Problem 1: Birthday Paradox

```
To design a program that can test the Birthday problem, by a series of experiments, on randomly generated birthdays which test t
his paradox for n = 5,10,15,20,25,30...200.
```

Step 1: Theoritical probability of at least two of the n persons having the same birthday can be calculated as follows:

In [1]:
```
def ProbabilityTheoritical(n):
    p= 1 - Factorial(365)/((365**n)*Factorial(365-n))
    """Reference: wikipedia"""
    return p

def Factorial(m):
    fac=1
    for i in range(1,m+1):
        fac= fac*i
    return fac
```

Step 2: Experimental probability for n person. Taking n persons over and over again and check the probability

In [2]:
```
import random

def ProbabilityExperimental(n):
    yescount=0
    lst=[]
    """To calculate experimental probability we need to repeat the experiment over and over again."""
    """so lets repeat it for 10,000 times for each value of n"""
    for i in range(10000):
        for j in range(n):
            """First lets generate n random birthdays.
            Each no. from 1 to 365 will represent a birthday, so we can select n random nos. from 1 to 365"""
            lst.append(random.randint(1,366))

        """Now check is their any pair with same birthday"""
        if(len(set(lst))<len(lst)):
            yescount +=1
        lst.clear()

    p=yescount/10000
    return p
```

Step 3: Now lets calculate the theoritical and experimental value of the probabibility of at least two of the n persons having the same birthday, taking n= 5,10,15,20,25,30...200.

In [3]:
```
y1=[]
y2=[]

falsecount=0
truecount=0

for n in range(5,205,5):
    a=ProbabilityTheoritical(n)
    b=ProbabilityExperimental(n)

    # y-axis for plotting the probability vs number of people graph
    y1.append(a)
    y2.append(b)

    # comparing theoritical and experimental probabilties
    if a-b<0.02:
        print("n = {}".format(n).ljust(7), "True".rjust(8),"     Theoritical prob ≈ {}".format(a).ljust(48),"Experimental prob ≈ {}".format(b))
        truecount+=1

    else:
        print("n = {}".format(n).ljust(7), "False".rjust(8),"     Theoritical prob ≈ {}".format(a).ljust(48),"Experimental prob ≈ {}".format(b))
        falsecount+=1


print("\nTrue count:",truecount)
print("False count:",falsecount)

if (falsecount==0):
    print("Since the false count is 0, so the Birthday Paradox is true.")

elif (falsecount<=5):
    print("Since the false count is so low, so the Birthday Paradox is true.")

else:
    print("Since the false count is more than 5, so the Birthday Paradox is false.\n")
```

```
n = 5       True      Theoritical prob ≈ 0.02713557369979358       Experimental prob ≈ 0.0285
n = 10      True      Theoritical prob ≈ 0.11694817771107768       Experimental prob ≈ 0.1121
n = 15      True      Theoritical prob ≈ 0.25290131976368635       Experimental prob ≈ 0.2488
n = 20      True      Theoritical prob ≈ 0.41143838358057994       Experimental prob ≈ 0.415
n = 25      True      Theoritical prob ≈ 0.5686997039694639        Experimental prob ≈ 0.5729
n = 30      True      Theoritical prob ≈ 0.7063162427192686        Experimental prob ≈ 0.7072
n = 35      True      Theoritical prob ≈ 0.8143832388747152        Experimental prob ≈ 0.8112
n = 40      True      Theoritical prob ≈ 0.891231809817949         Experimental prob ≈ 0.8939
n = 45      True      Theoritical prob ≈ 0.940975899465775         Experimental prob ≈ 0.9405
```

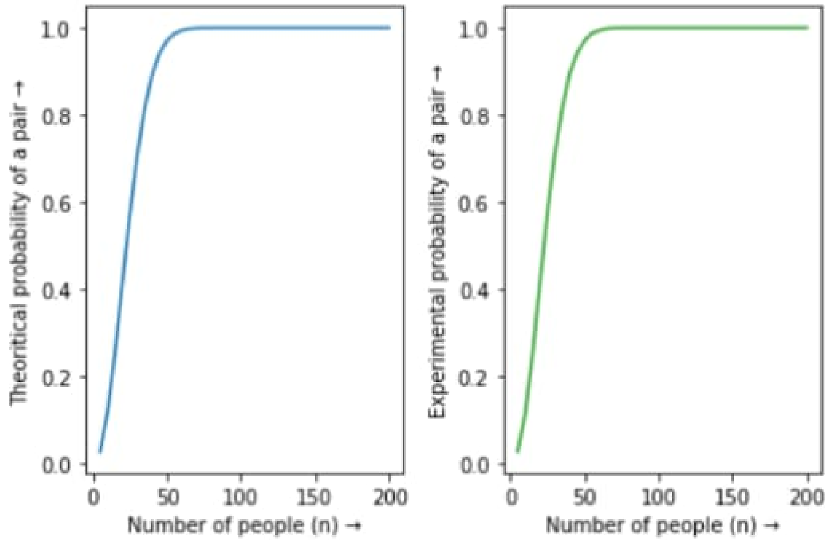
```
n = 50      True      Theoritical prob ≈ 0.9703735795779884       Experimental prob ≈ 0.9695
n = 55      True      Theoritical prob ≈ 0.9862622888164461       Experimental prob ≈ 0.9873
n = 60      True      Theoritical prob ≈ 0.994122660865348        Experimental prob ≈ 0.9939
n = 65      True      Theoritical prob ≈ 0.9976831073124921       Experimental prob ≈ 0.9981
n = 70      True      Theoritical prob ≈ 0.9991595759651571       Experimental prob ≈ 0.9996
n = 75      True      Theoritical prob ≈ 0.9997198781738114       Experimental prob ≈ 0.9998
n = 80      True      Theoritical prob ≈ 0.9999143319493135       Experimental prob ≈ 1.0
n = 85      True      Theoritical prob ≈ 0.9999759973260097       Experimental prob ≈ 1.0
n = 90      True      Theoritical prob ≈ 0.9999938483561236       Experimental prob ≈ 1.0
n = 95      True      Theoritical prob ≈ 0.9999985601708488       Experimental prob ≈ 1.0
n = 100     True      Theoritical prob ≈ 0.9999996927510721       Experimental prob ≈ 1.0
n = 105     True      Theoritical prob ≈ 0.9999999403276142       Experimental prob ≈ 1.0
n = 110     True      Theoritical prob ≈ 0.9999999894712943       Experimental prob ≈ 1.0
n = 115     True      Theoritical prob ≈ 0.9999999983154677       Experimental prob ≈ 1.0
n = 120     True      Theoritical prob ≈ 0.9999999997560852       Experimental prob ≈ 1.0
n = 125     True      Theoritical prob ≈ 0.9999999999681016       Experimental prob ≈ 1.0
n = 130     True      Theoritical prob ≈ 0.9999999999962403       Experimental prob ≈ 1.0
n = 135     True      Theoritical prob ≈ 0.9999999999996015       Experimental prob ≈ 1.0
n = 140     True      Theoritical prob ≈ 0.9999999999999621       Experimental prob ≈ 1.0
n = 145     True      Theoritical prob ≈ 0.9999999999999968       Experimental prob ≈ 1.0
n = 150     True      Theoritical prob ≈ 0.9999999999999998       Experimental prob ≈ 1.0
n = 155     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 160     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 165     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 170     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 175     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 180     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 185     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 190     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 195     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0
n = 200     True      Theoritical prob ≈ 1.0                      Experimental prob ≈ 1.0

True count: 40
False count: 0
Since the false count is 0, so the Birthday Paradox is true.
```

Note:

```
# True: Theoritical probability ≈ Experimental probabilbilty
# False: Theoritical probability ≠ Experimental probabilbilty
# These are approximate values of the probabilities, since the probability will be exact 1 only when no. of persons n >= 367

"""
From above values we may conclude that:
- If there are 23 person in a room, the probabilty that atleast 2 of them have same birthday is 50%
- If there are only 70 person in a room, the probabilty that atleast 2 of them have same birthday is 99.9%

These conclusions are based on the assumption that each day of the year (excluding February 29) is equally probable
for a birthday."""
```

Step 4: Plotting (Probability of a pair sharing a birthday) vs (Number of people):

In [4]:
```
import matplotlib.pyplot as plt

# x-axis scale
x=[]
for i in range(5,205, 5):
    x.append(i)


# plotting the theoritical probability of at least two people sharing a birthday vs the number of people
plt.subplot(1, 2, 1)
plt.plot(x, y1)
plt.xlabel('Number of people (n) →')
plt.ylabel('Theoritical probability of a pair →')


# plotting the experimental probability of at least two people sharing a birthday vs the number of people
plt.subplot(1, 2, 2)
plt.plot(x, y2, color ='tab:green')
plt.xlabel('Number of people (n) →')
plt.ylabel('Experimental probability of a pair →')


#printing the therotical and experimental plots side by side
plt.tight_layout()
plt.show()
```



Name - Yogesh Agarwala
EE19B130

EE4371 - ENDSEM

(2) Solution :

$4n\log n + 2n \longrightarrow O(n\log n)$

$3n + 100\log n \longrightarrow O(n)$

$n^2 + 10n \longrightarrow O(n^2)$

$2^{10} \longrightarrow O(1)$

$4n \longrightarrow O(n)$

$n^3 \longrightarrow O(n^3)$

$2^n \longrightarrow O(2^n)$

$n\log n \longrightarrow O(n\log n)$

$2^{\log n} = n^{\log 2}$
- Case 1: base of log is 2 $= O(n)$
- Case 2: base of log > 2 $= O(n^{\log 2})$

Case 1: base of log is 2

Now, $O(1) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$

$\boxed{2^{10}} < \boxed{2^{\log n} = 3n + 100\log n = 4n} < \boxed{n\log n = 4n\log n + 2n} < \boxed{n^2 + 10n} < \boxed{n^3} < \boxed{2^n}$

though
$2^{\log n} < 3n + 100\log n < 4n$
as n approaches ∞
but their asymptotic growth rate is same.

Case 2: base of log > 2

Now, $O(1) < O(n^{\log 2}) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$

$\boxed{2^{10}} < \boxed{2^{\log n}} < \boxed{3n + 100\log n = 4n} < \boxed{n\log n = 4n\log n + 2n} < \boxed{n^2 + 10n} < \boxed{n^3} < \boxed{2^n}$

③ Show that the running time of the merge-sort algorithm on n-element sequence is $O(n \log n)$, even when n is not a power of 2

Algorithm :

ⓐ Input : Array $A[1 \ldots n]$

30	10	18	3	2	16	50

ⓑ Divide into subarrays $A[1 \ldots m]$ and $A[m+1, \ldots n]$ where $m = \lceil n/2 \rceil$

30	10	18	3

2	16	50

ⓒ Recursively Mergesort $A[1 \ldots m]$ and $A[m+1, \ldots n]$

3	10	18	30

2	16	50

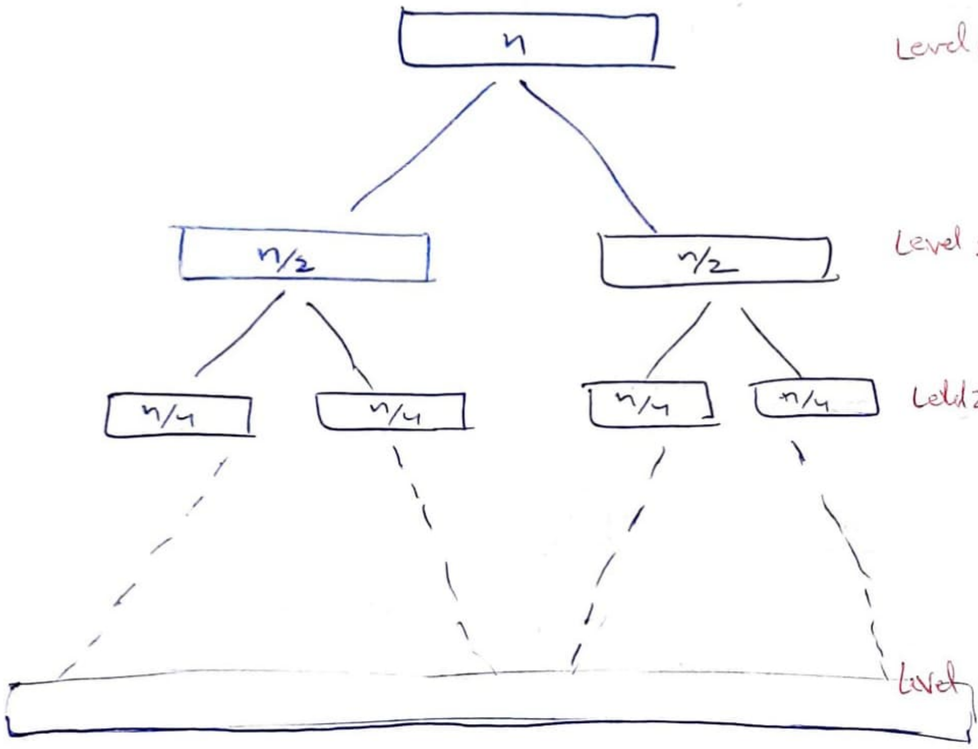
ⓓ Merge the sorted arrays

2	3	10	16	18	30	50

Time-complexity: (valid for both case when n is power of 2 or not)

$$T(n) = \underbrace{T(\lfloor n/2 \rfloor)}_{\text{sorting left-part}} + \underbrace{T(\lceil n/2 \rceil)}_{\text{sorting right-part}} + \underbrace{cn}_{\text{merging}}$$

When n is a power of 2:



Level 0:

Level 1: No. of operation for merging = cn

Level 2: no. of operation for merging = cn

Level $\log_2 n$: No. of operation for merging = cn

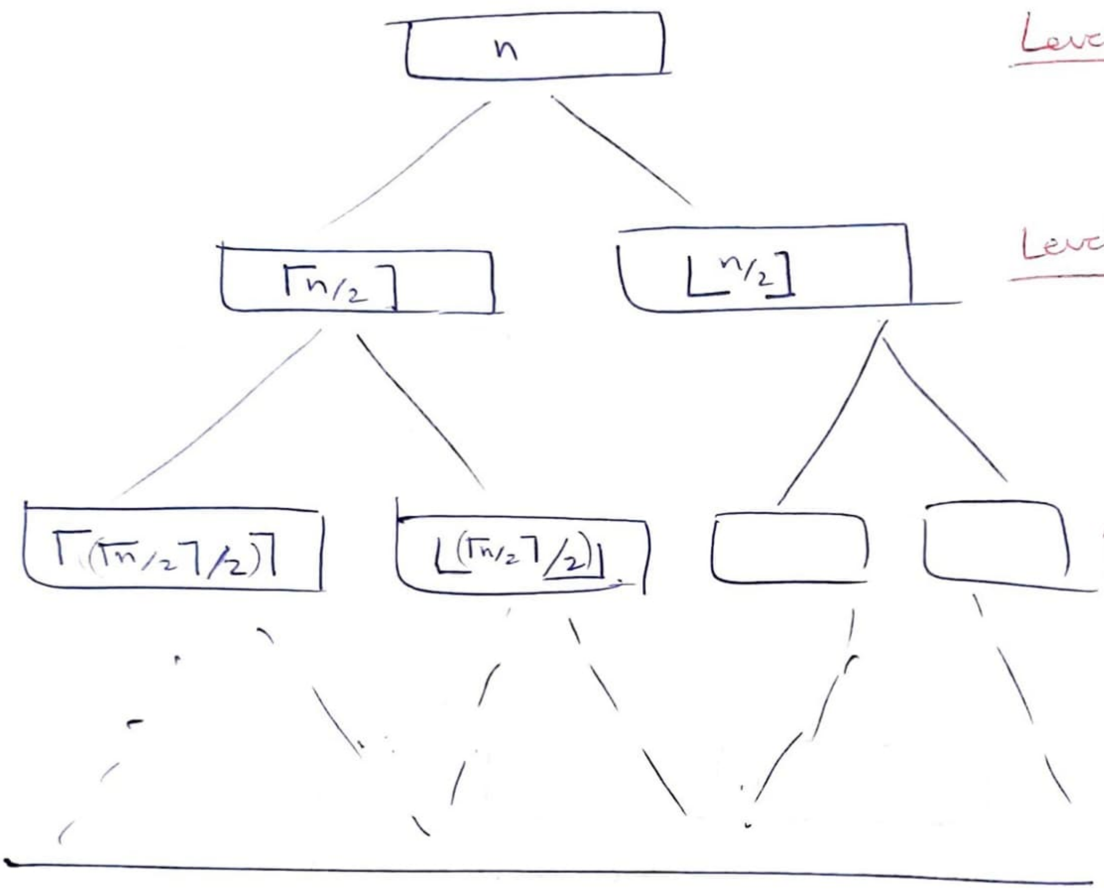
So no. of levels $= \log_2 n$

Work at each level (i.e no. of operations for merging the sorted arrays) $= cn$

$\therefore$ Total work $= (cn)(\log_2 n)$

$= O(n \log n)$

When n is not a power of 2 :



Level 0:

Level 1 : no. of operation for merging $= cn$

Level 2 : no. of operation for merging $= cn$

Level K : no. of operation for merging $= cn$

Here no. of levels $(K) = \lceil \log_2 n \rceil$

Work at each level is same as previous case $= cn$

(Since $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$)

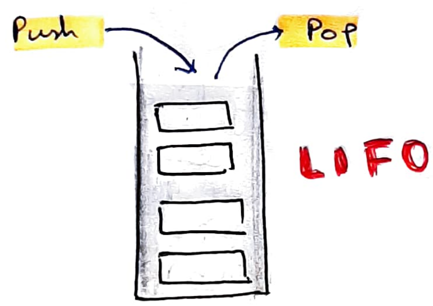
$\therefore$ Total work $= (cn)(\lceil \log_2 n \rceil)$

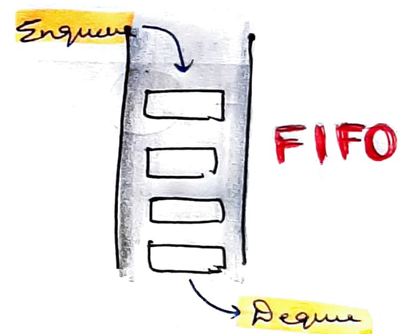
$= O(n \log_2 n)$

(4) Show how to implement a stack using two queues. Analyse the running time of the stack operations.

Concept: To construct a stack using two queues we need to simulate the stack operations by using queue operations

- Stack operations
  - Push (insert at top)
  - Pop (remove from top)
- queue operations
  - Enqueue (insert at rear)
  - Dequeue (remove from front)

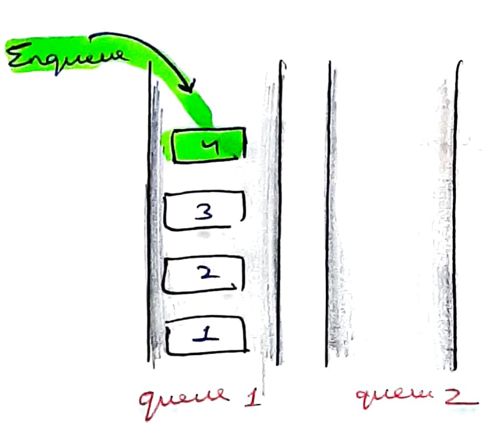


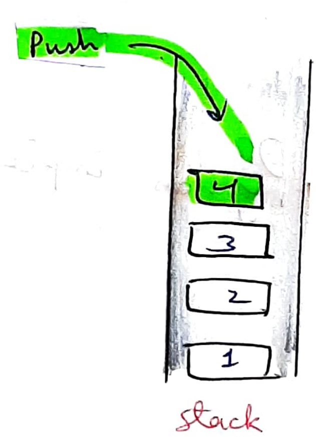


Implementation:

(1) Push operation : O(1)

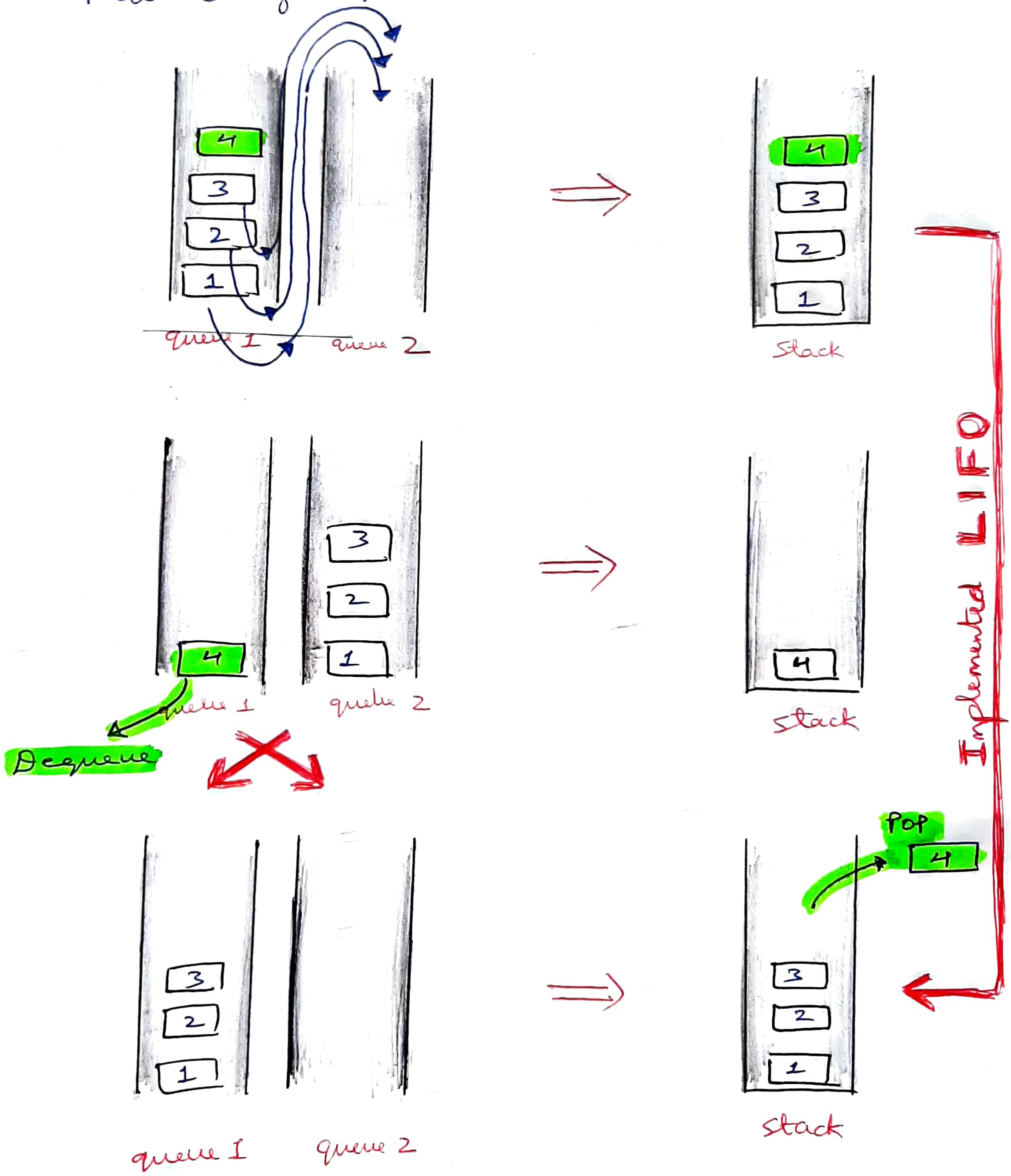
Push operation will be same as the enqueue operation, so push can be implemented in O(1) time





(II) Pop operation : $O(n)$

To implement pop, we dequeue each element from queue 1 and place it in queue 2, but stop before the last element. Then return the single element left in the original queue 1. Now change queue 2 to queue 1



Running time:

(I) Push operation is $O(1)$

(II) Pop operation is $O(n)$ since need to dequeue each element (except the last one) from queue 1 to queue 2.

⑤ You are given an array of n-elements, and you notice that some of the elements are duplicates, that is, they appear more than once in array. Show how to remove all duplicates from the array in time $O(n \log n)$

Step - ① Sort the array using mergesort/quicksort

$$\text{Time} = O(n \log n)$$

Step - ⑪ Traverse the sorted array linearly to find the duplicates and remove them

$$\text{Time} = O(n)$$

Time complexity = merge sort + linear traversal of the sorted array

$$= O(n \log n) + O(n)$$

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

Python implementation

Python Implementation:

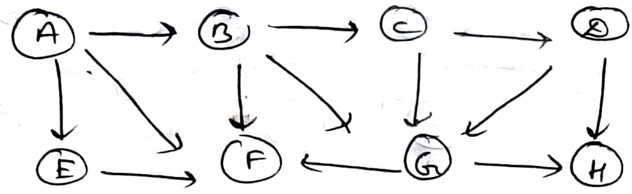
In [1]:
```python
"""Part 1: Sorting the array using mergesort"""
def merge_sort(A):
    n = len(A)

    # base-case of the recurssion
    if n==1:
        return A

    # sort the left and right halves of the array recursively
    mid = n//2
    L = merge_sort(A[:mid])
    R = merge_sort(A[mid:])
    merged_array = merge(L,R)
    return merged_array

def merge(L,R):
    i = 0
    j = 0
    answer = []

    # comparing the elements of the left and right sorted arrays,
and adding them to a new array (merged_array)
    while i<len(L) and j<len(R):
        if L[i]<=R[j]:
            answer.append(L[i])
            i += 1
        else:
            answer.append(R[j])
            j += 1

    # while comparing the left and right sorted arrays, it may ha
ppen that one of them is completly added to the merged_array
    # but in the other array still there are some elements left,
& since those are already sorted, add them to the merged_array
    if i<len(L):
            answer.extend(L[i:])
    if j<len(R):
            answer.extend(R[j:])

    # return the merged_array
    return answer
```

In [2]:
```python
"""Part 2: Removing duplicates from the sorted array"""
def remove_duplicates (sorted_array):
    arr = []
    i=0
    j=0
    n=len(sorted_array)

    while (i<len(sorted_array)):
        # copying elements to the output array from the sorted ar
ray
        arr.append(sorted_array[i])

        # don't copy the element to the output_array if it is a d
uplicate of the previous element
        # in while loop the condition i!=len(sorted_array)-1 is a
dded, to make sure
        # that sorted_array[i]==sorted_array[i+1] is not checked
for the last element
        while (i!=len(sorted_array)-1 and sorted_array[i] == sorte
d_array[i+1]):
            i+=1

        i+=1

    # sorted list with no duplicates
    return arr
```

In [3]:
```python
"""main function"""
input = [2,3,1,3,6,2,1,3]
sorted_array = merge_sort(input)
output = remove_duplicates(sorted_array)
print("Array without duplicates = ",output)
```

```
Array without duplicates =  [1, 2, 3, 6]
```

Q.6) Suppose Dijkstra's algorithm is run on the following graph, starting at node A.



ⓐ draw a table showing the intermediate values of all the nodes at each iteration of the algorithm.

ⓑ Show the final shortest path tree.

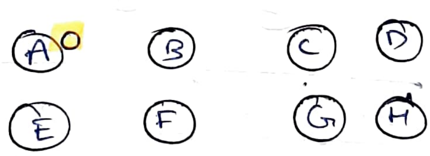
Solution:

Dijkstra's algorithm:

- Input: directed graph $G = (V, E)$
  where (i) each edge has a ~~neg~~ non-negative length
  (ii) source vertex

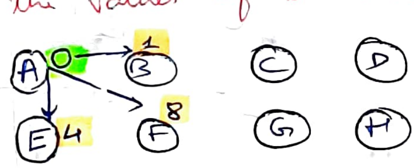
- output: for $v \in V$ compute $L(v)$ = length of shortest paths from source vertex to all other vertex.

Steps: (I) Starting node: (A) therefore → (I) initialize A with 0
→ (II) all other nodes with infinite value



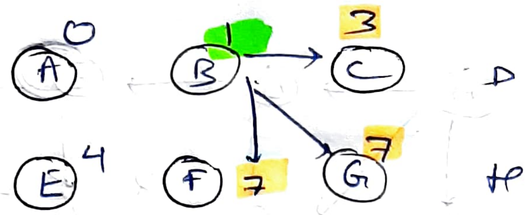
A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞

(II) Update the values of the nodes adjacent to node A



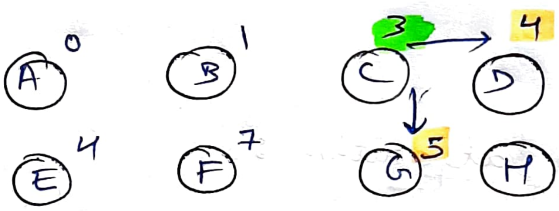
A	B	C	D	E	F	G	H
0	1	∞	∞	4	8	∞	∞

(III) Since minimum node is (B), so update its adj. nodes



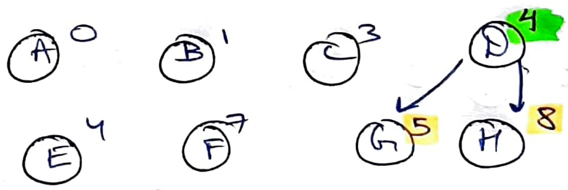
A	B	C	D	E	F	G	H
0	1	3	∞	∞	7	7	∞

(IV) Minimum node is (C)



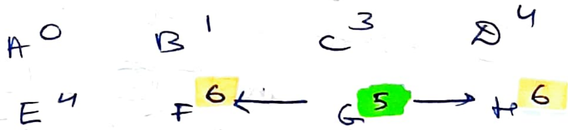
A	B	C	D	E	F	G	H
0	1	3	4	4	7	5	∞

(V) Minimum node is (D)



A	B	C	D	E	F	G	H
0	1	3	4	4	7	5	8

(VI) Minimum node is (G)



A	B	C	D	E	F	G	H
0	1	3	4	4	6	5	6

So, the table showing distance values at each iteration

Iteration	A	B	C	D	E	F	G	H
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	1	∞	∞	4	8	∞	∞
2	0	1	3	∞	4	7	7	∞
3	0	1	3	4	4	7	5	∞
4	0	1	3	4	4	7	5	8
5	0	1	3	4	4	7	5	8
6	0	1	3	4	4	6	5	6

(b) So, the final shortest - path tree is as follows