ADA

1812401160001
Vishwas Acharya

Assignment-2

Q. 1) Explain the efficiency of an algorithm.

Ans- The efficiency of an algorithm is a measure of the amount of resources consumed in solving a problem of size n.

- The important resource is time, i.e., time complexity.
- To measure the efficiency of an algorithm requires to measure its time complexity using any of the following approaches:

1. To run it and measure how much processor time is needed. [Empirical approach]
2. Mathematically computing how much time is needed as a function of input [Theoretically]

- Running time of an algorithm depends upon,
  (1.) Input size
  (2.) Nature of Input.
- Generally time grows with the size of input, for example, sorting 100 numbers will take time than sorting of 10,000 numbers.
- So, running time of an algorithm is usually measured as a function of input size.
- In theoretical computation of time complexity, Running time is measured in terms of number of steps/primitive operations performed.

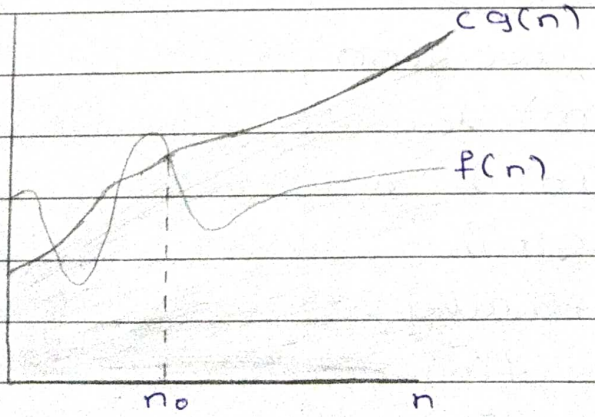
Q.2] Explain Asymptotic Notation in Detail.

Ans- Asymptotic Notations (Big O, Θ- Theta & Ω- omega) allow us to analyze an algorithm's running time.
- This is also known an algorithm's growth rate.
- Asymptotic Notation are used,

1. To characterize the complexity of an algorithm.
2. To compare the performance of two or more algorithms solving the same problem.

• O- Notation (Big O notation) (upper Bound)
- For a given function $g(n)$, we denote by $O(g(n))$ the set of functions,
$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that
$0 \leq f(n) \leq cg(n)$ for all $n_0 < n\}$

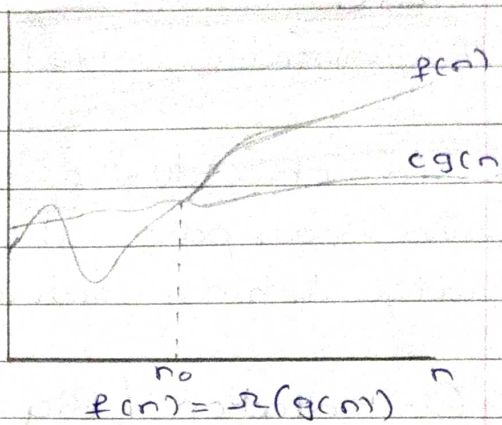
- $g(n)$ is an asymptotic upper bound for $f(n)$
- $f(n) = O(g(n))$ implies : $f(n)$ "$\leq$" $c \cdot g(n)$



$f(n) = o(g(n))$

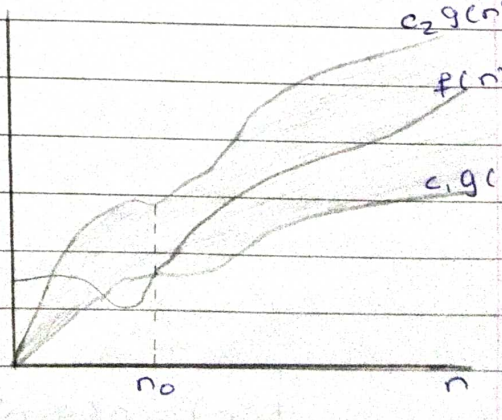
- Ω - Notation (omega notation) (Lower Bound)
- For a given notation $g(n)$, we denote by $\Omega(g(n))$ the set of functions,
$\Omega(g(n)) = \{ f(n) :$ these exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n_0 \le n \}$

- $g(n)$ is an asymptotically lower bound for $f(n)$
- $f(n) = \Omega(g(n))$ implies:
$f(n)$ "$\ge$" $c \cdot g(n)$'



- Θ - Notation (Thetha notation) (same orders)
- For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions,
$\Theta(g(n)) = \{ f(n) :$ these exist positive constants c_1, c_2 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n_0 \le n \}$

- $\Theta(g(n))$ is a set, we can write $f(n) \in \Theta(g(n))$ to indicate that $f(n)$ is a member of $\Theta(g(n))$
- $g(n)$ is an asymptotically tight bound for $f(n)$
- $f(n) = \Theta(g(n))$ implies:
$f(n)$ "$=$" $c \cdot g(n)$



Q.3) What is ~~Amortized~~ Amortized Analysis.

Ans- Amortized analysis is considers not just one operation, but a sequens of operation on a given data structure.
- The time sequired to perform a sequence of data ~~struct~~ structure operations is averaged over all operations performed.
- Amortized analysis can be used to show that the average cost of an operation is small even though a single operation might be expensive.

- These are three most common techniques of amortized analysis,

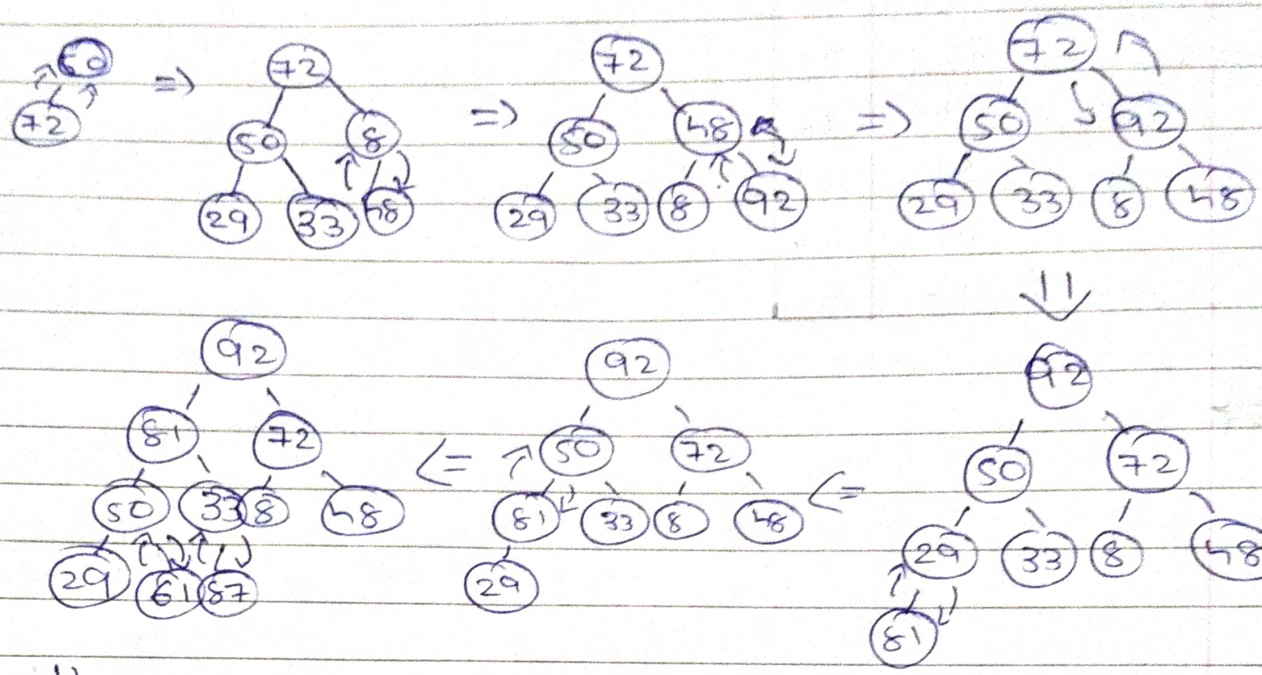
1. Aggregate method.
- A sequence of n operation takes worst case time T(n)
- Amortized cost per operation is T(n)/n.

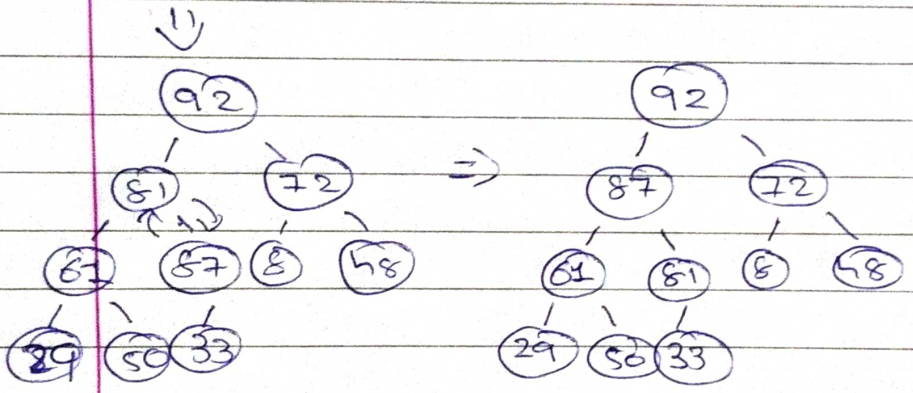
2. The accounting method.
- Assign each type of operations an amortized cost.
- Overcharge some operations.
- Stop the overcharges as credit on specific objects.
- Then use the credit for compensation for some later operations

3. Potential method
- Same as accounting method.
- But store the credit as "potential energy" and as a whole.

Q.4 Using heap sort algorithm sort the array
50, 72, 8, 29, 33, 48, 92, 81, 61, 87.

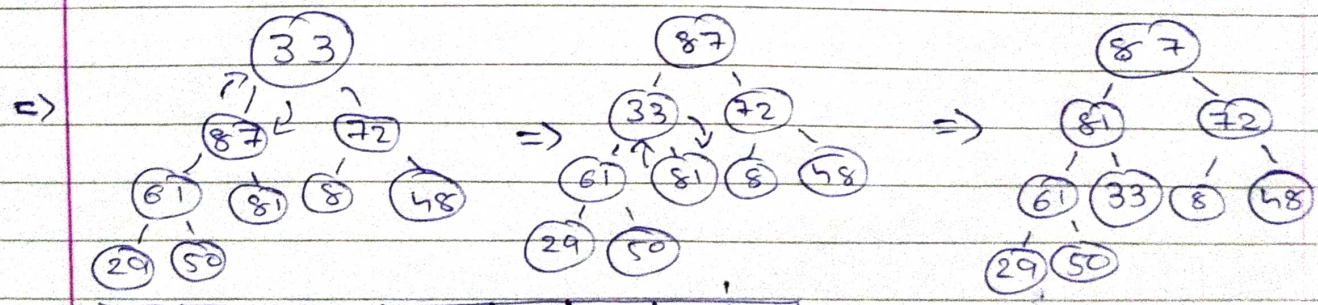
Ans Step-1 Building MAX HEAP



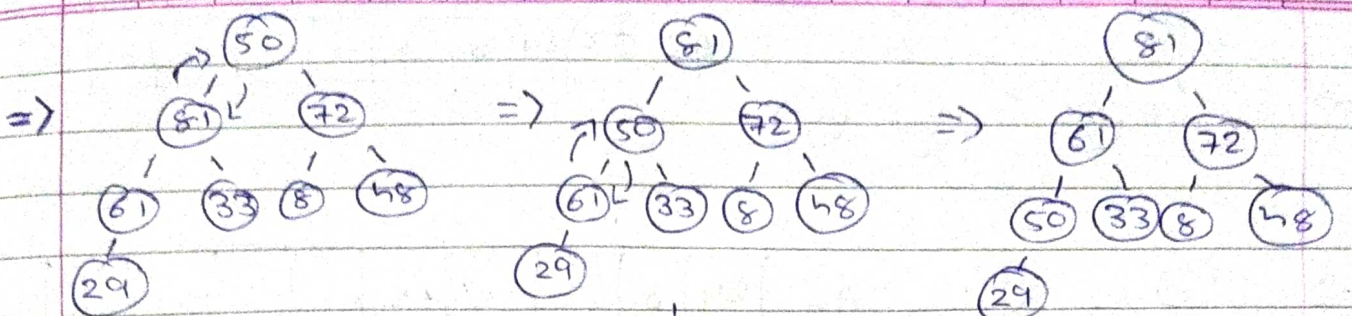


=> | 92 | 87 | 72 | 61 | 81 | 8 | 48 | 29 | 50 | 33 |

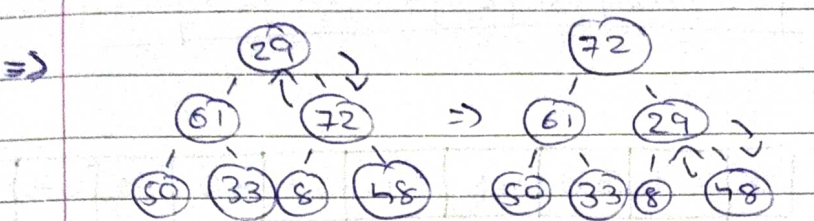
Step-2 Delete data from MAX HEAP



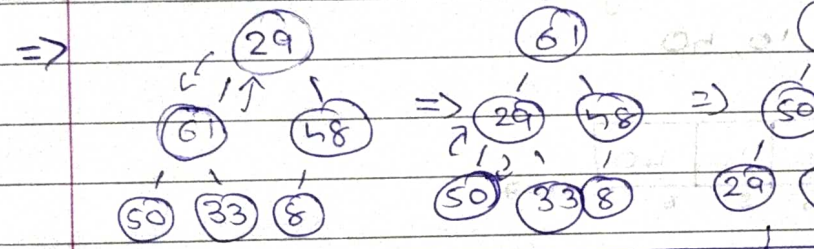
A = | 33 | 87 | 72 | 61 | 81 | 8 | 48 | 29 | 50 | 92 |



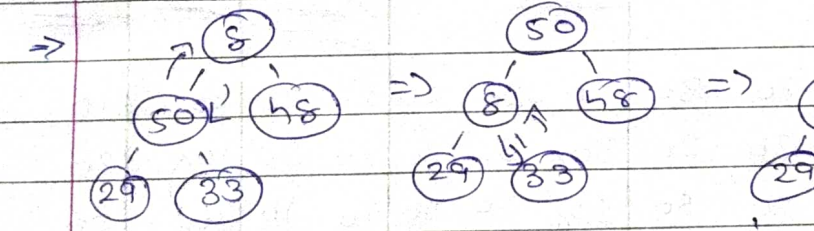
A = | 50 | 81 | 72 | 61 | 33 | 8 | 58 | 29 | 87 | 92 |



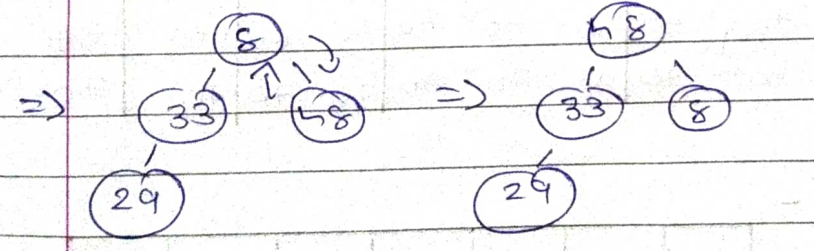
A = | 29 | 61 | 72 | 50 | 33 | 8 | 58 | 81 | 87 | 92 | => | 72 | 61 | 58 | 50 | 33 | 8 | 29 | 81 | 87 | 92 |



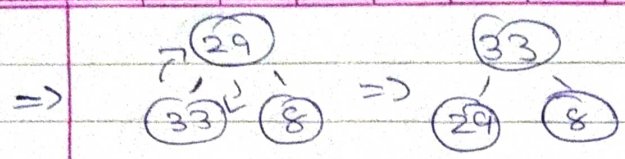
A = | 61 | 50 | 58 | 29 | 33 | 8 | 72 | 81 | 87 | 92 |



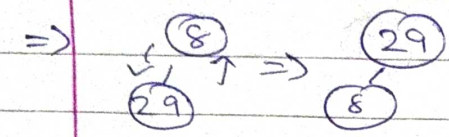
A = | 50 | 33 | 58 | 29 | 8 | 61 | 72 | 81 | 87 | 92 |



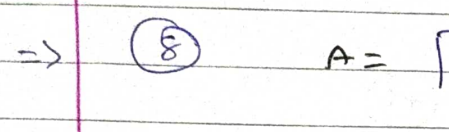
A = | 58 | 33 | 8 | 29 | 50 | 61 | 72 | 81 | 87 | 92 |



A= | 33 | 29 | 8 | 48 | 50 | 61 | 72 | 81 | 87 | 92 |



A= | 29 | 8 | 33 | 48 | 50 | 61 | 72 | 81 | 87 | 92 |



A= | 8 | 29 | 33 | 48 | 50 | 61 | 72 | 81 | 87 | 92 |

- Sorted Array :- | 8 | 29 | 33 | 48 | 50 | 61 | 72 | 81 | 87 | 92 |

Q.5) Using Bubble sort algorithm sort the array.
70, 30, 20, 50, 60, 10, 40

Ans

70	30	20	50	60	10	40
0	1	2	3	4	5	6

n=7

Pass:1							Pass:2			Pass:3		Pass:4	Pass:5
70	30	30	30	30	30	30	20	20	20	20	20	20	10
30	70	20	20	20	20	20	30	30	30	30	30	10	20
20	20	70	50	50	50	50	50	50	50	10	10	30	30
50	50	50	70	60	60	60	60	10	10	50	40	40	40
60	60	60	60	70	10	10	10	60	40	40	50	50	50
10	10	10	10	10	70	40	40	40	60	60	60	60	60
40	40	40	40	40	40	70	70	70	70	70	70	70	70

=> Sorted Array :-

10	20	30	40	50	60	70