# Experiment No: 1

**Experiment Name**: Implementation and Analysis of Linear Search Using recursive function.

**Objective:** Implement and analyse the recursive implementation of linear search to understand its recursive nature and assess its time and space complexity.

**Program:**

#include <stdio.h>

int recursiveLinearSearch(int arr[], int target, int index, int size) {

if (index == size) {

return -1;

}

if (arr[index] == target) {

return index;

}

return recursiveLinearSearch(arr, target, index + 1, size);

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; ++i) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

int targetElement;

printf("Enter the target element to search: ");

scanf("%d", &targetElement);

int result = recursiveLinearSearch(myArray, targetElement, 0, arraySize);

if (result != -1) {

printf("Element %d found at index %d.\n", targetElement, result);

} else {

printf("Element %d not found in the array.\n", targetElement);

}

return 0;

}

# Experiment No: 1

**Experiment Name**: Implementation and Analysis of Binary Search Using recursive function

**Objective:** Implement and analyse the recursive implementation of Binary search to understand its recursive nature and assess its time and space complexity.

**Program:**

#include <stdio.h>

int recursiveBinarySearch(int arr[], int target, int low, int high) {

if (low <= high) {

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

if (arr[mid] == target) {

return mid;

}

if (arr[mid] > target) {

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

} else {

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

}

}

return -1;

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d sorted elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; ++i) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

int targetElement;

printf("Enter the target element to search: ");

scanf("%d", &targetElement);

int result = recursiveBinarySearch(myArray, targetElement, 0, arraySize - 1);

if (result != -1) {

printf("Element %d found at index %d.\n", targetElement, result);

} else {

printf("Element %d not found in the array.\n", targetElement);

}

return 0;

}

# Experiment No: 2

**Experiment Name**: Implementation and Analysis of Insertion Sort .

**Objective:** To implement the Insertion Sort algorithm and analyze its efficiency in sorting data, evaluating its time complexity and practical performance.

**Program:**

#include <stdio.h>

#include <stdlib.h>

void insertionSort(int arr[], int size) {

int i, key, j;

for (i = 1; i < size; i++) {

key = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

insertionSort(myArray, arraySize);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}

# Experiment No: 2

**Experiment Name**: Implementation and Analysis of Bubble Sort.

**Objective:** Objective: To implement and analyse the Bubble Sort algorithm's efficiency and performance in sorting a given dataset.

**Program:**

#include <stdio.h>

#include <stdlib.h>

void bubbleSort(int arr[], int size) {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

bubbleSort(myArray, arraySize);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}

# Experiment No: 2

**Experiment Name**: Implementation and Analysis of Selection Sort.

**Objective:** Objective: To implement and analyse the Selection Sort algorithm's efficiency and performance in sorting a given dataset.

**Program:**

#include <stdio.h>

#include <stdlib.h>

void selectionSort(int arr[], int size) {

for (int i = 0; i < size - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < size; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

selectionSort(myArray, arraySize);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}

# Experiment No: 3

**Experiment Name**: Implementation and Analysis of Merge Sort.

**Objective:** To implement and analyse the Merge Sort algorithm for efficient sorting of data, assessing its time and space complexity.

**Program:**

#include <stdio.h>

#include <stdlib.h>

void merge(int arr[], int left, int middle, int right) {

int i, j, k;

int n1 = middle - left + 1;

int n2 = right - middle;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

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

for (j = 0; j < n2; j++)

R[j] = arr[middle + 1 + j];

i = 0;

j = 0;

k = left;

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

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

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int middle = left + (right - left) / 2;

mergeSort(arr, left, middle);

mergeSort(arr, middle + 1, right);

merge(arr, left, middle, right);

}

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

mergeSort(myArray, 0, arraySize - 1);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}

# Experiment No: 3

**Experiment Name**: Implementation and Analysis of Quick Sort.

**Objective:** To implement and analyse the Quick Sort algorithm for efficient sorting of data, assessing its time and space complexity.

**Program:**

#include <stdio.h>

#include <stdlib.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

quickSort(myArray, 0, arraySize - 1);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}

# Experiment No: 4

**Experiment Name**: Implementation and Analysis of Heap Sort.

**Objective:** To implement and analyse the Heap Sort algorithm for efficient sorting of data, assessing its time and space complexity.

**Program:**

#include <stdio.h>

#include <stdlib.h>

void heapify(int arr[], int size, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < size && arr[left] > arr[largest])

largest = left;

if (right < size && arr[right] > arr[largest])

largest = right;

if (largest != i) {

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, size, largest);

}

}

void heapSort(int arr[], int size) {

for (int i = size / 2 - 1; i >= 0; i--)

heapify(arr, size, i);

for (int i = size - 1; i > 0; i--) {

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

heapSort(myArray, arraySize);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}

# Experiment No: 4

**Experiment Name**: Implementation and Analysis of Counting Sort.

**Objective:** To implement and analyse the Counting Sort algorithm for efficient sorting of data, assessing its time and space complexity.

**Program:**

#include <stdio.h>

#include <stdlib.h>

void countingSort(int arr[], int size) {

int max = arr[0];

for (int i = 1; i < size; i++) {

if (arr[i] > max) {

max = arr[i];

}

}

int\* count = (int\*)malloc((max + 1) \* sizeof(int));

for (int i = 0; i <= max; i++) {

count[i] = 0;

}

for (int i = 0; i < size; i++) {

count[arr[i]]++;

}

int k = 0;

for (int i = 0; i <= max; i++) {

while (count[i] > 0) {

arr[k++] = i;

count[i]--;

}

}

free(count);

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

countingSort(myArray, arraySize);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}

# Experiment No: 5

**Experiment Name**: Implementation and Analysis of Radix Sort.

**Objective:** To implement and analyse the Radix Sort algorithm for efficient sorting of data, assessing its time and space complexity.

**Program:**

#include <stdio.h>

#include <stdlib.h>

int getMax(int arr[], int size) {

int max = arr[0];

for (int i = 1; i < size; i++) {

if (arr[i] > max) {

max = arr[i];

}

}

return max;

}

void countingSort(int arr[], int size, int place) {

const int max = 10;

int output[size];

int count[max];

for (int i = 0; i < max; i++) {

count[i] = 0;

}

for (int i = 0; i < size; i++) {

count[(arr[i] / place) % 10]++;

}

for (int i = 1; i < max; i++) {

count[i] += count[i - 1];

}

for (int i = size - 1; i >= 0; i--) {

output[count[(arr[i] / place) % 10] - 1] = arr[i];

count[(arr[i] / place) % 10]--;

}

for (int i = 0; i < size; i++) {

arr[i] = output[i];

}

}

void radixSort(int arr[], int size) {

int max = getMax(arr, size);

for (int place = 1; max / place > 0; place \*= 10) {

countingSort(arr, size, place);

}

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

radixSort(myArray, arraySize);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}

# Experiment No: 5

**Experiment Name**: Implementation and Analysis of Shell Sort.

**Objective:** To implement and analyse the Shell Sort algorithm for efficient sorting of data, assessing its time and space complexity.

**Program:**

#include <stdio.h>

#include <stdlib.h>

void shellSort(int arr[], int size) {

for (int gap = size / 2; gap > 0; gap /= 2) {

for (int i = gap; i < size; i++) {

int temp = arr[i];

int j;

for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {

arr[j] = arr[j - gap];

}

arr[j] = temp;

}

}

}

int main() {

int arraySize;

printf("Enter the size of the array: ");

scanf("%d", &arraySize);

int myArray[arraySize];

printf("Enter %d elements for the array:\n", arraySize);

for (int i = 0; i < arraySize; i++) {

printf("Element %d: ", i + 1);

scanf("%d", &myArray[i]);

}

shellSort(myArray, arraySize);

printf("Sorted array: ");

for (int i = 0; i < arraySize; i++) {

printf("%d ", myArray[i]);

}

printf("\n");

return 0;

}