Probabilistic Analysis :-

1) Probablistic Analysis is the use of probability in the Analysis of problems.

2) In order to perform a probabilistic analysis we must use knowledge of (or) make assumptions about, the distribution of inputs, then we analyse our algorithm computing an average case running time where we take the average over the distribution of the possible inputs.

Probability Theory :-

1) Probability Theory has the goal of characterizing the outcomes of natural or conceptual experiments.

2) Ex :- tossing a coin ten times

Rolling a die 3 times

playing a lottery

Picking a ball from bag containing white and Red balls and so on.

3) Each possible outcome of an experiment is called a Sample point and the set of all possible outcome is known as Sample Space.

4) In this text, we assume that capital S is finite.

5) An event e is a subset of Sample space S.

6) If the Simple space Consists of small n Sample points then there are 2^n possible events.

Definition: The probability of an event e is defined to

$\frac{|E|}{|S|}$ where S is sample space.

* Then the indicator random variable $I\{A\}$ associated with event a is defined as

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{if } A \text{ does not occur.} \end{cases}$$

The probability of Event E is denoted as $prob\{E\}$

Input :-

Ex :- Hiring Assistant

A sequence of n candidates for a position

Each has a distinct quality rating that can determined in an interview.

```
HIRE - ASSISTANT (n)
best = 0
For i = 1 to n
        interview Candidate i
        if Candidate i is better than Candidate best.
                best = i
                hire Candidate i
```

Cost :-

c_I , Cost of Interviewing (Low cost)

c_H , Cost of Hiring (High cost, i.e assume $c_H > c_I$)

n number of candidates to be interviewed

m number of people hired.

Total cost is $\boxed{O(n c_I + m c_H)}$

Probabilistic Analysis :-

1) Analysis of algorithm based on probability

2) An algorithm is said to be Randomized in behaviour of the output produced is dependent on the Random generator.

Problem statement :- Hiring problem.

Scenario :- Manager hiring an assistant with the help of agency.

Strategy for selection :-

Assume

1) Agency sends a Candidate per day.

2) Hire the new Candidate of he is better than (Fire) Current assistant.

Algorithm:-

```
Hire(N) - N. no. of candidates
{
    best := 0  // dummy candidate.
    for (i=1 to N)
    {
        if (candidate i better than existing candidate
            then best)
        {
            best = i;
            Hire candidate i (ci)
        }
    }
}
```

Objective:- Reduce the cost

$$\text{Total cost} = O(nc_i + mc_h)$$

c_i - Associate with interview people

c_h - Associate with hiring people

m = Candidates hired.

n = Candidates interviewed.

TPC

Worst Case:-

Agency is sending candidate in increasing order of quality.

Ex:-

C_i	1	2	3	4	5
Rank i	1	3	7	10	15
(Quality of candidates)	✓	✓	✓	✓	✓
	x	x	x	x	

here

Total cost = $n c_h + n c_i$

↑ To reduce this cost.

Problem:- No control over the order in which candidates are sent for interview.

Solution:-

1) Randomize the input distribution.
2) Apply probabilistic Analysis.

Assumption:-

Candidates will appear in random order.

1) Agency will only provide the list of candidates. Manager will call the candidate in random order. when input is randomized

Actual running time = Expected running time.

Indicator Random Variable (IRV):-

1) It is a method to convert probabilities to expectations and vice versa.
2) Indicator variable, $I\{A\}$

$$I\{A\} = \begin{cases} 1 & \text{If event } A \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

Ex:- Coin toss

$$X_h = I\{H\} = \begin{cases} 1 & \text{if } H \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

expectation

H = occurence of head

In a flip, we know that

$$E(X_h) = E(I\{A\})$$

$$= 1 \cdot \text{prob}\{H\} + 0 \cdot \text{prob}\{T\}$$

$$= 1 * \frac{1}{2} + 0 * \frac{1}{2}$$

$$E(X_h) = \frac{1}{2}$$

Lemma:-

Given a Sample space and event A

let $X_A = I\{A\}$

$$\boxed{E(X_A) = \text{Prob}\{A\}}$$

For n Flips:-

According to linearity of expectation:

$$X = \sum_{i=1}^{n} X_i$$

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right)$$

$$= \sum_{i=1}^{n} E(X_i)$$

$$= \sum_{i=1}^{n} \frac{1}{2}$$

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

$E(x) = \frac{n}{2}$

Analysing hiring problem ~ IRV :-

X - Random variable (no of times we hire a new assistant)

$$X = \sum X_{i\,(IRV)} \qquad i = 1, 2, 3, \ldots\ldots n$$

$$X_{i\,(IRV)} = \begin{cases} 1 & \text{if } i^{th} \text{ candidate hired} \\ 0 & \text{otherwise} \end{cases}$$

E - Expectancy

$E(X_i) = \text{prob}\{\text{Candidate } i \text{ hired}\}$

i^{th} candidate is hired if and only if c_i is better than c_1 to c_{i-1}.

$\text{Prob}\{\text{candidate } i \text{ hired}\} = \frac{1}{i}$

$E(X_i) = \text{prob}\{c_i \text{ hired}\} = \frac{1}{i}$

$\frac{1}{i}$ - nth harmonic

$$E(X) = \sum \left(\sum_{i=1}^{n} X_i \right)$$

$$= \sum_{i=1}^{n} E(X_i)$$

$$= \sum_{i=1}^{n} \frac{1}{i}$$

$$\boxed{E(X) = \log n + O(1)}$$

reduced n to log n.

For positive integers is harmonic series.

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = \log n + O(1)$$

is we need to hire $\log n$ people (out of n people who are interviewed)

$$\text{Total cost} = \log n \, C_h + {}^{n}C_i$$

Amortized Analysis:

* For finding sequence of operations.
* To find the total cost.

Description:-

* It is one of the strategy analysing a sequence of operations to show that an average cost for operation is small, eventhough a single operation within the sequence might be expensive.
* For sequence of 'n' operations the cost is

$$\frac{\text{Cost (n operations)}}{n} = \text{Cost(normal operations)} + \frac{\text{cost (Expensive operations)}}{n}$$

* Some operations may have lowest cost, some operations may have highest cost to perform, to analyse the sequence of n operations we have to use one strategy called Amortized Complexity.

* This Analysis shows that average cost per operation is small eventough a single operation in sequence might be expensive.

* The operation with highest cost is called worst case operations. The basic idea is that worst case operation can alter the state such a way that worst case operations cannot occur again.

* There are generally three methods used for performing Amortized Analysis.

(i) Aggregate method. Avg. of all Alg $\frac{T(n)}{n}$

(ii) Accountability method. — early (low), later (high)

(iii) Potential method. — Time, Space

Aggregate Method :-

1. The method determines the upper bound T(n) in file, total cost of a sequence of operations then calculate the Amortised of each operation to be $\frac{T(n)}{n}$.

Ex: 1 2 3 4 5

$$\frac{1+2+3+4+5}{5} = 3$$

Accountability Method:

* It is one of the form of aggregate method. In this method we have to assign amortized cost for each and every operation, for n number of operation such that the Amortized cost different from actual cost.
* There are two operations in Accountability method
  (i) Earlier operation (lowest cost Ex: 1)
  (ii) Later operation. (highest cost Ex: 5)
* Cost of Earlier operation is less than the cost of later operation.

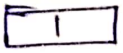
Potential Method:

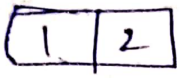
* Potential method is used to analyse the Time and space Complexities.

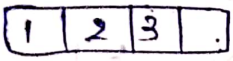
Ex: Dynamic Table.

Initially, the table is empty and size is zero.

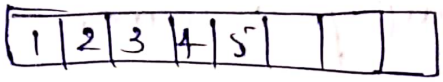
Everytime double the size of the table.

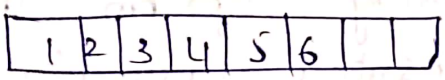
insert item 1 

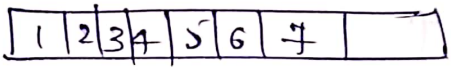
insert item 2 

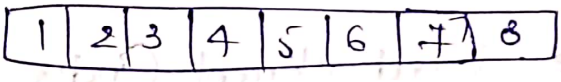
insert item 3 

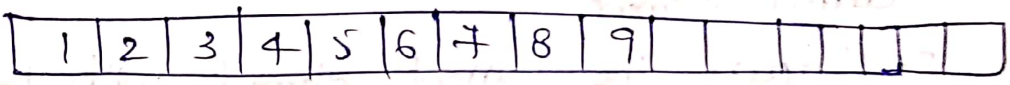
insert item 4 

insert item 5 

insert item 6 

insert item 7 

insert item 8 

insert item 9 

* Overflow would happen when we insert 9 table size is 16.

Uses:-

1. Useful for designing efficient algorithm for datastructs such as dynamic ~~arra~~ arrays, priority queues, Splay trees, disjoint set data structures etc ...

2. It provides a guarantee that the average case time Complexity of an operation is constant even if some operations may be expensive.

Insertion Sort :-

* Insertion sort is a very simple method to sort numbers in an ascending or descending order.
* This method follows the incremental method, It can be compared with the technique how cards are sorted at the time of playing a game.
* Insertion sort is sorting Algorithm that works by inserting each element in an unsorted array into its correct position in a sorted sub-array.

Methodology :-

1. Start with the second element (index 1) and consider it the current element.
2. Compare the current element with the elements to its left in the sorted position of the array.
3. Move the elements to the right until you find the correct position for the current element.
4. Repeat steps 1-3 for all elements in the array, one at a time.
5. The array is sorted when you have processed all elements.

Algorithm :-

Step 1: If it is the first element, it is already sorted. return 1
Step 2: Pick next element
Step 3: Compare with all elements in the sorted sublist.
Step 4: Shift all the elements in the sorted sublist. That is greate than the value to be sorted.
Step 5: Insert the value.
Step 6: Repeat until list is sorted.

Pseudocode :-

1. for j = 2 to A.length do
2. key = A[j]
3. //insert A[j] into the sorted sequence A[1 .... j-1]
4. i = j-1
5. while i > 0 and A[i] > key
6. A[i+1] = A[i]
7. i = i-1
8. A[i+1] = key.

Ex :-

A = (41, 22, 63, 14, 55, 36)

initially

41	22	63	14	55	36

1st iteration, key = 22, Compare a_1 wit a_0

41	22	63	14	55	36

Since $a_0 > a_1$, Swap

22	41	63	14	55	36

2nd iteration, set key = 63, Compare a_2 with a_1 & a_0

22	41	63	14	55	36

Since $a_1 > a_0$, keep array as it is :

22	41	63	14	55	36

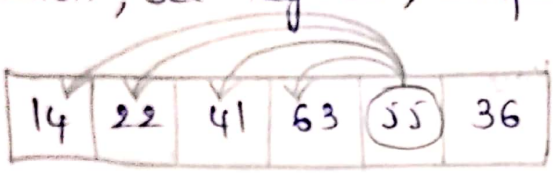
3rd iteration : set key = 14, Compare a_3 with a_2, a_1 & a_0.

22	41	63	14	55	36

Since a_3 is the small among all elements on the left hand side, place as at the beginning of the array

14	22	41	63	55	36

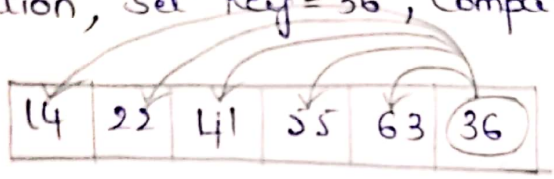
4th iteration, set key = 55, Compare a_4 with a_3, a_2, a_1, and a_0



As $a_4 < a_3$; swap both of them

14	22	41	55	63	36

5th iteration, set key = 36, Compare a_5 with a_4, a_3, a_2, a_1 & a_0



Since $a_5 < a_2$, so will place the elements in their correct positions.

14	22	36	41	55	63

Hence the array is arranged in ascending order, so no more swapping is required.

Complexity Analysis:-

Input: n input elements

Output: Number of steps incurred to sort a list

logic: If we are given n elements, then in the first pass, it will make n-1 comparisions, in the second pass, it will do n-2, in the third pass, it will do n-3 and so on

Thus, Total no of comparisions can be output:

$(n-1) + (n-2) + (n-3) + (n-4) + \cdots + 1$

i.e $O(n^2)$

Time Complexity	$O(n^2)$
Space Complexity	$O(1)$

Time Complexities :-

Best case = $O(n)$

Average case = $O(n^2)$

worst case = $O(n^2)$

Applications :-

1) When array contains only a few elements.
2) When there exist few elements to insert.

Advantages :-

* Simple to Implement.
* Efficient on small data sets.
* Stable.
* It is in-place (requires constant amount $O(1)$)

Heap Sort :-

Heap sort is a popular and efficient sorting algorithm. The concept of heap sort is to eliminate the elements one by one from the heap part of the list and then insert them into the sorted part of the list.

→ HeapSort is the in-place sorting algorithm.

Methodology :-

1) First convert the array into heap data structure using heapify.
2) Then one by one delete the root node of the Max-heap and replace it with the last node in the heap.
3) And then heapify the root of the heap.
4) Repeat this process until size of heap is greater than
5) Build a heap from the given input array.

Algorithm :-

1) Build a heap from the given input array.
2) Swap the root element of the heap with the last element of the heap.
3) Remove the last element of the heap. (which is now in the correct position)
4) Heapify the remaining elements of the heap.
5) Repeat 2, 3, 4 steps until the heap contains only one elem

Pseudo code:-

Algorithm : Heapsort (A)

BUILD-MAX-HEAP(A)

for i = A.length downto 2 do

exchange A[1] with A[i]

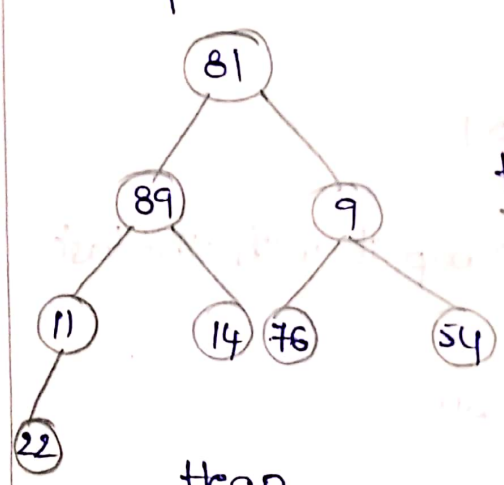
A.heap-size = A.heap-size-1

MAX-HEAPIFY (A,1)

Ex:-

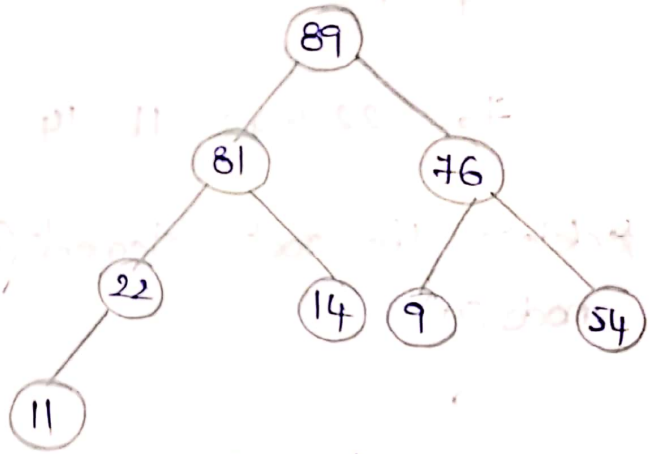
81	89	9	11	14	76	54	22

→ Construct a heap from array and convert it into max heap.



Heap

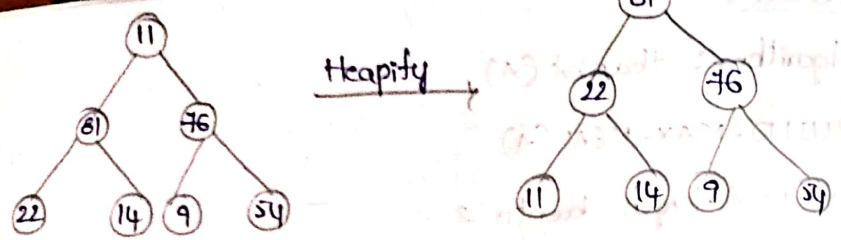
Heapify →



Max heap

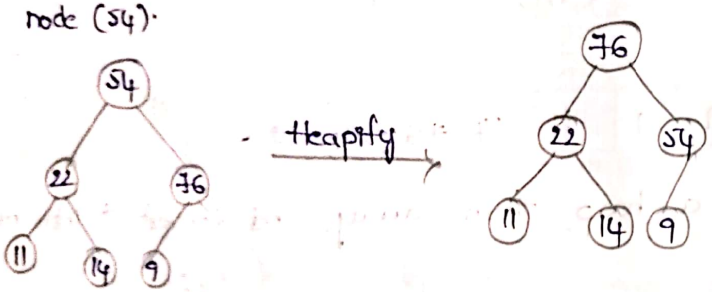
89	81	76	22	14	9	54	11

→ delete the root element (89) and swap it with the last node (11)



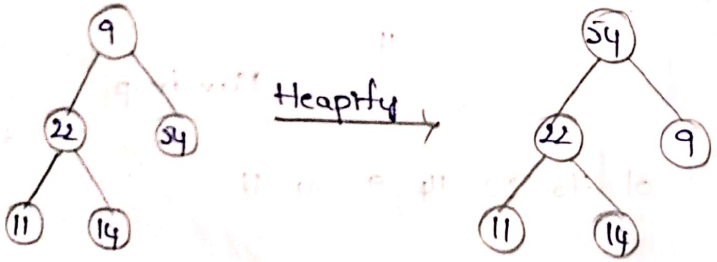
81	22	76	11	14	9	54	89

→ delete the root element (81) and swap it with the last node (54).



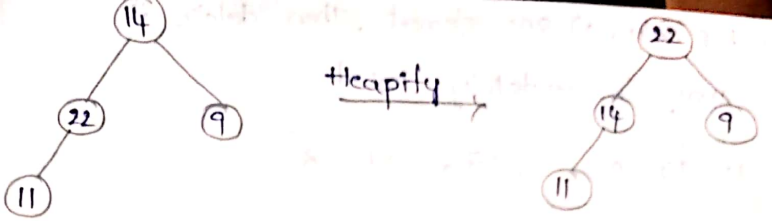
76	22	54	11	14	9	81	89

→ delete the root element (76) and swap it with the last node (9)



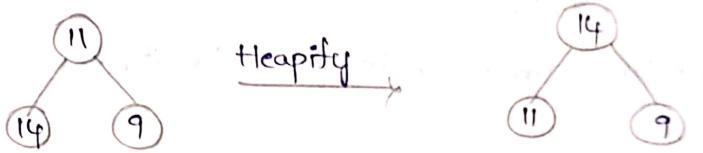
54	22	9	11	14	76	81	89

→ Delete the root element (54) and swap it with last node(14)



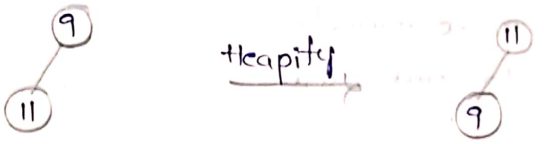
22	14	9	11	54	76	81	89

→ Delete the root node (22) and swap it with the last node(11



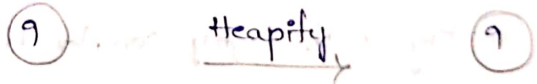
14	11	9	22	54	76	81	89

→ Delete the root element (14) and swap it with last element(



11	9	14	22	54	76	81	89

→ Delete the root element (11) and swap it with last element(



9	11	14	22	54	76	81	89

→ Heap has only (9) one element, then delete it.

The array is completely Sorted

9	11	14	22	54	76	81	89

Complexity Analysis:-

Time Complexity: $O(N \log N)$

Auxiliary Space:- $O(\log n)$, due to the recursive call Stack.

However, auxiliary space can be $O(1)$ for iterative implementation.

Advantages:-

1) Efficient Time Complexity

Time Complexity = $O(n \log n)$

2) Memory usage can be minimal.

3) Simplicity - Simple to understand.

Dis-advantages:-

1) Costly

2) Unstable

3) Efficient - not very efficient

- when writing with highly complex data.

Applications:-

1) When the Smallest (shortest) or highest (longest) value is needed instantly.

2) For finding the order in statistics.