

DESIGN AND ANALYSIS OF ALGORITHM

DAY-06 PROGRAMS

NAME : MALREDDY.P

REGISTER NO: 192372015

DAY-06 PROGRAMS

1. To Implement the Median of Medians algorithm ensures that you handle the worst-case time complexity efficiently while finding the k-th smallest element in an unsorted array. arr = [12, 3, 5, 7, 19] k = 2 Expected Output: 5
arr = [12, 3, 5, 7, 4, 19, 26] k = 3 Expected Output: 5
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6 Expected Output: 6.

```

main.py
1- def partition(arr, low, high, pivot):
2-     for i in range(low, high):
3-         if arr[i] == pivot:
4-             arr[i], arr[high] = arr[high], arr[i]
5-             break
6-     pivot = arr[high]
7-     i = low - 1
8-     for j in range(low, high):
9-         if arr[j] <= pivot:
10-             i += 1
11-             arr[i], arr[j] = arr[j], arr[i]
12-     arr[i+1], arr[high] = arr[high], arr[i+1]
13-     return i + 1
14-
15- def find_median(arr):
16-     arr.sort()
17-     return arr[len(arr) // 2]
18-
19- def median_of_medians(arr, k):
20-     if len(arr) <= 5:
21-         arr.sort()
22-         return arr[k-1]
23-
24-     medians = []
25-     for i in range(0, len(arr), 5):
26-         group = arr[i:i+5]
27-         medians.append(find_median(group))
28-
29-     median_of_medians_pivot = median_of_medians(medians, len(medians)//2 + 1)
30-     pivot_index = partition(arr, 0, len(arr) - 1, median_of_medians_pivot)
31-
32-     if pivot_index == k - 1:
33-         return arr[pivot_index]
34-     elif pivot_index > k - 1:
35-         return median_of_medians(arr[:pivot_index], k)
36-     else:
37-         return median_of_medians(arr[pivot_index + 1:], k - pivot_index - 1)
38-
39- arr1 = [12, 3, 5, 7, 19]
40- k1 = 2
41- print("Expected Output:", median_of_medians(arr1, k1))
42-
43- arr2 = [12, 3, 5, 7, 4, 19, 26]

```

Expected Output: 5
Expected Output: 5
Expected Output: 6
=== Code Execution Successful ===

2. To Implement a function median_of_medians(arr, k) that takes an unsorted array arr and an integer k, and returns the k-th smallest element in the array. arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6 arr = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27] k = 5 Output: An integer representing the k-th smallest element in the array.

main.py	Output
<pre>1 def partition(arr, low, high, pivot_index): 2 pivot_value = arr[pivot_index] 3 arr[pivot_index], arr[high] = arr[high], arr[pivot_index] 4 store_index = low 5 for i in range(low, high): 6 if arr[i] < pivot_value: 7 arr[store_index], arr[i] = arr[i], arr[store_index] 8 store_index += 1 9 arr[store_index], arr[high] = arr[high], arr[store_index] 10 return store_index 11 def median_of_medians(arr, k): 12 if len(arr) == 1: 13 return arr[0] 14 n = len(arr) 15 sublists = [arr[i:i+5] for i in range(0, n, 5)] 16 medians = [sorted(sublist)[len(sublist)//2] for sublist in sublists] 17 if len(medians) <= 5: 18 pivot = sorted(medians)[len(medians)//2] 19 else: 20 pivot = median_of_medians(medians, len(medians)//2) 21 pivot_index = arr.index(pivot) 22 pivot_index = partition(arr, 0, len(arr) - 1, pivot_index) 23 if k == pivot_index + 1: 24 return arr[pivot_index] 25 elif k < pivot_index + 1: 26 return median_of_medians(arr[:pivot_index], k) 27 else: 28 return median_of_medians(arr[pivot_index+1:], k - pivot_index - 1) 29 arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 30 k1 = 6 31 print(median_of_medians(arr1, k1)) 32 33 arr2 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27] 34 k2 = 5 35 print(median_of_medians(arr2, k2)) 36</pre>	<pre>6 21 === Code Execution Successful ===</pre>

3. Write a program to implement Meet in the Middle Technique. Given an array of integers and a target sum, find the subset whose sum is closest to the target. You will use the Meet in the Middle technique to efficiently find this subset. a) Set[] = {45, 34, 4, 12, 5, 2} Target Sum : 42 b) Set[] = {1, 3, 2, 7, 4, 6} Target sum = 10:

```
bin.py  [Icons]  Run  Output

from itertools import combinations

def get_subsets(arr):
    """Generate all possible subsets of the given array."""
    subsets = []
    n = len(arr)
    for r in range(n+1):
        for combo in combinations(arr, r):
            subsets.append(combo)
    return subsets

def meet_in_the_middle(arr, target_sum):
    """Find the subset whose sum is closest to the target sum using the Meet in the Middle technique."""
    n = len(arr)
    left_half = arr[:n//2]
    right_half = arr[n//2:]
    left_subsets = get_subsets(left_half)
    right_subsets = get_subsets(right_half)
    left_sums = {sum(subset): subset for subset in left_subsets}
    right_sums = {sum(subset): subset for subset in right_subsets}
    sorted_left_sums = sorted(left_sums.keys())
    sorted_right_sums = sorted(right_sums.keys())
    closest_sum = float('inf')
    best_subset = ()
    i, j = 0, len(sorted_right_sums) - 1
    while i < len(sorted_left_sums) and j >= 0:
        current_sum = sorted_left_sums[i] + sorted_right_sums[j]

        if abs(target_sum - current_sum) < abs(target_sum - closest_sum):
            closest_sum = current_sum
            best_subset = left_sums[sorted_left_sums[i]] + right_sums[sorted_right_sums[j]]
        if current_sum < target_sum:
            i += 1
        else:
            j -= 1
    return best_subset, closest_sum

arr1 = [45, 34, 4, 12, 5, 2]
target1 = 42
subset1, closest_sum1 = meet_in_the_middle(arr1, target1)
print(f"Subset: {subset1}, Closest Sum: {closest_sum1}")

arr2 = [1, 3, 2, 7, 4, 6]
target2 = 10

Subset: (34, 5, 2), Closest Sum: 41
Subset: (4, 6), Closest Sum: 10

=== Code Execution Successful ===
```

4. Write a program to implement Meet in the Middle Technique. Given a large array of integers and an exact sum E, determine if there is any subset that sums exactly to E. Utilize the Meet in the Middle technique to handle the potentially large size of the array. Return true if there is a subset that sums exactly to E, otherwise return false. a) $E = \{1, 3, 9, 2, 7, 12\}$ exact Sum = 15 b) $E = \{3, 34, 4, 12, 5, 2\}$ exact Sum = 15.

main.py	Run	Output
<pre> 1 from itertools import combinations 2 def get_all_subset_sums(arr): 3 """Generate all possible subset sums of the given array.""" 4 subset_sums = set() 5 n = len(arr) 6 for r in range(n+1): 7 for combo in combinations(arr, r): 8 subset_sums.add(sum(combo)) 9 return subset_sums 10 11 def meet_in_the_middle(arr, target_sum): 12 """Determine if there exists any subset that sums exactly to the target sum using Meet in the Middle 13 technique.""" 14 n = len(arr) 15 left_half = arr[:n//2] 16 right_half = arr[n//2:] 17 left_sums = get_all_subset_sums(left_half) 18 right_sums = get_all_subset_sums(right_half) 19 if target_sum in left_sums: 20 return True 21 if target_sum in right_sums: 22 return True 23 for left_sum in left_sums: 24 if (target_sum - left_sum) in right_sums: 25 return True 26 return False 27 arr1 = [1, 3, 9, 2, 7, 12] 28 target_sum1 = 15 29 print(meet_in_the_middle(arr1, target_sum1)) 30 31 arr2 = [3, 34, 4, 12, 5, 2] 32 target_sum2 = 15 33 print(meet_in_the_middle(arr2, target_sum2)) 34 35 arr3 = [3, 34, 4, 12, 5, 2] 36 target_sum3 = 100 37 print(meet_in_the_middle(arr3, target_sum3)) </pre>	Run	<pre> True True False === Code Execution Successful === </pre>

5. Given two 2×2 Matrices A and B A=(1 7 B=(1 3 3 5) 7 5) Use Strassen's matrix multiplication algorithm to compute the product matrix C such that C=A×B. Test Cases: Consider the following matrices for testing your implementation: Test Case 1: A=(1 7 B=(6 8 3 5), 4 2) Expected Output: C=(18 14 62 66.

main.py	Run	Output
<pre> 1 def strassen_multiplication(A, B): 2 a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1] 3 e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1] 4 M1 = (a + d) * (e + h) 5 M2 = (c + d) * e 6 M3 = a * (f - h) 7 M4 = d * (g - e) 8 M5 = (a + b) * h 9 M6 = (c - a) * (e + f) 10 M7 = (b - d) * (g + h) 11 p = M1 + M4 - M5 + M7 12 q = M3 + M5 13 r = M2 + M4 14 s = M1 - M2 + M3 + M6 15 C = [[p, q], [r, s]] 16 return C 17 18 A = [[1, 7], [3, 5]] 19 B = [[6, 8], [4, 2]] 20 C = strassen_multiplication(A, B) 21 print("Resultant Matrix C:") 22 for row in C: 23 print(row) </pre>	Run	<pre> Resultant Matrix C: [34, 22] [38, 34] === Code Execution Successful === </pre>

6. Given two integers X=1234 and Y=5678: Use the Karatsuba algorithm to compute the product Z=X x Y Test Case 1: Input: x=1234,y=5678 Expected Output: z=1234×5678=7016652.

main.py	Output
<pre>1 def karatsuba(x, y): 2 if x < 10 or y < 10: 3 return x * y 4 n = max(len(str(x)), len(str(y))) 5 m = n // 2 6 high1, low1 = divmod(x, 10**m) 7 high2, low2 = divmod(y, 10**m) 8 z0 = karatsuba(low1, low2) 9 z1 = karatsuba((low1 + high1), (low2 + high2)) 10 z2 = karatsuba(high1, high2) 11 return (z2 * 10**(2*m)) + ((z1 - z2 - z0) * 10**m) + z0 12 13 14 x = 1234 15 y = 5678 16 z = karatsuba(x, y) 17 print(f"{x} × {y} = {z}")</pre>	<pre>1234 × 5678 = 7006652 === Code Execution Successful ===</pre>