

1. Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found. Input : N= 8, a[] = {5,7,3,4,9,12,6,2} Output : Min = 2, Max = 12 Test Cases : Input : N= 9, a[] = {1,3,5,7,9,11,13,15,17} Output : Min = 1, Max = 17 Test Cases : Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17} Output : Min 12, Max 67

```
main.py  Run Output Clear
1 import heapq
2 import copy
3
4 N = 4
5
6 # State space tree node
7 class Node:
8     def __init__(self, x, y, assigned, parent):
9         self.parent = parent
10        self.pathCost = 0
11        self.cost = 0
12        self.workerID = x
13        self.jobID = y
14        self.assigned = copy.deepcopy(assigned)
15        if y != -1:
16            self.assigned[y] = True
17
18 # Custom heap class with push and pop functions
19 class CustomHeap:
20     def __init__(self):
21         self.heap = []
22
23     def push(self, node):
24         heapq.heappush(self.heap, (node.cost, node))
25
26     def pop(self):
27         if self.heap:
28             _, node = heapq.heappop(self.heap)
```

```
Assign Worker A to Job 1
Assign Worker B to Job 0
Assign Worker C to Job 2
Assign Worker D to Job 3

Optimal Cost is 13

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

2. Consider an array of integers sorted in ascending order: 2,4,6,8,10,12,14,18. Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found. Input : N=8, 2,4,6,8,10,12,14,18. Output : Min = 2, Max =18 Test Cases : Input : N= 9, a[] = {11,13,15,17,19,21,23,35,37} Output : Min = 11, Max = 37 Test Cases : Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17} Output : Min 12, Max 67

```
main.py  Run  Clear
40 arr[k] = R[j]
41 j += 1
42 k += 1
43
44 def merge_sort(arr, left, right):
45     if left < right:
46         mid = (left + right) // 2
47
48         merge_sort(arr, left, mid)
49         merge_sort(arr, mid + 1, right)
50         merge(arr, left, mid, right)
51
52 def print_list(arr):
53     for i in arr:
54         print(i, end=" ")
55     print()
56
57 # Driver code
58 if __name__ == "__main__":
59     arr = [12, 11, 13, 5, 6, 7]
60     print("Given array is")
61     print_list(arr)
62
63     merge_sort(arr, 0, len(arr) - 1)
64
65     print("\nSorted array is")
66     print_list(arr)
67
```

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

Sorted array is
5 6 7 11 12 13

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

3. You are given an unsorted array 31,23,35,27,11,21,15,28. Write a program for Merge Sort and implement using any programming language of your choice. Test Cases : Input : N= 8, a[] = {31,23,35,27,11,21,15,28} Output : 11,15,21,23,27,28,31,35 Test Cases : Input : N= 10, a[] = {22,34,25,36,43,67, 52,13,65,17} Output : 13,17,22,25,34,36,43,52,65,67

4. Implement the Merge Sort algorithm in a programming language of your choice and test it on the array 12,4,78,23,45,67,89,1. Modify your implementation to count the number of comparisons made during the sorting process. Print this count along with the sorted array. Test Cases : Input : N= 8, a[] = {12,4,78,23,45,67,89,1} Output : 1,4,12,23,45,67,78,89 Test Cases : Input : N= 7, a[] = {38,27,43,3,9,82,10} Output : 3,9,10,27,38,43,82,

```
main.py  Run  Clear
40 arr[k] = R[j]
41 j += 1
42 k += 1
43
44 def merge_sort(arr, left, right):
45     if left < right:
46         mid = (left + right) // 2
47
48         merge_sort(arr, left, mid)
49         merge_sort(arr, mid + 1, right)
50         merge(arr, left, mid, right)
51
52 def print_list(arr):
53     for i in arr:
54         print(i, end=" ")
55     print()
56
57 # Driver code
58 if __name__ == "__main__":
59     arr = [12, 11, 13, 5, 6, 7]
60     print("Given array is")
61     print_list(arr)
62
63     merge_sort(arr, 0, len(arr) - 1)
64
65     print("\nSorted array is")
66     print_list(arr)
67
```

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

Sorted array is
5 6 7 11 12 13

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

5. Given an unsorted array 10,16,8,12,15,6,3,9,5 Write a program to perform Quick Sort. Choose the first element as the pivot and partition the array accordingly. Show the array after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the array after each recursive call until the entire array is sorted. Input : N= 9, a[] = {10,16,8,12,15,6,3,9,5} Output : 3,5,6,8,9,10,12,15,16 Test Cases : Input : N= 8, a[] = {12,4,78,23,45,67,89,1} Output : 1,4,12,23,45,67,78,89 Test Cases : Input : N= 7, a[] = {38,27,43,3,9,82,10} Output : 3,9,10,27,38,43,82,

6. Implement the Quick Sort algorithm in a programming language of your choice and test it on the array 19,72,35,46,58,91,22,31. Choose the middle element as the pivot and partition the array accordingly. Show the array after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the array after each recursive call until the entire array is sorted. Execute your code and show the sorted array. Input : N= 8, a[] = {19,72,35,46,58,91,22,31} Output : 19,22,31,35,46,58,72,91 Test Cases : Input : N= 8, a[] = {31,23,35,27,11,21,15,28} Output : 11,15,21,23,27,28,31,35 Test Cases : Input : N= 10, a[] = {22,34,25,36,43,67, 52,13,65,17} Output : 13,17,22,25,34,36,43,52,65,67

```

main.py  Run  Output  Clear
21 # The QuickSort function implementation
22 def quick_sort(arr, low, high):
23     if low < high:
24         # pi is the partition return index of pivot
25         pi = partition(arr, low, high)
26
27         # Recursion calls for smaller elements
28         # and greater or equals elements
29         quick_sort(arr, low, pi - 1)
30         quick_sort(arr, pi + 1, high)
31
32 # Function to print an array
33 def print_array(arr):
34     for i in arr:
35         print(i, end=" ")
36     print()
37
38 # Driver code
39 if __name__ == "__main__":
40     arr = [10, 7, 8, 9, 1, 5]
41     print("Given array is")
42     print_array(arr)
43
44     quick_sort(arr, 0, len(arr) - 1)
45
46     print("\nSorted array is")
47     print_array(arr)
48

```

Given array is
10 7 8 9 1 5

Sorted array is
1 5 7 8 9 10

=== Code Execution Successful ===

7. Implement the Binary Search algorithm in a programming language of your choice and test it on the array 5,10,15,20,25,30,35,40,45 to find the position of the element 20. Execute your code and provide the index of the element 20. Modify your implementation to count the number of comparisons made during the search process. Print this count along with the result. Input : N= 9, a[] = {5,10,15,20,25,30,35,40,45}, search key = 20 Output : 4 Test cases Input : N= 6, a[] = {10,20,30,40,50,60}, search key = 50 Output : 5 Input : N= 7, a[] = {21,32,40,54,65,76,87}, search key = 32 Output : 2

```

main.py
1 # Python3 code to implement iterative Binary
2 # Search.
3
4
5 # It returns location of x in given array arr
6 def binarySearch(arr, low, high, x):
7
8     while low <= high:
9
10         mid = low + (high - low) // 2
11
12         # Check if x is present at mid
13         if arr[mid] == x:
14             return mid
15
16         # If x is greater, ignore left half
17         elif arr[mid] < x:
18             low = mid + 1
19
20         # If x is smaller, ignore right half
21         else:
22             high = mid - 1
23
24     # If we reach here, then the element
25     # was not present
26     return -1
27
28

```

Output

Element is present at index 3

=== Code Execution Successful ===

8. You are given a sorted array 3,9,14,19,25,31,42,47,53 and asked to find the position of the element 31 using Binary Search. Show the mid-point calculations and the steps involved in finding the element. Display, what would happen if the array was not sorted, how would this impact the performance and correctness of the Binary Search algorithm? Input : N= 9, a[] = {3,9,14,19,25,31,42,47,53}, search key = 31 Output : 6 Test cases Input : N= 7, a[] = {13,19,24,29,35,41,42}, search key = 42 Output : 7 Test cases Input : N= 6, a[] = {20,40,60,80,100,120}, search key = 60 Output : 3

```

main.py
1 # Python3 code to implement iterative Binary
2 # Search.
3
4
5 # It returns location of x in given array arr
6 def binarySearch(arr, low, high, x):
7
8     while low <= high:
9
10         mid = low + (high - low) // 2
11
12         # Check if x is present at mid
13         if arr[mid] == x:
14             return mid
15
16         # If x is greater, ignore left half
17         elif arr[mid] < x:
18             low = mid + 1
19
20         # If x is smaller, ignore right half
21         else:
22             high = mid - 1
23
24     # If we reach here, then the element
25     # was not present
26     return -1
27
28

```

Output

Element is present at index 3

=== Code Execution Successful ===

9. Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0). (i) Input : points = [[1,3],[-2,2],[5,8],[0,1]],k=2 Output:[[-2, 2], [0, 1]] (ii) Input: points = [[1, 3], [-2, 2]], k = 1 Output: [[-2, 2]] (iii) Input: points = [[3, 3], [5, -1], [-2, 4]], k = 2 Output: [[3, 3], [-2, 4]] 10. Given four lists A, B, C, D of integer values, Write a program to compute how many tuples n(i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero. (i) Input: A = [1, 2], B = [-2, -1], C = [-1, 2], D = [0, 2] Output: 2 (ii) Input: A = [0], B = [0], C = [0], D = [0] Output: 1

main.py



Share

Run

Output

Clear

```
1 # Python3 program for implementation of
2 # above approach
3
4 # Function to return required answer
5 def pClosest(points, K):
6
7     points.sort(key = lambda K: K[0]**2 + K[1]**2)
8
9     return points[:K]
10
11 # Driver program
12 points = [[3, 3], [5, -1], [-2, 4]]
13
14 K = 2
15
16 print(pClosest(points, K))
```

```
[[3, 3], [-2, 4]]
```

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