الطالب /عزالدين نعمان محمد احمد حسين

نشاط الخوارزميات

ملحوضة جميع الاكواد التنفيذية للخوارزميات كتبت بلغة بايثون

---

١. خوارزمية اقليدس لحساب القاسم المشترك الأكبر:

الشفرة الزايفه :

While n ≠ *0* **do**

R <- m mod n

M<-n

N<- r

return *m*

الكود :

```
def gcd(m,n):
    while n !=0:
        r=m%n
        m=n
        n=r
    return m
#الحصول على المدخلات من المستخدم
m=int(input ("inter numb 1:"))
n=int(input ("inter numb 2:"))
result=gcd(m,n)
print (f"gcd(P{m},{n}) the : {result}")
```

خطوات عمل البرنامج : تكراري

٢-خوارزمية فيبوناتشي التكرارية :

الشفر الزايفه :

F[0]<-0;f[1]<-1

For i>-2 to n do

F[i]<-f[i-1]+f[i-2]

Return f[n]

الكود:تكراري

```
def fib(n):
    if n==0:
        return [0]

    elif n==1:
        return [0,1]
    f=[0,1]
    for i in range(2,n):
        f.append(f[i-1]+ f[i-2])
    return f
m=int (input("enter number"));
 print(fib(m))
```

٣- خوارزمية فيبوناتشي التعاودية:

If (n<=1)

Return 1

Return fib(n-1)+fib(n-2)

الكود التنفيذي:تعاودي

```
def fib(n):
    if n<=1:
        return n
    return fib (n-1 )+fib(n-2)
# الحصول علىn n من المستخدم
n=int(input("enter number"));
for i in range(n):
     print (fib(i),end=" ")
```

٤-خوارزمية ضرب المصفوفات :

For i <-0 to n-1 do

For j <- 0 to n-1

C[I,j]<-0.0

For k <- 0 to n-1 do

C[I,j]<- c[I,j]+a[I,k]*b[k,j]

Return c

الكود:

```
def multi(a,b):
    n=len(a)
    c=[[0]*n for _ in range (n)]
    for i in range(n):
        for j in range(n):
            c[i][j]=sum(a[i][k]*b[k][j] for k in range (n ))
    return c
 #الحصول علئ حجم المصفوفة من المستخدم
n=int(input(" input size array ")) ;
a=[]
b=[]
#ادخال قيم المصفوفتين
print (" input array a");
for i in range (n ):
    row=list(map (int ,input().split()))
    a.append(row)
print (" input array b");
for i in range (n ):
    row=list(map (int ,input().split()))
    b.append(row)
#استدعاء الدالة وعرض النتيجة
c=multi(a,b)
print ("result multi a*b")
for row in c:
    print (" ".join (map(str,row)))
```

٥-مشكلة تفرد العناصر :

الشفرة الزايفه :

Function is _unique(a,n)

For I from 0 to n-2

For j from i+1 to n-1

If a[i]==a[j]

Return false

Return true

الكود: تكراريا

```
def is_unique(a,n):
    for i in range(n-1):
        for j in range(i+1,n):
            if a[i]==a[j]:
                return False
    return True#مما يدل علء ان جكميع العناصر فريدة
def main():
    n=int(input("how many number do yuo want:"))
     #اطلب من المستخدم عدد العناص
    print ("==================================")
    a=[]
    #print ("inter numbers 1:")
    for i  in range(n):
        number = int (input(f"enter number {i+1}: "))
        a.append(number)
    if is_unique(a,n):
       print ("difrent number:")
    else :
       print ("not difrent")
if __name__ =="__main__":main()
```

٦-خوارزمية البحث التعاقبي تكراريا:

```
def serch(a,k):
    for i in range (len(a)):
        if a[i]==k:
            return i
    return -1
def main():
    n=int (input("how many string  do you wantto enter :"))
    print("==============================")
    a=[input (f"entrr string {i+1}:")for i in range (n)]
    k=input ("inter string to serch for :")
    index=serch(a,k)
    if index !=-1:
        print (f"index of the string found :{index}")
    else:
        print (f"index of the string not found")
if __name__=="__main__":
main()
```

٧- لغز أبراج هانوي :

الكود الزايف :

```
Function Hanoi(n,sourcefirst,second,thred):
  If n==1 :
        Print "move diske from                              ,
sourcefirst,"to",thred

   Else:
       honi(n-1, sourcefirst,second,thred)
            Print "move diske from,  sourcefirst,"to",thred
        honi(n-1, sourcefirst,second,thred)
```

الكود : الية العمل تعاوديا

```
def honi(n,sour,second,three,towers):
    if n==1:
        disk =towers[sour].pop()
        towers[second].append(disk)
        print (f"move disk {disk} from  {sour} to {second}")
        print("towers:",towers)
    else:
        honi(n-1,sour,second,three,towers)
        honi(1,sour,three,second,towers)
        honi(n-1,three,sour,second,towers)
        n=3
        towers={'a':list(range(n,0,-1)),'b':[],'c':[]}
        honi(n,'a','b','c',towers)
        if towers['c']==list(range(n,0,-1)):
           print ("all disks:")
        else:
          print("not.")

#n=int (input("enter number:"))
 honi(3,'a', 'b', 'c')
```

خوارزمية ابرج هانوي :الية العمل تعاقبيا :

لحساب عدد الحركات اللزمة لترتب الأقراص قي العمود عن طريق أدخال عدد الاعمدة:

```
def hanoi(n):
    if n==1:
        return 1
    else:
        return 2* hanoi(n-1)+1
n=int(input("enter the number of disk:"))
 print(hanoi(n))
```

٨-خوارزمية إيجاد العنصر الاكبر :الية العمل تعاودي

```
def findmax(arr,n):
    maxval=arr[0]
    for i in range(n
        if maxval == arr[i]:
            maxval=arr[i]
            return maxval

n=int(input("how many number do you want:"))
a=[]
print("enter numbers")
for i in range(n):
    num=int (input())
    a.append(num)
 print("the number max is :",findmax(a,n))
```

٩-حساب عدد الارقام الثنائية تعاوديا:

```
def  binary(n):
    if n==1:
        return 1
    else:
```

```
        return binary(n//2)+1
n=int(input("enter positiv integer:"))
result=binary(n)
print("the number of binary digist is : :",result)
```

١٠- حساب عدد الارقام الثنائية نكراريا:

```
def binary(n):
    count=1
    while n>1:
      count+=1
      n=n//2#القسمة الصحيحه
    return count
n=int(input("enter positiv integer:"))
result=binary(n)
 print("the number of binary digist is : :",result)
```

11-#خوارزمية الترتيب بالاختيار

```
def selection_sort(arr):
    for i in range(len(arr) - 1):
        min_index = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

# استخدام الدالة
#ادخل الارقام مفصولة بمسافات
arr =[int(x) for x in input("input numbers must space:").split()]
sorted_arr = selection_sort(arr)
 print("numbers after sort", sorted_arr)
```

١٢-خوترزمية الترتيب الفقاعي

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if arr[j + 1] < arr[j]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

# استخدام الدالة
arr = [int(x) for x in input("input numbers must space:").split()]
sorted_arr =bubble_sort(arr)
print("numbers after sort", sorted_arr)
#////////////////////////////////////////////////////////////////--------
------------------------------------------------------------------
```

13-#وارزمية تطابق الخيط الرمزي:

```
def brute_force_match(text, pattern):
```

```
    m = len(text)
    n = len(pattern)

    for i in range(m - n + 1):
        j = 0
        while j < n and pattern[j] == text[i + j]:
            j += 1
        if j == n:
            return i  # الموقع الذي يبدأ فيه التطابق

    return -1  # في حال عدم وجود تطابق

# استخدام الدالة
text = input("enter text: ")
pattern = input("enter text compart: ")
location = brute_force_match(text, pattern)

if location != -1:
    print(f"yes text find  {location}")
else:
     print("no text find")
```

١٤-خوارزمية ايجاد الزوج الاقرب :

```
import math

def closest_pair(points):
    n = len(points)
    d = float('inf')  # تعيين المسافة القصوى كبداية

    for i in range(n - 1):
        for j in range(i + 1, n):
            distance = math.sqrt((points[i] - points[j]) ** 2)
            d = min(d, distance)

    return d

# استخدام الدالة
points = [int(x) for x in input("enter point must space:").split()]
min_distance = closest_pair(points)
 print( min_distance)
```

١٥-الترتيب بالدمج:

```
###خوارزمية الترتيب بالدمج


def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
```

```
            i += 1
        else:
            result.append(right[j])
            j += 1

    # إضافة العناصر المتبقية في كل جزء
    result.extend(left[i:])
    result.extend(right[j:])
    return result

def mergesort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = mergesort(arr[:mid])
    right = mergesort(arr[mid:])
    return merge(left, right)

# استخدام الدالة
arr = [int(x) for x in input("أدخل العناصر التي تريد ترتيبها مفصولة بمسافات: ").split()]
sorted_arr = mergesort(arr)
print("العناصر بعد الترتيب:", sorted_arr)
##////////////////////////////////////////////////////////////////////////////
////////////////////////////
```

١٦-خوارزمية التريب السريع:

```
##خوارزمية الترتيب السريع

def partition(arr, low, high):
    pivot = arr[low]
    s = low

    for i in range(low + 1, high + 1):
        if arr[i] < pivot:
            s += 1
            arr[s], arr[i] = arr[i], arr[s]

    arr[low], arr[s] = arr[s], arr[low]
    return s

def quicksort(arr, low, high):
    if low < high:
        s = partition(arr, low, high)
        quicksort(arr, low, s - 1)
        quicksort(arr, s + 1, high)

# استخدام الدالة
arr = [int(x) for x in input("أدخل العناصر التي تريد ترتيبها مفصولة بمسافات: ").split()]
quicksort(arr, 0, len(arr) - 1)
print("العناصر بعد الترتيب:", arr)

##////////////////////////////////////////////////////////////////////////////
////////////////////
```

١٧-خوارزمية البحث الثنائي: تعاوديا

```
##خوارزمية البحث الثنائي بالاستدعاء الذاتي
def binary_search(arr, left, right, k):
    if left > right:
        return -1  # العنصر غير موجود

    mid = (left + right) // 2
```

```
    if arr[mid] == k:
        return mid  # تم العثور على العنصر
    elif arr[mid] > k:
        return binary_search(arr, left, mid - 1, k)  # البحث في النصف الأيسر
    else:
        return binary_search(arr, mid + 1, right, k)  # البحث في النصف الأيمن

# استخدام الدالة
arr = [int(x) for x in input("أدخل العناصر المرتبة التي تريد البحث فيها مفصولة بمسافات:
").split()]
k = int(input("أدخل العنصر الذي تريد البحث عنه: "))

location = binary_search(arr, 0, len(arr) - 1, k)
if location != -1:
    print(f"موقع العنصر هو: {location}")
else:
    print("لم يتم العثور على العنصر")
##////////////////////////////////////////////////////////////////////////////
////////////////////
```

١٨-خوارزمية االبحث الخطي:

```
##خوارزمية البحث الخطي التكراري
def linear_search(arr, k):
    for element in arr:
        if element == k:
            return True  # تم العثور على العنصر
    return False  # لم يتم العثور على العنصر

# استخدام الدالة
arr = [int(x) for x in input("أدخل عناصر المصفوفة مرتبة بفواصل: ").split()]
k = int(input("أدخل العنصر المراد البحث عنه: "))

if linear_search(arr, k):
    print("العنصر موجود في المصفوفة.")
else:
   print("العنصر غير موجود.")
```

```
##////////////////////////////////////////////////////////////////////////////
//
```

١٩-خوارزمية لحساب الاس: تعاوديا:

```
###خوارزمية لحساب الاس  الية العمل التعاودي
def exp(a, n):
    if n == 0:
        return 1
    else:
        return exp(a, n - 1) * a

# الإدخال
a = int(input("Enter the number: "))
n = int(input("Enter the exponent: "))

print("=======================================")
print("The result =", exp(a, n))
##////////////////////////////////////////////////////////////////////////////
////////////////////////////
```

٢٠-حساب الاس باستخدام تقنية التقليل والفتح:

```python
#حساب الاس بعامل ثابت بأستخدام تقنية التقليل والفتح
def exp(a, n):
    if n == 0:
        return 1
    if n == 1:
        return a
    else:
        half_exp = exp(a, n // 2)
        return half_exp * half_exp

# الإدخال
a = int(input("Enter the number: "))
n = int(input("Enter the exponent: "))

print("=======================================")
print("The result =", exp(a, n))
##////////////////////////////////////////////////////////////////////
```

٢١-خوارزمية الترتيب بالحشو:تعاوديا

```python
##خوارزمية الترتيب بالحشو التعاودي
def insert(A, n):
    key = A[n]
    j = n - 1
    while j >= 0 and key < A[j]:
        A[j + 1] = A[j]
        j -= 1
    A[j + 1] = key

def insertion_sort_rec(A, n):
    if n > 0:
        insertion_sort_rec(A, n - 1)
        insert(A, n)

# البرنامج الرئيسي
n = int(input("Enter size of Array: "))
A = [int(input("Enter element: ")) for _ in range(n)]
print("================================")

insertion_sort_rec(A, n - 1)
print("The elements of the array are:")
print(" ".join(map(str, A)))
##////////////////////////////////////////////////////////////////////
////////////////////
```

٢٢-خوارزمية البحث بالعمق أولا: تعاوديا

```python
##البحث بالعمق اولا


n = int(input("Enter number of nodes n: "))
k = int(input("Enter number of edges k: "))
print("============================")

# إنشاء مصفوفة تمثل الجيران (الحواف)
graph = [[] for _ in range(n+1)]
visited = [False] * (n+1)
ns = 0

def dfs(node):
    visited[node] = True
    for neighbor in graph[node]:
        if not visited[neighbor]:
            dfs(neighbor)
```

```
# قراءة الحواف
for _ in range(k):
    x = int(input("Enter x: "))
    y = int(input("Enter y: "))
    print("============================")
    graph[x].append(y)
    graph[y].append(x)

# تنفيذ DFS للعقد غير المزارة
for i in range(1, n+1):
    if not visited[i]:
        dfs(i)
        ns += 1

print("\nNumber of connected components:", ns - 1)
print("============================")
##////////////////////////////////////////////////////////////////////////
//////////////////////////
```

٢٣-خوارزمية اجتياز الشجرة السابق والداخل والاحق: تعاوديا

```
#خوارزمية اجتياز الشجرة السابق والداخل واللاحق
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# اجتياز الشجرة بالترتيب داخل (Inorder)
def inorder(root):
    if root:
        inorder(root.left)
        print(root.data, end=" ")
        inorder(root.right)

# اجتياز الشجرة بترتيب لاحق (Postorder)
def post_order(root):
    if root:
        post_order(root.left)
        post_order(root.right)
        print(root.data, end=" ")

# اجتياز الشجرة بترتيب سابق (Preorder)
def pre_order(root):
    if root:
        print(root.data, end=" ")
        pre_order(root.left)
        pre_order(root.right)

# إنشاء العقد (Node) وإضافة الب يانات
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(8)

# تنفيذ الاجتيازات المختلفة
print("inorder traversal:")
inorder(root)
print("\npostorder traversal:")
post_order(root)
print("\npreorder traversal:")
```

```
pre_order(root)
##/////////////////////////////////////////
```

٢٤-خوارزمية البحث لجاوس : تكراريا

```
##خوارزمية الحذف لجاوس
def gauss_elimination(matrix, n):
    # تحويل المصفوفة إلى مصفوفة مثلثية علوية
    for i in range(n):
        for j in range(i + 1, n):
            factor = matrix[j][i] / matrix[i][i]
            for k in range(i, n + 1):
                matrix[j][k] -= factor * matrix[i][k]

    # الحل العكسي لإيجاد قيم المتغيرات
    x = [0 for _ in range(n)]
    for i in range(n - 1, -1, -1):
        x[i] = matrix[i][n] / matrix[i][i]
        for j in range(i - 1, -1, -1):
            matrix[j][n] -= matrix[j][i] * x[i]

    return x

# إدخال البيانات
n = int(input("Enter number of variables (n): "))
matrix = []

print("Enter the augmented matrix:")
for i in range(n):
    row = list(map(float, input().split()))
    matrix.append(row)

# تطبيق الحذف لجاوس
solution = gauss_elimination(matrix, n)

# طباعة الحل
print("Solution:")
for i in range(n):
    print(f"x{i + 1} = {solution[i]}")

##/////////////////////////////////////////////////////////////////////////
//////////////////////
```

25–قاعدة هورنر :تكراريا

```
#قاعدز هورنر

def horner(coefficients, n, x):
    result = coefficients[n]
    for i in range(n - 1, -1, -1):
        result = result * x + coefficients[i]
    return result

# إدخال البيانات
n = int(input("Enter the number of coefficients: "))
coefficients = []
print("Enter the coefficients:")
for _ in range(n + 1):
    coefficients.append(float(input()))

x = float(input("Enter the value of x: "))

# حساب قيمة متعددة الحدود باستخدام قاعدة هورنر
```

```
value = horner(coefficients, n, x)
print("The value of the polynomial:", value)
##////////////////////////////////////////////////////////////////////////
////////////////////////////
```

٢٦-خوارزمية بناء الكومة من اسفل لأعلئ: تكراريا

```
##بناء الكومة من اسفل لاعلئ

def left(i):
    return 2 * i

def right(i):
    return 2 * i + 1

def max_heapify(H, i, n):
    largest = i
    l = left(i)
    r = right(i)

    if l <= n and H[l] > H[i]:
        largest = l
    if r <= n and H[r] > H[largest]:
        largest = r

    if largest != i:
        H[i], H[largest] = H[largest], H[i]
        max_heapify(H, largest, n)

def build_heap_bottom_up(H, n):
    for i in range(n // 2, 0, -1):
        max_heapify(H, i, n)

# إدخال البيانات
n = int(input("How many elements do you want: "))
H = [0] + [int(input(f"Enter element {i+1}: ")) for i in range(n)]

build_heap_bottom_up(H, n)

print("\n========================================")
print("The elements of Heap are:")
print(" ".join(map(str, H[1:])))
print("========================================")
###////////////////////////////////////////////////////////////////////////
//////////////////////////
```

٢٧- إضافة عنصر والحذف علئ الكومة: تكراريا

```
##اضافة عنصر والحذف علئ الكومة
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[l] > arr[largest]:
        largest = l

    if r < n and arr[r] > arr[largest]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)
```

```python
def delete_root(arr, n):
    arr[0] = arr[n - 1]
    n -= 1
    heapify(arr, n, 0)
    return n

def insert_node(arr, n, key):
    n += 1
    arr.append(key)
    i = n - 1
    while i != 0 and arr[(i - 1) // 2] < arr[i]:
        arr[i], arr[(i - 1) // 2] = arr[(i - 1) // 2], arr[i]
        i = (i - 1) // 2

def print_array(arr):
    print(" ".join(map(str, arr)))

# اختبار الكود
n = int(input("How many numbers do you want: "))
arr = [int(input(f"Enter element {i + 1}: ")) for i in range(n)]

print("Array before any operation:")
print_array(arr)

key = 18
insert_node(arr, n, key)
n += 1
print("Array after inserting a node:")
print_array(arr)

n = delete_root(arr, n)
print("Array after deleting the root:")
print_array(arr)
             ##////////////////////////////////////////////////////////////
```

٢٨-خوارزمية lcs بالبرمجة الديناميكية:تكراريا

```python
def lcs_length(X, Y):
    m, n = len(X), len(Y)
    # إنشاء جدول ثنائي الأبعاد لتخزين أطوال الـ LCS
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # بناء الجدول باستخدام طريقة من الأسفل إلى الأعلى
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if X[i - 1] == Y[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp

# استخدام:
X = "president"
Y = "providence"
dp_table = lcs_length(X, Y)
print("LCS Length:", dp_table[-1][-1])  # إخراج طول أطول تسلسل من آخر خلية
```

٢٩-خوارزمية التتبع الخلفي :تكراريا

```
def get_lcs(dp, X, Y):
    i, j = len(X), len(Y)
    lcs_sequence = []

    # تتبع المسار من الخلية الأخيرة
    while i > 0 and j > 0:
        if X[i - 1] == Y[j - 1]:
            lcs_sequence.append(X[i - 1])
            i -= 1
            j -= 1
        elif dp[i - 1][j] > dp[i][j - 1]:
            i -= 1
        else:
            j -= 1

    # عكس التسلسل لأنه تم جمعه من النهاية للبداية
    return ''.join(reversed(lcs_sequence))

# استخدام:
lcs_seq = get_lcs(dp_table, X, Y)
 print("LCS Sequence:", lcs_seq)
```

٣٠-خوارزمية ديجسترا:

```
import heapq

def dijkstra(graph, start):
    # تهيئة المسافات لكل عقدة لتكون لا نهائية، وجعل المسافة لنقطة البداية تساوي 0
    distances = {vertex: float('infinity') for vertex in graph}
    distances[start] = 0
    priority_queue = [(0, start)]
    previous_nodes = {vertex: None for vertex in graph}

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)

        if current_distance > distances[current_vertex]:
            continue

        # تحديث المسافات للعقد المجاورة إذا كانت أقصر من المسافة الحالية
        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous_nodes[neighbor] = current_vertex
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances, previous_nodes

# دالة لإدخال المسارات من قبل المستخدم
def create_graph():
    graph = {}
    n = int(input("أدخل عدد العقد: "))
    for _ in range(n):
        node = input("أدخل اسم العقدة: ")
        graph[node] = {}
        m = int(input(f"أدخل عدد العقد المتصلة بالعقدة {node}: "))
        for _ in range(m):
```

```
            neighbor, weight = input(f"أدخل العقدة المتصلة والوزن (مثال: B 4):
").split()
            weight = int(weight)
            graph[node][neighbor] = weight
    return graph

# تنفيذ المثال
graph = create_graph()
start_node = input("أدخل عقدة البداية: ")
distances, previous_nodes = dijkstra(graph, start_node)

# عرض النتائج
print("Distances from start node:", distances)
print("Previous nodes in shortest path tree:", previous_nodes)
 #////////////////////////////////////////////////////////////////////////////
```

٣١-خوارزمية بريم:

```
import heapq

def prim_mst(graph, start):
    mst = {}  # الشجرة الناتجة
    for vertex in graph:
        mst[vertex] = None  # تهيئة MST بتعيين الوالد لكل عقدة إلى None

    # تهيئة مفتاح وأولوية كل عقدة
    key = {vertex: float('infinity') for vertex in graph}
    key[start] = 0
    priority_queue = [(0, start)]
    in_mst = set()

    while priority_queue:
        current_key, u = heapq.heappop(priority_queue)
        if u in in_mst:
            continue
        in_mst.add(u)

        for v, weight in graph[u].items():
            if v not in in_mst and weight < key[v]:
                key[v] = weight
                mst[v] = u  # تعيين u كوالد للعقدة v في الشجرة
                heapq.heappush(priority_queue, (key[v], v))

    return mst
def create_graph():
    graph = {}
    n = int(input("أدخل عدد العقد: "))
    for _ in range(n):
        node = input("أدخل اسم العقدة: ")
        graph[node] = {}
        m = int(input(f"أدخل عدد العقد المتصلة بالعقدة {node}: "))
        for _ in range(m):
            neighbor, weight = input(f"أدخل العقدة المتصلة والوزن (مثال: B 4):
").split()
            weight = int(weight)
            graph[node][neighbor] = weight
    return graph
graph = create_graph()
start_node = 'A'
```

```
mst = prim_mst(graph, start_node)

# عرض MST الناتجة
print("Minimum Spanning Tree:", mst)
```

٣٢-خوارزمية بيلمان:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices  # عدد العقد
        self.edges = []  # قائمة الحواف

    def add_edge(self, u, v, weight):
        # إضافة حافة بين u و v بوزن معين
        self.edges.append((u, v, weight))

    def bellman_ford(self, source):
        # تهيئة المسافات والأباء
        distance = [float("inf")] * self.V
        distance[source] = 0
        parent = [None] * self.V

        # تطبيق خطوة الاسترخاء لجميع الحواف |V| - 1 مرة
        for _ in range(self.V - 1):
            for u, v, weight in self.edges:
                if distance[u] != float("inf") and distance[u] + weight <
distance[v]:
                    distance[v] = distance[u] + weight
                    parent[v] = u

        # التحقق من عدم وجود دورة سالبة
        for u, v, weight in self.edges:
            if distance[u] != float("inf") and distance[u] + weight <
distance[v]:
                print("الرسم البياني يحتوي على دورة سالبة.")
                return None

        return distance, parent
g = Graph(5)
g.add_edge(0, 1, -1)
g.add_edge(0, 2, 4)
g.add_edge(1, 2, 3)
g.add_edge(1, 3, 2)
g.add_edge(1, 4, 2)
g.add_edge(3, 2, 5)
g.add_edge(3, 1, 1)

# تشغيل خوارزمية بيلمان-فورد من العقدة 0
result = g.bellman_ford(0)

if result:
    distances, parents = result
    print("المسافات من العقدة المصدر 0:", distances)
    print("الأباء في مسار الشجرة الأقصر:", parents)
```

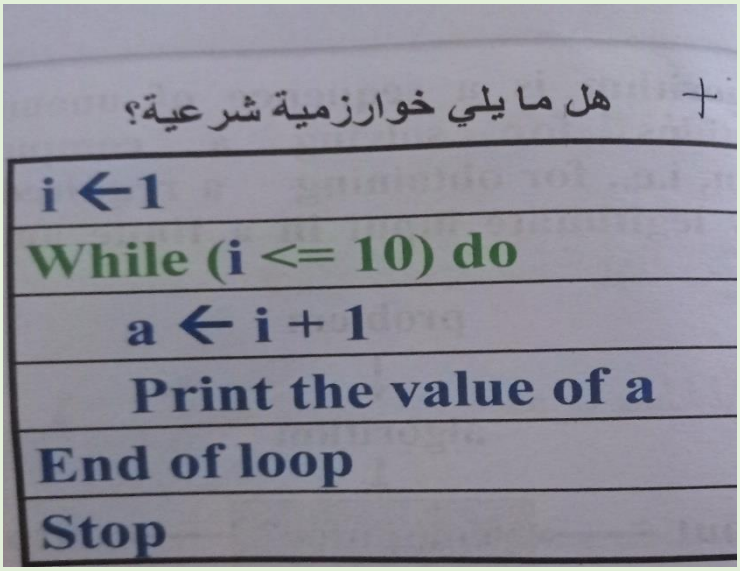
النتائج المتوقعة:

المسافات من العقدة المصدر 0: [0, -1, 2, 1, 1]

الأباء في مسار الشجرة الأقصر : [None, 0, 1, 1, 1]

ثانيا :الأسئلة والاكواد الزايفه للخوارزميات التي طلبت منا ايجادها في المحاضرات:

الفصل الأول :



الاجابه :
لا لان هذه الخوارزمية غير منتهية بسبب ثبات قيمة المتغير i مما يسبب عدم تحقق شرط التوقف

طريقة ثانية لخوارزمية اقليدس:من خلال التحقق المتتالي للاعداد الصحبحة :

```
t □ min(m,n)
while t =! 0 do
  if (t mod m = 0)
    if( t mod n = 0 )
      return t
   else
    goto s4
 else
   goto s4
s4: t□t-1
###//////////////////////////////////////
```

الشفرة الزايف لخوارزمية أقليدس من خلال المدرسة المتوسطة:

```
function gcd(m, n):
    factors_m = prime_factors(m)
    factors_n = prime_factors(n)
    common_factors = find_common_factors(factors_m, factors_n)
    gcd_result = 1
    for each factor in common_factors:
        gcd_result = gcd_result * (factor ^ min_count_in_both_lists)
    return gcd_result
                                                                    //////////////
```

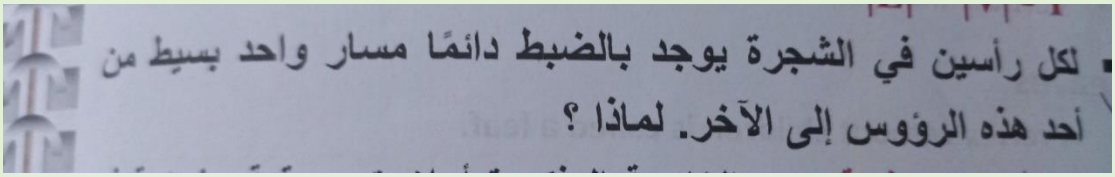
الكود التنفيذي بلغة بايثون:

```
from collections import Counter

def prime_factors(n):
    factors = []
    divisor = 2
    while n > 1:
        while n % divisor == 0:
            factors.append(divisor)
            n //= divisor
        divisor += 1
    return factors

def gcd_using_prime_factors(m, n):
    factors_m = Counter(prime_factors(m))
    factors_n = Counter(prime_factors(n))
    common_factors = factors_m & factors_n  # Get common factors with
minimum counts
    gcd_result = 1
    for factor in common_factors:
        gcd_result *= factor ** common_factors[factor]
    return gcd_result

# اختبار على الأعداد 60 و 24
print(gcd_using_prime_factors(60, 24))  # يجب أن تكون النتيجة 12
                ------------------------------------------------------------/
```



لان من شروط تسمية الرسم البياني شجرة ان يكون متصل فاي شجرة تكون متصلة

سؤال صـــــ٣٥ ــــــــ

لماذا لم يتم تضمين القيم n> 10^2 ل في الجدول للدوال من الرتب 2^n و n!
؟!

لانها قد تصل الى مليارات من السنوات بسبب التضاعف الكبير لزمن المعالجة

سؤال في المحاضرة

اكتب خوارزمية لايجاد ثاني اكبر عنصر في مصفوفة وقم بإيجاد كفاءة الخوارزمية من حيث تعقيدات الوقت والخزن ؟

خوارزمية إيجاد ثاني اكبر عنصر مصفوفة :

```
FindSecondLargest(matrix):
    max_val = -∞
    second_max = -∞

    for each row in matrix:
        for each element in row:
            if element > max_val:
                second_max = max_val
                max_val = element
            elif element > second_max and element < max_val:
                second_max = element

    return second_max
```

تعقيدات الوقت O(n)

تعقيدات الخزن O(1)

الكود بلغة بايثون :

```
def find_second_largest(matrix):
    max_val = float('-inf')
    second_max = float('-inf')

    for row in matrix:
        for element in row:
            if element > max_val:
                second_max = max_val
                max_val = element
            elif element > second_max and element < max_val:
                second_max = element

    return second_max

##مثال على الاستخدام

# 3 مصفوفة مربعةx3
matrix = [
    [1, 5, 3],
    [4, 8, 2],
    [7, 6, 9]
]
result = find_second_largest(matrix)
print("Second largest element is:", result)
```

#//////////////////////

خوارزمية لغز أبراج هانوي: الشفرة الزايفة

```
Hanoi(n, source, target, auxiliary):
    if n == 1:
        Print "Move disk 1 from", source, "to", target
    else:
        Hanoi(n - 1, source, auxiliary, target)
        Print "Move disk", n, "from", source, "to", target
        Hanoi(n - 1, auxiliary, target, source)
        #////////////////////////////////////
```

الكود بلغة بايثون :

```
def hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {target}")
    else:
        # نقل n-1 قرص من الوتد المصدر إلى الوتد المساعد
        hanoi(n - 1, source, auxiliary, target)

        # نقل القرص الأخير إلى الوتد الهدف
        print(f"Move disk {n} from {source} to {target}")

        # نقل n-1 قرص من الوتد المساعد إلى الوتد الهدف
        hanoi(n - 1, auxiliary, target, source)

# اختبار الكود بنقل 3 أقراص من الوتد A إلى الوتد C باستخدام الوتد B كمساعد
```

```
hanoi(3, 'A', 'C', 'B')
```

سؤال في المحاضرة

قم بتعديل خوارزمية إيجاد اكبر عنصر في قائمة لتوجد الخوارزمية اكبر و اصغر عنصر واحسب كفاءة الخوارزمية؟

```
Algorithem Maxminelements(A[ 0 … n-1])
//تحديد قيمة اكبر واصغر عنصر في مصفوفة
// مصفوفة من الأرقام الحقيقة  A[ 0 … n-1]
//قيمة اكبر واصغر عنصر في مصفوفة

Maxval  A[0], minval  A[0]
for I 1 to n-1 do
  if A[i]> maxval then
      maxval ← A[i]
  if A[i]< minval then
     minval ← A[i]
return maxval , minval
#////////////////////////////
```

تكليف في المحاضرة

، البحث عن المسار الأقصر " دائرة هاميلتونيان اكتب الخوارزمية + الشفرة الزائفة صـــ٦١ـ ؟؟

// إيجاد المسار الأقصر في دائرة هاميلتونيان

//المدخلات :مصفوفة A [V][V]ثنائية حيث عدد الرؤوس في الرسم البياني والرسم البياني [V] [V] هو تمثيل مصفوفة مجاورة للرسم البياني. الرسم البياني للقيمة [j] [i] هو ١ إذا كان هناك حافة مباشرة من i إلى j ، وإلا فإن الرسم البياني [j] [i] يساوي ٠.

// عدد الرؤوس في الرسم البياني والرسم البياني [V] [V] هو تمثيل مصفوفة مجاورة للرسم البياني. الرسم البياني للقيمة [j] [i] هو ١ إذا كان هناك حافة مباشرة من i إلى j ، وإلا فإن الرسم البياني [j] [i] يساوي ٠.

```
Algorithm Sortestpath(A[V][V])
 do while there are untried configuration
      generate the next configuration
   if ( there are edges between two consecutive vertices of this
      configuration and there is an edge from the last vertex to
    the first).
       print this configuration
      break
  ##////////////////////////////////////////////
```

تعقيدات الوقت O(!V) و تعقيدات الخزن O(1)

مشكلة حقيبة الظهر صـــــــــــ٦٢ـــــ

الخطوة الأولى :إيجاد كل المجموعات الجزئية التي يمكن انشاءها .

الخطوة الثانية :حساب الوزن الكلي والقيمة الكلية لكل المجموعات الجزئية.

الخطوة الثالثة:البحث في المجموعات الجزئية التي يكون وزنها الكلي أصغر أو مساوي لسعة الحقيبة
.الخطوة الرابعة :اختيار المجموعة الجزئية التي تمتلك أكبر مقدار للقيمة

Algorithm knapsack (W, wt[0..n-1], itemsval [0..n-1], n)

//المدخلات :مصفوفة اوزان العناص wtمصفوفة قيم العناصر itemsvalعدد العناصر n سعة الحقيبة w

//المخرجات : المجموعة الأكثر قيمة والتي لا تتعدى حجم الحقيبة

```
if n == 0 or W == 0 then
 return 0
 if wt[n-1] > W then
 return knapsack (W, wt, itemval, n-1)
 else return max (knapsack (W-wt[n-1], wt, val, n-1) + val[n-1], knapsack
(W, wt, val, n-1))
##////////////////////////////////////////
```

سؤال في المحاضرة

خوارزمية مشكلة التخصيص صـــــــــ٦٣ـــــ

الخطوة ١: اطرح أصغر عنصر تكلفة لكل صف من جميع العناصر الموجودة في صف مصفوفة التكلفة المحددة. تأكد من أن كل صف يحتوي على صفر واحد على الأقل.

الخطوة ٢: اطرح أصغر عنصر تكلفة لكل عمود من جميع العناصر الموجودة في عمود مصفوفة التكلفة الناتجة التي تم الحصول عليها في الخطوة ١ وتأكد من احتواء كل عمود على صفر واحد على الأقل

الخطوة ٣: (تعيين الأصفار)

(أ) افحص الصفوف على التوالي حتى يتم العثور على صف بصفر واحد غير محدد بالضبط. قم بتخصيص هذا الصفر غير المميز بتطويقه. اعبر جميع الأصفار الأخرى في عمود هذا الصفر المطوق ، حيث لن يتم أخذها في الاعتبار لأي تعيين مستقبلي. استمر بهذه الطريقة حتى يتم فحص جميع الصفوف.

(ب) افحص الأعمدة على التوالي حتى يتم العثور على عمود بصفر واحد غير محدد بالضبط. قم بتعيين تعيين لهذا الصفر الوحيد غير المميز عن طريق تطويقه وعبور أي صفر آخر في صفه. استمر حتى يتم فحص جميع الأعمدة.

الخطوة ٤: (تطبيق الاختبار الأمثل)

(أ) إذا كان كل صف وكل عمود يحتويان بالضبط على صفر واحد محاط بدائرة ، فإن التخصيص الحالي هو الأمثل.

(ب) إذا كان صف أو عمود واحد على الأقل بدون تخصيص (أي إذا كان هناك صف أو عمود واحد على الأقل بدون صفر محاط بدائرة) ، فإن التخصيص الحالي ليس هو الأمثل. انتقل إلى الخطوة ٥. اطرح أصغر عنصر تكلفة لكل عمود من جميع العناصر الموجودة في عمود مصفوفة التكلفة الناتجة التي تم الحصول عليها في الخطوة ١ وتأكد من احتواء كل عمود على صفر واحد على الأقل.

الخطوة ٥: قم بتغطية جميع الأصفار عن طريق رسم أقل عدد ممكن من الخطوط المستقيمة على النحو التالي:

(أ) حدد الصفوف التي لا تحتوي على تخصيص.

(ب) قم بتمييز الأعمدة (التي لم يتم تمييزها بالفعل) التي تحتوي على أصفار في الصفوف المحددة. (ج) ضع علامة على الصفوف (التي لم يتم تمييزها بالفعل) التي تحتوي على تخصيصات

الأعمدة المميزة. (د) كرر (ب) و (ج) حتى لا تكون هناك حاجة إلى مزيد من العلامات.

(هـ) رسم خطوط من خلال جميع الصفوف غير المميزة والأعمدة المميزة. إذا كان عدد هذه الأسطر مساويًا لترتيب المصفوفة ، فهذا هو الحل الأمثل وإلا لا.

الخطوة ٦: حدد أصغر عنصر تكلفة لا تغطيه الخطوط المستقيمة. اطرح عنصر التكلفة الأصغر هذا من جميع العناصر غير المغطاة وأضف هذا إلى كل تلك العناصر الموجودة في تقاطع هذه الخطوط المستقيمة ولا تغير العناصر المتبقية التي تقع على الخطوط المستقيمة.

• الخطوة ٧: كرر الخطوات من (١) إلى (٦) ، حتى يتم الحصول على التخصيص الأمثل.

لم استطع كتابة الطود الزائق

تعقيدات الوقت $O(n!)$

تعقيدات الخزن $O(n^2)$

الفصل السادس:

إجابات اسئلة صـــ٧٣ـــ

كم عدد مقارنات العاناصر المطلوبة ؟ Θ(n log n)

ما هو مقدار الذاكرة الإضافية المطلوبة ؟ Θ(n)

هل خوارزمية ترتيب الدمج خوارزمية في المكان؟ لا

هل الخوارزمية مستقرة ؟ نعم

إجابات أسئلة صـــ٧٦ـــ

خوارزمية التقسيم ..

هل يمكن حذف "=" من '≤' '≥' ؟

لا .لانه في حال وجود عناصر متكررة لن يتم ترتيبها بالشكل الصحيح اذا حذفنا المساواة ..

خوارزمية الترتيب السرع ..

كم عدد مقارنات العناصر المطلوبة؟

افضل حالة والحالة المتوسطة **Θ(n log n)**

الحالة الأسوأ $\Theta(n^2)$

ما هو مقدار الذاكرة االضافية المطلوب؟ **Θ(n log n)**

هل خوارزمية ترتيب الدمج خوارزمية في المكان؟ نعم

هل الخوارزمية مستقرة؟ نعم

اكتب خوارزمية نعاودية لاجتياز شجرة ثنائية بترتيب الداخل وترتيب اللاحق ؟؟

أولا بترتيب الداخل:

الخطوة الأولى : طباعة الابن الايسر للشجرة الفرعية

الخطوة الثانية : طباعة الاب للابن الذي تمت طباعته فب الخطوة ١

الخطوة الثالثة: طباعة الابن الايمن للاب الذي تمت طباعته في الخطوة الثانية

تكرار الثلاث الخطوات تعاوديا الى ان يتم المرور على جميع عقد الشجرة

ALGORITHM internal(T)

//إجتياز الشجرة بترتيب الداخل

المدخالت: شجرة ثنائية T مع تسميات لعقدها//

// المخرجات: تسميات العقد بقائمة في ترتيب الداخل

if T ≠ Θ then

internal(TL)

write label of T's root

internal(TR)

ثانيا بترتيب اللاحق:

الخطوة الأولى : طباعة الابن الايسر للشجرة الفرعية

الخطوة الثانية : طباعة الابن الأيمن للشجرة الفرعية

الخطوة الثالثة: طباعة الاب للاب الذي تمت طباعته في الخطوة ١

تكرار الثلاث الخطوات تعاوديا الى ان يتم المرور على جميع عقد الشجرة

ALGORITHM after (T)

//إجتياز الشجرة بترتيب الداخل

المدخالت: شجرة ثنائية T مع تسميات لعقدها//

// المخرجات: تسميات العقد بقائمة في ترتيب اللاحق

if T ≠ Θ then

after (TL)

write label of T's root

after (TR)

سؤال صـــــ١٢٣ــــ a^n

أولا :الاس الثنائي من اليسار الى اليمين ..

الخطوة الأولى :فحصn اس ثنائي من اليسار الى اليمين

الخطوة الثانية :اذا كان الرقم الثنائي الحالي هو ٠ يتم تربيع قيمة المجمع واذا كان الرقم الثنائي هو ١ يتم تربيع قيمة المجمع ويضرب فيa

ALGORITHM powfrom left to right(a, b(n))

// حساب a^n ايستخدام التعبير من اليسار الى اليمن

// المدخلات : رقم a وقائمة b(n) من الأرقام الثنائية في التوسع الثنائي لعدد صحيح

// قيمة a^n

for i ← l − 1 downto 0 do

mult ← mult ∗ mult

if b[i] = 1

mult ← mult ∗ a

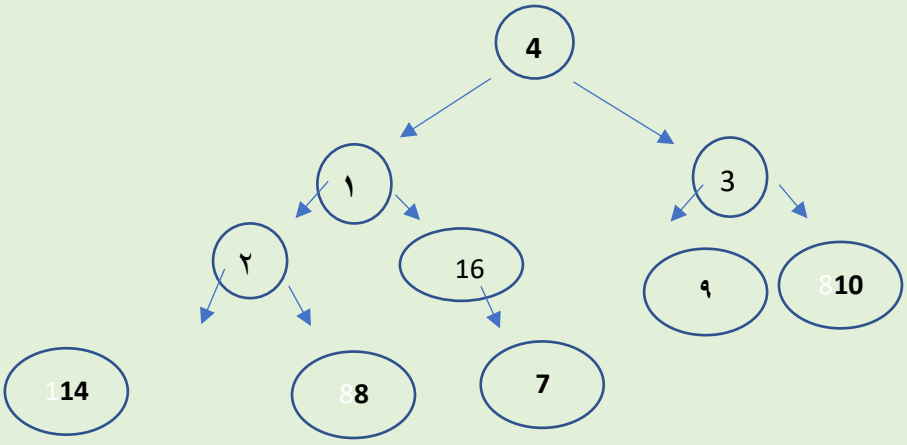
return mult----------

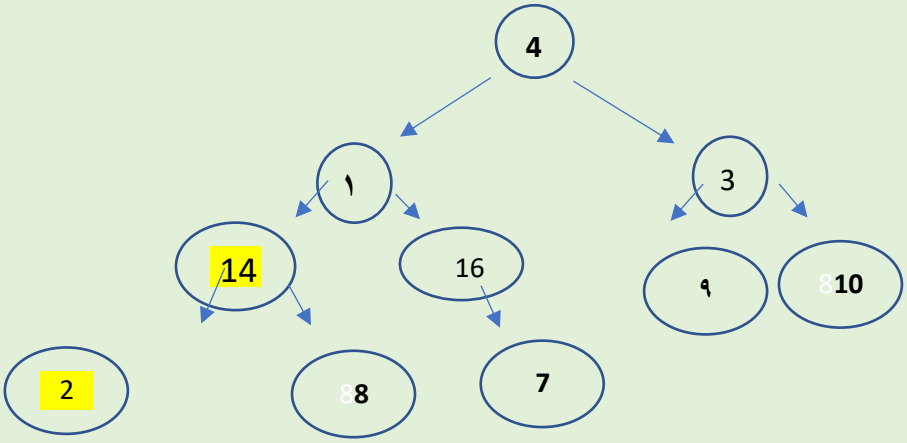
أنشاء كومة من القائمة التالية

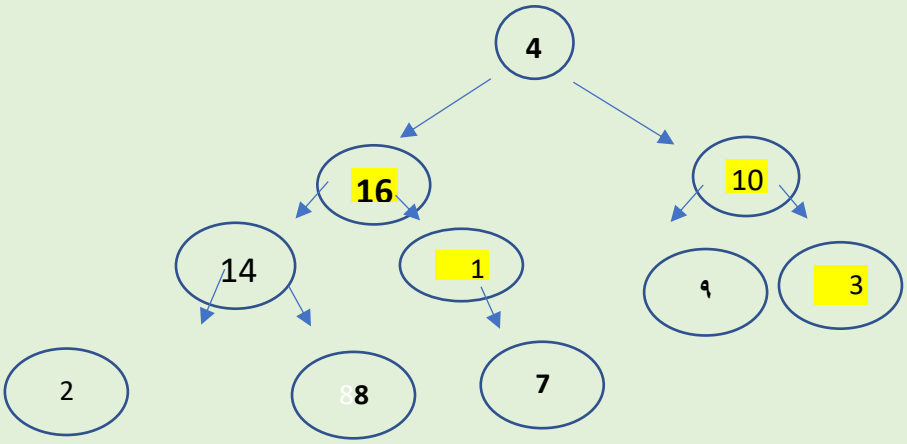
7 8 14 10 9 16 2 3 1 4

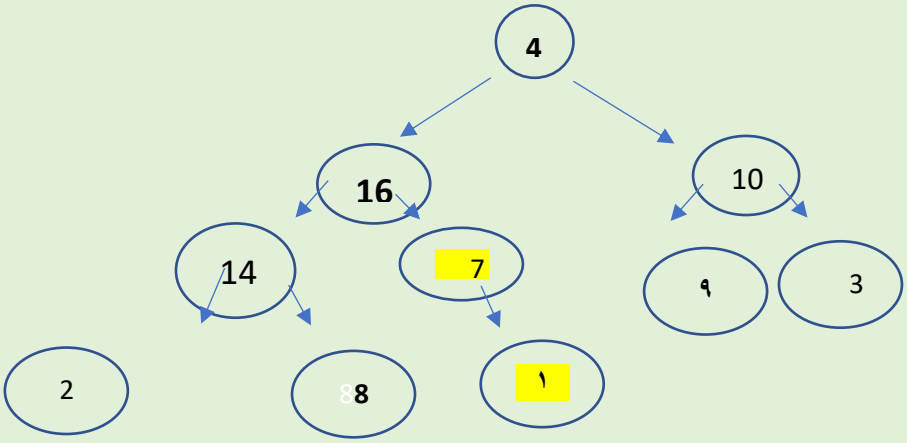
سؤال

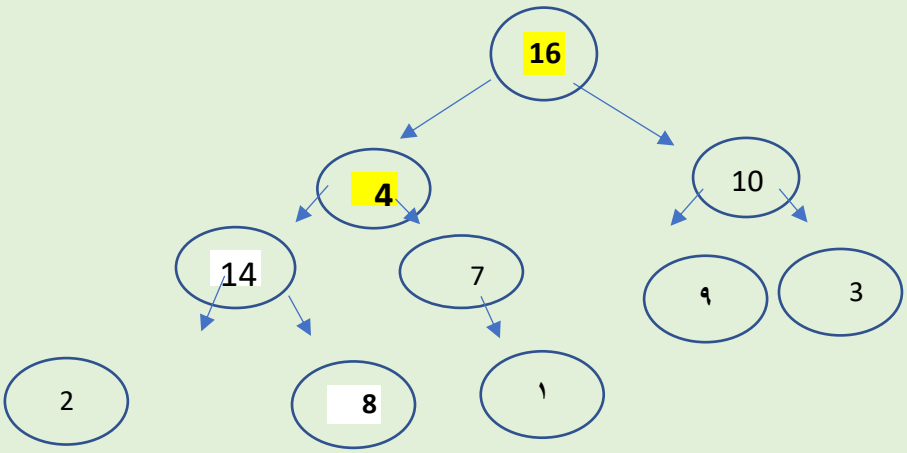
صـــ١٢٩ــــ

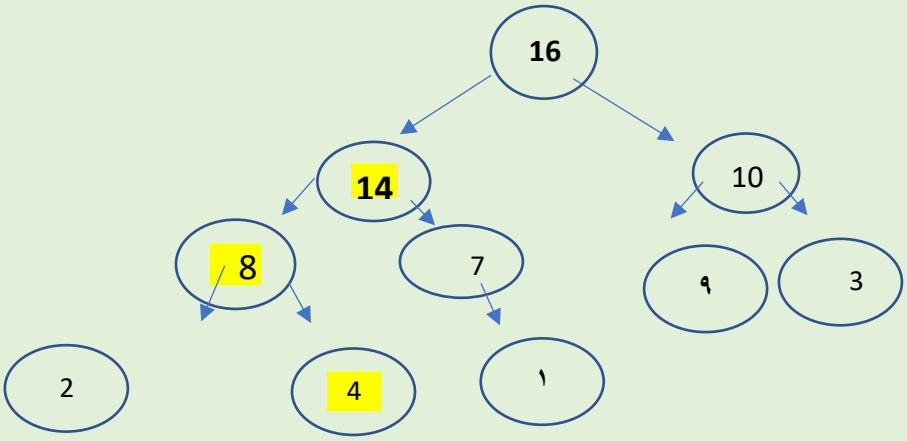





4
16
10
14
1
٩
3
2
8
7


4
16
10
14
7
٩
3
2
8
١


16
4
10
14
7
٩
3
2
8
١



الناتج النهائي

إجابات أسئلة صـــــــ١٣٣ و ١٣٤ــــــــــ

إضافة عنصر للكومة:

الخطوة الأولى :اضف العنصر في اخر موقع في الكومة.

الخطوة الثانية :قارن العنصر المضاف مع والده، و قم بالتبديل بينهما إذا كان ينتهك شرط الهيمنة الابوية

الخطوة الثالثة :استمر في مقارنة العنصر الجديد بالعقد الموجودة أعلى الشجرة حتى يتم استيفاء حالة الهيمنة الابوية.

```
Algorithm AddNewElementHeap (H[1..n], element)
// إضافة عنصر جديد إلى الكومة
// المدخالت: مصفوفة عناصر الكومة والعنصر المراد اضافته
// المخرجات: مصفوفة عناصر الكومة مضافة إليها العنصر الجديد
H[n+1] = element
for i= (n+1)/2 to 1 do
(i, H (MaxHeapify  // إعادة ترتيب عناصر الكومة
return H[1..n+1]
```

حذف الجذر من الكومة :

الخطوة الأولى : استبدل الجذر بالورقة ألاخيرة.

الخطوة الثانية : قلل حجم الكومة بـ ١

الخطوة الثالثة : كوم الشجرة ألصغر بنفس طريقةMax ()

Algorithm DeleteRootHeap (H[1..n]) حذف الجذر من الكومة

// حذف الجذر من الكومة

// المدخالت: مصفوفة عناصر الكومة

// المخرجات: مصفوفة عناصر الكومة محذوف منها الجذر السابق

swap H[1] and H[n]

delete H[n]

for i← (n-1)/2 to 1 do

MaxHeapify (H[1..n-1], i) // إ عناصر ترتيب الكومة

ترتيب الكومة

:أولا :أنشيء الكومة لمصفوفة مع ينة اما بطريقة اسفل- العلى او بطريقة اعلى-السفل

ثانيا :طبق عملية حذف الجذر1-n مرة على الكومة المتبقية حتى تحتوي الكومة على غقدة واحدة فقط.

Algorithm OrderHeap (H[1..n]) ترتيب عناصر الكومة

// ترتيب الكومة

//المدخالت: مصفوفة عناصر الكومة

//المخرجات: عناصر الكومة مرتبة في المصفوفة result

HeapBottomUp (H[1..n])

m = n-1

for i←1 to m do

result [i] = H[1]

H = DeleteRootHeap (H)

result [n] = H[1]

return result

إجابة سوال ١٤٠

حساب lcm (n,m) المضاعف المشترك ألصغر (مقارنة بحساب gcd (n,m) القاسم المشترك الأعظم

نقوم بإيجاد المضاعف المشترك ألصغر عن طريق ضرب العددين والقسمة عل ى القاسم المشترك ألعظم

Algorithm lcm(m,n)

\\خوارزمية ايجاد المضاعف المشترك ألصغر بواسطة القاسم المشترك ألعظم

المدخالت: عددين صحيحين m و\\ n

المخرجات: المضاعف المشترك ألصغرLcm

Lcm= (m*n)/gcd(m,n)

Return lcm

إجابة سؤال صـــــــــ١٧٢ـــــ

هل الخوارزمية الجشعة دائما تعطي الحلول المثلى ؟

ليس بالضرورة ان تعطي الخزارومية الجشعة الحلول المثلى فهو على حسب الهدف الذي صممت من اجله تأخذ المسار مثلا لو كان الهدف المدينة الأقرب فسيتم تجاهل مدن من المفترض زيارتها فقط لسبب وجود مدينة اقرب ويتم زيارة هذه المدن المتجاهلة أخير ومن الممكن اتخاذ طريق أطول في حال لو تم زيارتها مسبقا كان الطريق ليصبح اقصر