**UCS415 : Design and Analysis of Algorithms**

**Lab Assignment 01**

**Vaibhav Srivastva 102316037 2P12(C2) 25 Jan 2025**

Write the solutions to the following problems using iterative, recursive, and divide-and-conquer approaches.

1. Consider an array arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}. Using Binary Search, find the target 23.

1. import java.util.Scanner;

2.

3. public class Main {

4.

5. public static void main(String[] args) {

6.

7. int[] myArr = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};

8.

9. // Divide-and-conquer part

10. Scanner scanner = new Scanner(System.in);

11. System.out.print("Enter target element : ");

12. int key = scanner.nextInt();

13. System.out.print("\nDivide-and-conquer answer : ");

14. System.out.printf("%d is present at index %d of the array.", key, DnQBinarySearch(myArr, key));

15.

16. // Iterative and recursive portion

17. int len = myArr.length - 1;

18. System.out.print("\nIterative answer : ");

19. System.out.print(iterativeBinarySearch(myArr, 23));

20. System.out.print("\nRecursive answer : ");

21. System.out.print(recursiveBinarySearch(myArr, 0, len, 23));

22.

23. }

24.

25. private static int iterativeBinarySearch(int[] arr, int target) {

26. int low = 0;

27. int high = arr.length - 1;

28.

29. while (low <= high) {

30. int mid = low + (high - low)/2;

31. int testNum = arr[mid];

32.

33. if (testNum == target) {

34. return mid;

35. }

36. if (testNum < target) {

37. low = mid + 1; // take the right part

38. } else {

39. high = mid - 1; // take the left part

40. }

41. }

42. return -1; // default case

43. }

44.

45. private static int recursiveBinarySearch(int[] arr, int low, int high, int target) {

46. if (low <= high) {

47. int mid = low + (high - low)/2;

48. int testNum = arr[mid];

49. if (testNum == target) {

50. return mid;

51. }

52. if (testNum < target) {

53. return recursiveBinarySearch(arr, mid + 1, high, target);

54. // undershoot

55. } else {

56. return recursiveBinarySearch(arr, low, mid - 1, target);

57. // overshoot

58. }

59. }

60. return -1; // default case

61. }

62.

63. private static int DnQBinarySearch(int[] arr, int target) {

64. int low = 0;

65. int high = arr.length - 1;

66.

67. while (low <= high) {

68. int mid = low + (high - low)/2; // prevents overflow

69. int testNum = arr[mid];

70.

71. if (testNum == target) {

72. return mid;

73. }

74. if (testNum < target) {

75. low = mid + 1; // take the right part

76. } else {

77. high = mid - 1; // take the left part

78. }

79. }

80. return -1; // default case

81. }

82.

83. }

84.

// OUTPUT  
  
Enter target element : 23  
  
Divide-and-conquer answer : 23 is present at index 5 of the array.  
Iterative answer : 5  
Recursive answer : 5

1. Implement Merge sort for the given array int arr[] = {12, 11, 13, 5, 6, 7}.

1. public class MergeSort {

2.

3. public static void main(String[] args) {

4. int[] arr = {12, 11, 13, 5, 6, 7};

5. System.out.println("Original array : ");

6. printArray(arr, ", ");

7. System.out.println();

8.

9. mergeSort(arr, 0, arr.length - 1);

10.

11. System.out.println("\nSorted array : ");

12. printArray(arr, ", ");

13. }

14.

15. private static void mergeSort(int[] arr, int left, int right) {

16. if (left < right) {

17. int mid = left + (right - left)/2; // prevents overflow

18.

19. mergeSort(arr, left, mid); // sort left half

20. mergeSort(arr, mid + 1, right); //sort right half

21.

22. merge(arr, left, mid, right); // merge the sorted halves

23. }

24. }

25.

26. private static void merge(int[] arr, int left, int mid, int right) {

27. int n1 = mid - left + 1; // size of 1st subarray

28. int n2 = right - mid; // size of 2nd subarray

29.

30. int[] L = new int[n1]; // temporary left array

31. int[] R = new int[n2]; // temporary right array

32.

33. // copying data to temporary arrays

34. for (int i = 0; i < n1; ++i)

35. L[i] = arr[left + i];

36. for (int j = 0; j < n2; ++j)

37. R[j] = arr[mid + 1 + j];

38.

39.

40. // merging the temporary arrays

41. int i = 0, j = 0; // initial index of 1st and 2nd arrays

42. int k = left; // initial index of merged subarray

43. while (i < n1 && j < n2) {

44. if (L[i] <= R[j]) {

45. arr[k] = L[i];

46. i++;

47. } else {

48. arr[k] = R[j];

49. j++;

50. }

51. k++;

52. }

53.

54. // Copy any remaining elements of L

55. while (i < n1) {

56. arr[k] = L[i];

57. i++;

58. k++;

59. }

60.

61. // Copy any remaining elements of R

62. while (j < n2) {

63. arr[k] = R[j];

64. j++;

65. k++;

66. }

67. }

68.

69. private static void printArray(int[] arr, String sep) {

70. int n = arr.length;

71. for (int i = 0; i < n - 1; i++)

72. System.out.print(arr[i] + sep);

73. System.out.print(arr[n - 1]);

74. }

75. }

76.

// OUTPUT

Original array :

12, 11, 13, 5, 6, 7

Sorted array :

5, 6, 7, 11, 12, 13

1. Implement Quick Sort for arr[] = {4, 2, 6, 9, 2}.

1. public class QuickSort {

2.

3. public static void main(String[] args) {

4. int[] myArr = {4, 2, 6, 9, 2};

5. System.out.print("Unsorted array : ");

6. printArray(myArr, ", ");

7. iterativeQuickSort(myArr, 0, myArr.length - 1);

8. System.out.print("\nIterative sorting : ");

9. printArray(myArr, ", ");

10. System.out.print("\nRecursive sorting : ");

11. printArray(myArr, ", ");

12. }

13.

14. public static void recursiveQuickSort(int[] arr, int low, int high) {

15. if (low < high) {

16. int pi = partition(arr, low, high);

17. recursiveQuickSort(arr, low, pi - 1);

18. recursiveQuickSort(arr, pi + 1, high);

19. }

20. }

21.

22. public static void iterativeQuickSort(int[] arr, int low, int high) {

23. int[] stack = new int[high - low + 1];

24. int top = -1;

25.

26. stack[++top] = low;

27. stack[++top] = high;

28.

29. while (top >= 0) {

30. high = stack[top--];

31. low = stack[top--];

32.

33. int pi = partition(arr, low, high);

34.

35. if (pi - 1 > low) {

36. stack[++top] = low;

37. stack[++top] = pi - 1;

38. }

39. if (pi + 1 < high) {

40. stack[++top] = pi + 1;

41. stack[++top] = high;

42. }

43. }

44. }

45.

46. public static int partition(int[] arr, int low, int high) {

47. int pivot = arr[high];

48. int i = low - 1;

49. for (int j = low; j < high; j++) {

50. if (arr[j] < pivot) {

51. i++;

52. int temp = arr[i];

53. arr[i] = arr[j];

54. arr[j] = temp;

55. }

56. }

57. int temp = arr[i + 1];

58. arr[i + 1] = arr[high];

59. arr[high] = temp;

60. return i + 1;

61. }

62.

63. private static void printArray(int[] arr, String sep) {

64. int n = arr.length;

65. for (int i = 0; i < n - 1; i++)

66. System.out.print(arr[i] + sep);

67. System.out.print(arr[n - 1]);

68. }

69. }

70.

71. // OUTPUT

72. Unsorted array : 4, 2, 6, 9, 2

73. Iterative sorting : 2, 2, 4, 6, 9

74. Recursive sorting : 2, 2, 4, 6, 9

75.

1. You are given a one-dimensional array that may contain both positive and negative integers, find the sum of contiguous subarray of numbers which has the largest sum. For example, if the given array is {-2, -5, 6, -2, -3, 1, 5, -6}, then the maximum subarray sum is 7.

1. public class SubarraySum {

2.

3. public static void main(String[] args) {

4. int[] myArr = {-2, -5, 6, -2, -3, 1, 5, -6};

5.

6. int maxSum = findMaxSubarraySum(myArr);

7.

8. System.out.println("Maximum contiguous subarray sum = " + maxSum + ".");

9. }

10.

11. private static int findMaxSubarraySum(int[] arr) {

12.

13. // Kadane's Algorithm

14.

15. int maxSoFar = arr[0];

16. int maxEndingHere = arr[0];

17.

18. for (int i = 1; i < arr.length; i++) {

19. maxEndingHere = max(arr[i], maxEndingHere + arr[i]);

20. maxSoFar = max(maxSoFar, maxEndingHere);

21. }

22.

23. return maxSoFar;

24. }

25.

26. private static int max(int a, int b) {

27. if (b > a) {

28. return b;

29. } else {

30. return a;

31. }

32. }

33. }

34.

35. // OUTPUT

36. Maximum contiguous subarray sum = 7.

37.