KEGIATAN PRAKTIKUM 6
SORTING

Impelementasikan program pengurutan berikut:

1. Insertion Sort

```
// C program for insertion sort
// Source: geeksforgeeks.com
#include <math.h>
#include <stdio.h>

/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
  int i, key, j;
  for (i = 1; i < n; i++) {
     key = arr[i];
     j = i - 1;

     /* Move elements of arr[0..i-1], that are
     greater than key, to one position ahead
     of their current position */
     while (j >= 0 && arr[j] > key) {
          arr[j + 1] = arr[j];
          j = j - 1;
     }
     arr[j + 1] = key;
  }
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
  int i;
  for (i = 0; i < n; i++)
     printf("%d ", arr[i]);
  printf("\n");
}

/* Driver program to test insertion sort */
int main()
{
  int arr[] = { 12, 11, 13, 5, 6 };
  int n = sizeof(arr) / sizeof(arr[0]);

  insertionSort(arr, n);
  printArray(arr, n);

  return 0;
}
```

2. Merge Sort

```
/*
C program for Merge Sort
Source: geeksforgeeks.com
*/

#include<stdlib.h>
#include<stdio.h>

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
  int i, j, k;
  int n1 = m - l + 1;
  int n2 = r - m;

  /* create temp arrays */
  int L[n1], R[n2];

  /* Copy data to temp arrays L[] and R[] */
  for (i = 0; i < n1; i++)
     L[i] = arr[l + i];
  for (j = 0; j < n2; j++)
     R[j] = arr[m + 1+ j];

  /* Merge the temp arrays back into arr[l..r]*/
  i = 0; // Initial index of first subarray
  j = 0; // Initial index of second subarray
  k = l; // Initial index of merged subarray
  while (i < n1 && j < n2)
  {
     if (L[i] <= R[j])
     {
           arr[k] = L[i];
           i++;
     }
     else
     {
           arr[k] = R[j];
           j++;
     }
     k++;
  }

  /* Copy the remaining elements of L[], if there
  are any */
  while (i < n1)
  {
     arr[k] = L[i];
     i++;
     k++;
  }
```

```
/* Copy the remaining elements of R[], if there
   are any */
   while (j < n2)
   {
      arr[k] = R[j];
      j++;
      k++;
   }
}

/* l is for left index and r is right index of the
sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
   if (l < r)
   {
      // Same as (l+r)/2, but avoids overflow for
      // large l and h
      int m = l+(r-l)/2;

      // Sort first and second halves
      mergeSort(arr, l, m);
      mergeSort(arr, m+1, r);

      merge(arr, l, m, r);
   }
}

/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
   int i;
   for (i=0; i < size; i++)
      printf("%d ", A[i]);
   printf("\n");
}

/* Driver program to test above functions */
int main()
{
   int arr[] = {12, 11, 13, 5, 6, 7};
   int arr_size = sizeof(arr)/sizeof(arr[0]);

   printf("Given array is \n");
   printArray(arr, arr_size);

   mergeSort(arr, 0, arr_size - 1);

   printf("\nSorted array is \n");
   printArray(arr, arr_size);
   return 0;
}
```

3. Quick Sort

```
/* C implementation QuickSort */
#include<stdio.h>

// A utility function to swap two elements
void swap(int* a, int* b)
{
  int t = *a;
  *a = *b;
  *b = t;
}

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
  array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition (int arr[], int low, int high)
{
  int pivot = arr[high]; // pivot
  int i = (low - 1); // Index of smaller element

  for (int j = low; j <= high- 1; j++)
  {
     // If current element is smaller than the pivot
     if (arr[j] < pivot)
     {
           i++; // increment index of smaller element
           swap(&arr[i], &arr[j]);
     }
  }
  swap(&arr[i + 1], &arr[high]);
  return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
  if (low < high)
  {
     /* pi is partitioning index, arr[p] is now
     at right place */
     int pi = partition(arr, low, high);

     // Separately sort elements before
     // partition and after partition
     quickSort(arr, low, pi - 1);
     quickSort(arr, pi + 1, high);
  }
}
```

```
/* Function to print an array */
void printArray(int arr[], int size)
{
  int i;
  for (i=0; i < size; i++)
     printf("%d ", arr[i]);
  printf("\n");
}

// Driver program to test above functions
int main()
{
  int arr[] = {10, 7, 8, 9, 1, 5};
  int n = sizeof(arr)/sizeof(arr[0]);
  quickSort(arr, 0, n-1);
  printf("Sorted array: \n");
  printArray(arr, n);
  return 0;
}
```

Bentuklah kelompok yang beranggotakan maksimal 3 orang, lalu kerjakan tugas-tugas berikut!

1. Telusurilah bagaimana fungsi insertionSort, mergeSort, dan quickSort bekerja pada program yang Anda implementasikan! Tuliskan hasil penelusuran Anda!

2. Lakukan modifikasi terhadap program Insertion Sort, Merge Sort, dan Quick Sort yang telah Anda implementasikan agar dapat melakukan pengurutan secara menaik (ascending) dan menurun (descending)! Tambahkan pada program-program tersebut pertanyaan kepada pengguna untuk memilih jenis pengurutan ascending atau descending!

3. Lakukan modifikasi terhadap program Insertion Sort, Merge Sort, dan Quick Sort yang telah Anda implementasikan agar dapat membaca input dari fail dan menulis ke fail, sehingga data yang akan diurutkan diambil dari fail dan hasil pengurutan ditulis ke fail terpisah!

4. Tuliskan jawaban Anda pada fail dengan nama: **NamaKelompok_laporan_Praktikum_6.pdf**. Jangan lupa tuliskan daftar nama dan NIM anggota kelompok Anda dalam fail tersebut!

5. Jawaban dikumpulkan selambatnya hari Jumat, 10 Maret 2023, Pukul 23.59 WIB.

===SELAMAT BEKERJA===