor = \lfloor 4.5 \rfloor = 4$

2	4	5	7

1	2	3	6

(15) $i \leftarrow i+1$

$i \leftarrow 3$.

(12) $K=6$

(13) $L[3] \leq R[4]$

$5 \leq 6$ (true)

(14) $A[6] \leftarrow L[3]$ (5)

1	2	2	3	4	5		

(15) $i \leftarrow i+1$

$i \leftarrow 4$.

(12) $K=7$

(13) $L[4] \leq R[4]$

$7 \leq 6$ (false)

(16) $A[7] \leftarrow R[4]$ (6)

1	2	2	3	4	5	6	

(17) $j \leftarrow j+1$

$j \leftarrow 5$

(12) $K=8$

(13) $L[4] \leq R[5]$

$7 \leq \infty$ (true)

(14) $A[8] \leftarrow L[4]$ (7)

1	2	2	3	4	5	6	7

(15) $i \leftarrow i+1$

$i \leftarrow 5$

(12) $K=9$ (false)

1	2	2	3	4	5	6	7

Analysis of Merge Sort :

Best Case: when $n=1$ (i.e, size of array is 1)

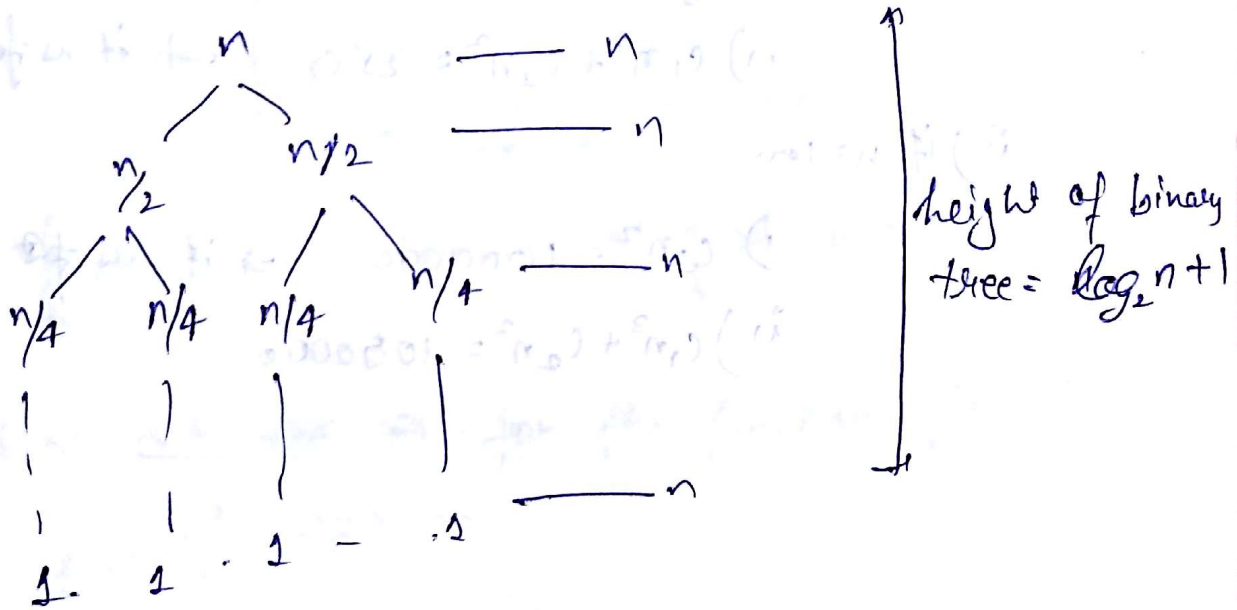
$$\Omega(1)$$

Worst Case:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

in Merge Sort, $a=2$, $b=2$.

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



$$T(n) = n(\log_2 n + 1) \Rightarrow T(n) = n\log_2 n + n$$

$$\Rightarrow \boxed{O(n\log_2 n)}$$

Average Case :

Same as worst case

$$\Theta(n \log_2 n)$$

Growth of Function / Order of Growth / Asymptotic Notation

eg) i) $C_3 n^2$

ii) $C_1 n^3 + C_2 n^2$

if $C_3 = 100$, $C_1 = 1$, $C_2 = 5$.

So i) if $n = 5$

i) $C_3 n^2 = 2500$

ii) $C_1 n^3 + C_2 n^2 = 250$ ⇒ it is fas

ii) if $n = 100$

i) $C_3 n^2 = 1000000$ ⇒ it is slo

ii) $C_1 n^3 + C_2 n^2 = 105000$

Notations are used to describe the algorithm efficiency and performance in a meaningful w these notation are called asymptotic notation

Asymptotic notation how the running time of an algorithm increase with the size of input.

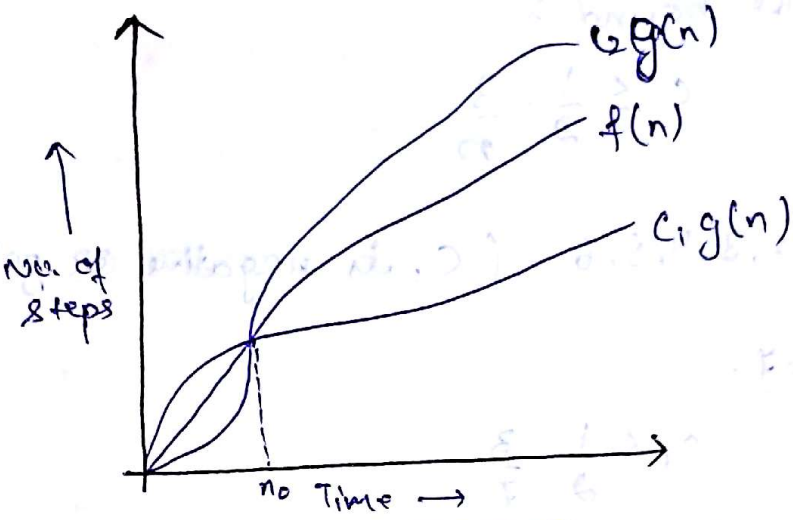
There are three types of notations which are used to describe the efficiency of the algorithm

1) Big Oh (O) ⇒ Worst case (upper bound)

2) Theta (Θ) ⇒ Average case (Sandwich)

3) Big Omega (Ω) ⇒ Best Case (Lower bound)

1) Theta (Θ): Average Case: (sandwich):



$$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \quad \forall \; n \ge n_0$$

Q Find out the 'Θ' for the function $f(n)$

$$f(n) = \frac{n^2}{2} - 3n$$

Sol

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$$

$$c_1 n^2 \le \frac{n^2}{2} - 3n \le c_2 n^2$$ [$\because g(n) = n^2$ (neglect for $f(n)$)]

for upper bound.

divide by n^2.

$$\boxed{C_1 \leq \frac{1}{2} - \frac{3}{n} \leq C_2}$$

for Upper bound:

$$\frac{1}{2} - \frac{3}{n} \leq C_2$$

$$\Rightarrow C_2 \geq \frac{1}{2} - \frac{3}{n} \quad \text{---(2)}$$

$$\Rightarrow \boxed{C_2 = 1/2}, \; n_0 = 7 \text{ (as it is maximum)}$$

for lower bound:

$$C_1 \leq \frac{1}{2} - \frac{3}{n}$$

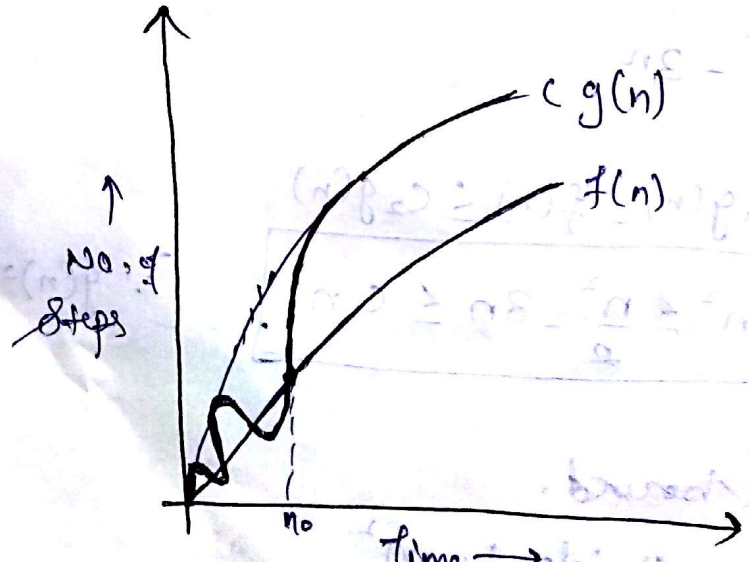
for $n = 1, 2, 3, 4, 5, 6$ (C_1 is negative or zero (not possible))

so, $\underline{n_0 = 7}$.

$$C_1 \leq \frac{1}{2} - \frac{3}{7}$$

$$\boxed{C_1 = \frac{1}{14}}, \; n_0 = 7.$$

2) Big Oh (O): worst case: Upper bound



$0 \le f(n) \le c(g(n)) \quad \forall\, n \ge n_0$

Q Find out 'O' for function $f(n) = 3n+2$

Sol $0 \le f(n) \le c(g(n))$

~~$0 \le 3n+2 \le c8n$~~

~~divide by n~~

~~$0 \le 3 + \frac{2}{n} \le 8c$~~

$f(n) = 3n+2$

$0 \le f(n) \le c\, g(n)$

$0 \le 3n+2 \le 4n$ [as 4n is near greater than 3n]

$\Rightarrow \boxed{c=4}$

for $n_0 = 1$

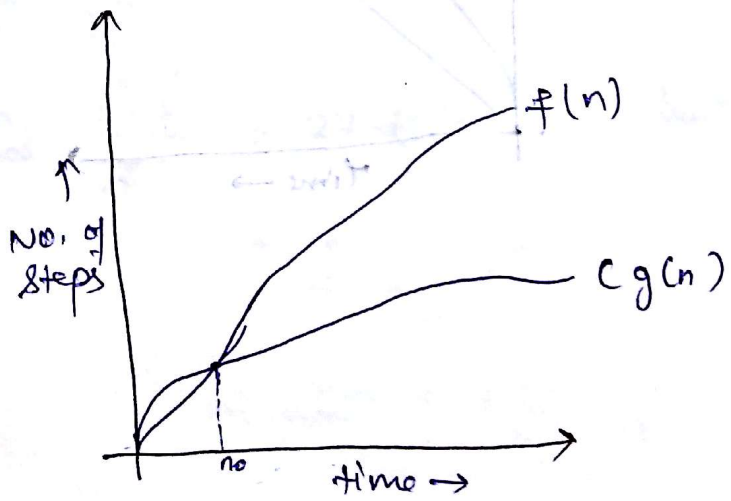
$0 \le 4 \not\le 5$

for $n_0 = 2$

$8 \le 8$ (true)

$\Rightarrow \boxed{c=4}, \; n_0 = 2$

3) Big Omega (Ω): Best Case : Lower bound



$$0 \le cg(n) \le f(n) \quad \forall \; n \ge n_0$$

Q find out 'Ω' for function.

$$f(n) = 3n + 2$$

Sol $0 \le cg(n) \le f(n)$

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

$0 \le 3n \le 3n+2$ [as $3n$ is just less than $3n+2$]

$\Rightarrow \boxed{c=3}$

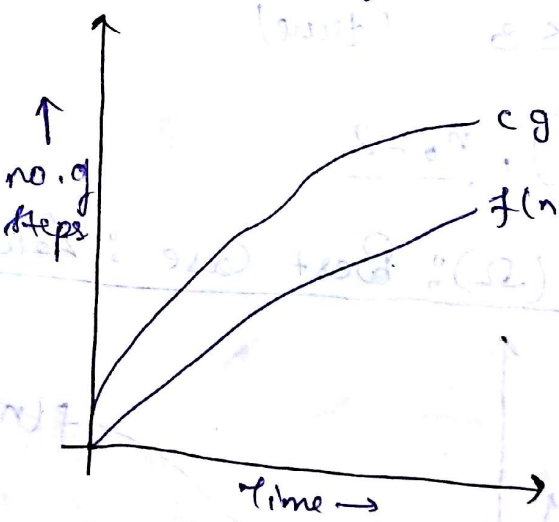
for $n_0 = 1$

$3 \le 5$ (true)

$\therefore \boxed{c=3}$, $n_0 = 1$

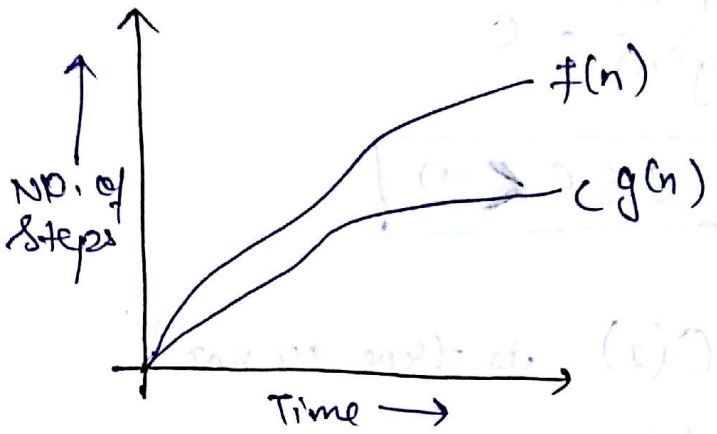
Tightly upper bound : (small oh) (o)

$$0 \le f(n) < cg(n)$$



Tightly Lower bound : (small omega) (ω)

$0 \leq c\,g(n) < f(n)$



* Theta (θ) [theorem]

Let $f(n)$ & $g(n)$ are the functions such that

~~(tail n → ∞)~~

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$

$$\boxed{0 < c < \infty}$$

Q Is $3^{n+3} = \Theta(3^n)$ is true or not?

Sol

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c \quad \text{where } 0 < c < \infty$$

$$\lim_{n \to \infty} \frac{3^{n+3}}{3^n}$$

$$\lim_{n \to \infty} \frac{3^n \cdot 3^3}{3^n} = 27 \text{ (it is constant)}$$

true.

* Big Oh (O) [theorem]

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

$$\boxed{0 \le c < \infty}$$

Q. Is $2^{1000} = O(1)$ is true or not?

Sol

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad \text{where } 0 \le c < \infty$$

$$\rightarrow \lim_{n\to\infty} \frac{2^{1000}}{1} = 2^{1000}$$ (it is constant)

true.

* Big Omega (Ω) [theorem]

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

$$\boxed{0 < c \le \infty}$$

OR.

$$\boxed{0 < c \le \infty - 1}$$

Q Is $(12 \times 3^n) + n^2 = \Omega(3^n)$ is true or not?

Sol

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad \text{where } 0 < c \le \infty$$

$$\lim_{n\to\infty} \frac{(12 \times 3^n) + n^2}{3^n}$$

$$\lim_{n\to\infty} \frac{12 \times 3^n}{3^n} + \lim_{n\to\infty} \frac{n^2}{3^n}$$

$= 12 + \lim_{n\to\infty} \frac{n^3}{3^n} \to 0$

$= 12$ (It is constant)

true :

* Small Oh (o) [theorem]

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Q Is $3^n = o(3^{3n})$ is true or not?

Sol $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$\lim_{n\to\infty} \frac{3^n}{3^{3n}} = 0$

$\lim_{n\to\infty} \frac{3^n}{3^n \cdot 3^{2n}} = 0$

$= \lim_{n\to\infty} \frac{1}{3^{2n}} = 0$

$= \frac{1}{\infty} = 0$ (true)

* Small Omega (ω) [theorem]

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Q Is $2^{2n} = \omega(2^n)$ is true or not?

Sol $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \infty$

$$\lim_{n\to\infty} \frac{2^n \cdot 2^n}{2^n} = \infty$$

$\infty = \infty$ (true).

Q. Is $2^{2n} = O(2n)$ is true or not?

Sol $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C \quad 0 \le C < \infty$

$\lim_{n\to\infty} \frac{2^{2n}}{2n} \neq C$ (false)

Q The running time of algorithm is atleast $O(n^2)$. Statement is correct or not?

Sol It is incorrect as 'O' indicates the upper bound.

Compairing Order of Growth

- for; $\lim_{n\to\infty} \frac{T_a(n)}{T_b(n)} = 0$

$T_b(n)$ is large value as compare to $T_a(n)$ it means $T_a(n)$ is fast.

OR.

we can say that order of growth of $T_a(n)$ slower than $T_b(n)$

for; $\lim_{n \to \infty} \frac{T_a(n)}{T_b(n)} = c$

$T_a(n)$ & $T_b(n)$ has same order of growth

for; $\lim_{n \to \infty} \frac{T_a(n)}{T_b(n)} = \infty$

$T_a(n)$ is large value as compare to $T_b(n)$ it means $T_b(n)$ is fast.

Q. Two algo. with running time $\log_2 n$ & $\sqrt{n}$. Compare the order of growth.

Q. Two algos. with running time $\frac{n}{4}(n-1)$ and n^2. Compare the order of growth

Q. $T_a(n) = n!$, $T_b(n) = 3^n$. Compare the order of growth

Q. Prove that $(n+a)^b = \Theta(n^b)$ where $b > 0$.

Sol (i) $T_a(n) = \log_2 n$. , $T_b(n) = \sqrt{n}$

$$\lim_{n \to \infty} \frac{\log_2 n}{\sqrt{n}}$$

$$\lim_{n \to \infty} \frac{\log_2 e \cdot \frac{1}{n}}{\frac{1}{2\sqrt{n}}}$$

$$= 2 \log_2 e \cdot \lim_{n \to \infty} \frac{\sqrt{n}}{n}$$

$= 0$

So, order of growth of $T_b(n)$ is faster than $T_a(n)$

$\log_2 n = \log_2 e \log_e n$

$= \frac{\log_e n}{\log_e 2}$

(2) $T_a(n) = \frac{n}{4}(n-1)$

$T_b(n) = n^2$

$\lim_{n\to\infty} \frac{\frac{n}{4}(n-1)}{n^2} = \frac{1}{4} \lim_{n\to\infty} \frac{n^2 - n}{n^2}$

$= \frac{1}{4} \lim_{n\to\infty} \left(1 - \frac{1}{n}\right)$

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

So, both algo. have same order of growth

(3) $\lim_{n\to\infty} \frac{n!}{3^n} = \lim_{n\to\infty} \frac{1 \cdot 2 \cdot 3 \cdots n}{3 \cdot 3 \cdots 3} \lim_{n\to\infty} \frac{n^n}{3^n}$

$\lim_{n\to\infty} \left(\frac{1}{3} \cdot \frac{2}{3} \cdot \frac{3}{3} \cdots \frac{n}{3}\right) = \lim_{n\to\infty} \left(\frac{n}{3}\right)^n$

$= \infty$

So, order of growth of $T_a(n)$ is faster than $T_b(n)$

Sterling's Approximation

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \simeq n^n$$

(4) $(n+a)^b = \Theta(n^b)$ $[b>0]$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{(n+a)^b}{n^b} = \lim_{n\to\infty} n^b \left(1+\frac{a}{n}\right)^b$$

$$= \lim_{n\to\infty} \left(1+\frac{a}{n}\right)^b = \lim_{n\to\infty} \left(1+\frac{a}{n}\right)^b$$

$$= \lim_{n\to\infty} (1)^b = 1 \qquad [\text{it is const.}]$$

So, $(n+a)^n = \Theta(n^b)$

Q) (i) $n! = O(n^n)$, prove, (ii) $n! = o(n^n)$

Sol (i) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ where $0 \le c < \infty$ for big Oh 'O'

$$\lim_{n\to\infty} \frac{n!}{n^n} = \lim_{n\to\infty} \frac{n^n}{n^n} \qquad [\text{by sterling's}]$$

$= 1$ = constant.

∴ True

(ii) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ for small oh 'o'

$\lim_{n\to\infty} \frac{n!}{n^n} = \lim_{n\to\infty} \frac{n^n}{n^n}$ [by sterling's]

$1 \neq 0$

∴ false

$$\Rightarrow \lim_{n\to\infty} \frac{n!}{n^n} = \lim_{n\to\infty} \frac{n(n-1)(n-2)\ldots\ldots 3\cdot 2\cdot 1}{n^n}$$

$$\lim_{n\to\infty} \frac{n^n}{n^n}\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\cdots\cdots\frac{3}{n}\cdot\frac{2}{n}\cdot\frac{1}{n}\right)$$

$$= 0$$

$$\Rightarrow \quad n! = o(n^n)$$

Q. Prove that $\log_2 n! = O(n \log n)$

Sol $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ where $0 \le c < \infty$

$$\lim_{n\to\infty} \frac{\log n!}{n\log n} = \frac{\lim \log(n^n)}{n\log n} \quad \left[\text{by Sterlings approx. } n! \approx n^n\right]$$

$$= \lim_{n\to\infty} \frac{n\log n}{n\log n} = \underline{1} = \text{Constant}$$

$$\Rightarrow \log n! = O(n\log n)$$

Q List the following f^n in increasing order of growth.

$\log n$, $n!$, $\frac{n^2}{\log n}$, $n\cdot 2^n$, $(\log)^{\log n}$, 3^n

Recurrence Relation :

There are five method to find complexity.

1) Substitution Method
2) Iteration Method
3) Changing Variable Method
4) Recursion Tree Method
5) Master Method

NOTE: Recursion tree & Master Method. both are used with recurrence relation

when an algorithm contains a recursive call to itself until a given condition then its running time can be discussed by recurrence relation.

(used for divide & conquer Approach)

$$T(n) = aT\left(\frac{n}{b}\right) + D(n) + C(n)$$

where a = no. of sub problems
b = size of sub problem
$D(n)$ = Dividing time
$C(n)$ = Combining time

For GP

$$1 + x + x^2 + x^3 + x^4 \ldots\ldots n$$

$$= \frac{a(x^n - 1)}{x - 1} \quad \text{if } x > 1$$

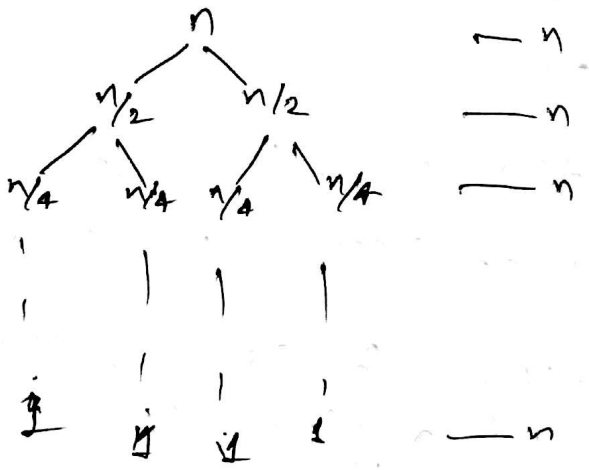
$$= \frac{1}{1 - x} \quad \text{if } x < 1$$

$$b^{\log_b a} = a \quad \text{Or} \quad a = b^{\log_b a}$$

Recursion Tree:

(1) Merge Sort:

(1) $T(n) = 2T\left(\frac{n}{2}\right) + n \quad \therefore a = 2,\ b = 2$

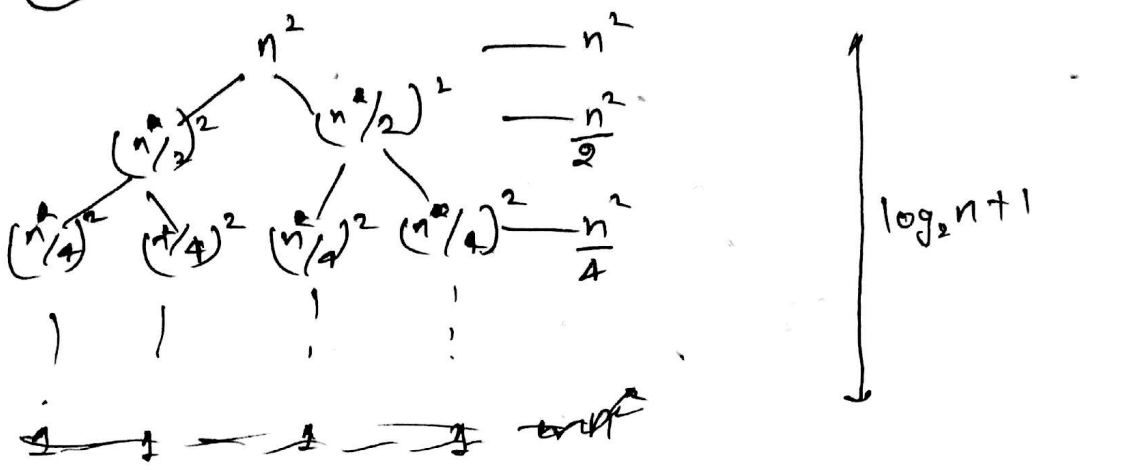


$$T(n) = (\log_2 n + 1)\, n$$

$$= n \log_2 n + n$$

$$= O(n \log_2 n)$$

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



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

$= n^2 \log_2 n + n^2$

$\Rightarrow O(n^2 \log_2 n)$

$T(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \ldots \log_2 n + 1$

$T(n) = n^2 \left(1 + \frac{1}{2} + \frac{1}{4} + \ldots \frac{\log_2 n + 1}{}\right)$

$T(n) = n^2 \left[\frac{1}{1 - 1/2}\right]$ [Sum of GP]

$T(n) = 2n^2$

$\Rightarrow \boxed{T(n) = O(n^2)}$

OR

$T(n) = n^2 \left[1 + \frac{1}{2} + \frac{1}{4} + \ldots \log_2 n + 1\right]$

$T(n) \leq n^2 \sum_{i=0}^{\infty} \frac{1}{2^i}$ [tends to infinity]

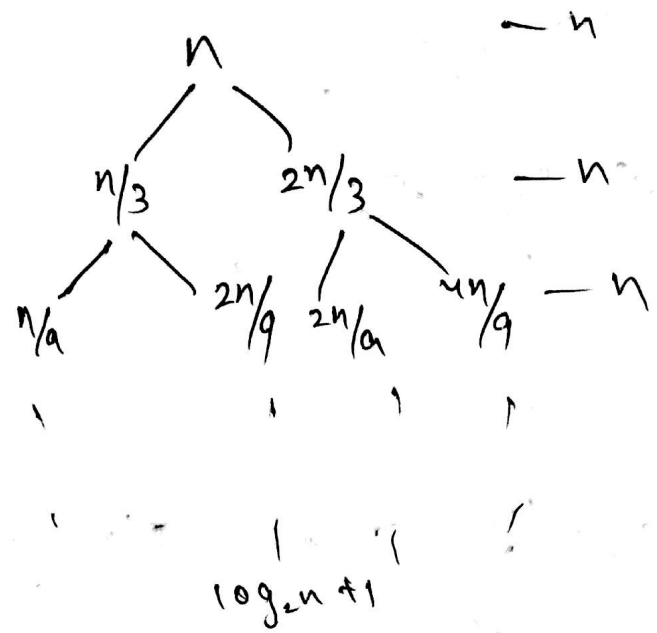
$T(n) \leq n^2 \left[\frac{1}{1 - 1/2}\right]$

$T(n) \leq 2n^2$

$\boxed{T(n) = O(n^2)}$

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

(a)



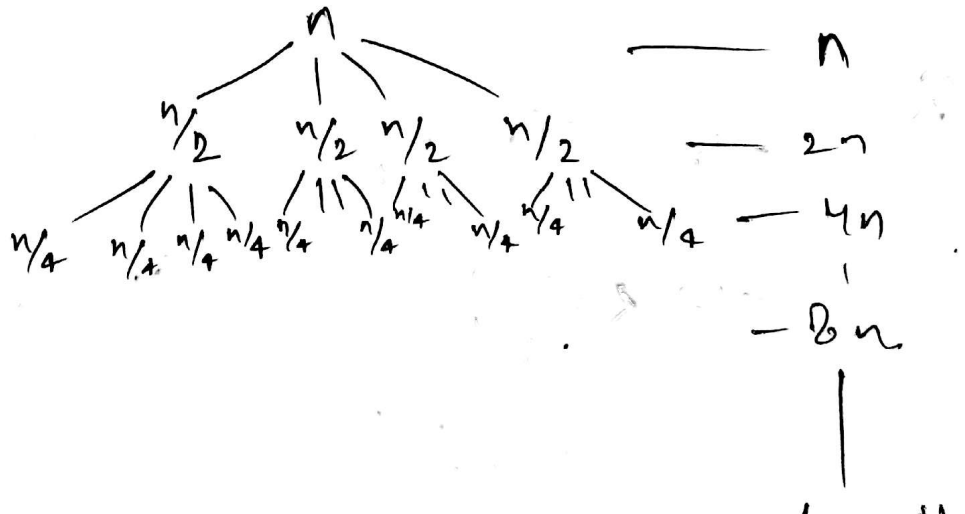
$T(n) = (\log_2 n + 1)\, n$

$= n \log_2 n + n$

$\therefore T(n) = O(n \log_2 n)$ A

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

(a)



$T(n) = n + 2n + 4n + 8n + \ldots (\log_2 n + 1)$

$= n \left[1 + 2 + 4 + 8 + \ldots \log_2 n \right]$

~~$T(n) \leq n[1+2+4+8+\ldots\infty]$~~

~~$T(n) \leq n \sum_{i=0}^{\infty} 2^i$~~

~~$T(n) \leq n \left[\frac{1}{1-1/2}\right]$~~

~~$T(n) \leq 2n$~~

~~$T(n) \leq n$~~ [

$T(n) = n[1+2+4+8+\ldots+(\log_2 n+1)]$

$T(n) = \underline{n\left[2^{\log_2 n}-1\right]}$
$2-1$

$T(n) = n2^{\log_2 n} - n$

$T(n) = n(n) - n \quad [\because b^{\log_b a} = a]$

$T(n) = n^2$

$\Rightarrow \boxed{T(n) = O(n^2)}$

Master Method :

Master Method is used for recurrence relation.

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1, b > 1$

Step. (i) find $n^{\log_b a}$

case if. $n^{\log_b a} = f(n) \Rightarrow \Theta(f(n)\cdot \log n)$

- $n^{\log_b a} > f(n) = \Theta(n^{\log_b a})$
- $n^{\log_b a} < f(n) = \Theta(f(n))$

(i) $T(n) = 9T\left(\frac{n}{3}\right) + n$

Sol $\Rightarrow a = 9, \; b = 3.$

then $n^{\log_b a} = n^{\frac{\log 9}{\log 3}} = n^2$

$n^2 > n$

$\therefore \boxed{T(n) = \Theta(n^2)}$

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

Sol $\Rightarrow a = 3, \; b = 4, \; f(n) = n \log n$

$n^{\log_b a} = n^{\log_4 3} = n^{\frac{\log 3}{\log 4}} \approx n^{0.8}$

$\Rightarrow f(n) > n^{\log_b a}$

$n \log n > n^{0.8}$

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

(iii) $T(n) = T\left(\frac{2n}{3}\right) + 1$

Sol $\Rightarrow a = 1, \; b = \frac{3}{2}, \; f(n) = 1$

$n^{\log_b a} = n^{\log_{3/2} 1} = n^{\frac{\log 1}{\log 3 - \log 2}} = 1$

$f(n) = n^{\log_b a}$

$1 = 1$

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

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

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

Sol $a = 2,\ b = 2,\ f(n) = n\log n$

$n^{\log_b a} = n^{\log_2 2} = n^{\log 2/\log 2} = n$

$\Rightarrow n^{\log_b a} < f(n)$

$n < n\log n$

$T(n) = \theta(n\log n)$ wrong ✗

master method cannot apply in this example.

Master's Method:

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ where $a \geq 1,\ b > 1$

① $T(n) = \theta\left(n^{\log_b a}\right)$

if $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ where $\epsilon > 0$

② $T(n) = \theta\left(f(n)\cdot\log n\right)$

if $f(n) = \theta\left(n^{\log_b a}\right)$

③ $T(n) = \theta\left(f(n)\right)$

if $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ where $\epsilon > 0$

and $af\left(\frac{n}{b}\right) \leq cf(n)$ & $c < 1$

Q $T(n) = 9T\left(\frac{n}{3}\right) + n$

Sol $a = 9,\ b = 3,\ f(n) = n.$

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

Acc. to 1st rule of master's Method

if $f(n) = O\left(n^{\log_b a - \epsilon}\right)$

$\Rightarrow \quad n = \Theta(n^{2-\epsilon})$

$\Rightarrow \boxed{\epsilon = 1}$

So, $\boxed{T(n) = \Theta(n^2)}$

Q $T(n) = 3T\left(\frac{n}{3} + 5\right) + n/2$

Sol $\Rightarrow T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$ (neglect 15)

because as the value of n increases, we can neglect (+5).

$a = 3, \quad b = 3, \quad f(n) = n/2$

then $n^{\log_b a} = n^{\log_3 3} = n$

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

Q $T(n) = 16T\left(\frac{n}{4}\right) + n^3$

Sol $a = 16, \quad b = 4, \quad f(n) = n^3$

then $n^{\log_b a} = n^{\log_4 16} = n^2$

Acc. to 3rd rule of Master's Method

if $f(n) = \Omega(n^{\log_b a + \epsilon}) \quad \epsilon > 0$

$\Rightarrow n^3 = \Omega(n^{2+\epsilon})$

$\Rightarrow \boxed{\epsilon = 1}$

$af\left(\frac{n}{b}\right) \leq cf(n)$

$16\left(\frac{n^3}{4^3}\right) \leq cn^3$

$\Rightarrow c \geq 1/4$

$\Rightarrow \boxed{c = 1/4}$

So, $\boxed{T(n) = \theta(n^3)}$

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

Sol $a = 4, \quad b = 2, \quad f(n) = n^2 \log n.$

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

Acc. to 3rd rule of master's method.

if $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$

$n^2 \log n = \Omega\left(n^{2+\epsilon}\right) \Rightarrow \boxed{\epsilon = 0}$ not possible.

So, master's method can not be applied on this.

Q $T(n) = T\left(\frac{9n}{10}\right) + n$

Sol $a = 1, \quad b = 10/9 \quad f(n) = n$

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

Acc. to 3rd rule of master's method.

if $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$

$n = \Omega\left(n^{0+\epsilon}\right)$

$\Rightarrow \boxed{\epsilon = 1}$

& $af\left(\frac{n}{b}\right) \leq cf(n)$

$\Rightarrow 1\left(\frac{9n}{10}\right) \leq c(n) \Rightarrow c \geq \frac{9}{10}$

$\Rightarrow \boxed{c = 9/10}$

$\therefore$ $\boxed{T(n) = \theta(n^1)}$

Changing Variable:

Q) $T(n) = T(\sqrt{n}) + 1$, solve by master's theorem

Sol $a = 1$, $b = 1$, $f(n) = 1$

cannot apply as $b \neq 1$

Let $n = 2^m$

$\Rightarrow \boxed{\sqrt{n} = 2^{m/2}}$

$\log_2 n = \log_2 2^m$

$\log_2 n = m \log_2 2$

$\Rightarrow \boxed{m = \log_2 n}$

$\therefore T(2^m) = T(2^{m/2}) + 1$

now let $S(m) = T(2^m)$

then $S(m) = S\left(\frac{m}{2}\right) + 1$

$\therefore a = 1$, $b = 2$, $f(m) = 1$

$m^{\log_b a} = m^0 = 1$

$\therefore T(m) = \theta(1 \cdot \log m)$

$\therefore T(m) = \theta(\log m)$

then $T(n) = \theta(\log(\log n))$ $\{\because m = \log n\}$

Quick Sort:

It is based on divide and Conquer approach.

Quick Sort (A, P, R)

1) If $P < R$ then
2) $q \leftarrow$ partition (A, P, R)
3) Quick Sort $(A, P, q-1)$
4) Quick Sort $(A, q+1, R)$

partition (A, P, R)

1) $x \leftarrow A[R]$
2) $i \leftarrow P-1$

} Initialise

3) for $j \leftarrow p$ to $(R-1)$
4) If $A[j] \leq x$
5) $i \leftarrow i+1$
6) $A[i] \leftrightarrow A[j]$

} loop

7) $i \leftarrow i+1$
8) $A[i] \leftrightarrow A[R]$

Q:

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

Implement the quick sort.

Sol:

$P = 1$, $R = 8$.

① $p < R$ (true)

② partition (A, 1, 8)

① $x \leftarrow A[8]$ (4)

② $i \leftarrow P-1 \Rightarrow \boxed{i = 0}$

③ for $j \leftarrow 1$ to 7. $(n-1)$

④ $A[j] \leq x$ $2 \leq 4$ (true)

⑤ $i \leftarrow 1$

⑥ $A[1] \leftrightarrow A[1]$

③ $j \leftarrow 2$

④ $A[2] \leq 4$ $8 \leq 4$ (false)

③ $j \leftarrow 3$

④ $A[3] \leq 4$ $7 \leq 4$ (false)

⑧ $j \leftarrow 4$

④ $A[4] \leq 4$ $1 \leq 4$ (true)

⑤ $i \leftarrow 2$.

⑥ $A[2] \leftrightarrow A[4]$

2	1	7	8	3	5	6	4

③ $j \leftarrow 5$

④ $A[5] \leq 4$ $3 \leq 4$ (true)

⑤ $i \leftarrow 3$

⑥ $A[3] \leftrightarrow A[5]$

2	1	3	8	7	5	6	4

③ $j \leftarrow 6$

④ $A[6] \leq 4$ $5 \leq 4$ (false)

③ $j \leftarrow 7$

④ $A[7] \leq 4$ $6 \leq 4$ false

③ $i \leftarrow 8$ (false)

(7) $i \leftarrow i+1$

$\Rightarrow i \leftarrow 4$.

(8) $A[4] \leftrightarrow A[8]$

2	1	3	4	7	5	6	8

2	1	3	4	7	5	6	8

Pivot element

Analysis of Quick Sort:

(1) Best Case:

$a = 2$, $b = 2$.

ans. $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

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

$\Rightarrow T(n) = \Omega(n \log n)$ by Master's Method

Best case of quick sort is when data is divided into equal parts.

(2) Worst Case:

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

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

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

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

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

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

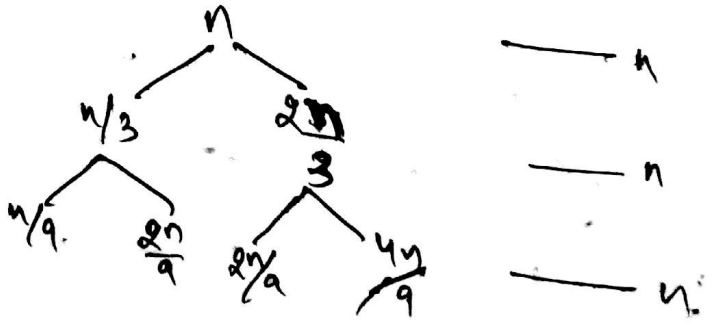
$$T(n) = \frac{n^2 + 3n - 2}{2}$$

$$\Rightarrow \boxed{T(n) = O(n^2)}$$

Note: Worst case of quick sort is $O(n^2)$ is not the best because insertion sort takes less time $\Omega(n)$ in best case when data is already sorted. It means insertion sort is best as compared to quick sort ~~and~~ when the data is already sorted (best case of insertion sort)

(3) Average case

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



$$T(n) = (n + n + n + \cdots \text{upto } (\log_2 n + 1))$$

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

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

$$\Rightarrow T(n) = n\log_2 n$$

$$\boxed{T(n) = \Theta(n\log_2 n)}$$

Heap Sort :-

Heap Sort maintain the complete binary tree except the leaf node.

Heap maintain the two properties :-

1. Max heap (Parent value is greater than child)
2. Min heap (Parent value is Smaller than child)

for max-heap $A[\text{Parent}(i) \geq A(i)]$

for min-heap $A[\text{parent}(i) \leq A(i)]$

* The index of left child of any parent whose index is 'i' is given by

$$\boxed{\text{left child}(\text{parent}(i)) = 2i}$$

* The index of right child

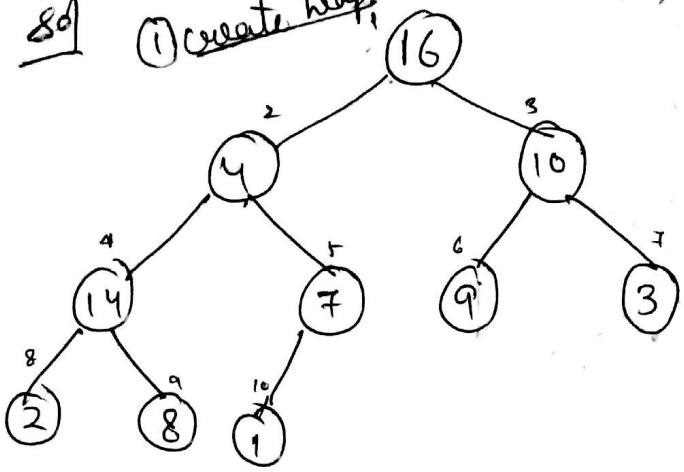
$$\boxed{\text{right child}(\text{parent}(i)) = 2i + 1}$$

* Parent of a child on index 'i' is given by

$$\text{parent}(\text{child}(i)) = \lfloor i/2 \rfloor$$

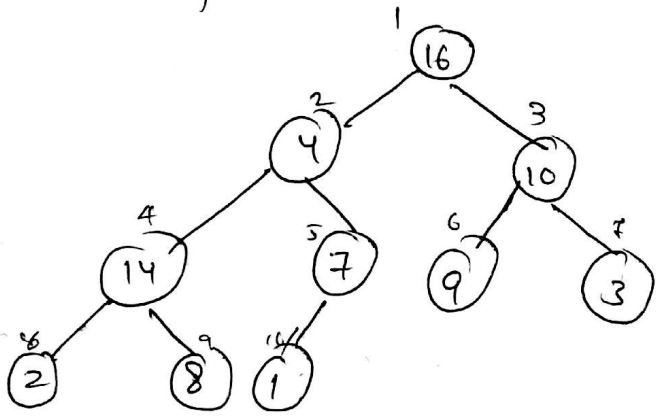
Q | 16 | 4 | 10 | 14 | 7 | 9 | 3 | 2 | 8 | 1 |

Sol (1) create heap,

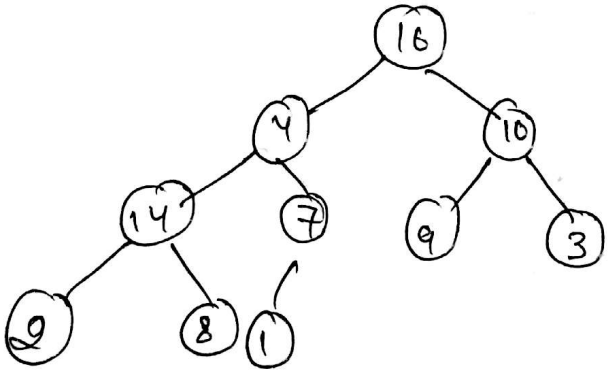


(2) Call max-heapify on $\lfloor 9/2 \rfloor$.

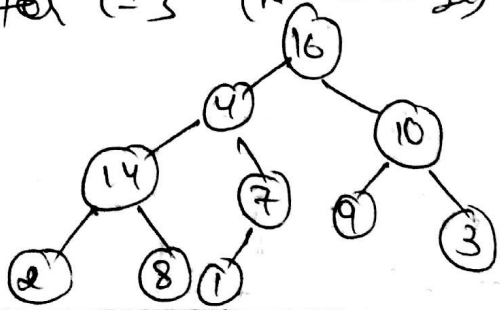
for $i = 5$ (No change)



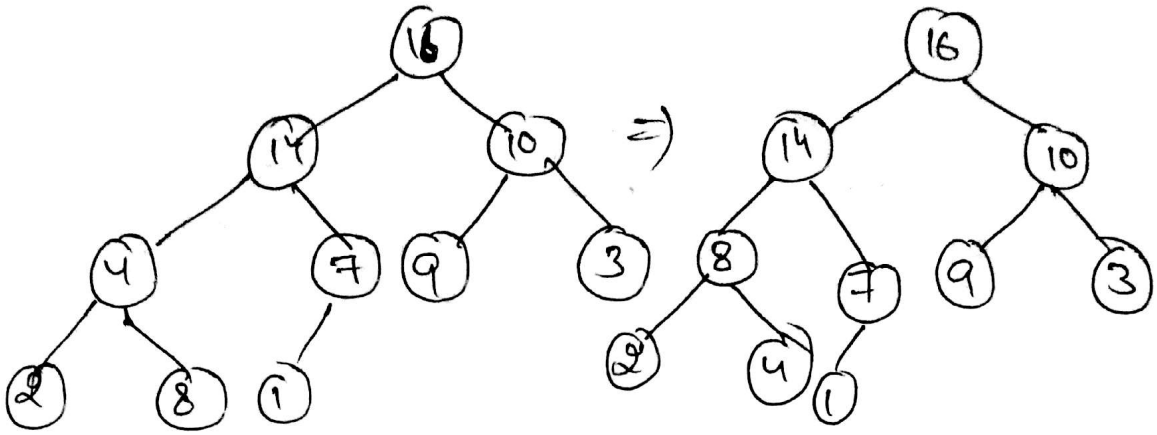
for $i = 4$ (No change)



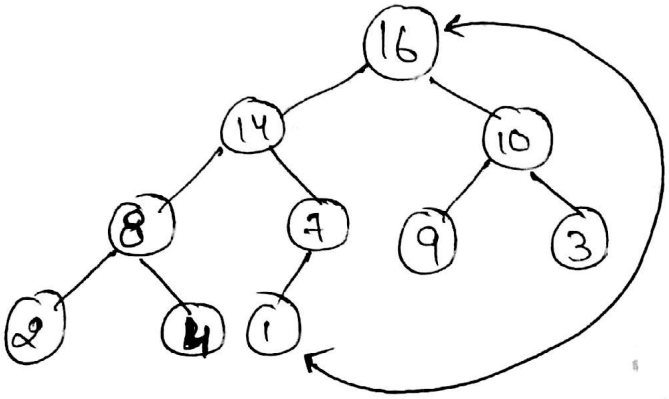
for $i = 3$ (No change)



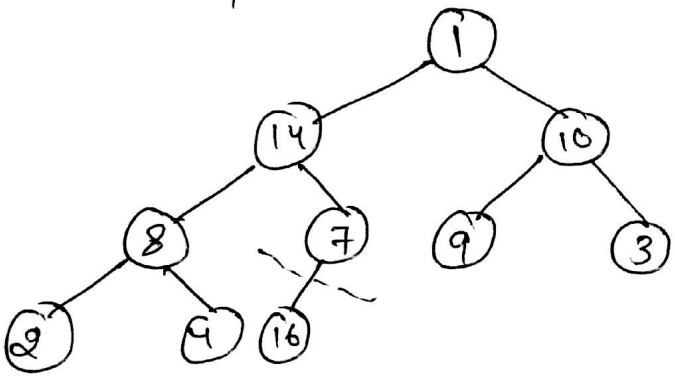
for i = 2.

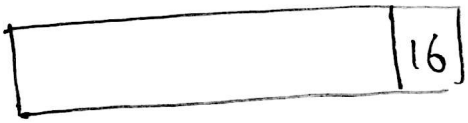


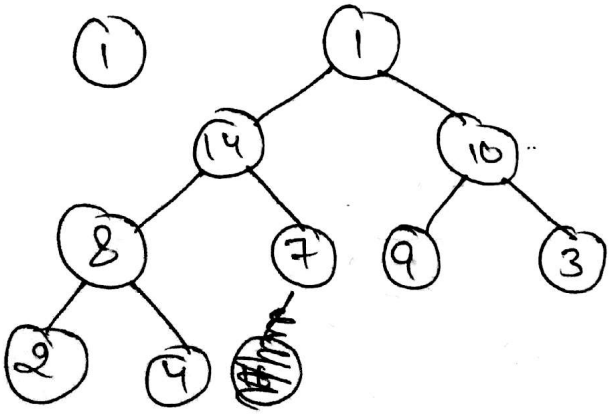
for i = 1



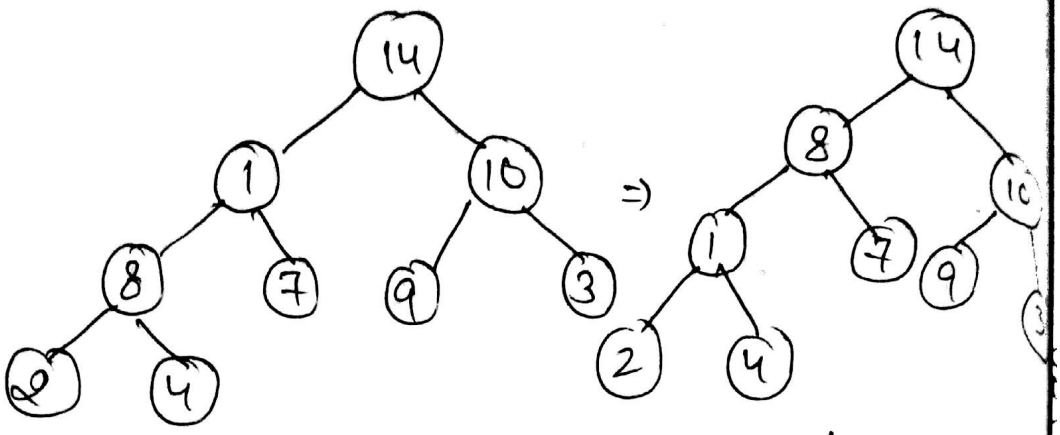
(3) Heap sort

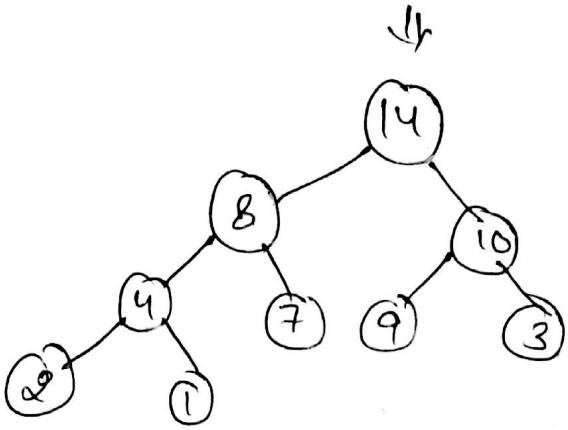




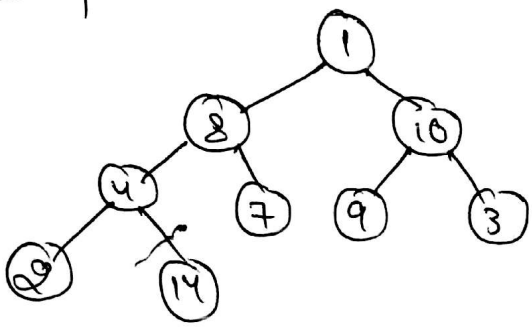


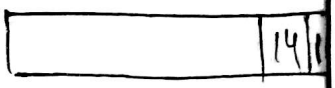
(2) Call max-heapify on root node.

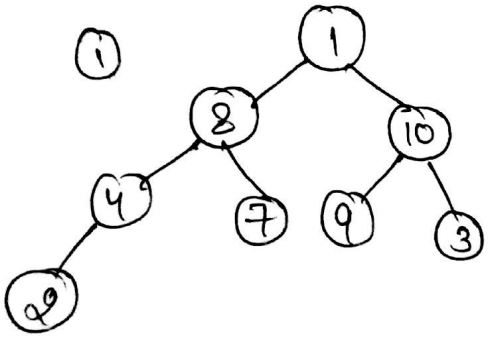




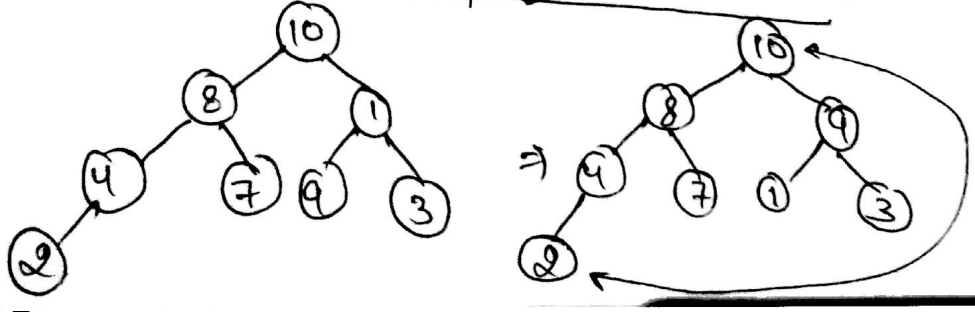
(3) Heap Sort



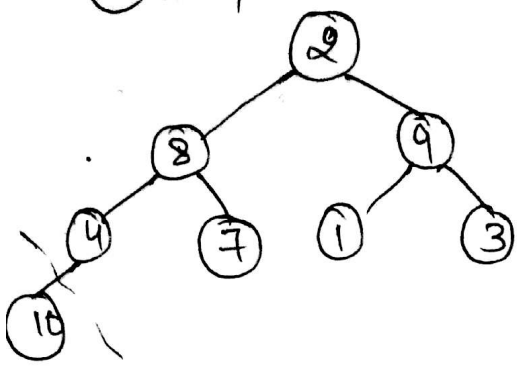




(2) Call max-heapify on root node

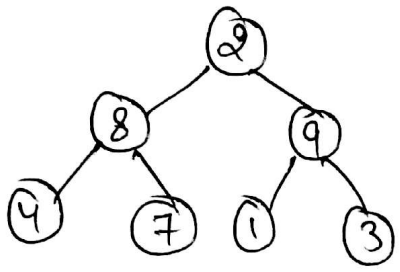


(3) Heap Sort

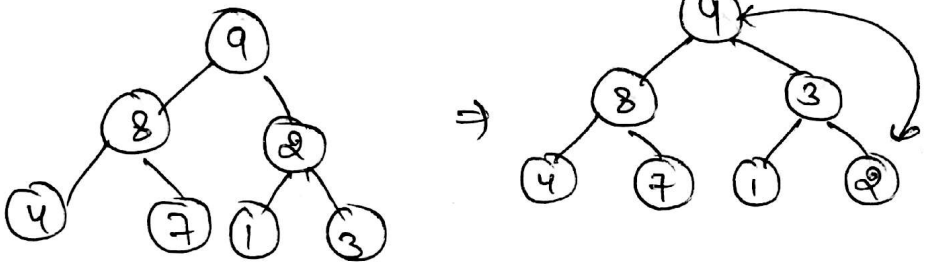


										10	14	16			

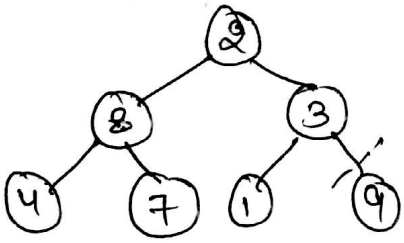
(1)



(2) Call-maxheapify on root node

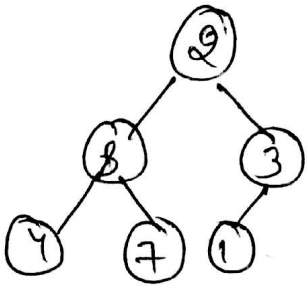


(3) Heap-Sort.

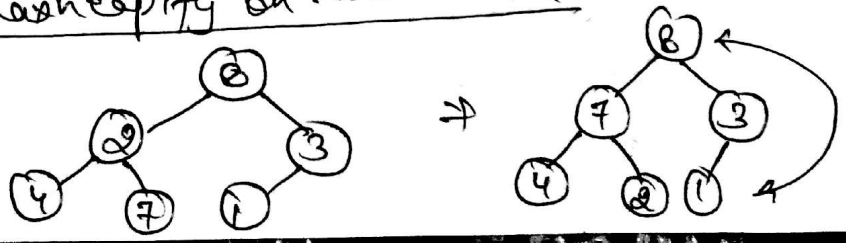


									9	10	14	16			

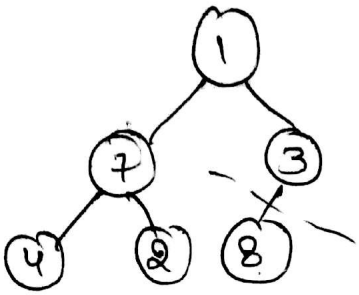
(1)



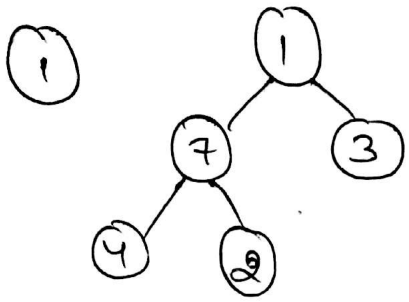
(2) Call-maxheapify on root node



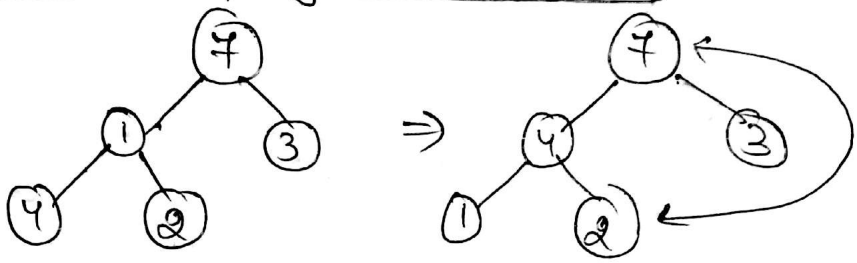
③ heap sort



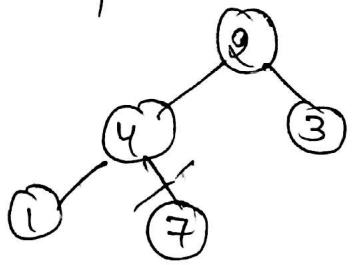
	8	9	10	14	16



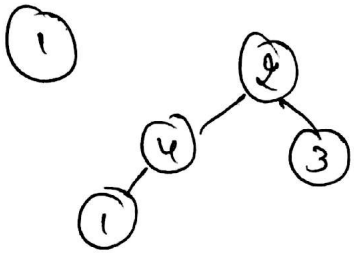
② Call maxheapify on root node



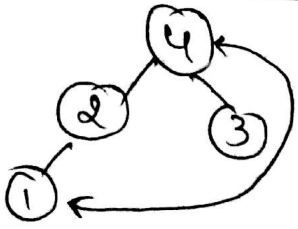
③ heap-sort



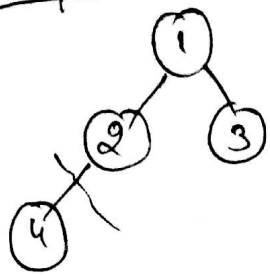
	7	8	9	10	14	16



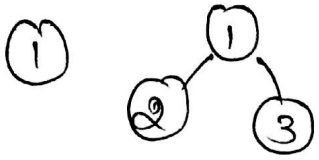
② Call-Maxheapify on root node



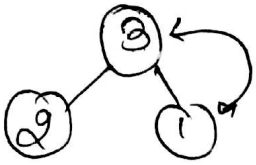
③ heap sort



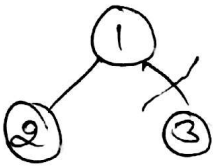
	4	7	8	9	10	14	16



② call max-heapify on root node

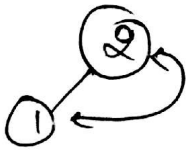


③ heap sort.

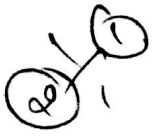


	3	4	7	8	9	10	14	16

② call maxheapify on root node



③ heap sort



	2	3	4	7	8	9	10	14	16

③ heapsort

→

1	2	3	4	7	8	9	10	14	16

Algorithm for Heap Sort

Build Max-HEAP [A]

1. heap-size [A] ← length [A]
2. for i ← ⌊length [A] / 2⌋ down to 1
3. do max-heapify (A, i)

MAX-HEAPIFY (A, i)

1. l = LEFT (i)
2. r = RIGHT (i)
3. If l ≤ heap size [A] and A[l] > A[i]
4. then largest ← l
5. else largest ← i
6. If r ≤ heapsize [A] and A[r] > A[largest]
7. then largest ← r
8. If largest ≠ i
9. then exchange A[largest] ↔ A[i]
10. MAX-HEAPIFY (A, largest)

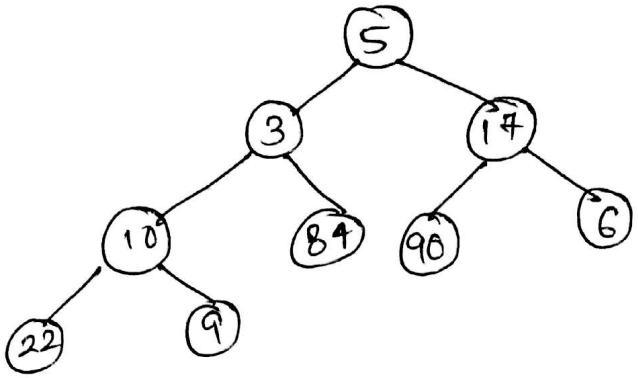
HEAP SORT (A)

1. Build MAX-HEAP (A)
2. for i ← length (A) down to 2.
3. do exchange A[1] ↔ A[i]
4. heap size [A] ← heap size [A] - 1
5. MAX HEAPIFY (A, 1)

eg)

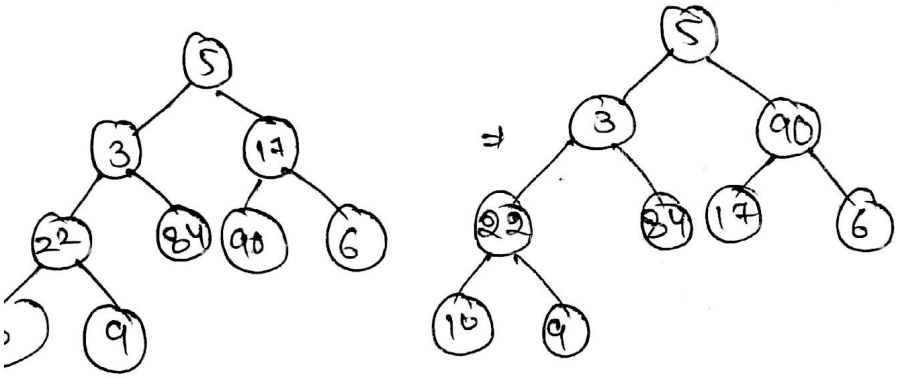
5	3	17	10	84	90	6	22	9

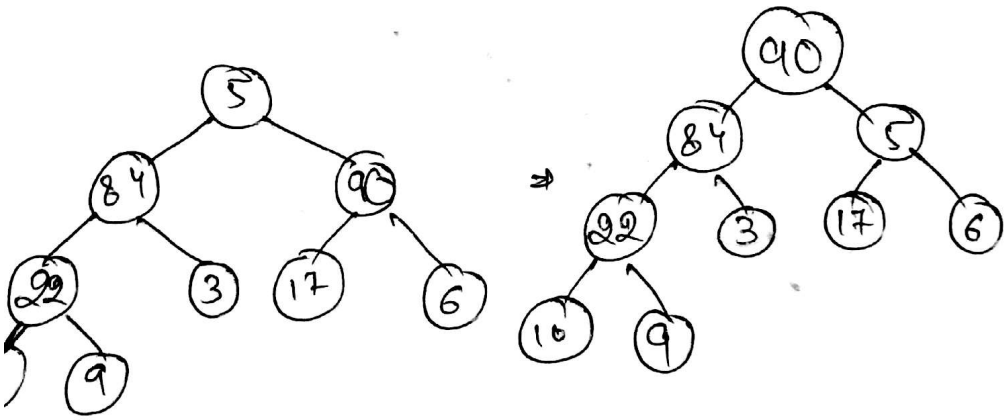
Sol (1) Create heap

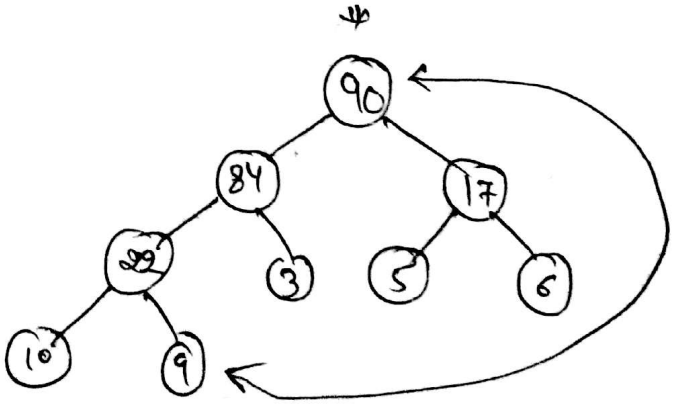


(2) call Max-heapify on i/2.

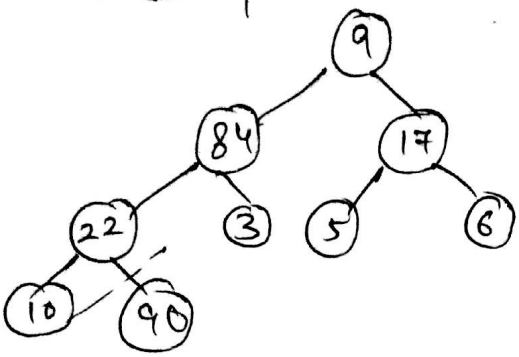
for i = 4.





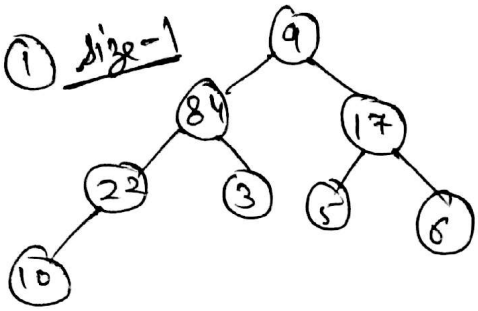


③ Heap Sort

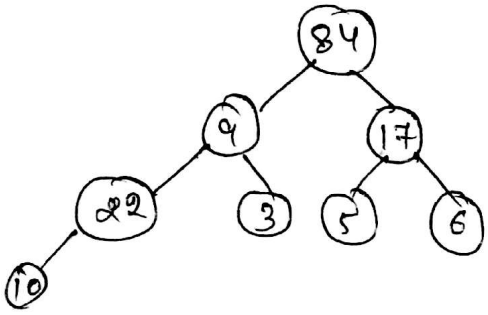


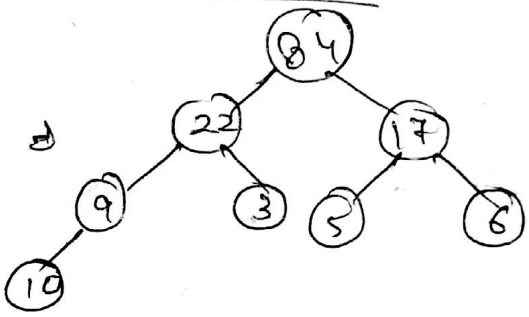
190

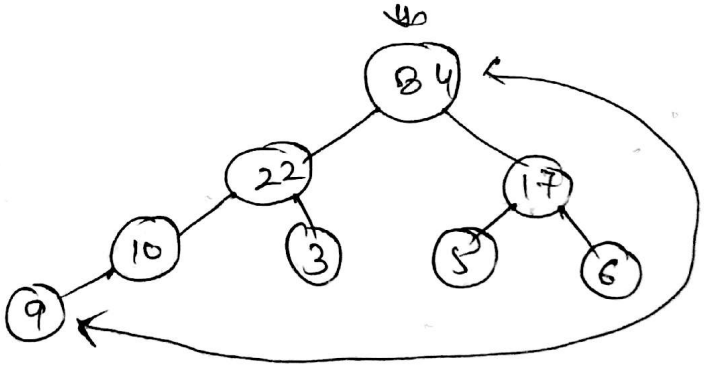
① size-1



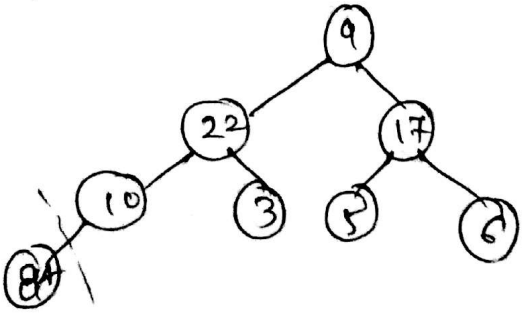
② Call max-heapify on root node





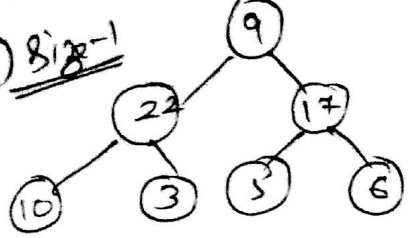


③ Heap Sort

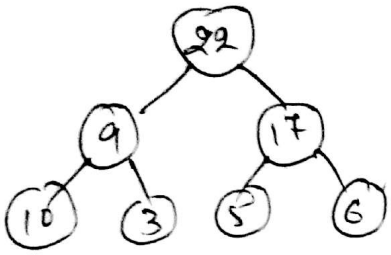


84 90

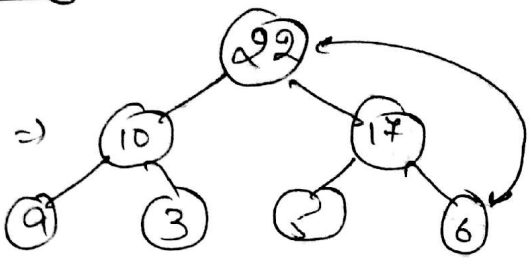
(1) Size-1



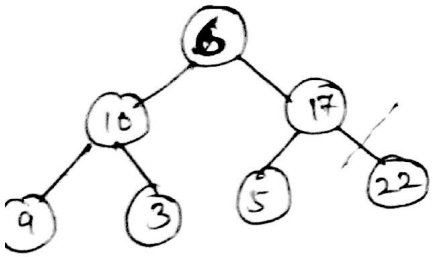
(2) Call Max-heapify on root node

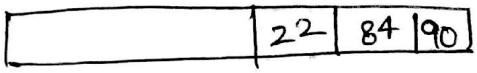


=>

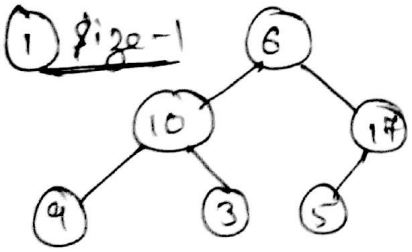


(3) Heap-Sort

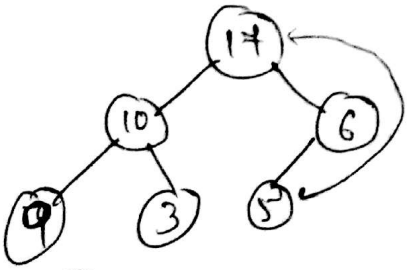




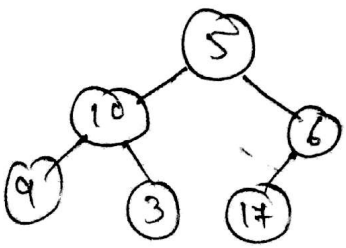
(1) Size-1

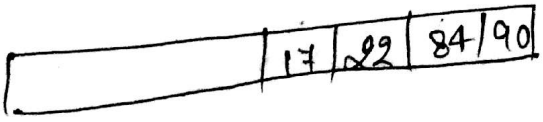


(2) Call max-heapify on root node

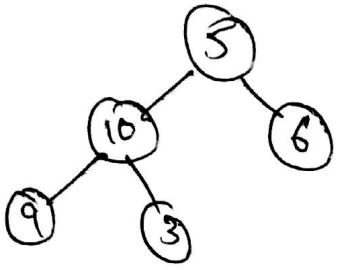


(3) Heap-Sort



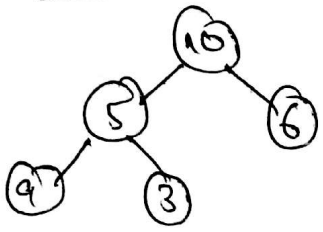


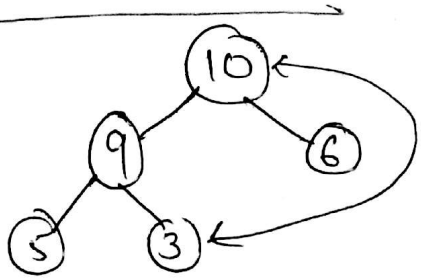
(1) Heap Size - 1



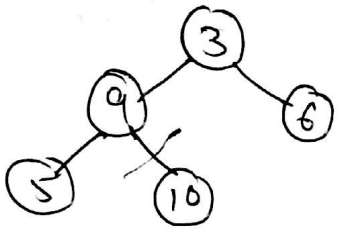
10

(2) Call max-heapify on root node



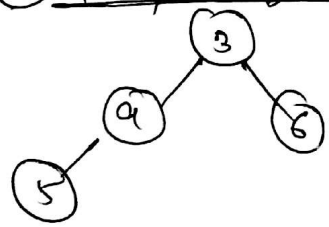


(3) Heap - Sort

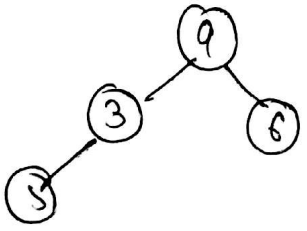


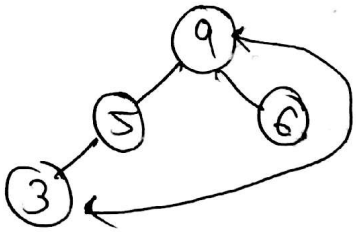
	10	17	22	84	90

(1) Heap - Size - 1

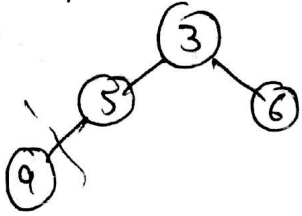


(2) Call maxheapify on root node



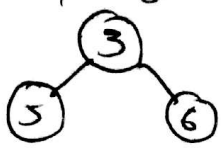


(3) Heap Sort

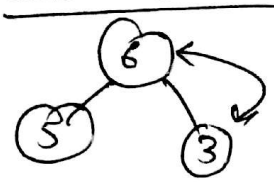


	9	10	17	22	84	90

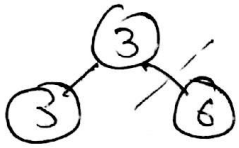
① Heap.size-1



② Call max-heapify on root node

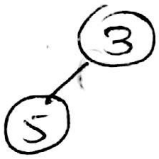


③ Heapsort



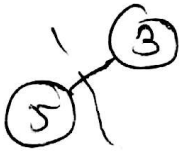
	6	9	10	17	22	84	90

① size-1



② call max-heapify on root node

③ Heap sort



	5	6	9	10	17	22	84	90

① Heap size-1

③ ~~call max~~ Heap sort

3	5	6	9	10	17	22	84	90

Complexity of heap sort

$$T(n) = O(n \log_2 n)$$

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

$$T(n) = \Omega(n \log_2 n)$$

Sorting in Linear time

We have discussed several algorithms such as insertion sort, heap sort, merge sort, etc. with running time $\Theta(n^2)$, $\Theta(n \log n)$ or both in worst case.

These algorithms have an interesting property. they based on comparison b/w the array elements only.

We call these ~~sorting~~ algorithms as Comparison sorting

Stable sorting

It is based on some operations for sorted.

eg] Counting sort.

Counting Sort:

Counting Sort (A, B, K)

```
(1) for i ← 0 to K.
(2)     do C[i] ← 0
(3) for j ← 1 to length [A]
```

④ do $C[A[j]] \leftarrow C[A[j]] + 1$ }

⑤ for $i \leftarrow 1$ to $|C|$. }

⑥ do $C[i] \leftarrow C[i] + C[i-1]$ }

⑦ for $j \leftarrow$ length[A] down to 1

⑧ do $B[C[A[j]]] \leftarrow A[j]$

⑨ $C[A[j]] \leftarrow C[A[j]] - 1$

A)

2	5	3	0	2	3	0	3

Id

2	5	3	0	2	3	0	3

Step 1 - ②

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

Step 2 - ④

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

updated C

Step 3 - ⑥

C =

2	2	4	7	7	8
0	1	2	3	4	5

updated C

⑦ for $j = 8$ to 1.

⑧ $B[C[A[8]]] \leftarrow A[8]$ (3)

B =

						3	
1	2	3	4	5	6	7	8

⑨ C =

2	2	4	6	7	8
0	1	2	3	4	5

⑦ for $j = 7$

⑧ $B[C[A[7]]] \leftarrow A[7]$ (0)

B =

	0					3	
1	2	3	4	5	6	7	8

⑨ C = | 1 | 2 | 4 | 6 | 7 | 8 |

⑦ for j = 6

⑧ B[C[A[6]]] ← A[6] (3)

B = | | 0 | | | | 3 | 3 | |
1 2 3 4 5 6 7 8

⑨ C = | 1 | 2 | 4 | 5 | 7 | 8 |
0 1 2 3 4 5

⑦ for j = 5

⑧ B[C[A[5]]] ← A[5] (2)

B = | | 0 | | 2 | | 3 | 3 | |
1 2 3 4 5 6 7 8

⑨ C = | 1 | 2 | 3 | 5 | 7 | 8 |
0 1 2 3 4 5

⑦ for j = 4

⑧ B[C[A[4]]] ← A[4] (0)

B = | 0 | 0 | | 2 | | 3 | 3 | |
1 2 3 4 5 6 7 8

⑨ C = | 0 | 2 | 3 | 5 | 7 | 8 |
1 2 3 4 5

⑦ for j = 3

⑧ B[C[A[3]]] ← A[3] (3)

B = | 0 | 0 | | 2 | 3 | 3 | 3 | |
1 2 3 4 5 6 7 8

⑨ C = | 0 | 2 | 3 | 4 | 7 | 8 |
0 1 2 3 4 5

(7) for j = 2

(6) B[C[A[2]]] ← A[2] (5)

B = | 0 | 0 | | 2 | 3 | 3 | 3 | 5 |
1 2 3 4 5 6 7 8

(4) C = | 0 | 2 | 3 | 4 | 7 | 7 |
0 1 2 3 4 5

(7) for j = 1

(8) B[C[A[1]]] ← A[1] (2)

B = | 0 | 0 | 2 | 2 | 3 | 3 | 3 | 5 |
1 2 3 4 5 6 7 8

(4) C = | 0 | 2 | 2 | 4 | 7 | 7 |
0 1 2 3 4 5

(7) for j = 0 (false)

| 0 | 0 | 2 | 2 | 3 | 3 | 3 | 5 |

(1)

| 3 | 6 | 4 | 1 | 3 | 4 | 1 | 4 |

(2)

A = | 3 | 6 | 4 | 1 | 3 | 4 | 1 | 4 |

size = 7
because max = 6.

Step (1)-(2)

C = | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
0 1 2 3 4 5 6

Step (3)-(4)

C = | 0 | 2 | 0 | 2 | 3 | 0 | 1 |
0 1 2 3 4 5 6

Step (5)-(6)

C = | 0 | 2 | 2 | 4 | 4 | 7 | 7 |
0 1 2 3 4 5 6

(7) B =

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

Complexity of Counting Sort

$T(n) = O(n)$

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

$T(n) = \Omega(n)$

Q.

6	0	2	0	1	3	4	6	1	3	2

Sol

A =

6	0	2	0	1	3	4	6	1	3	2

Size = 7 because ma

step ① - ②

C =

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

step ③ - ④

C =

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

step ⑤ - ⑥

C =

2	4	6	8	9	9
0	1	2	3	4	5	6

(7)

B =

0	0	1	1	2	2	3	3	4	6	6

updated C =

0	2	4	6	8	9	9

Radix Sort

Radix Sort (A, d)

① for $i \leftarrow 1$ to d

② use a stable sort / counting sort

complexity:

$$T(n) = O(n)$$

Q. 326, 453, 608, 835, 751, 435, 704, 690

Sol

3	2	6
4	5	3
6	0	8
8	3	5
7	5	1
4	3	5
7	0	4
6	9	0

① for $i = 1$ using counting sort

②

6	9	0
7	5	1
4	5	3
7	0	4
8	3	5
4	3	5
3	2	6
6	0	8

① for $i = 2$ using a counting sort

②

7	0	4
6	0	8
3	2	6
8	3	5
4	3	5
7	5	1
4	5	3
6	9	0

① for $i = 3$ using counting sort

②

3	2	6
4	3	5
4	5	3
6	0	8
6	9	0
7	0	4
7	5	1
8	3	5

Bucket Sort :

Bucket sort is used for floating point numbers and range is $0 \leq A[i] < 1$

where $n = 10$, fixed, on the basis of index

Bucket Sort

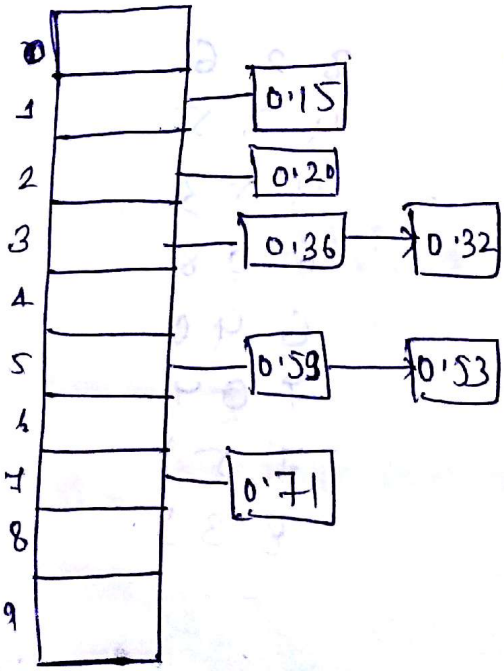
$0 \leq A[i] < 1$

1. $n \leftarrow$ length (A)
2. for $i \leftarrow 1$ to n
3. do insert $A[i]$ into $B[n * A[i]]$
4. for $i = 0$ to $n-1$
5. Sort $B(i)$ with insertion sort
6. Can concatenate list $B(i)$

Q. 0.36, 0.15, 0.20, 0.59, 0.53, 0.71, 0.32

Sol

① $n = 10$ (fixed)



④ for i=0

⑤ B(0) with insertion sort.

No Change.

④ for i=1

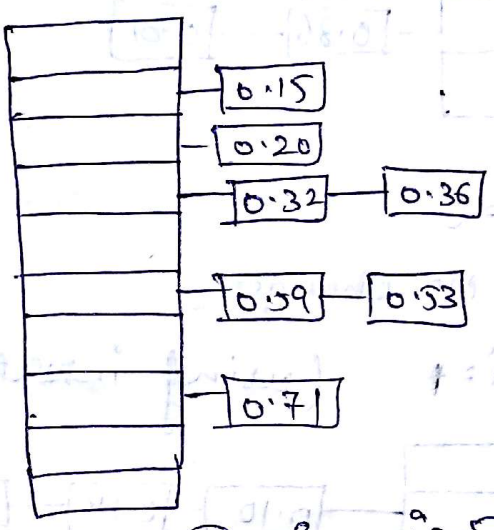
⑤ No Change

④ for i=2

⑤ No change

④ for i=3

⑤ B(3) with insertion sort.

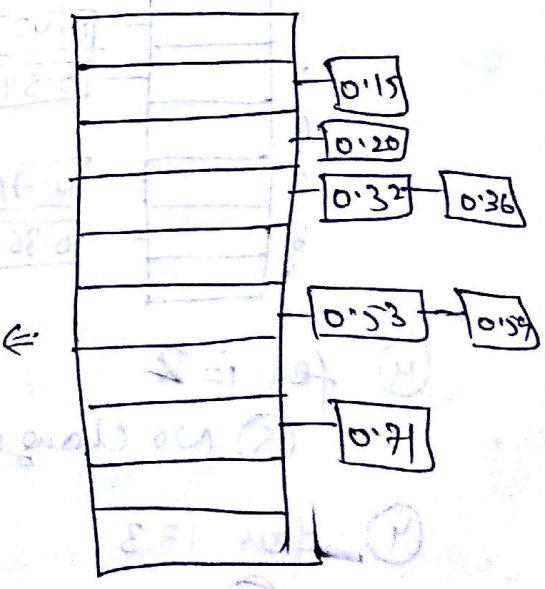


④ for i=4

⑤ No Change.

④ for i=5

⑤ B(5) with insertion



④ for i=6

③ No Change

④ for i=7.

⑤ No Change.

④ for i=8

⑤ No Change.

④ for i=9

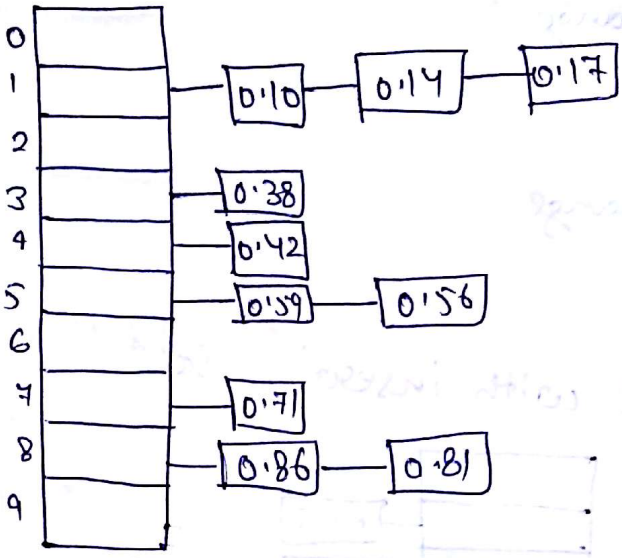
⑤ No Change

0.10, 0.20, 0.32, 0.36, 0.53, 0.59, 0.71

Q. 0.42, 0.71, 0.10, 0.14, 0.86, 0.38, 0.59, 0.81, 0.56

Sol ① n=10 fixed.

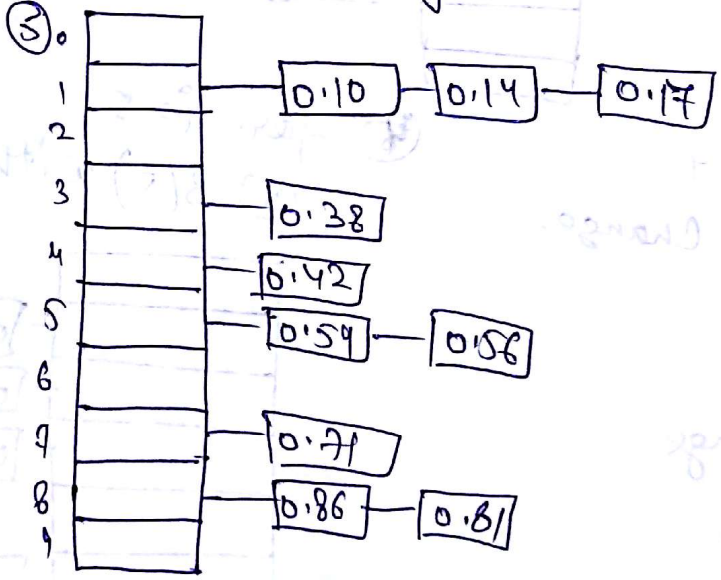
② & ③



④ for i=0

⑤ No change.

④ for i=1 (using insertion sort)



④ for i=2

⑤ No change.

④ for i=3

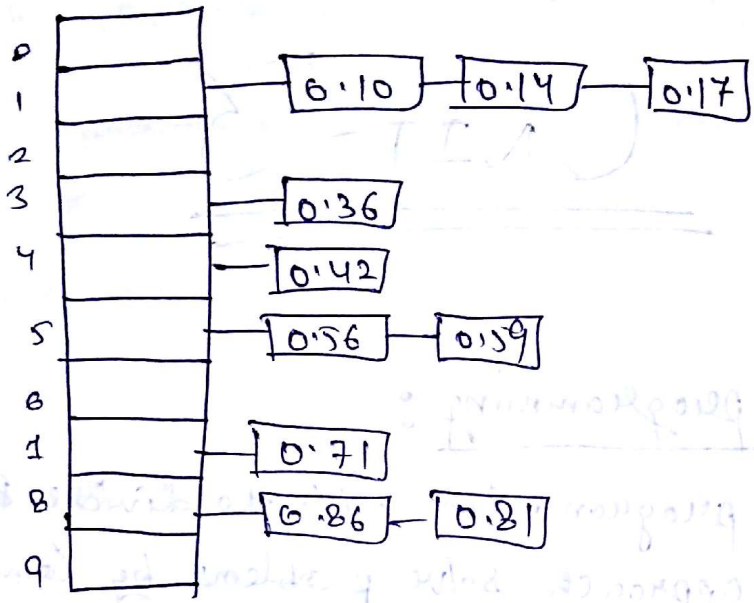
⑤

(4) for i=4

(5) No change

(4) for i=5 (using insertion sort)

(5)



(4) for i=6

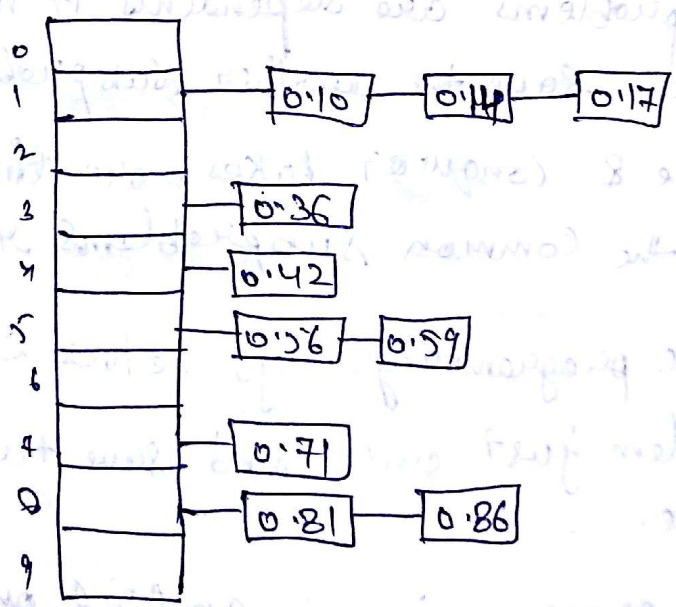
(5) No change.

(4) for i=7.

(5) No change.

(4) for i=8 (using insertion sort)

(5)



(4) for i=9

(5) No change.

(4) for i=10 (false)

0·10, 0·14, 0·17, 0·36, 0·42, 0·58, 0·59, 0·71

0·81, 0·86.