

Practical - 6

Aim:- Programs to select the Nth Max/Min element in a list by using various algorithms. Compare the efficiency of algorithms.

Description:- Write an algorithm.

- 1) If n is odd then initialize min and max as the first element.
- 2) If n is even then initialize min and max as minimum and maximum of the first two elements respectively.
- 3) For the rest of the elements, pick them in pairs and compare their maximum and minimum with max and min respectively.

Code:-

```
def getMinMax(arr):  
    n = len(arr)  
    if (n % 2 == 0):  
        mx = max(arr[0], arr[1])  
        mn = min(arr[0], arr[1])  
        i = 2  
    else:  
        mx = mn = arr[0]  
        i = 1  
    while (i < n - 1):  
        if arr[i] < arr[i + 1]:
```

```
mx = max(mx, arr[i+1])  
mn = min(mn, arr[i])
```

```
else:
```

```
mx = max(mx, arr[i])
```

```
mn = min(mn, arr[i+1])
```

```
i+1
```

```
return (mx, mn)
```

```
if __name__ == "__main__":
```

```
arr = [1000, 11, 445, 1, 330, 3000]
```

```
mx, mn = getMinMax(arr)
```

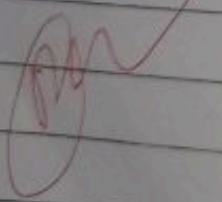
```
print("Minimum element is", mn)
```

```
print("Maximum element is", mx)
```

OUTPUT:-

Minimum element is 1

Maximum element is 3000



Practical - 7

- *) Aim:- Programs to find a pattern in a given string - general way and brute force technique. Compare the efficiency of algorithms.

Description :- Write an algorithm.

- 1) Start
- 2) Define the pattern that you are searching for.
- 3) Loop through each character in the string.
- 4) For each character, compare it to the first character in the string and pattern is fully matched.
- 5) If the character match continue to compare subsequent character in the string is fully matched.
- 6) If the pattern is fully matched, print it.
- 7) Stop.

Code :-

```
n = int(input("Enter the number of cities :"))
city = ()
for i in range(n):
    (input("Enter city :"))
    city += (c,)
print(city)
put = input("Enter pattern you want to search for ?")
for c in city:
    if (find(put) != 1):
        print(c)
```

OUTPUT :-

Enter number of cities : 1

Enter city : Mumbai
(Mumbai)

Enter pattern you want to search for ?

Mumbai

Practical - 8

Aim:- Programs on recursion, like factorial, fibonacci, tower of Hanoi. Compare algorithms to find factorial / fibonacci using iterative and recursive approaches.

Description:- Write an Algorithm

- 1) Create a function TowerOfHanoi where pass the N (current number of disk), from_rod, to_rod, aux_rod.
- 2) Make a function call for N - 1th disk.
- 3) Then print the current the disk along with from_rod and to_rod.
- 4) Again make a function call for N - 1th disk.

Code:-

```
# tower of hanoi
def TowerOfHanoi(n, from_rod, to_rod, aux_rod):
    if n == 1:
        print("Move disk 1 from rod", from_rod, "to rod",
              to_rod)
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print("Move disk", n, "from rod", from_rod, "to rod",
          to_rod)
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)

# main
n = 3
```



```
TowerOfHanoi(n, 'A', 'C', 'B')  
#A, B, C are the rod  
print("Sorted array is:")  
for i in range(n):  
    print(arr[i], end = " ")
```

OUTPUT:-

Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C

Practical - 9

Aim:- Program to implement merge sort, Strassen's Matrix Multiplication using D-n-C Algorithm and to understand time complexity.

Description:- Write an algorithm.

- 1) If the size of the matrix is 1 then multiply the two matrix and return the result.
- 2) Divide each matrix into four submatrix each of size $n/2$.
- 3) Compute the following seven product.

Code:-

```
import numpy as np
def strassen_algorithm(x, y):
    if x.size == 1 or y.size == 1:
        return x * y
    n = x.shape[0]

    if n % 2 == 1:
        x = np.pad(x, (0, 1), mode = 'constant')
        y = np.pad(y, (0, 1), mode = 'constant')

    m = int(np.ceil(n / 2))
    a = x[:m, :m]
    b = x[:m, m:]
    c = x[m:, :m]
```



```

d = x[m:, m:]
e = y[:, m:]
f = y[:, m:]
g = y[m:, m:]
h = y[m:, m:]
p1 = strassen_algorithm(a, f - h)
p2 = strassen_algorithm(a + b, h)
p3 = strassen_algorithm(c + d, e)
p4 = strassen_algorithm(d, g - e)
p5 = strassen_algorithm(a + d, e + h)
p6 = strassen_algorithm(b - d, g + h)
p7 = strassen_algorithm(a - c, e + f)
result = np.zeros((2 * m, 2 * m), dtype=np.int32)

```

```

result[:m, :m] = p5 + p4 - p2 + p6
result[:, m:] = p1 + p2
result[m:, :m] = p3 + p4
result[m:, m:] = p1 + p5 - p3 - p7

```

```

return result[:n, :n]

```

```

if __name__ == "__main__":

```

```

x = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
y = np.array([[1, 0, 0], [0, -1, 0], [0, 0, -1]])
print('Matrix multiplication result:')
print(strassen_algorithm(x, y))

```

OUTPUT :-

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$