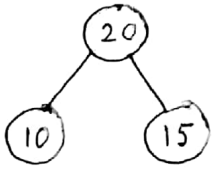
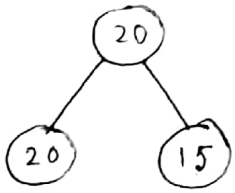
Heap Sort Algorithm

Max heap Tree:



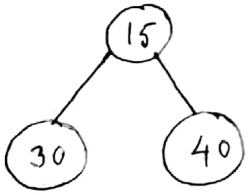
Max heap Tree



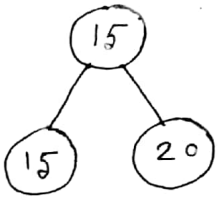
Max heap Tree

father $\geq$ child
(For Ascending order)

Min heap Tree:



Min heap Tree



Min heap Tree

father $\leq$ child
(For Descending order)

Algorithm

step1: Construct a Binary Tree with given list of elements

step2: Transform the Binary Tree into Max heap

step3: Delete the root element from Max heap using Heapify method

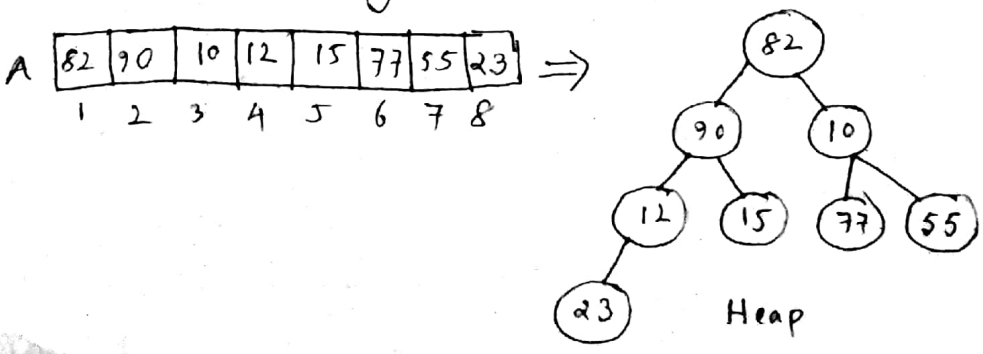
step4: Put the deleted element into the sorted list

step5: Repeat the same until the Max-heap becomes empty

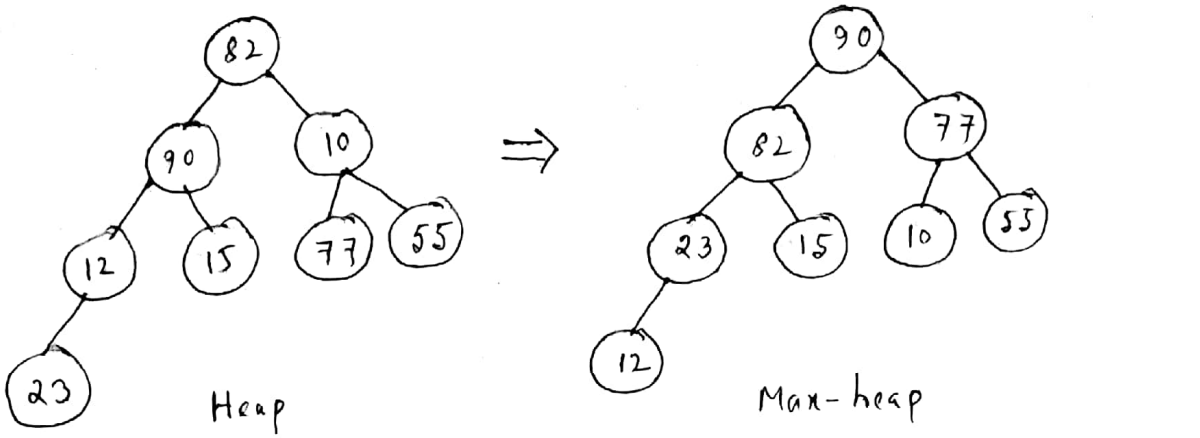
step6: Display the sorted list

Example:

step1: Construct a Binary Tree with given list of elements



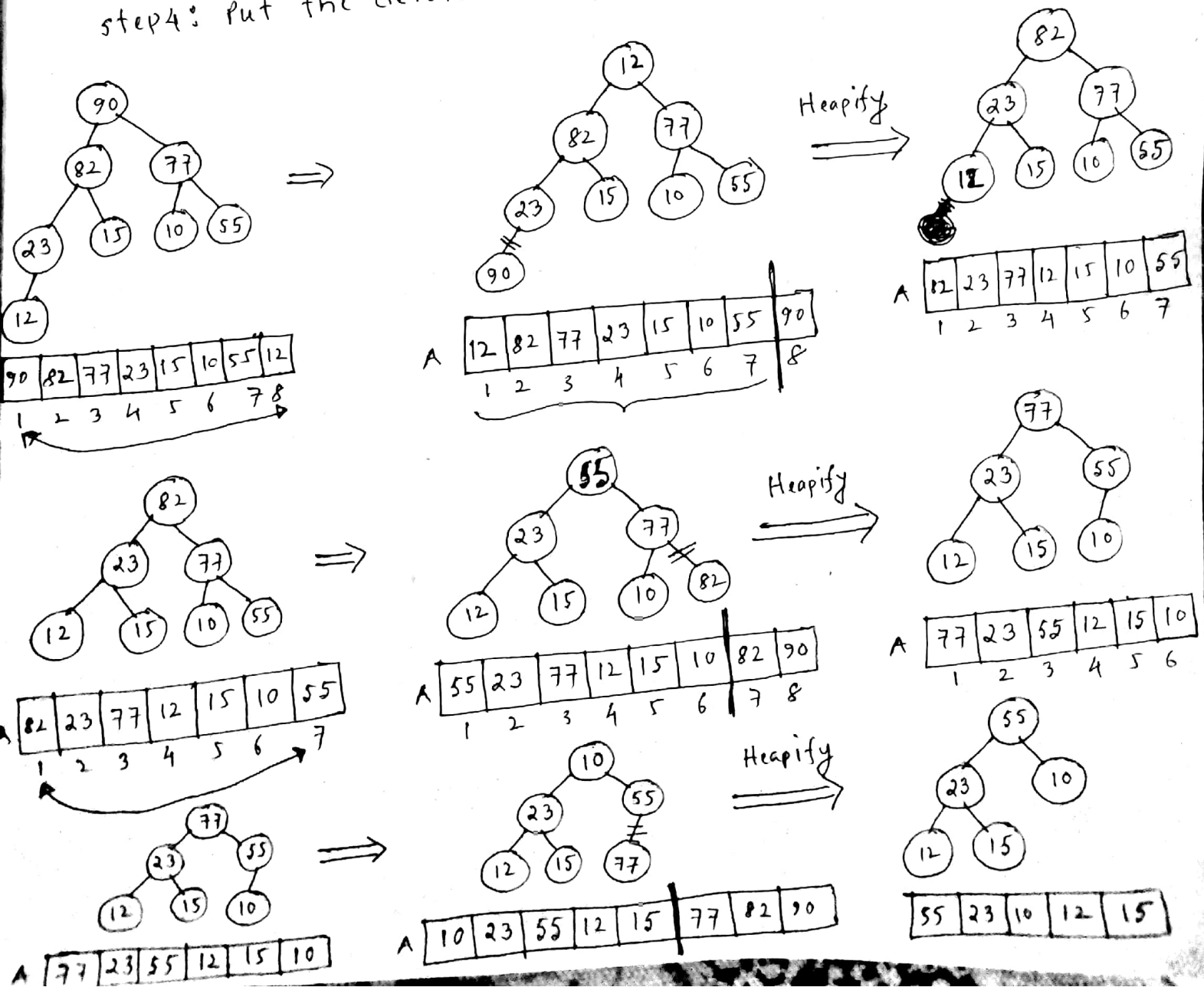
step2: Transform the binary tree into Max heap

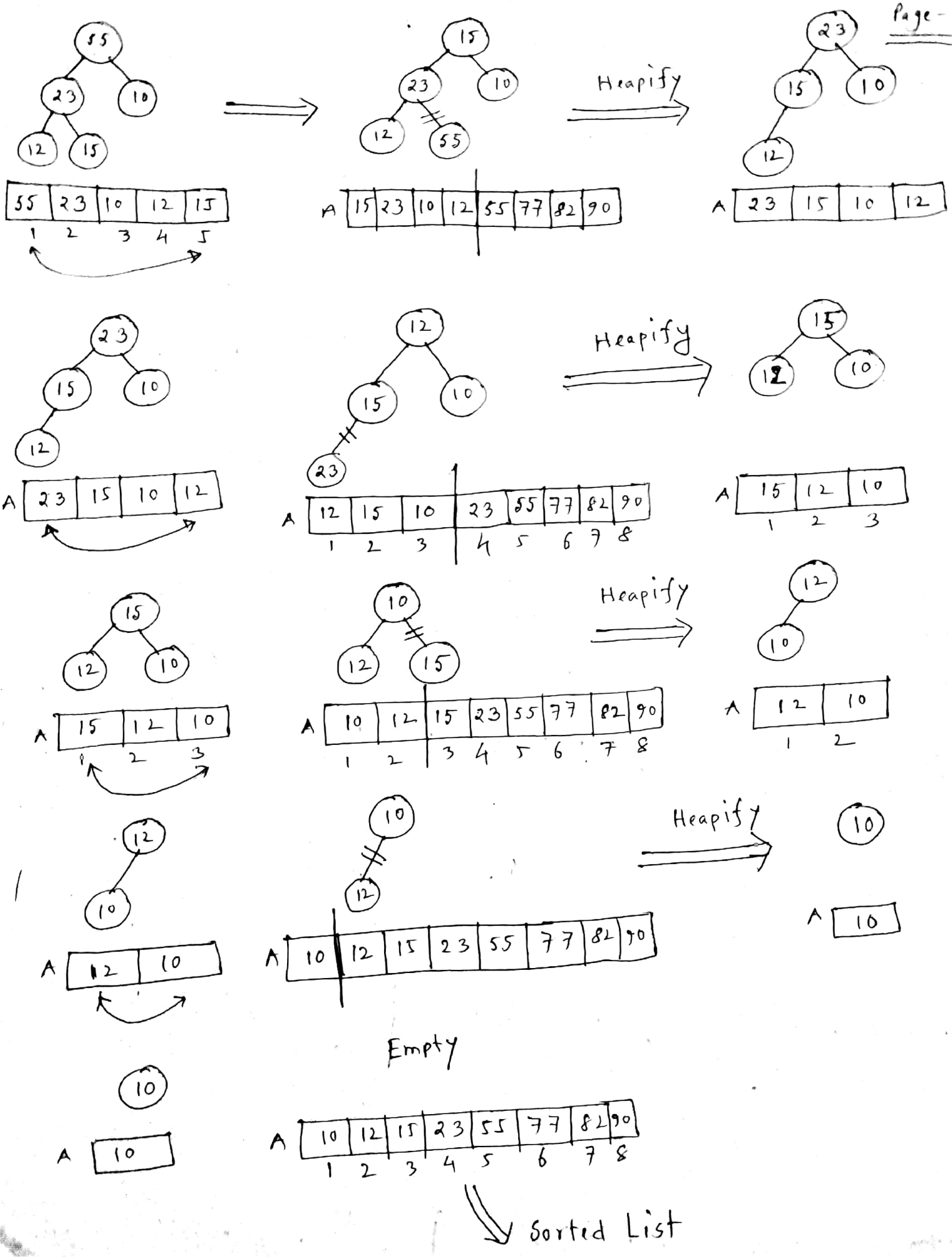


A	90	82	77	23	15	10	55	12
	1	2	3	4	5	6	7	8

step3: Delete the root element from Max heap using Heapify method

step4: Put the deleted element into the sorted list





Procedure HeapSort (A, n)

for i = $\lfloor \frac{n}{2} \rfloor$ downto 1
    Adjust (A, i, n)
end for

↑ for drawing first max heap tree

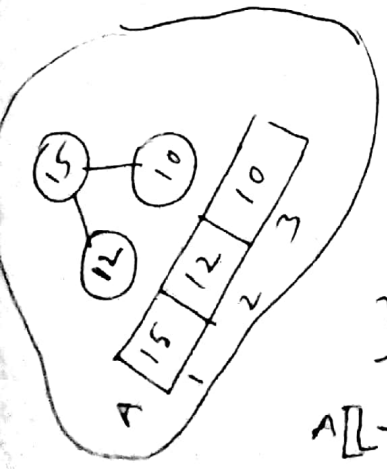
for i = n downto 2
    A[1] ⟷ A[i] → for interchanging first and last
    Adjust (A, 1, i) → for heapify
end for

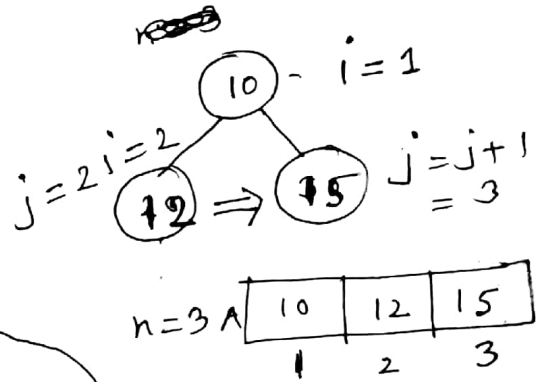
End Procedure

Procedure Adjust (A, i, n)

j = 2i; item = A[i];
while (j <= n) {
    if (j < n) and (A[j] < A[j+1])
        j = j+1
    if item >= A[j]
        exit loop
    A[⌊j/2⌋] = A[j]
    j = 2j
}
A[⌊j/2⌋] = item

end Procedure





j = 2i = 2
item = A[1] = 10
while (True)
if (True) and (12 < 15)
j = 3
if (10 >= 15)
False
A[3/2] = A[3]
⇒ A[1] = A[3]
⇒ A[1] = 15

A
15
12
15
1
2
3

j = 2j = 2*3 = 6

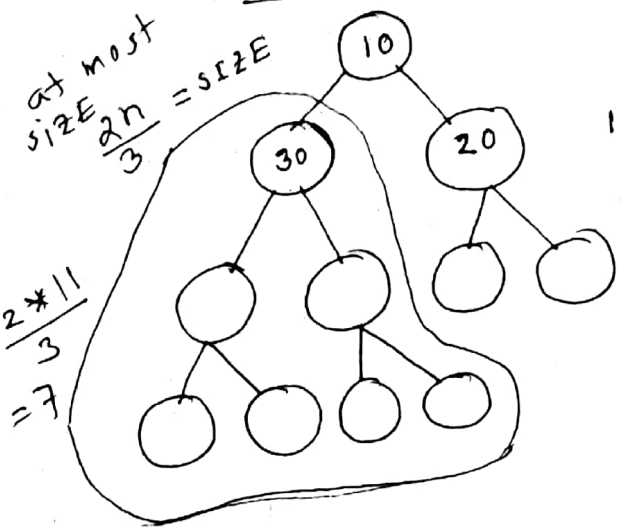
while (False)
A[6/2] = 10
⇒ A[3] = 10

A
15
12
10
1
2
3

Time Complexity Analysis

Time complexity:

Adjust or Max-heap



1. compare 30 and 20
2. Put replace 10 by 30

} required time = C, n = 1

$$T(n) = \begin{cases} D & \\ T\left(\frac{2n}{3}\right) + C & \text{when } n > 1 \end{cases}$$

Solution

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

Replace n by $\frac{2n}{3}$

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

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

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

$T(n) = D$ when $n = 1$

$\Rightarrow T(1) = D$

⋮

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

$$= T(1) + \log_{1.5} 2 * \log_2 n$$

$$= D + \log_{1.5} 2 * \log_2 n$$

$$= O(\log_2 n)$$

$\frac{2^k}{3^k} n = 1$

$\Rightarrow n = \left(\frac{3}{2}\right)^k$

$\Rightarrow \log_{1.5} n = \log_{1.5} (1.5)^k$

$\Rightarrow \log_{1.5} n = k$

$\Rightarrow k = \log_{1.5} n$

$= \log_{1.5} n * \log_2 1.5$

$= \log_2 n$

Heapsort

First for loop ⇒ 2nd for loop

$$\frac{n}{2} \log_2 n + (n-2+1)\{c_1 + \log_2 n\}$$

$$= \frac{n}{2} \log_2 n + c_1(n-1) + (n-1)\log_2 n$$

$O(n \log_2 n)$

Find time complexity of following algorithm

```
for(i=1; i<=n; ++i) {
    for(j=1; j<=n; ++j) {
        printf("%d", j);
    }
    printf("%d", i);
}
```

Ans:

statement	iteration	Time/iteration	Total
i = 1	1	c_1	$c_1 * 1$
i<=n	$(n-1+1)$ + 1 (True / False)	c_2	$c_2 * (n+1)$
++i	$(n-1+1)$ (True)	c_3	$c_3 * n$
j = 1	$n * 1$	c_4	$c_4 * n$
j<=n	$n * ((n-1+1)+1)$ (True / False)	c_5	$c_5 * n(n+1)$
++j	$n * (n-1+1)$ (True)	c_6	$c_6 * n * n$
printf(j)	$n * (n-1+1)$	c_7	$c_7 * n * n$
printf(i)	n	c_8	$c_8 * n$

Total running Time $= c_1 + c_2(n+1) + c_3 n + c_4 n + c_5 n(n+1) + c_6 n^2 + c_7 n^2 + c_8 n$

$= O(n^2)$

```
sum = 0;
for(i = 2; i <= n; i++){
    sum = sum + i;
}
```

Find time complexity

statement	iteration	Time/ iteration	Total
sum = 0	1	c_1	$c_1 * 1$
i = 2	1	c_2	$c_2 * 1$
i <= n	(n-2+1)+1 (True, false)	c_3	$c_3 n$
i++	(n-2+1) (True)	c_4	$c_4(n-1)$
sum = sum + i	(n-2+1) (True)	c_5	$c_5(n-1)$

Total running time, $T(n) = c_1 + c_2 + c_3 n + c_4(n-1) + c_5(n-1)$

$= O(n)$