# Week1

**Practicelab–AlgorithmsandProblemSolving(15B17CI471)**

**Weare given anarrayofndistinctnumbers;where n is large numbers are randomlygenerated.**

**The task is to**

1. **Sorttheentirearrayusingselectionsort,bubblesort,insertionsort,quicksortandmerge sort. Print the total number of comparisons done in each of the sorting algorithm.**

#include <iostream>

#include <vector>

#include <ctime>

#include <cstdlib>

using namespace std;

int selectionSort(vector<int>& arr)

{

int comparisons = 0;

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

{

int minIndex = i;

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

{

comparisons++;

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

{

minIndex = j;

}

}

swap(arr[i], arr[minIndex]);

}

return comparisons;

}

int bubbleSort(vector<int>& arr)

{

int comparisons = 0;

bool swapped;

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

{

swapped = false;

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

{

comparisons++;

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

{

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

if (!swapped) break;

}

return comparisons;

}

int insertionSort(vector<int>& arr)

{

int comparisons = 0;

for (int i = 1; i < arr.size(); i++)

{

int key = arr[i];

int j = i - 1;

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

{

comparisons++;

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

j--;

}

arr[j + 1] = key;

}

return comparisons;

}

int partition(vector<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;

}

int quickSortHelper(vector<int>& arr, int low, int high, int& comparisons)

{

if (low < high)

{

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

comparisons += high - low;

quickSortHelper(arr, low, pi - 1, comparisons);

quickSortHelper(arr, pi + 1, high, comparisons);

}

return comparisons;

}

int quickSort(vector<int>& arr)

{

int comparisons = 0;

return quickSortHelper(arr, 0, arr.size() - 1, comparisons);

}

void merge(vector<int>& arr, int left, int mid, int right, int& comparisons)

{

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> L(n1), R(n2);

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2)

{

comparisons++;

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++;

}

}

int mergeSortHelper(vector<int>& arr, int left, int right, int& comparisons)

{

if (left < right)

{

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

mergeSortHelper(arr, left, mid, comparisons);

mergeSortHelper(arr, mid + 1, right, comparisons);

merge(arr, left, mid, right, comparisons);

}

return comparisons;

}

int mergeSort(vector<int>& arr)

{

int comparisons = 0;

return mergeSortHelper(arr, 0, arr.size() - 1, comparisons);

}

vector<int> generateRandomArray(int size)

{

vector<int> arr(size);

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

arr[i] = rand() % 10000;

return arr;

}

int main()

{

int n = 1000;

vector<int> arr = generateRandomArray(n);

vector<int> arrCopy = arr;

int selectionComparisons = selectionSort(arrCopy);

cout << "Selection Sort Comparisons: " << selectionComparisons << endl;

arrCopy = arr;

int bubbleComparisons = bubbleSort(arrCopy);

cout << "Bubble Sort Comparisons: " << bubbleComparisons << endl;

arrCopy = arr;

int insertionComparisons = insertionSort(arrCopy);

cout << "Insertion Sort Comparisons: " << insertionComparisons << endl;

arrCopy = arr;

int quickComparisons = quickSort(arrCopy);

cout << "Quick Sort Comparisons: " << quickComparisons << endl;

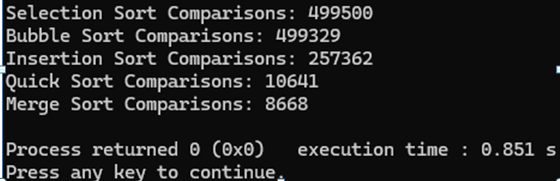
arrCopy = arr;

int mergeComparisons = mergeSort(arrCopy);

cout << "Merge Sort Comparisons: " << mergeComparisons << endl;

}

**Output :**

****

1. **Takesortedarrayfromparta)andagainrunall thesortingalgorithmfunctions.Printthe total number of comparisons done in each of the sorting algorithm.**

#include <iostream>

#include <vector>

#include <ctime>

#include <cstdlib>

using namespace std;

int selectionSort(vector<int>& arr)

{

int comparisons = 0;

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

{

int minIndex = i;

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

{

comparisons++;

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

{

minIndex = j;

}

}

swap(arr[i], arr[minIndex]);

}

return comparisons;

}

int bubbleSort(vector<int>& arr)

{

int comparisons = 0;

bool swapped;

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

{

swapped = false;

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

{

comparisons++;

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

{

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

if (!swapped) break;

}

return comparisons;

}

int insertionSort(vector<int>& arr)

{

int comparisons = 0;

for (int i = 1; i < arr.size(); i++)

{

int key = arr[i];

int j = i - 1;

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

{

comparisons++;

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

j--;

}

arr[j + 1] = key;

}

return comparisons;

}

int partition(vector<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;

}

int quickSortHelper(vector<int>& arr, int low, int high, int& comparisons)

{

if (low < high)

{

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

comparisons += high - low;

quickSortHelper(arr, low, pi - 1, comparisons);

quickSortHelper(arr, pi + 1, high, comparisons);

}

return comparisons;

}

int quickSort(vector<int>& arr)

{

int comparisons = 0;

return quickSortHelper(arr, 0, arr.size() - 1, comparisons);

}

void merge(vector<int>& arr, int left, int mid, int right, int& comparisons)

{

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> L(n1), R(n2);

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2)

{

comparisons++;

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++;

}

}

int mergeSortHelper(vector<int>& arr, int left, int right, int& comparisons)

{

if (left < right)

{

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

mergeSortHelper(arr, left, mid, comparisons);

mergeSortHelper(arr, mid + 1, right, comparisons);

merge(arr, left, mid, right, comparisons);

}

return comparisons;

}

int mergeSort(vector<int>& arr)

{

int comparisons = 0;

return mergeSortHelper(arr, 0, arr.size() - 1, comparisons);

}

vector<int> generateRandomArray(int size)

{

vector<int> arr(size);

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

arr[i] = rand() % 10000;

return arr;

}

int main()

{

int n = 1000;

vector<int> arr = generateRandomArray(n);

vector<int> arrCopy = arr;

int selectionComparisons = selectionSort(arrCopy);

cout << "Selection Sort Comparisons: " << selectionComparisons << endl;

arrCopy = arr;

int bubbleComparisons = bubbleSort(arrCopy);

cout << "Bubble Sort Comparisons: " << bubbleComparisons << endl;

arrCopy = arr;

int insertionComparisons = insertionSort(arrCopy);

cout << "Insertion Sort Comparisons: " << insertionComparisons << endl;

arrCopy = arr;

int quickComparisons = quickSort(arrCopy);

cout << "Quick Sort Comparisons: " << quickComparisons << endl;

arrCopy = arr;

int mergeComparisons = mergeSort(arrCopy);

cout << "Merge Sort Comparisons: " << mergeComparisons << endl;

cout<<"Comparisons for sorted array : "<<endl;

vector<int> arrCopy2=arrCopy;

int selectionComparisons2 = selectionSort(arrCopy2);

cout << "Selection Sort Comparisons: " << selectionComparisons2 << endl;

arrCopy2=arrCopy;

int bubbleComparisons2 = bubbleSort(arrCopy2);

cout << "Bubble Sort Comparisons: " << bubbleComparisons2 << endl;

arrCopy2=arrCopy;

int insertionComparisons2 = insertionSort(arrCopy2);

cout << "Insertion Sort Comparisons: " << insertionComparisons2 << endl;

arrCopy2=arrCopy;

int quickComparisons2 = quickSort(arrCopy2);

cout << "Quick Sort Comparisons: " << quickComparisons2 << endl;

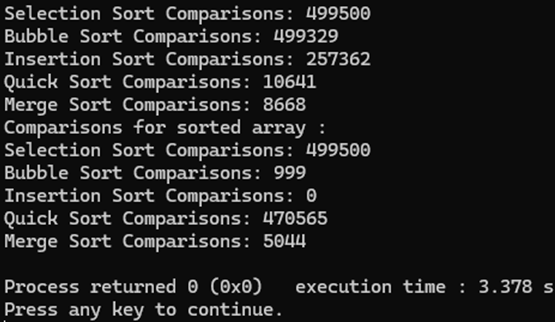
arrCopy2=arrCopy;

int mergeComparisons2 = mergeSort(arrCopy2);

cout << "Merge Sort Comparisons: " << mergeComparisons2 << endl;

}

**Output :**

****

1. **Change the functions totake a flag “order” as argument. This order can be „d‟ or „a‟for descendingandascendingrespectively.Thefunctionwillsortthearrayindescendingand ascending order depending on the flag value.**

#include <iostream>

#include <vector>

#include <ctime>

#include <cstdlib>

using namespace std;

int selectionSort(vector<int>& arr,char order)

{

int comparisons = 0;

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

{

int minIndex = i;

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

{

comparisons++;

if(order=='a')

{

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

minIndex = j;

}

if(order=='d')

{

if (arr[j]>arr[minIndex])

minIndex = j;

}

}

swap(arr[i], arr[minIndex]);

}

return comparisons;

}

int bubbleSort(vector<int>& arr,char order)

{

int comparisons = 0;

bool swapped;

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

{

swapped = false;

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

{

comparisons++;

if(order=='a')

{

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

{

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

if(order=='d')

{

if (arr[j]<arr[j + 1])

{

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

}

if (!swapped) break;

}

return comparisons;

}

int insertionSort(vector<int>& arr,char order)

{

int comparisons = 0;

for (int i = 1; i < arr.size(); i++)

{

int key = arr[i];

int j = i - 1;

while (j >= 0 && ((order == 'a' && arr[j] > key) || (order == 'd' && arr[j] < key)))

{

comparisons++;

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

j--;

}

arr[j + 1] = key;

}

return comparisons;

}

int partition(vector<int>& arr, int low, int high,char order)

{

int pivot = arr[high];

int i = low - 1;

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

{

if(order=='a')

{

if (arr[j] < pivot)

{

i++;

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

}

}

if(order=='d')

{

if (arr[j] > pivot)

{

i++;

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

}

}

}

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

return i + 1;

}

int quickSortHelper(vector<int>& arr, int low, int high, int& comparisons,char order)

{

if (low < high)

{

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

comparisons += high - low;

quickSortHelper(arr, low, pi - 1, comparisons,order);

quickSortHelper(arr, pi + 1, high, comparisons,order);

}

return comparisons;

}

int quickSort(vector<int>& arr,char order)

{

int comparisons = 0;

return quickSortHelper(arr, 0, arr.size() - 1, comparisons,order);

}

void merge(vector<int>& arr, int left, int mid, int right, int& comparisons,char order)

{

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> L(n1), R(n2);

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2)

{

comparisons++;

if((order == 'a' && L[i] <= R[j]) || (order == 'd' && 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++;

}

}

int mergeSortHelper(vector<int>& arr, int left, int right, int& comparisons,char order)

{

if (left < right)

{

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

mergeSortHelper(arr, left, mid, comparisons,order);

mergeSortHelper(arr, mid + 1, right, comparisons,order);

merge(arr, left, mid, right, comparisons,order);

}

return comparisons;

}

int mergeSort(vector<int>& arr,char order)

{

int comparisons = 0;

return mergeSortHelper(arr, 0, arr.size() - 1, comparisons,order);

}

vector<int> generateRandomArray(int size)

{

vector<int> arr(size);

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

arr[i] = rand() % 10000;

return arr;

}

int main()

{

int n = 1000;

char order='a';

vector<int> arr = generateRandomArray(n);

vector<int> arrCopy = arr;

cout<<"Comparisons when order = 'a' :"<<endl;

int selectionComparisons = selectionSort(arrCopy,order);

cout << "Selection Sort Comparisons: " << selectionComparisons << endl;

arrCopy = arr;

int bubbleComparisons = bubbleSort(arrCopy,order);

cout << "Bubble Sort Comparisons: " << bubbleComparisons << endl;

arrCopy = arr;

int insertionComparisons = insertionSort(arrCopy,order);

cout << "Insertion Sort Comparisons: " << insertionComparisons << endl;

arrCopy = arr;

int quickComparisons = quickSort(arrCopy,order);

cout << "Quick Sort Comparisons: " << quickComparisons << endl;

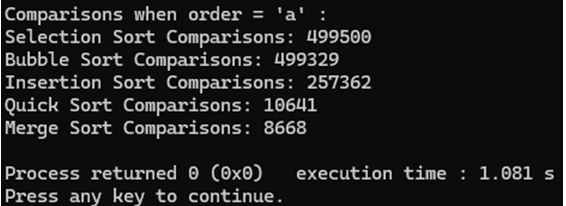
arrCopy = arr;

int mergeComparisons = mergeSort(arrCopy,order);

cout << "Merge Sort Comparisons: " << mergeComparisons << endl;

}

**Output :**

****

1. **Take sorted array from part a). Again run all the sorting algorithm functions with the “order”flagchanged.Ifarrayisalreadyinascendingorderpassflagvalueas „d‟andvice versa. Print the total number of comparisons done in each of the sorting algorithm.**

#include <iostream>

#include <vector>

#include <ctime>

#include <cstdlib>

using namespace std;

int selectionSort(vector<int>& arr,char order)

{

int comparisons = 0;

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

{

int minIndex = i;

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

{

comparisons++;

if(order=='a')

{

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

minIndex = j;

}

if(order=='d')

{

if (arr[j]>arr[minIndex])

minIndex = j;

}

}

swap(arr[i], arr[minIndex]);

}

return comparisons;

}

int bubbleSort(vector<int>& arr,char order)

{

int comparisons = 0;

bool swapped;

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

{

swapped = false;

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

{

comparisons++;

if(order=='a')

{

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

{

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

if(order=='d')

{

if (arr[j]<arr[j + 1])

{

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

}

if (!swapped) break;

}

return comparisons;

}

int insertionSort(vector<int>& arr,char order)

{

int comparisons = 0;

for (int i = 1; i < arr.size(); i++)

{

int key = arr[i];

int j = i - 1;

while (j >= 0 && ((order == 'a' && arr[j] > key) || (order == 'd' && arr[j] < key)))

{

comparisons++;

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

j--;

}

arr[j + 1] = key;

}

return comparisons;

}

int partition(vector<int>& arr, int low, int high,char order)

{

int pivot = arr[high];

int i = low - 1;

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

{

if(order=='a')

{

if (arr[j] < pivot)

{

i++;

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

}

}

if(order=='d')

{

if (arr[j] > pivot)

{

i++;

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

}

}

}

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

return i + 1;

}

int quickSortHelper(vector<int>& arr, int low, int high, int& comparisons,char order)

{

if (low < high)

{

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

comparisons += high - low;

quickSortHelper(arr, low, pi - 1, comparisons,order);

quickSortHelper(arr, pi + 1, high, comparisons,order);

}

return comparisons;

}

int quickSort(vector<int>& arr,char order)

{

int comparisons = 0;

return quickSortHelper(arr, 0, arr.size() - 1, comparisons,order);

}

void merge(vector<int>& arr, int left, int mid, int right, int& comparisons,char order)

{

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> L(n1), R(n2);

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2)

{

comparisons++;

if((order == 'a' && L[i] <= R[j]) || (order == 'd' && 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++;

}

}

int mergeSortHelper(vector<int>& arr, int left, int right, int& comparisons,char order)

{

if (left < right)

{

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

mergeSortHelper(arr, left, mid, comparisons,order);

mergeSortHelper(arr, mid + 1, right, comparisons,order);

merge(arr, left, mid, right, comparisons,order);

}

return comparisons;

}

int mergeSort(vector<int>& arr,char order)

{

int comparisons = 0;

return mergeSortHelper(arr, 0, arr.size() - 1, comparisons,order);

}

vector<int> generateRandomArray(int size)

{

vector<int> arr(size);

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

arr[i] = rand() % 10000;

return arr;

}

int main()

{

int n = 1000;

char order='d';

vector<int> arr = generateRandomArray(n);

vector<int> arrCopy = arr;

cout<<"Comparisons when order = 'd' :"<<endl;

int selectionComparisons = selectionSort(arrCopy,order);

cout << "Selection Sort Comparisons: " << selectionComparisons << endl;

arrCopy = arr;

int bubbleComparisons = bubbleSort(arrCopy,order);

cout << "Bubble Sort Comparisons: " << bubbleComparisons << endl;

arrCopy = arr;

int insertionComparisons = insertionSort(arrCopy,order);

cout << "Insertion Sort Comparisons: " << insertionComparisons << endl;

arrCopy = arr;

int quickComparisons = quickSort(arrCopy,order);

cout << "Quick Sort Comparisons: " << quickComparisons << endl;

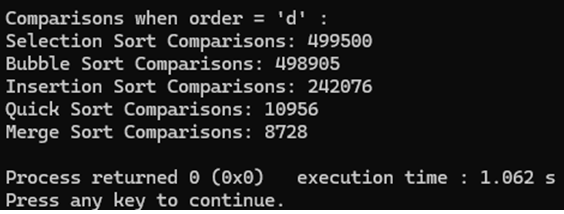
arrCopy = arr;

int mergeComparisons = mergeSort(arrCopy,order);

cout << "Merge Sort Comparisons: " << mergeComparisons << endl;

}

**Output :**

****

1. **Implementquicksortwiththreeoverloaded functions.1sttakingpivotasfirst index,2nd taking pivot as last index and 3rd taking pivot as middle index.**

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

void swap(int &a, int &b)

{

int temp = a;

a = b;

b = temp;

}

int partitionFirst(int arr[], int low, int high, int &comparisons)

{

int pivot = arr[low];

int i = low + 1;

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

{

comparisons++;

if (arr[j] < pivot)

{

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

i++;

}

}

swap(arr[low], arr[i - 1]);

return i - 1;

}

int partitionLast(int arr[], int low, int high, int &comparisons)

{

swap(arr[low], arr[high]);

return partitionFirst(arr, low, high, comparisons);

}

int partitionMiddle(int arr[], int low, int high, int &comparisons)

{

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

swap(arr[low], arr[mid]);

return partitionFirst(arr, low, high, comparisons);

}

void quickSortFirst(int arr[], int low, int high, int &comparisons)

{

if (low < high)

{

int pi = partitionFirst(arr, low, high, comparisons);

quickSortFirst(arr, low, pi - 1, comparisons);

quickSortFirst(arr, pi + 1, high, comparisons);

}

}

void quickSortLast(int arr[], int low, int high, int &comparisons)

{

if (low < high)

{

int pi = partitionLast(arr, low, high, comparisons);

quickSortLast(arr, low, pi - 1, comparisons);

quickSortLast(arr, pi + 1, high, comparisons);

}

}

void quickSortMiddle(int arr[], int low, int high, int &comparisons)

{

if (low < high)

{

int pi = partitionMiddle(arr, low, high, comparisons);

quickSortMiddle(arr, low, pi - 1, comparisons);

quickSortMiddle(arr, pi + 1, high, comparisons);

}

}

int main()

{

const int n = 1000;

int arr[n];

for (int i = 0; i < n; i++)

{

arr[i] = rand() % 100;

}

int arr1[n];

copy(arr, arr + n, arr1);

int comparisonsFirst = 0;

quickSortFirst(arr1, 0, n - 1, comparisonsFirst);

cout << "Sorted with pivot as first index: ";

cout << "\nComparisons: " << comparisonsFirst << endl;

int arr2[n];

copy(arr, arr + n, arr2);

int comparisonsLast = 0;

quickSortLast(arr2, 0, n - 1, comparisonsLast);

cout << "Sorted with pivot as last index: ";

cout << "\nComparisons: " << comparisonsLast << endl;

int arr3[n];

copy(arr, arr + n, arr3);

int comparisonsMiddle = 0;

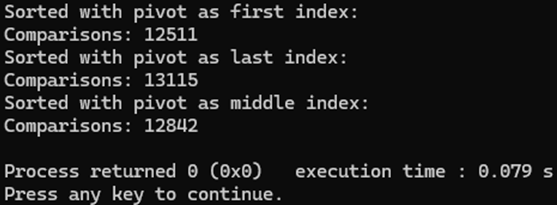
quickSortMiddle(arr3, 0, n - 1, comparisonsMiddle);

cout << "Sorted with pivot as middle index: ";

cout << "\nComparisons: " << comparisonsMiddle << endl;

}

**Output :**

****

1. **Analysecomplexityofquicksort andwrite downyourobservationofbestandworst case**

 **Best Case:** O(nlogn)

* Occurs when the pivot divides the array into two nearly equal parts at every step.
* Recursion happens approximately logn, and at each level, the partitioning requires O(n) operations.

 **Worst Case:** O(n2)

* Occurs when the pivot divides the array extremely unevenly, e.g., one part has n−1elements and the other has 0.
* This happens when the array is already sorted or reverse sorted with a poor pivot selection.

1. **Analysecomplexityofmergesortandwritedownyourobservationofbestandworst case**

**Best Case and Worst Case:**O(nlogn)

* Both cases have the same complexity because the algorithm always divides the array into two equal parts and merges them.
* The division takes logn levels of recursion, and merging at each level costs O(n)

1. **Analysecomplexityofbubblesortandwritedownyourobservationofbestandworst case**

 **Best Case:** O(n)

* Occurs when the array is already sorted. The algorithm only needs one pass to confirm no swaps are needed.

 **Worst Case:** O(n2)

* Occurs when the array is sorted in reverse order. The algorithm makes n−1passes, and each pass involves up to n−i−1 comparisons.

1. **Analysecomplexityofselectionsortandwritedown yourobservationofbestandworst case**

**Best Case and Worst Case:** O(n2)

Regardless of the initial arrangement, the algorithm always performs n−1passes, selecting the minimum

element and swapping it with the current position.

* The number of comparisons is fixed at n(n−1)/2, independent of input arrangement.

1. **Analysecomplexityofinsertionsortandwritedown yourobservationofbestandworst case**

 **Best Case:** O(n)

* Occurs when the array is already sorted. Only one comparison per element is needed to confirm its position.

 **Worst Case:** O(n2)

* Occurs when the array is sorted in reverse order. Each element must be compared and shifted up to its correct position in the sorted portion.

1. **Write afunctiontosortalleven-placednumbersinincreasingandodd-placenumbersin decreasing order. The modified array should contain all sorted even-placed numbers followed by reverse sorted odd-placed numbers. Analyse the complexity of your implemented approach**

**Notethatthefirstelementisconsideredasevenbecause ofitsindex0.**

#include <iostream>

#include <vector>

#include <algorithm>

#include <cstdlib>

#include <ctime>

using namespace std;

void sortEvenOdd(int arr[], int n)

{

vector<int> evenIndices, oddIndices;

for (int i = 0; i < n; i++)

{

if (i % 2 == 0)

evenIndices.push\_back(arr[i]);

else

oddIndices.push\_back(arr[i]);

}

sort(evenIndices.begin(), evenIndices.end());

sort(oddIndices.rbegin(), oddIndices.rend());

for (int i = 0; i < evenIndices.size(); i++)

arr[i] = evenIndices[i];

for (int i = 0; i < oddIndices.size(); i++)

arr[evenIndices.size() + i] = oddIndices[i];

}

int main()

{

int n=8;

int arr[8]={0, 1, 2, 3, 4, 5, 6, 7};

cout << "Original array: ";

for (int i = 0; i < n; i++)

cout << arr[i] << " ";

cout << endl;

sortEvenOdd(arr, n);

cout << "Modified array: ";

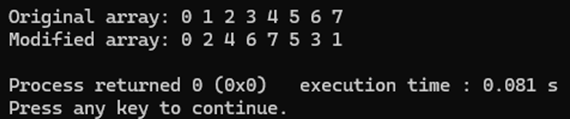
for (int i = 0; i < n; i++)

cout << arr[i] << " ";

cout << endl;

}

**Output :**

****

1. Given an integer array of which both first half and second half are sorted. Task is to mergetwosortedhalvesofarrayintosinglesortedarray. Analysethecomplexityof your implemented approach

# Example:

Input:A[]={2,3,8,-1,7,10}

Output: -1,2,3,7,8,10

Input:A[]={-4,6,9,-1,3}

Output:-4,-1,3,6,9

#include <iostream>

using namespace std;

void mergeSortedHalves(int A[], int n)

{

int mid = n / 2;

int i = 0, j = mid, k = 0;

int \*temp = new int[n];

while (i< mid && j < n)

{

if (A[i] <= A[j])

temp[k++] = A[i++];

else

temp[k++] = A[j++];

}

while (i< mid)

temp[k++] = A[i++];

while (j < n)

temp[k++] = A[j++];

for (int p = 0; p < n; p++)

A[p] = temp[p];

delete[] temp;

}

int main()

{

int n;

cout<<"Input the sie of the array : ";

cin>>n;

int arr[n];

cout<<"Input the elements : ";

for(int i=0;i<n;i++)

cin>>arr[i];

mergeSortedHalves(arr, n);

cout<<"Sorted array : ";

for (int i = 0; i< n; i++)

cout<<arr[i] << " ";

}

