Department of Information and Communication Engineering

Pabna University of Science and Technology

Faculty of Engineering and Technology

B.Sc. (Engineering) 2nd Year 2nd Semester Examination-2019

Session: 2017-2018

Course Code: **ICE-2202** Course Title: **Data Structure and Algorithm Sessional**

Lab on Data structure and Algorithm Sessional

Submitted by Md Rokonujjaman

Roll: 180618

Registration No: 1065301

Dept. of Information and Communication Engineering

Pabna University of Science and Technology

Pabna-6600, Bangladesh

Email: rokonice180618pust@gmail.com

Mobile: 01953955939

Submitted to Md. Anwar Hossain

Assistant Professor

Dept. of Information and Communication Engineering

Pabna University of Science and Technology

Pabna-6600, Bangladesh

Email: manwar.ice@gmail.com

Index

1 Write a Program to insert and delete an element into a linear array

Illustration of logic:

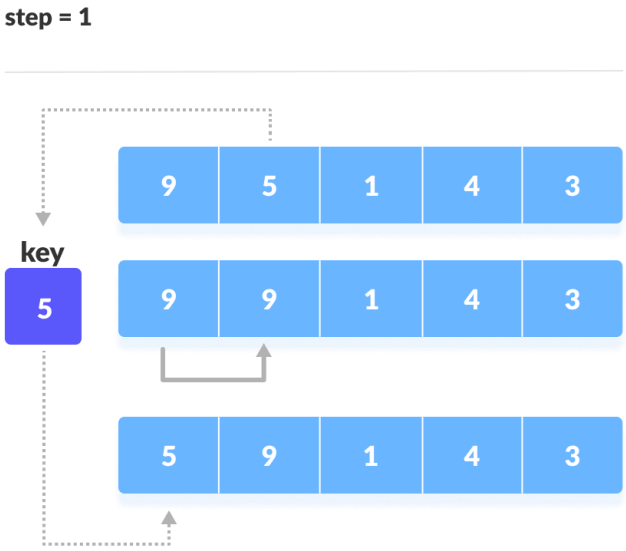
Insert:

Suppose we need to sort the following array.

Initial array

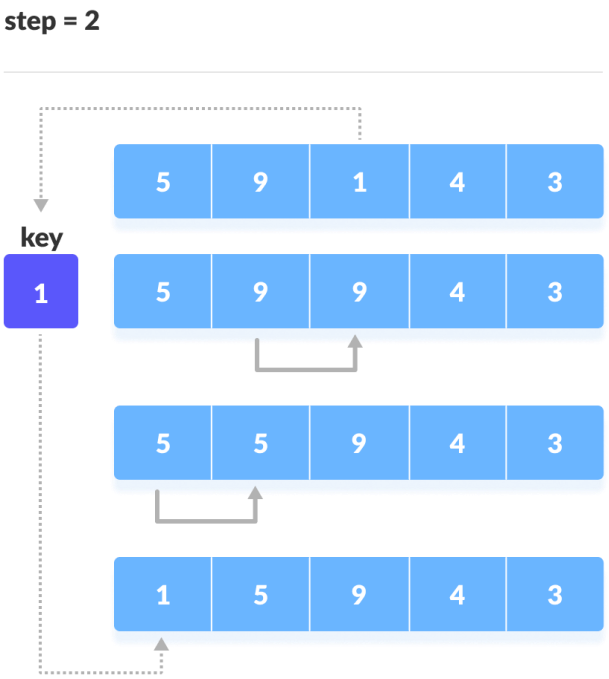
1. The first element in the array is assumed to be sorted. Take the second element and store it separately in key.

   Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.



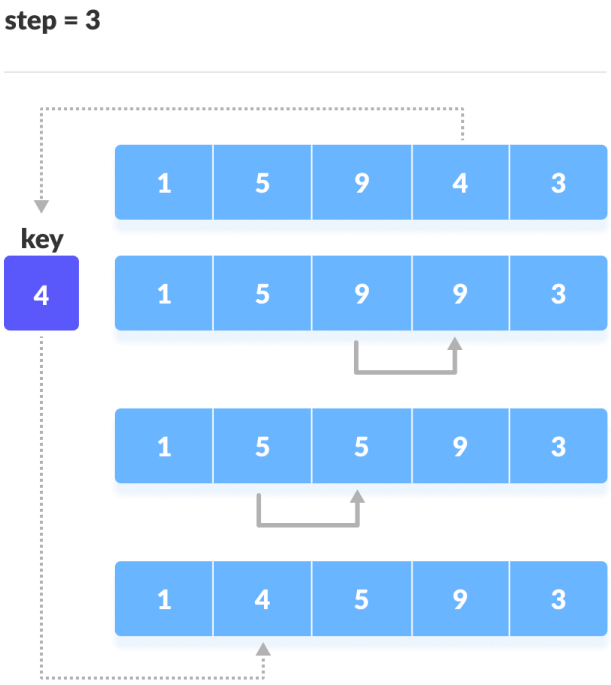
If the first element is greater than key, then key is placed in front of the first element.

2. Now, the first two elements are sorted.
   Take the third element and compare it with the elements on left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at beginning of the array.

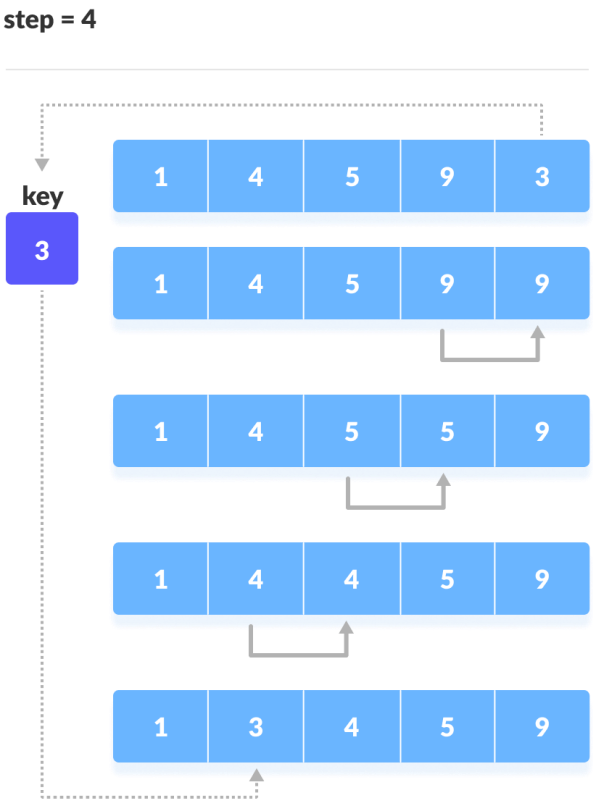


Place 1 at the beginning

3. Similarly, place every unsorted every unsorted element at its correct position.



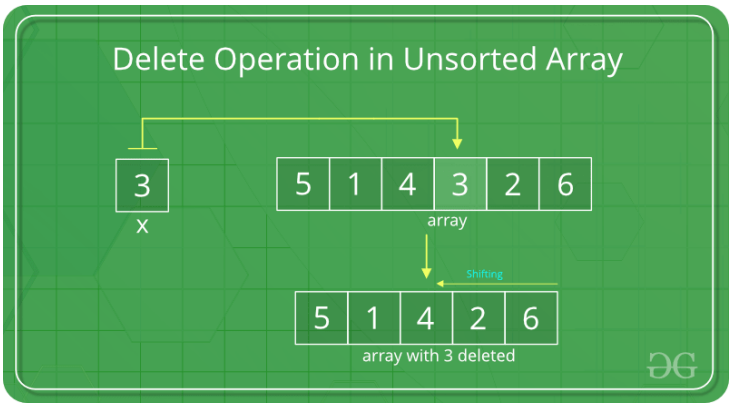
Place 4 behind 1



Place 3 behind 1 and the array is sorted

Delete:

In delete operation, the element to be deleted is searched using the linear search and then delete operation is performed followed by shifting the elements.



Algorithm:

Insertion:

1. Get the **element value** which needs to be inserted.
2. Get the **position** value.
3. Check whether the position value is valid or not.
4. If it is **valid**,
   Shift all the elements from the last index to position index by 1 position to the **right**.

   insert the new element in **arr[position]**

5. Otherwise,

   Invalid Position

Deletion:

1. Start
2. Set J = K
3. Repeat steps 4 and 5 while J < N
4. Set LA[J] = LA[J + 1]
5. Set J = J+1
6. Set N = N-1
7. Stop

Source code:

```
#Problem-1: Write a Program to insert and delete an element into a linear array
from array import *
#insert the element
def insertElements(arr):
    idx = int(input("Enter the index number : "))
    ele = int(input("Enter the element : "))
    arr.insert(idx, ele)
    print("After insertion the array : ")
    for i in range(0,len(arr)):
        print(arr[i], end=" ")
    print("\n")

#delete the element
def deleteElements(ar):
    pos = int(input("Enter the delete index number : "))
    ar.remove(arr[pos])
    print("After the delete the element : ")
    for i in range(0,n):
        print(ar[i], end=" ")

#main function
n = int(input("Enter the number : "))
arr = array('i', [])
print("Enter elements are : ")
for i in range(n):
    x = int(input())
    arr.append(x)
while True:
    print("\nIf press 1 then go to insert option.")
    print("If press 2 then go to delete option.")
    print("If press 0 then go to exit.")
    press = int(input());
    if press==1:
        insertElements(arr)
    elif press==2:
        deleteElements(arr)
    else:
        print("Exit")
        break
```

Input and Output:

```
Enter the number : 5
Enter elements are :
9
5
1
4
3

If press 1 then go to insert option.
If press 2 then go to delete option.
If press 0 then go to exit.
1
Enter the index number : 4
Enter the element : 386
After insertion the array :
9 5 1 4 386 3

If press 1 then go to insert option.
If press 2 then go to delete option.
If press 0 then go to exit.
1
Enter the index number : 1
Enter the element : 90
After insertion the array :
9 90 5 1 4 386 3

If press 1 then go to insert option.
If press 2 then go to delete option.
If press 0 then go to exit.
2
Enter the delete index number : 4
After the delete the element :
9 90 5 1 386
If press 1 then go to insert option.
If press 2 then go to delete option.
If press 0 then go to exit.
2
Enter the delete index number : 1
After the delete the element :
9 5 1 386 3
If press 1 then go to insert option.
If press 2 then go to delete option.
If press 0 then go to exit.
0
Exit
```

2 Write a program to sort a linear array using the bubble sort algorithm

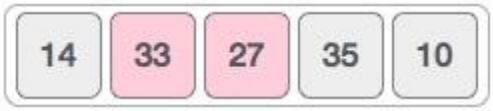
Illustration of logic:

We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping it short and precise.

Bubble sort starts with very first two elements, comparing them to check which one is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this −

Next we compare 33 and 35. We find that both are in already sorted positions.

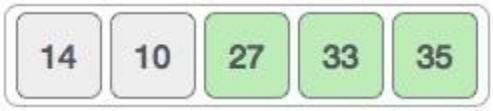
Then we move to the next two values, 35 and 10.

We know then that 10 is smaller 35. Hence they are not sorted.

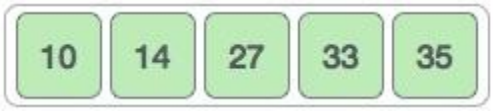
We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this −

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this −

Notice that after each iteration, at least one value moves at the end.



And when there's no swap required, bubble sorts learns that an array is completely sorted.



Pseudocode:

```
procedure bubbleSort( list : array of items )

  loop = list.count;

  for i = 0 to loop-1 do:
    swapped = false

    for j = 0 to loop-1 do:

      /* compare the adjacent elements */
      if list[j] > list[j+1] then
        /* swap them */
        swap( list[j], list[j+1] )
        swapped = true
      end if

    end for

    /*if no number was swapped that means
    array is sorted now, break the loop.*/

    if(not swapped) then
      break
    end if
  end for
end procedure return list
```

Source code:

```
from array import *
def bubbleSort(arr):
    n = len(arr)
    for i in range(0,n):
        for j in range(0,n-i-1):
            if arr[j] > arr[j+1]:
                arr[j],arr[j+1] = arr[j+1],arr[j]

#main function
while True:
    n = int(input("Enter the number : "))
    arr = array('i', [])
    print("Enter the array elements : ")
    for i in range(n):
        x = int(input())
        arr.append(x)

    bubbleSort(arr)
    print("After Bubble sorting : ")
    for i in range(0,len(arr)):
        print(arr[i], end=" ")
```

Input and Output:

Input:

Enter the number : 5
Enter the array elements :
14
33
27
35
10

Output:

After Bubble sorting :
10 14 27 33 35

3 Write a program to find an element using a linear search algorithm

Illustration of logic:

Consider the following list of elements and the element to be searched...

search element 12

Step 1:

search element (12) is compared with first element (65)

Both are not matching. So move to next element

Step 2:

search element (12) is compared with next element (20)

Both are not matching. So move to next element

Step 3:

search element (12) is compared with next element (10)

Both are not matching. So move to next element

Step 4:

search element (12) is compared with next element (55)

Both are not matching. So move to next element

Step 5:

search element (12) is compared with next element (32)

Both are not matching. So move to next element

Step 6:

search element (12) is compared with next element (12)

Both are matching. So we stop comparing and display element found at index 5.

Algorithm:

Step 1: Select the first element as the current element.

Step 2: Compare the current element with the target element. ...

Step 3: If there is a next element, then set current element to next element and go to step 2.

Step 4: Target element not found. ...

Step 5: Target element found and return location.

Step 6: Exit **process**.

Source code:

```
#Problem-3: Write a program to find an element using a linear search algorithm.
from array import *
def linearSeacrch(n,x,arr):
    for i in range(0,n):
        if(arr[i]==x):
            return i
    return -1
#main function
n = int(input("Enter the number : "))
arr = array('i',[])
print("Enter the elements : ")
for i in range(0,n):
    m = int(input())
    arr.append(m)
while True:
    print("If press 1 then go to search.")
    print("If press 0 then exit.")
    press = int(input())
    if press==1:
        x = int(input("Enter the element number : "))
        ans = linearSeacrch(n, x, arr)
        if ans == -1:
            print(x, " are not present in this array")
        else:
            print(x, " are present in this ", ans, " number index array")
    else:
        print("Exit")
        break
```

Input and Output:

Input and Output:

```
Enter the number : 5
Enter the elements :
10
20
30
40
50
If press 1 then go to search.
If press 0 then exit.
1
Enter the element number : 30
30  are present in this  3  number index array
If press 1 then go to search.
If press 0 then exit.
1
Enter the element number : 40
40  are present in this  4  number index array
If press 1 then go to search.
If press 0 then exit.
0
Exit
```

4 Write a program to find an element using the binary search algorithm

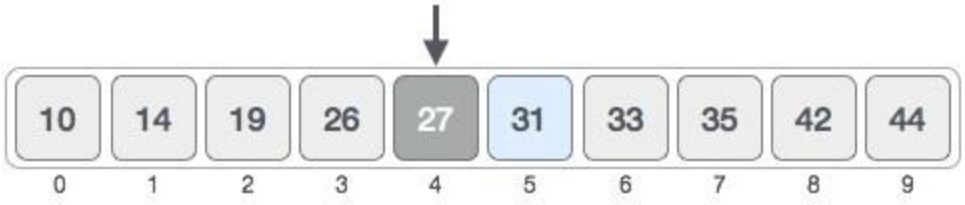
Illustration of logic:

For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.

First, we shall determine half of the array by using this formula −

mid = low + (high - low) / 2

Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.



Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.

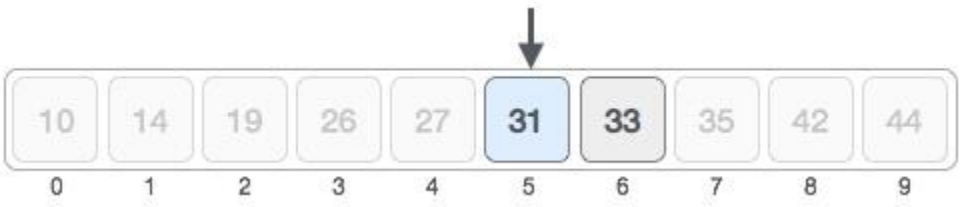
We change our low to mid + 1 and find the new mid value again.

low = mid + 1
mid = low + (high - low) / 2

Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.

The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.

Hence, we calculate the mid again. This time it is 5.



We compare the value stored at location 5 with our target value. We find that it is a match.

We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Algorithm:

Step-1: Start with the middle element:

- If the **target** value is equal to the middle element of the array, then return the index of the middle element.
- If not, then compare the middle element with the target value,
  - If the target value is greater than the number in the middle index, then pick the elements to the right of the middle index, and start with Step 1.
  - If the target value is less than the number in the middle index, then pick the elements to the left of the middle index, and start with Step 1.

Step-2: When a match is found, return the index of the element matched.

Step-3: If no match is found, then return -1

Source code:

```
#Problem-4: Write a program to find an element using the binary search algorithm
from array import *
def binrySearch(arr, n, x):
    lo=0
    hi=n-1
    while lo<=hi:
        mid = lo+(hi-1)//2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            hi = mid-1
        else:
            lo = mid+1
    return -1
#main function
 while True:
    n = int(input("Enter the number : "))
```

```
arr1 = array('i',[])
for i in range(0, n):
    x = int(input())
    arr1.append(x)
r = int(input("Enter the search number : "))
arr = sorted(arr1)
result = binrySearch(arr, n, r)
if result==-1:
    print("Element are not present in array.")
else:
    print("Element are present at index ",result)
```

Input and Output:

Input:

Enter the number : 4
30
20
10
50
Enter the search number : 50

Output:

Element are present at index 4

5 Write a program to sort a linear array using the merge sort algorithm

Illustration of logic:

To understand merge sort, we take an unsorted array as the following −

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

We further divide these arrays and we achieve atomic value which can no more be divided.

Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

After the final merging, the list should look like this −

Now we should learn some programming aspects of merge sorting.

Algorithm:

Step 1 − if it is only one element in the list it is already sorted, return.

Step 2 − divide the list recursively into two halves until it can no more be divided.

Step 3 − merge the smaller lists into new list in sorted order.

Source code:

```
1  #Problem-5: Write a program to sort a linear array using the merge sort algorithm
2  from array import *
3  # Merge sort implementation
4  def mergeSorting(arr):
5      if len(arr) > 1:
6          # base case
7          mid = len(arr) // 2
8          lo = arr[:mid]
9          hi = arr[mid:]
10         mergeSorting(lo)
11         mergeSorting(hi)
12         i = j = k = 0
13         while i < len(lo) and j < len(hi):
14             if lo[i] < hi[j]:
15                 arr[k] = lo[i]
16                 i += 1
17             else:
18                 arr[k] = hi[j]
19                 j += 1
20             k += 1
21         while i < len(lo):
22             arr[k] = lo[i]
23             i += 1
24             k += 1
25         while j < len(hi):
26             arr[k] = hi[j]
27             j += 1
28             k += 1
29
30
31 #printing function
32 def printList(arr):
33     for i in range(len(arr)):
34         print(arr[i], end=" ")
35     print()
36
37 #main function
38 while True:
39     n = int(input("Enter the number : "))
40     arr = array('i', [])
41     print("Enter the array elements : ")
42     for i in range(0, n):
43         x = int(input())
44         arr.append(x)
45     print("Given an array : ", end="\n")
46     printList(arr)
47     mergeSorting(arr)
48     print("After sorted array : ", end="\n")
49     printList(arr)
```

Input and Output:

Input:

Enter the number : 5
Enter the array elements :
10
40
20
50
30

Output:

Given an array :
10 40 20 50 30
After sorted array :
10 20 30 40 50

6 Write a program to sort a linear array using the Selection Sort algorithm

Illustration of logic:

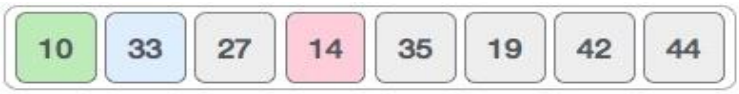
Consider the following depicted array as an example.

For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.



After two iterations, two least values are positioned at the beginning in a sorted manner.

The same process is applied to the rest of the items in the array.

Following is a pictorial depiction of the entire sorting process −

Now, let us learn some programming aspects of selection sort.

Algorithm:

Step 1 − Set MIN to location 0

Step 2 − Search the minimum element in the list

Step 3 − Swap with value at location MIN

Step 4 − Increment MIN to point to next element

Step 5 − Repeat until list is sorted

Source code:

```
1   from array import *
2
3   #Selection sorting
4   def selectionSort(arr):
5       for i in range(len(arr)):
6           mi = i
7           for j in range(i+1, len(arr)):
8               if arr[mi] > arr[j]:
9                   mi = j
10          arr[i], arr[mi] = arr[mi], arr[i]
11
12  #for printing
13  def printList(arr):
14      for i in range(len(arr)):
15          print(arr[i], end=" ")
16
17  #main function
18  while True:
19      n = int(input("Enter the number : "))
20      arr = array('i',[])
21      print("Enter the array elements : ")
22      for i in range(0,n):
23          x = int(input())
24          arr.append(x)
25
26      print("Given an array : ")
27      printList(arr)
28      selectionSort(arr)
29      print("\nAfter sorting an array : ")
30      printList(arr)
```

Input and Output:

Input:
Enter the number : 8
Enter the array elements :
14
33
27
10
35
19
42
44
Output:
Given an array :
14 33 27 10 35 19 42 44
After sorting an array :
10 14 19 27 33 35 42 44

7 Write a program to find a given pattern from text using the first pattern matching algorithm

Illustration of logic:

Knuth Morris Pratt (KMP) is an algorithm, which checks the characters from left to right. When a pattern has a sub-pattern appears more than one in the sub-pattern, it uses that property to improve the time complexity, also for in the worst case.

pat[] = "**AAACAAAA**"

len = 0, i = 0.
lps[0] is always 0, we move
to i = 1

len = 0, i = 1.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 1, **lps[1] = 1**, i = 2

len = 1, i = 2.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 2, **lps[2] = 2**, i = 3

len = 2, i = 3.
Since pat[len] and pat[i] do not match, and len > 0,
set len = lps[len-1] = lps[1] = 1

len = 1, i = 3.
Since pat[len] and pat[i] do not match and len > 0,
len = lps[len-1] = lps[0] = 0

len = 0, i = 3.
Since pat[len] and pat[i] do not match and len = 0,
Set **lps[3] = 0** and i = 4.
We know that characters pat
len = 0, i = 4.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 1, **lps[4] = 1**, i = 5

len = 1, i = 5.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 2, **lps[5] = 2**, i = 6

len = 2, i = 6.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 3, **lps[6] = 3**, i = 7

len = 3, i = 7.
Since pat[len] and pat[i] do not match and len > 0,
set len = lps[len-1] = lps[2] = 2

len = 2, i = 7.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 3, **lps[7] = 3**, i = 8

We stop here as we have constructed the whole lps[].

Pseudocode:

```
Begin
   n := size of text
   m := size of pattern
   call findPrefix(pattern, m, prefArray)

   while i < n, do
      if text[i] = pattern[j], then
         increase i and j by 1
      if j = m, then
         print the location (i-j) as there is the pattern
         j := prefArray[j-1]
```

```
    else if i < n AND pattern[j] ≠ text[i] then
      if j ≠ 0 then
        j := prefArray[j - 1]
      else
        increase i by 1
  done
End
```

Source code:

```
#Problem-7: Write a program to find a given pattern from text using the first pattern
matching algorithm
###### KMP Algorithm ##########
def PatternSearch(pat, txt):
    M = len(pat)
    N = len(txt)
    lps = [0] * M
    computeLPSarray(pat, M, lps)
    i = 0
    j = 0
    cnt=0
    while i < N:
        if pat[j] == txt[i]:
            i += 1
            j += 1
        else:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
        if j == M:
            print("Pattern found at position ",i - j)
            j = lps[j - 1]
            cnt += 1
    if cnt>0:
        print("Total pattern match : ", cnt)
    else:
        print("Pattern Not found")
def computeLPSarray(pat, M, lps):
    len = 0
    i = 1
    lps[0] = 0
    while i < M:
        if pat[i] == pat[len]:
            len += 1
            lps[i] = len
            i += 1
```

```
        else:
            if len != 0:
                len = lps[len - 1]
            else:
                lps[i] = 0
                i += 1
#main function
while True:
    print("\nPress 1 then go to pattern matching field.")
    print("Press 0 then exit.")
    press = int(input())
    if press==1 :
        txt = input("Enter the pattern field : ")
        pat = input("Enter the pattern : ")
        PatternSearch(pat, txt)
    else:
        print("Exit")
        break
```

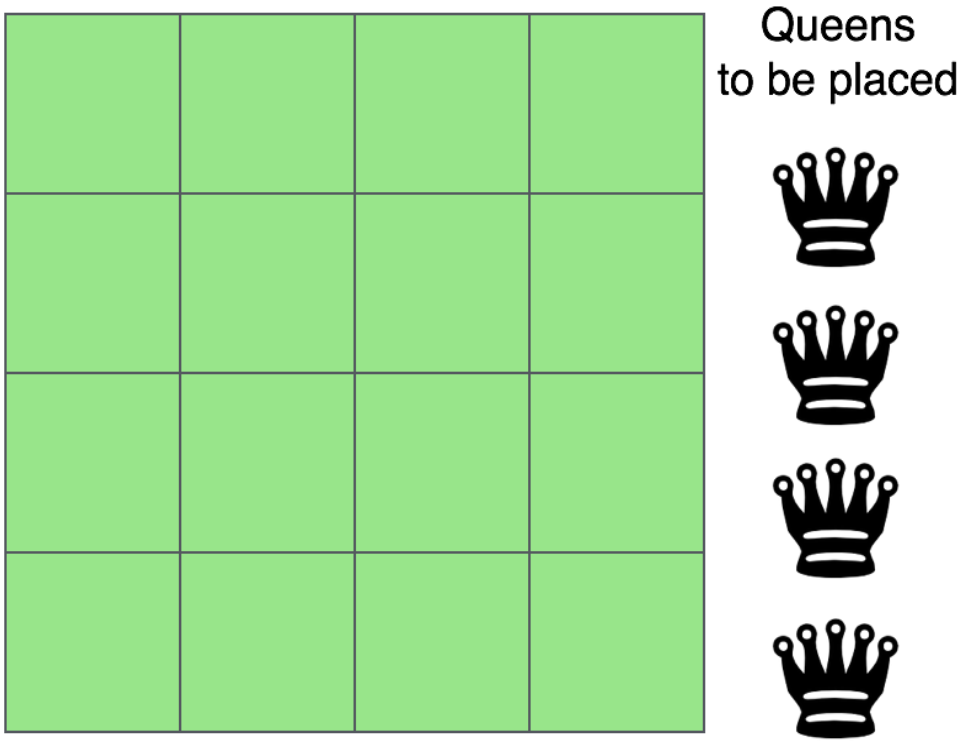
Input and Output:

```
Press 1 then go to pattern matching field.
Press 0 then exit.
1
Enter the pattern field : MYNAMEISMOHAMMEDROKONUJJAMAN
Enter the pattern : MOHAMMED
Pattern found at position  8
Total pattern match : 1

Press 1 then go to pattern matching field.
Press 0 then exit.
1
Enter the pattern field : MYFATHERNAMEISRAFIQULISLAM
Enter the pattern : ISLAM
Pattern found at position :21
Total pattern match :  1

Press 1 then go to pattern matching field.
Press 0 then exit.
0
Exit
```

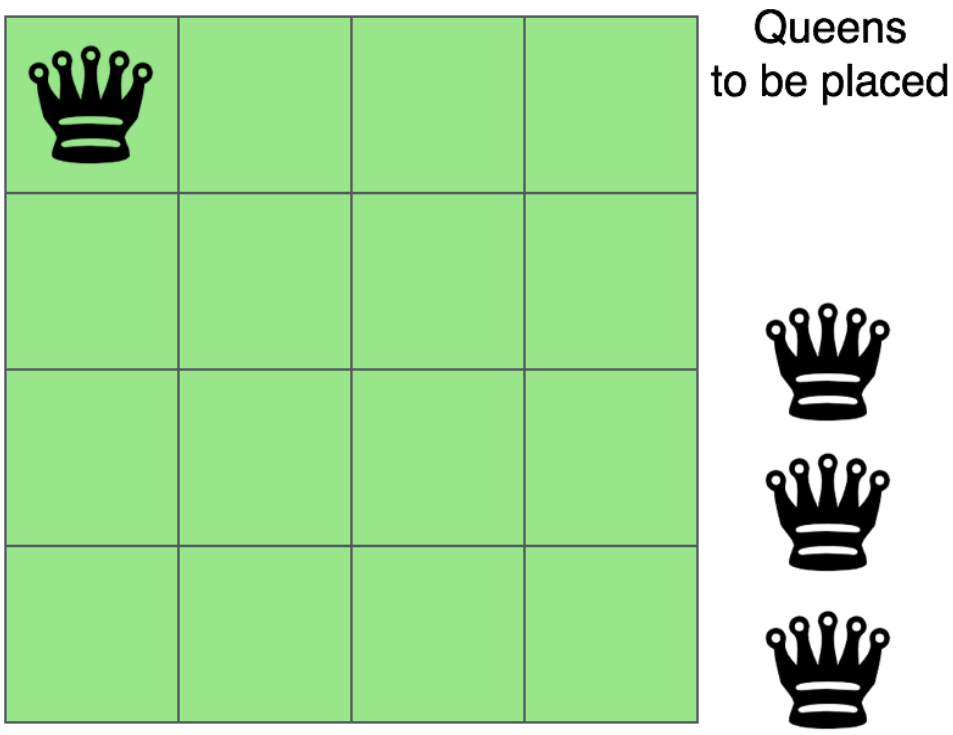
8 Write a program to solve n queen's problem using backtracking

Illustration of logic:

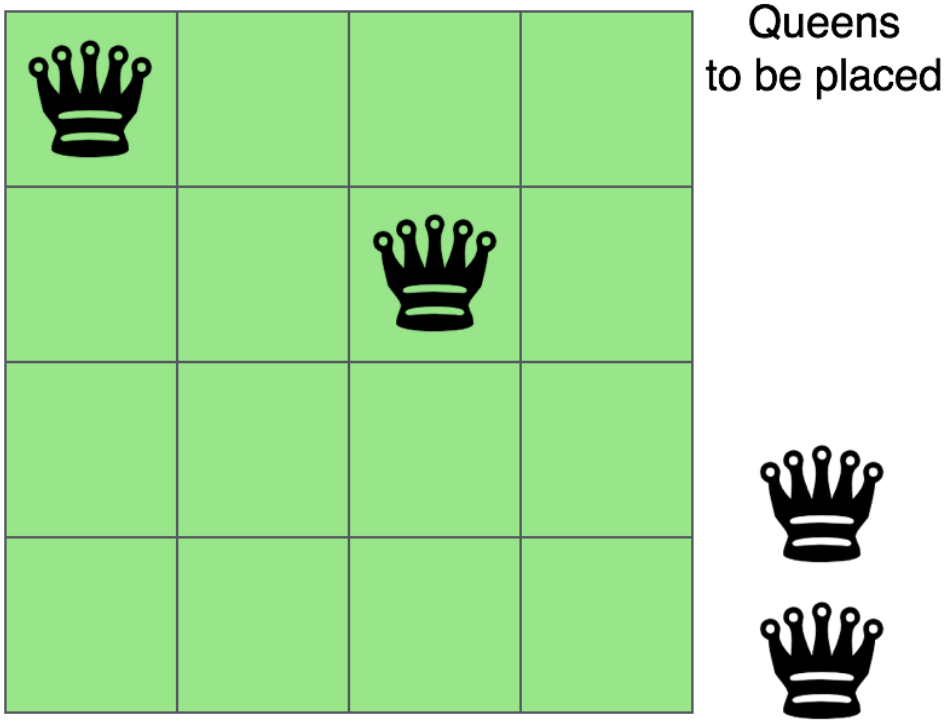
One of the most common examples of the backtracking is to arrange N queens on an NxN chessboard such that no queen can strike down any other queen. A queen can attack horizontally, vertically, or diagonally. The solution to this problem is also attempted in a similar way. We first place the first queen anywhere arbitrarily and then place the next queen in any of the safe places. We continue this process until the number of unplaced queens becomes zero (a solution is found) or no safe place is left. If no safe place is left, then we change the position of the previously placed queen.

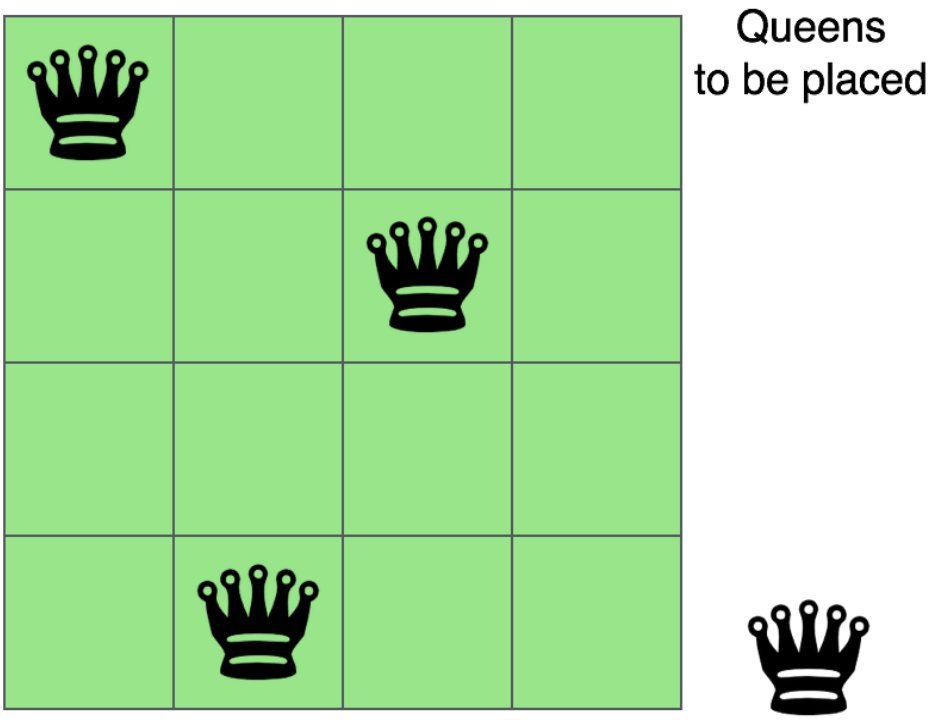


The above picture shows an NxN chessboard and we have to place N queens on it. So, we will start by placing the first queen.



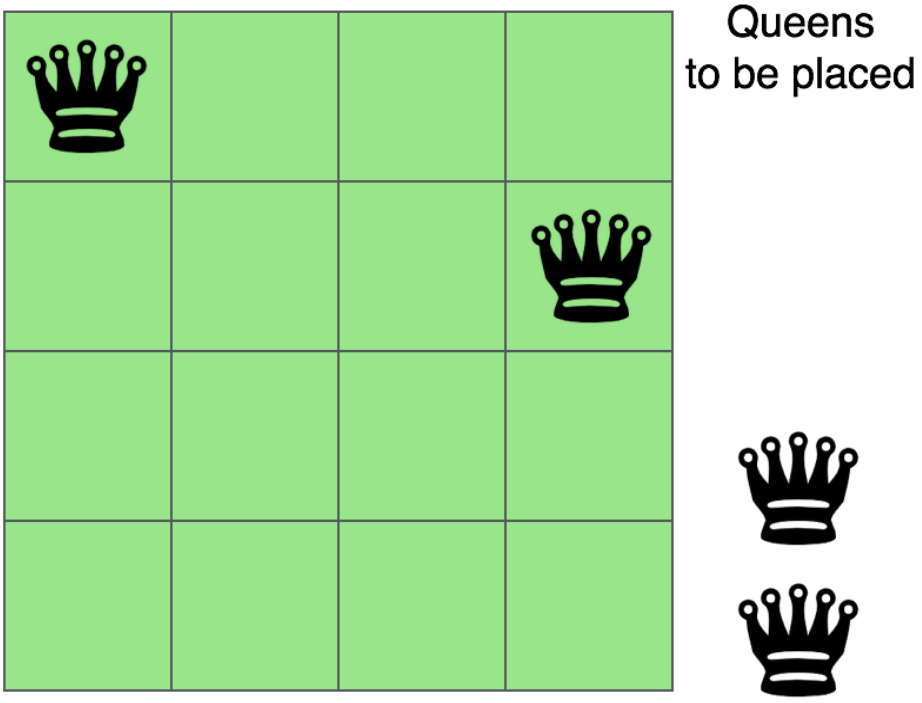
Now, the second step is to place the second queen in a safe position and then the third queen.



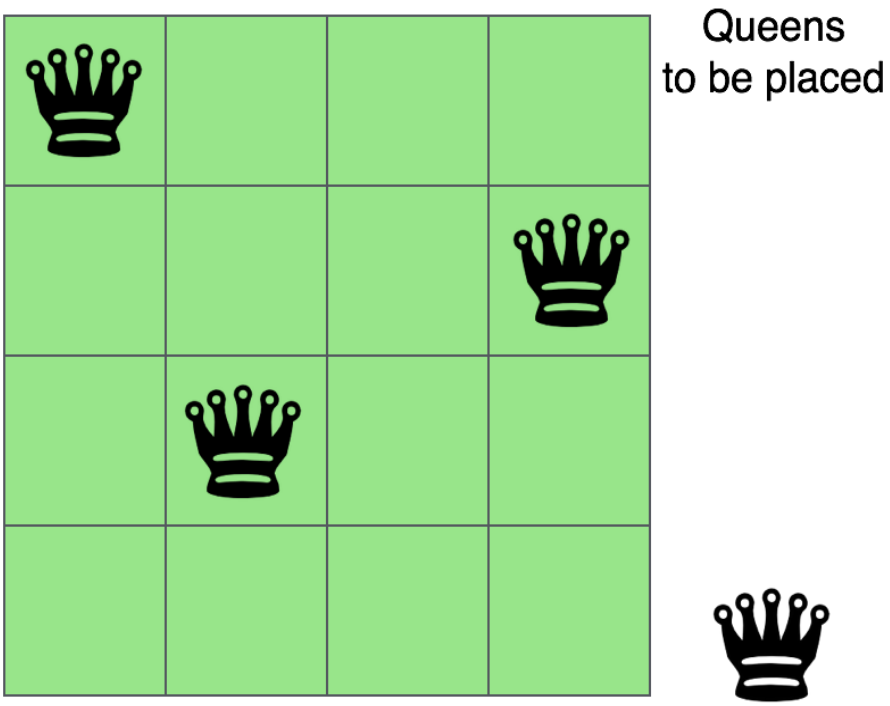


Now, you can see that there is no safe place where we can put the last queen. So, we will just change the position of the previous queen. And this is backtracking.

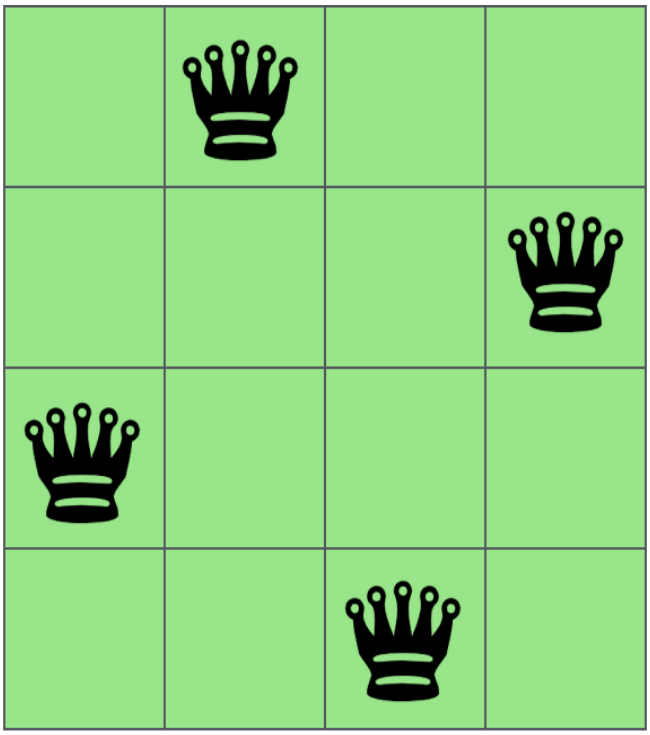
Also, there is no other position where we can place the third queen so we will go back one more step and change the position of the second queen.



And now we will place the third queen again in a safe position until we find a solution.



We will continue this process and finally, we will get the solution as shown below.



As now you have understood backtracking, let us now code the above problem of placing N queens on an NxN chessboard using the backtracking method.

Algorithm:

Step-1: Start in the leftmost column
Step-2: If all queens are placed
Return true
Step-3: Try all rows in the current column
Do following for every tried row.
a) If the queen can be placed safely in this row
Then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
b) If placing the queen in [row, column] leads to a solution then return true.
c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
Step-4: If all rows have been tried and nothing worked, return false to trigger backtracking.

Source code:

```
#Problem-8: Write a program to solve n queen's problem using
backtracking
print("Enter the number of queens")
N = int(input())
#NxN matrix with all elements 0
board = [[0]*N for _ in range(N)]

def is_attack(i, j):
    #checking if there is a queen in row or column
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return 1
    #checking diagonals
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return 1
    return 0

def N_queen(n):
    #if n is 0, solution found
    if n==0:
        return 1
    for i in range(0,N):
        for j in range(0,N):
            '''checking if we can place a queen here or not
            queen will not be placed if the place is being attacked
            or already occupied'''
            if (not(is_attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1
                #recursion
```

```
33              #wether we can put the next queen with this arrangment or not
34              if N_queen(n-1)==1:
35                  return 1
36              board[i][j] = 0
37      return 0
38
39  N_queen(N)
40  for i in board:
41      print (i)
```

Input and output:

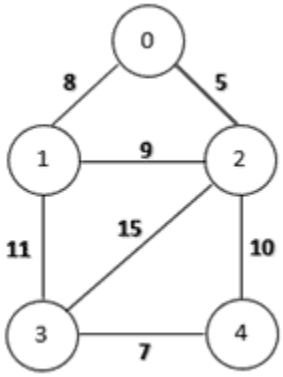
Input and Output:

Enter the number of queens
4
[0, 1, 0, 0]
[0, 0, 0, 1]
[1, 0, 0, 0]
[0, 0, 1, 0]

9 Write a program to find the shortest path from a graph using Kruskal's Algorithm

Illustration of logic:

Let us take the following graph as an example:



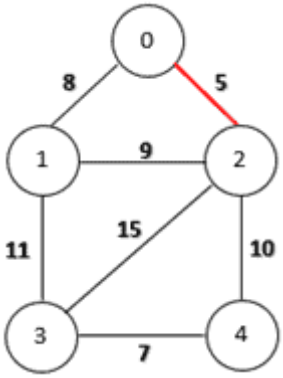
Move: 0

Now, the edges in ascending order of their weights are:

- **0-2 :** 5
- **3-4 :** 7

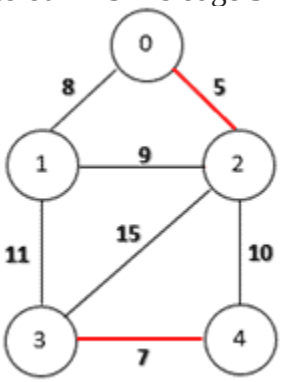
- **0-1 :** 8
- **1-2 :** 9
- **2-4 :** 10
- **1-3 :** 11

Now, we will select the minimum weighted edge, i.e. **0-2**, and add it to MST:



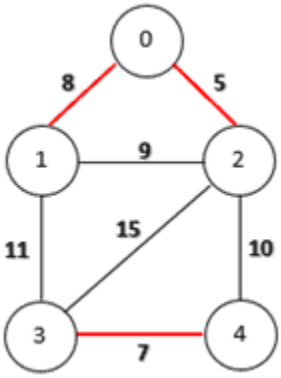
Move: 1

The next edge that we will add to our MST is edge **3-4**:



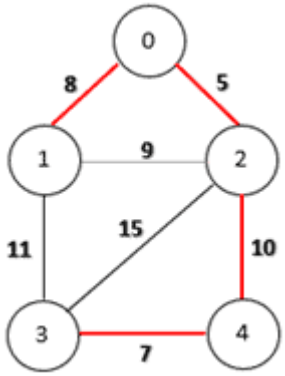
Move: 2

Now, we will add the edge **0-1**:



Move: 3

Now, we have the edge **1-2** next, however, we add this edge then a cycle will be created. As a result, this edge will be rejected.
After adding the edge **2-4,** the MST is complete, as all the vertices are now included.



Minimum Spanning Tree

Algorithm:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat Step-2 until there are (V-1) edges in the spanning tree.

Source code:

```
#P-9: Write a program to find the shortest path from a graph using Kruskal Algorithm
# Kruskal Algorithm  in Python
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    # Search function

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def apply_union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
```

```
            parent[yroot] = xroot
            rank[xroot] += 1

    #  Applying Kruskal algorithm
    def kruskal_algo(self):
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.apply_union(parent, rank, x, y)
        print("The sortest path : ")
        for u, v, weight in result:
            print("%d - %d: %d" % (u, v, weight))


g = Graph(6)
g.add_edge(0, 1, 4)
g.add_edge(0, 2, 4)
g.add_edge(1, 2, 2)
g.add_edge(1, 0, 4)
g.add_edge(2, 0, 4)
g.add_edge(2, 1, 2)
g.add_edge(2, 3, 3)
g.add_edge(2, 5, 2)
g.add_edge(2, 4, 4)
g.add_edge(3, 2, 3)
g.add_edge(3, 4, 3)
g.add_edge(4, 2, 4)
g.add_edge(4, 3, 3)
g.add_edge(5, 2, 2)
g.add_edge(5, 4, 3)
g.kruskal_algo()
```

Input and Output:

Input and Output:

The sortest path :
1 - 2: 2
2 - 5: 2
2 - 3: 3
3 - 4: 3
0 - 1: 4

10 Write a program using greedy method to solve this problem when no of job n = 5, profits ($P1$, $P2$, $P3$, $P4$, $P5$) = (3,25,1,6,30) and deadlines ($d1$,$d2$, $d3$, $d4$, $d5$) = (1,3,2,1,2)

Illustration of logic:

Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes the single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

Example:

Input: Five jobs following deadlines and profits

Job ID	Deadline	Profit
a	1	3
b	3	25
c	2	1
d	1	6
e	2	30

Algorithm:

Step-1: Sort all jobs in decreasing order of profit

Step-2: Iterate on jobs in decreasing order of profit. For each job, do the following:

a) Find a time slot i, such that is empty and i < deadline and i is greatest. Put the job in this slot and mark this slot filled.
b) If no such i exits, then ignore the job.

Source code:

```
    #Problem-10: Write a program using greedy method to solve this problem when no of
1   job n = 5, profits (P1, P2, P3, P4, P5)=(3,25,1,6,30) and deadlines (d1,d2, d3, d4,
2   d5)=(1,3,2,1,2).
3   def printJobScheduling(arr, t):
4       # length of array
5       n = len(arr)
6       # Sort all jobs according to
7       # decreasing order of profit
8       for i in range(n):
9           for j in range(n - 1 - i):
10              if arr[j][2] < arr[j + 1][2]:
11                  arr[j], arr[j + 1] = arr[j + 1], arr[j]
12                  # To keep track of free time slots
13      result = [False] * t
14      # To store result (Sequence of jobs)
15      job = ['-1'] * t
16      # Iterate through all given jobs
17      for i in range(len(arr)):
18
19          # Find a free slot for this job
20          # (Note that we start from the
21          # last possible slot)
22          for j in range(min(t - 1, arr[i][1] - 1), -1, -1):
23
24              # Free slot found
25              if result[j] is False:
26                  result[j] = True
27                  job[j] = arr[i][0]
28                  break
29      # print the sequence
30      print(job)
31
32  # main function
33  arr = [['a', 1, 3],
34         ['b', 3, 25],
35         ['c', 2, 1],
36         ['d', 1, 6],
37         ['e', 2, 30]]
38  print("Following is maximum profit sequence of jobs")
39  # Function Call
40  printJobScheduling(arr, 3)
```

Input and Output:

Input & Output:
Following is maximum profit sequence of jobs ['d', 'e', 'b']

11 Write a program to solve the following 0/1 Knapsack using dynamic programming approach profits P = (15,25,13,23), weight W = (2,6,12,9), Knapsack C = 20, and the number of items n=4

Illustration of logic:

In this item cannot be broken which means thief should take the item as a whole or should leave it. That's why it is called **0/1 knapsack Problem**.

- Each item is taken or not taken.
- Cannot take a fractional amount of an item taken or take an item more than once.
- It cannot be solved by the Greedy Approach because it is enable to fill the knapsack to capacity.
- **Greedy Approach** doesn't ensure an Optimal Solution.

Example: The maximum weight the knapsack can hold is W is 11. There are five items to choose from. Their weights and values are presented in the following table:

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$												
$w_2 = 2\ v_2 = 6$												
$w_3 = 5\ v_3 = 18$												
$w_4 = 6\ v_4 = 22$												
$w_5 = 7\ v_5 = 28$												

The [i, j] entry here will be V [i, j], the best value obtainable using the first "i" rows of items if the maximum capacity were j. We begin by initialization and first row.

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0											
$w_3 = 5\ v_3 = 18$	0											
$w_4 = 6\ v_4 = 22$	0											
$w_5 = 7\ v_5 = 28$	0											

$V [i, j] = \max \{V [i - 1, j], v_i + V [i - 1, j - w_i]$

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0											
$w_4 = 6\ v_4 = 22$	0											
$w_5 = 7\ v_5 = 28$	0											

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0											
$w_5 = 7\ v_5 = 28$	0											

The value of V [3, 7] was computed as follows:

$V [3, 7] = \max \{V [3 - 1, 7], v_3 + V [3 - 1, 7 - w_3]$

$= \max \{V [2, 7], 18 + V [2, 7 - 5]\}$

$= \max \{7, 18 + 6\}$

$= 24$

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7\ v_5 = 28$	0											

Finally, the output is:

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7\ v_5 = 28$	0	1	6	7	7	18	22	28	29	34	35	40

Algorithm:

KNAPSACK (n, W)

1. for w = 0, W
2. do V [0,w] ← 0
3. for i=0, n
4. do V [i, 0] ← 0
5. for w = 0, W
6. do if ($w_i \leq w$ & v_i + V [i-1, w - w_i]> V [i -1,W])
7. then V [i, W] ← v_i + V [i - 1, w - w_i]
8. else V [i, W] ← V [i - 1, w]

Source code:

```
#Problem-11: Write a program to solve the following 0/1 Knapsack using dynamic
programming approach profits P = (15,25,13,23), weight W = (2,6,12,9), Knapsack
C = 20, and the number of items n=4
def Knapsack (numitems, capacity, weight, value):

    # No item can be put in the sack of capacity 0 so maximum value for sack of
capcacity 0 is 0
    if (capacity == 0):
        return 0

    # If 0 items are put in the sack, then maximum value for sack is 0
    if (numitems == 0):
        return 0

    # Note : Here the number of item is limited (unlike coin change / integer partition
problem)
    # hence the numitems -> (numitems - 1) when the item is included in the knapsack
    if (capacity >= weight[numitems-1]):
        return max ( Knapsack (numitems-1, capacity, weight, value), # Item is not
included.
```

```
                Knapsack (numitems-1, capacity-weight[numitems-1], weight, value) +
value[numitems-1] )# Item included.
    else:
        return Knapsack (numitems-1, capacity, weight, value)

# DP approach to 0-1 Knapsack problem
def DPKnapsack (capacity, weight, value):

    numitems = len(weight)
    maxval = [0] * (numitems+1)

    for r in range (numitems+1) :
        maxval[r] = [0] * (capacity+1)

    # If 0 items are put in the sack of capacity 'cap' then maximum value for each sack
is 0
    for cap in range (capacity+1) :
        maxval[0][cap] = 0

    # No item can be put in the sack of capacity 0 so maximum value for sack of
capcacity 0 is 0
    for item in range (numitems+1) :
        maxval[item][0] = 0

    # Note : Here the number of item is limited (unlike coin change / integer partition
problem)
    # hence the numitems -> (numitems - 1) when the item is included in the knapsack
    for item in range (1, numitems+1) :
        for cap in range (1, capacity+1) :
            if (cap >= weight[item-1]) :
                maxval[item][cap] = max (maxval[item-1][cap], maxval[item-1][cap-
weight[item-1]] + value[item-1])
            else:
                maxval[item][cap] = maxval[item-1][cap]

    return maxval[numitems][capacity]

weight = [2,6,12,9]
value =  [15,25,13,23,]
capacity = 10
print("Maximum value of 0-1 Knapsack using DP : " + str( DPKnapsack (capacity,
weight, value)))
```

Input and Output:

Maximum value of 0-1 Knapsack using DP: 40

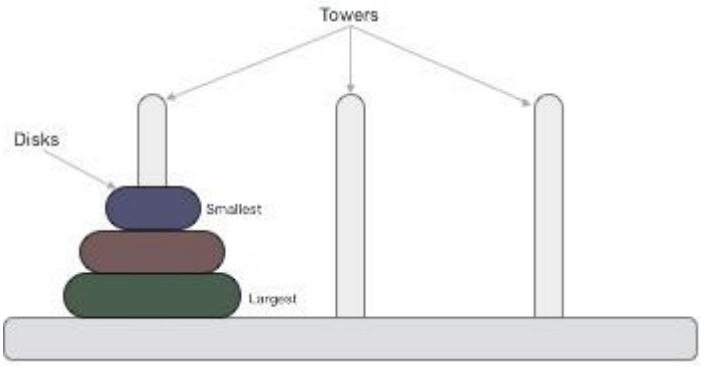
12 Write a program to solve the Tower of Hanoi problem for the N disk

Illustration of logic:

Before getting started, let's talk about what the Tower of Hanoi problem is. Well, this is a fun puzzle game where the objective is to move an entire stack of disks from the source position to another position. Three simple rules are followed:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack. In other words, a disk can only be moved if it is the uppermost disk on a stack.
3. No larger disk may be placed on top of a smaller disk.

   Now, let's try to imagine a scenario. Suppose we have a stack of three disks. Our job is to move this stack from **source A** to **destination C**. How do we do this? Before we can get there, let's imagine there is an **intermediate point B**.



Three disks

We can use B as a helper to finish this job. We are now ready to move on. Let's go through each of the steps:

1. Move the first disk from A to C
2. Move the first disk from A to B
3. Move the first disk from C to B
4. Move the first disk from A to C

5. Move the first disk from B to A
6. Move the first disk from B to C
7. Move the first disk from A to C

Algorithm:

Step 1: Shift 'n-1' disks from 'A' to 'B'.

Step 2: Shift last disk from 'A' to 'C'.

Step 3: Shift 'n-1' disks from 'B' to 'C'.

Source code:

```
#Problem-12: Write a program to solve the Tower of Hanoi problem for the N disk.
def towerHanoi(n, A, B, C):
    if n==1:
        print("Move disc ",n," from ",A," to ",B)
        print(A, B, C)
    else:
        towerHanoi(n-1, A, C, B)
        print("Move disc ",n," from ",B," to ",A)
        print(A, B, C)
        towerHanoi(n-1, C, B, A)

while True:
    print("Press 1 then go to work.")
    print("Press 0 them exit.")
    n = int(input())
    if n==0:
        print("Exit.")
        break
    else:
        number = int(input("How many discs : "))
        a = 'A'
        b = 'B'
        c = 'C'
        print(a, b, c)
        towerHanoi(number, a, b, c)
```

Input and Output:

Input and Output:

```
Press 1 then go to work.
Press 0 them exit.
1
How many discs : 3
A B C
Move disc  1  from  A  to  B
A B C
Move disc  2  from  C  to  A
A C B
Move disc  1  from  B  to  C
B C A
Move disc  3  from  B  to  A
A B C
Move disc  1  from  C  to  A
C A B
Move disc  2  from  B  to  C
C B A
Move disc  1  from  A  to  B
A B C

Press 1 then go to work.
Press 0 them exit.
1
How many discs : 5
A B C
Move disc  1  from  A  to  B
A B C
Move disc  2  from  C  to  A
A C B
Move disc  1  from  B  to  C
B C A
Move disc  3  from  B  to  A
A B C
Move disc  1  from  C  to  A
C A B
Move disc  2  from  B  to  C
C B A
Move disc  1  from  A  to  B
A B C
Move disc  4  from  C  to  A
A C B
Move disc  1  from  B  to  C
B C A
Move disc  2  from  A  to  B
B A C
Move disc  1  from  C  to  A
C A B
```

```
Move disc 3 from C to B
B C A
Move disc 1 from A to B
A B C
Move disc 2 from C to A
A C B
Move disc 1 from B to C
B C A
Move disc 5 from B to A
A B C
Move disc 1 from C to A
C A B
Move disc 2 from B to C
C B A
Move disc 1 from A to B
A B C
Move disc 3 from A to C
C A B
Move disc 1 from B to C
B C A
Move disc 2 from A to B
B A C
Move disc 1 from C to A
C A B
Move disc 4 from B to C
C B A
Move disc 1 from A to B
A B C
Move disc 2 from C to A
A C B
Move disc 1 from B to C
B C A
Move disc 3 from B to A
A B C
Move disc 1 from C to A
C A B
Move disc 2 from B to C
C B A
Move disc 1 from A to B
A B C

Press 1 then go to work.
Press 0 them exit.
0
Exit.
```

13 Write a program to implement a queue data structure along with its typical operations

Illustration of logic:

Queues, like the name suggests, follow the **First-in-First-Out** (FIFO) principle. As if waiting in a queue for the movie tickets, the first one to stand in line is the first one to buy a ticket and enjoy the movie.

A Queue supports the following standard operations:

1. **append**: Inserts an element at the rear (right side) of the queue.
2. **pop**: Removes the element from the front (left side) of the queue.
3. **peek**: Returns the element at the front of the queue without removing it.
4. **isEmpty**: Checks whether the queue is empty.
5. **size**: Returns the number of elements present in the queue.

Algorithm:

peek():

```
begin procedure peek
   return queue[front]
end procedure
```

isfull()

```
begin procedure isfull

   if rear equals to MAXSIZE
      return true
   else
      return false
   endif

end procedure
```

isEmpty()

```
begin procedure isempty

   if front is less than MIN  OR front is greater than rear
      return true
   else
      return false
```

```
  endif

end procedure
```

Source code:

```
1  #Problem-13: Write a program to implement a queue data structure along with its
2  typical operation.
3  class Queue:
4
5      # Initialize queue
6      def __init__(self, size):
7          self.q = [None] * size  # list to store queue elements
8          self.capacity = size  # maximum capacity of the queue
9          self.front = 0  # front points to front element in the queue if present
10         self.rear = -1  # rear points to last element in the queue
11         self.count = 0  # current size of the queue
12
13     # Function to remove front element from the queue
14     def pop(self):
15         # check for queue underflow
16         if self.isEmpty():
17             print("Queue UnderFlow!! Terminating Program.")
18             exit(1)
19
20         print("Removing element : ", self.q[self.front])
21
22         self.front = (self.front + 1) % self.capacity
23         self.count = self.count - 1
24
25     # Function to add a value to the queue
26     def append(self, value):
27         # check for queue overflow
28         if self.isFull():
29             print("OverFlow!! Terminating Program.")
30             exit(1)
31
32         print("Inserting element : ", value)
33
34         self.rear = (self.rear + 1) % self.capacity
35         self.q[self.rear] = value
36         self.count = self.count + 1
37
38     # Function to return front element in the queue
39     def peek(self):
40         if self.isEmpty():
41             print("Queue UnderFlow!! Terminating Program.")
```

```
42            exit(1)
43
44        return self.q[self.front]
45
46    # Function to return the size of the queue
47    def size(self):
48        return self.count
49
50    # Function to check if the queue is empty or not
51    def isEmpty(self):
52        return self.size() == 0
53
54    # Function to check if the queue is full or not
55    def isFull(self):
56        return self.size() == self.capacity
57
58
59 if __name__ == '__main__':
60
61    # create a queue of capacity 100
62    q = Queue(100)
63    while True:
64        print("Press 0 then exit.")
65        print("Press 1 then go to append.")
66        print("Press 2 then go to pop.")
67        print("Press 3 then go to check isEmpty?")
68        n = int(input())
69        if n==0:
70            print("EXIT.")
71            break;
72        elif n==1:
73            x = int(input("Enter the element : "))
74            q.append(x)
75        elif n==2:
76            print("Queue size is", q.size())
77            print("Front element is", q.peek())
78            q.pop()
79        elif n==3:
80            if q.isEmpty():
81                print("Queue is empty")
82            else:
83                print("Queue is not empty")
```

Input and Output:

Input & Output:

```
Press 0 then exit.
Press 1 then go to append.
Press 2 then go to pop.
Press 3 then go to check isEmpty?
1
Enter the element : 36
Inserting element :  36
Press 0 then exit.
Press 1 then go to append.
Press 2 then go to pop.
Press 3 then go to check isEmpty?
1
Enter the element : 30
Inserting element :  30
Press 0 then exit.
Press 1 then go to append.
Press 2 then go to pop.
Press 3 then go to check isEmpty?
1
Enter the element : 10
Inserting element :  10
Press 0 then exit.
Press 1 then go to append.
Press 2 then go to pop.
Press 3 then go to check isEmpty?
1
Enter the element : 20
Inserting element :  20
Press 0 then exit.
Press 1 then go to append.
Press 2 then go to pop.
Press 3 then go to check isEmpty?
2
Queue size is 4
Front element is 36
Removing element :  36
Press 0 then exit.
Press 1 then go to append.
Press 2 then go to pop.
Press 3 then go to check isEmpty?
3
Queue is not empty
Press 0 then exit.
Press 1 then go to append.
Press 2 then go to pop.
Press 3 then go to check isEmpty?
0
EXIT.
```