LAB-7

DATE:13/06/2024

1. Finding the maximum and minimum

CODING:

```
3 usages
def max_min_divide_conquer(arr, low, high):
    class Pair:
        def __init__(self):
            self.max = 0
            self.min = 0
    result = Pair()
    if low == high:
        result.max = arr[low]
        result.min = arr[low]
        return result
    if high == low + 1:
        if arr[low] < arr[high]:
            result.min = arr[low]
            result.max = arr[high]
        else:
            result.min = arr[high]
            result.max = arr[low]
        return result
    mid = (low + high) // 2
    left = max_min_divide_conquer(arr, low, mid)
    right = max_min_divide_conquer(arr, mid + 1, high)

    result.max = max(left.max, right.max)

    result.min = min(left.min, right.min)
    return result
arr = [6, 4, 26, 14, 33, 64, 46]
result = max_min_divide_conquer(arr, low: 0, len(arr) - 1)
print("Maximum element is:", result.max)
print("Minimum element is:", result.min)
```

OUTPUT:

```
C:\Users\vinot\PycharmProjects\pythonProject
Maximum element is: 64
Minimum element is: 4

Process finished with exit code 0
```

2. Merge sort

CODING:

```
1 usage
def merge(array, left, mid, right):
    subArrayOne = mid - left + 1
    subArrayTwo = right - mid
    leftArray = [0] * subArrayOne
    rightArray = [0] * subArrayTwo
    for i in range(subArrayOne):
        leftArray[i] = array[left + i]
    for j in range(subArrayTwo):
        rightArray[j] = array[mid + 1 + j]
    indexOfSubArrayOne = 0
    indexOfSubArrayTwo = 0
    indexOfMergedArray = left

    while indexOfSubArrayOne < subArrayOne and indexOfSubArrayTwo < subArrayTwo:
        if leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]:
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne]
        if leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]:
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne]
            indexOfSubArrayOne += 1
        else:
            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo]
            indexOfSubArrayTwo += 1
        indexOfMergedArray += 1

    while indexOfSubArrayOne < subArrayOne:
        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne]
        indexOfSubArrayOne += 1
        indexOfMergedArray += 1
    while indexOfSubArrayTwo < subArrayTwo:
        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo]
        indexOfSubArrayTwo += 1
        indexOfMergedArray += 1
```

```
def mergeSort(array, begin, end):
    if begin >= end:
        return
    mid = begin + (end - begin) // 2
    mergeSort(array, begin, mid)
    mergeSort(array, mid + 1, end)
    merge(array, begin, mid, end)
2 usages
def printArray(array, size):
    for i in range(size):
        print(array[i], end=" ")
    print()
if __name__ == "__main__":
    arr = [12, 11, 13, 5, 6, 7]
    arr_size = len(arr)
    print("Given array is")
    printArray(arr, arr_size)
    print("\nSorted array is")
    printArray(arr, arr_size)
```

OUTPUT:

```
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13

Process finished with exit code 0
```

3. QUICK SORT

CODING:

```
def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])
    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1
3 usages
def quicksort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quicksort(array, low, pi - 1)
        quicksort(array, pi + 1, high)
if __name__ == '__main__':
    array = [10, 7, 8, 9, 1, 5]
    N = len(array)
    quicksort(array, low: 0, N - 1)
    print('Sorted array:')
    for x in array:
        print(x, end=" ")
```

OUTPUT:

```
Sorted array:
1 5 7 8 9 10
Process finished with exit code 0
```

4. Binary search

CODING:

```
def binarySearch(arr, low, high, x):
    while low <= high:
        mid = low + (high - low) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
if __name__ == '__main__':
    arr = [2, 3, 4, 10, 40]
    x = 10
    result = binarySearch(arr, low: 0, len(arr)-1, x)
    if result != -1:
        print("Element is present at index", result)
```

```
    if result != -1:
        print("Element is present at index", result)
    else:
        print("Element is not present in array")
```

OUTPUT:

```
Element is present at index 3

Process finished with exit code 0
```

5. Strassens matrix multiplication

CODING:

```
import numpy as np

def strassen(A, B):
    n = len(A)

    if n <= 2:  # Base case
        return np.dot(A, B)

    # Partition matrices into submatrices
    mid = n // 2
    A11 = A[:mid, :mid]
    A12 = A[:mid, mid:]
    A21 = A[mid:, :mid]
    A22 = A[mid:, mid:]
    B11 = B[:mid, :mid]
    B12 = B[:mid, mid:]
    B21 = B[mid:, :mid]
    B22 = B[mid:, mid:]

    # Recursive multiplication
    P1 = strassen(A11, B12 - B22)
    P2 = strassen(A11 + A12, B22)
    P3 = strassen(A21 + A22, B11)
    P4 = strassen(A22, B21 - B11)
    P5 = strassen(A11 + A22, B11 + B22)
    P6 = strassen(A12 - A22, B21 + B22)
    P7 = strassen(A11 - A21, B11 + B12)

    # Combine results to form C
    C11 = P5 + P4 - P2 + P6
    C12 = P1 + P2
    C21 = P3 + P4
    C22 = P5 + P1 - P3 - P7
```

```
    # Combine results to form C
    C11 = P5 + P4 - P2 + P6
    C12 = P1 + P2
    C21 = P3 + P4
    C22 = P5 + P1 - P3 - P7

    # Combine quadrants to form C
    C = np.vstack((np.hstack((C11, C12)), np.hstack((C21, C22))))
    return C

# Example usage:
A = np.array([[1, 3], [7, 5]])
B = np.array([[6, 8], [4, 2]])
C = strassen(A, B)
print("Matrix C (Result of A * B):\n", C)
```

OUTPUT:

```
Matrix C (Result of A * B):
 [[18 14]
 [62 66]]
```

6. Karatsuba algorithm for multiplication

CODING:

```python
import math
4 usages
def karatsuba(X, Y):
    if X < 10 and Y < 10:
        return X * Y
    size = max(get_size(X), get_size(Y))
    if size < 10:
        return X * Y
    size = (size // 2) + (size % 2)
    multiplier = 10 ** size
    b = X // multiplier
    a = X - (b * multiplier)
    d = Y // multiplier
    c = Y - (d * size)
    u = karatsuba(a, c)
    z = karatsuba(a + b, c + d)
    v = karatsuba(b, d)
```

```
    d = Y // multiplier
    c = Y - (d * size)
    u = karatsuba(a, c)
    z = karatsuba(a + b, c + d)
    v = karatsuba(b, d)
    return u + ((z - u - v) * multiplier) + (v * (10 ** (2 * size)))
2 usages
def get_size(value):
    count = 0
    while value > 0:
        count += 1
        value //= 10
    return count
x = 145623
y = 653324
print("The final product is: ", end="")
product = karatsuba(x, y)
print(product)
```

OUTPUT:

```
The final product is: 95139000852

Process finished with exit code 0
```

7. Closest pair of points using divide and conquer

CODING:

```
import math
2 usages
def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
1 usage
def strip_closest(strip, d):
    min_dist = d
    strip.sort(key=lambda point: point[1])
    for i in range(len(strip)):
        for j in range(i + 1, len(strip)):
            if (strip[j][1] - strip[i][1]) < min_dist:
                min_dist = min(min_dist, distance(strip[i], strip[j]))
            else:
                break
    return min_dist
3 usages
def closest_util(points_sorted_by_x):
```

```
def closest_util(points_sorted_by_x):
    if len(points_sorted_by_x) <= 3:
        min_dist = float('inf')
        for i in range(len(points_sorted_by_x)):
            for j in range(i + 1, len(points_sorted_by_x)):
                min_dist = min(min_dist, distance(points_sorted_by_x[i], points_sorted_by_x[j]))
        return min_dist
    mid = len(points_sorted_by_x) // 2
    mid_point = points_sorted_by_x[mid]
    dl = closest_util(points_sorted_by_x[:mid])
    dr = closest_util(points_sorted_by_x[mid:])
    d = min(dl, dr)
    strip = []
    for point in points_sorted_by_x:
        if abs(point[0] - mid_point[0]) < d:
            strip.append(point)
```

```
        if abs(point[0] - mid_point[0]) < d:
            strip.append(point)
    return min(d, strip_closest(strip, d))
1 usage
def closest_pair_of_points(points):
    points_sorted_by_x = sorted(points, key=lambda point: point[0])
    return closest_util(points_sorted_by_x)
points = [(2.1, 3.2), (3.4, 5.1), (5.2, 2.8), (1.3, 4.7), (4.1, 1.2), (2.5, 3.9)]
print("The smallest distance is:", closest_pair_of_points(points))
```

OUTPUT:

```
The smallest distance is: 0.8062257748298546

Process finished with exit code 0
```

8. Median of medians

CODING:

```
def partition(arr, low, high, pivot):
    pivot_value = arr[pivot]
    arr[pivot], arr[high] = arr[high], arr[pivot]
    store_index = low
    for i in range(low, high):
        if arr[i] < pivot_value:
            arr[store_index], arr[i] = arr[i], arr[store_index]
            store_index += 1
    arr[store_index], arr[high] = arr[high], arr[store_index]
    return store_index
6 usages
def select(arr, low, high, k):
    if low == high:
        return arr[low]
    pivot_index = median_of_medians(arr, low, high)
    pivot_index = partition(arr, low, high, pivot_index)
```

```
        pivot_index = partition(arr, low, high, pivot_index)
        if k == pivot_index:
            return arr[k]
        elif k < pivot_index:
            return select(arr, low, pivot_index - 1, k)
        else:
            return select(arr, pivot_index + 1, high, k)
    1 usage
    def median_of_medians(arr, low, high):
        n = high - low + 1
        if n <= 5:
            return partition5(arr, low, high)
        medians = []
        for i in range(low, high + 1, 5):
            sub_right = min(i + 4, high)
            median5 = partition5(arr, i, sub_right)
            medians.append(arr[median5])
        median_of_medians_index = select(medians, low: 0, len(medians) - 1, len(medians) // 2)
        return arr.index(median_of_medians_index)
    2 usages
    def partition5(arr, low, high):
        i = low + 1
        while i <= high:
            j = i
            while j > low and arr[j - 1] > arr[j]:
                arr[j - 1], arr[j] = arr[j], arr[j - 1]
                j -= 1
            i += 1
        return (low + high) // 2
    1 usage
    def find_median(arr):
    def find_median(arr):
        n = len(arr)
        if n % 2 == 1:
            return select(arr, low: 0, n - 1, n // 2)
        else:
            return 0.5 * (select(arr, low: 0, n - 1, n // 2 - 1) + select(arr, low: 0, n - 1, n // 2))
    arr = [12, 3, 5, 7, 4, 19, 26]
    print("Median is:", find_median(arr))
```

Output:

```
Median is: 7

Process finished with exit code 0
```

9. Meet in middle technique

Coding:

```python
from itertools import combinations
2 usages
def subset_sums(arr):
    subset_sum_list = []
    n = len(arr)
    for i in range(n + 1):
        for combo in combinations(arr, i):
            subset_sum_list.append(sum(combo))
    return subset_sum_list
1 usage
def meet_in_the_middle(arr, target):
    n = len(arr)
    left_half = arr[:n // 2]
    right_half = arr[n // 2:]
    left_sums = subset_sums(left_half)
    right_sums = subset_sums(right_half)
    right_sums_set = set(right_sums)
    for left_sum in left_sums:
        if (target - left_sum) in right_sums_set:
            return True
    return False
arr = [3, 34, 4, 12, 5, 2]
target = 9
if meet_in_the_middle(arr, target):
    print(f"A subset with the sum {target} exists.")
else:
    print(f"No subset with the sum {target} exists.")
```

Output:

```
A subset with the sum 9 exists.

Process finished with exit code 0
```