Experiment 17
Date 22-12-2022

# SORTING ALGORITHMS

## Aim

Write a menu driven C program to implement the following sorting algorithms for sorting the numbers in ascending order. Implement each algorithm as a separate function.
a. Insertion Sort
b. Selection Sort
c. Heap Sort
d. Merge Sort
e. Quick Sort

## 1 Sorting Algorithms

### 1.1 Algorithm

```
Step 1: Start
Step 2: Print ''Menu''
        Print ''1. Insertion Sort''
        Print ''2. Selection Sort''
        Print ''3. Quick Sort''
        Print ''4. Heap Sort''
        Print ''5. Merge Sort''
        Print ''6. Exit''
Step 3: Input choice
Step 4: If choice = 1, then input len, input(arr,len), insertionsort(arr,len),
        display(arr,len), go to step 2
Step 5: If choice = 2, then input len, input(arr,len), selectionsort(arr,len),
         display(arr,len), go to step 2
Step 6: If choice = 3, then input len, input(arr,len), quicksort(arr,len),
         display(arr,len), go to step 2
Step 7: If choice = 4, then input len, input(arr,len), heapsort(arr,len),
         display(arr,len), go to step 2
Step 8: If choice = 5, then input len, input(arr,len), mergesort(arr,len),
        display(arr,len), go to step 2
Step 9: Stop
```

```
Start of function input(arr,len)
Step 1: Let i ← 0
Step 2: If i = len, go to step 4
Step 3: Input arr[i], let i ← i + 1, go to step 2
Step 4: Return

Start of function swap(a,b)
Step 1: Let temp ← a, a ← b, b ← temp, return

Start of function insertionsort(arr,len)
Step 1: Let i ← 1
Step 2: If i = len, return
Step 3: Let key ← arr[i], j ← i - 1
Step 4: If j < 0 or arr[j] ¡ key, go to step 6
Step 5: Let arr[j+1] ← arr[j], j ← j - 1, go to step 4
Step 6: Let arr[j+1] ← key, go to step 2

Start of function selectionsort(arr,len)
Step 1: Let i ← 0
Step 2: If i = len - 1, return
Step 3: Let min ← i
Step 4: Let j ← i + 1
Step 5: If j = len, go to step 7
Step 6: If arr[j] < arr[min], let min ← j, go to step 5
Step 7: swap(arr[min],arr[i]), go to step 2
Start of function partition(arr,low,high)
Step 1: Let pivot ← arr[high], i ← low - 1, j ← low
Step 2: If j = high, go to step 5
Step 3: If arr[j] <= pivot, let i ← i + 1, swap(arr[i],arr[j]
Step 4: Let j ← j + 1, go to step 2
Step 5: swap(arr[i],arr[j]), return i + 1

Start of function quicksort(arr,low,high)
Step 1: If low = high, return
Step 2; Let p ← partition(arr,low,high)
Step 3: quicksort(arr,low,p - 1), quicksort(arr,p + 1,high), return
Start of function quicksort(arr,n,i)
Step 1: Let max ← i, left ← 2 * i + 1, right ← 2 * i + 2
Step 2: If left < n and arr[left] > arr[max], let max ← left
Step 3: If right < n and arr[right] > arr[max], let max ← right
Step 4: If max != i, swap(arr[i],arr[max],heap(arr,n,max), return

Start of function heapsort(arr,len)
Step 1: Let i ← n / 2 - 1
Step 2: If i < 0, go to step 4
Step 3: heap(arr,n,i), let i ← i + 1, go to step 2
```

```
Step 4: 1. Let i ← n - 1
Step 5: If i < 0, go to step return
Step 6: swap(arr[0],arr[i], heap(arr,i,0), let i ← i + 1, go to step 5

Start of function merge(arr,l,m,r)
Step 1: Let n1 ← m - l + 1, n2 ← r - m, i ← 0
Step 2: If i = n1, go to step 4
Step 3: Let L[i] ← arr[l + i], i ← i + 1, go to step 2
Step 4: Let j ← 0
Step 5: If j = n2, go to step 7
Step 6: Let R[j] ← arr[m + j + 1], j ← j + 1, go to step 5
Step 7: Let i ← 0, j ← 0, k ← 1
Step 8: If i = n1 or j = n2, go to step 11
Step 9: If L[i] ¡= R[j], let arr[k] ← L[i], k ← k + 1, i ← i + 1, go to step 8
Step 10: Let arr[k] ← R[j], k ← k + 1, j ← j + 1, go to step 8
Step 11: If i = n1, go to step 13
Step 12: Let arr[k] ← L[i], k ← k + 1, i ← i + 1, go to step 11
Step 13: Let arr[k] ← R[j], k ← k + 1, j ← j + 1, go to step 8

Start of function mergesort(arr,l,r)
Step 1: If l < r, then let m ← l + (r - 1) / 2, mergesort(arr,l,m), merge-
sort(arr,m + 1,r), merge(arr,l,m,r), return

Start of function display(arr,len)
Step 1: Let i ← 0
Step 2: If i = len, return
Step 3: Print arr[i], let i ← i + 1, go to step 2\
```

## 1.2 Program

```
#include <stdlib.h>
#include <stdio.h>
void swap(int *a, int *b)
{
int t;
t = *a;
*a = *b;
*b = t;
}
void heapify(int arr[], int n, int i)
{
int largest = i;
int l = 2 * i + 1;
int r = 2 * i + 2;
if (l < n && arr[l] > arr[largest])
```

```
largest = l;
if (r < n && arr[r] > arr[largest])
largest = r;
if (largest != i)
{
swap(&arr[i], &arr[largest]);
heapify(arr, n, largest);
}
}
void heapSort(int arr[], int n)
{
for (int i = n / 2 - 1; i >= 0; i--)
heapify(arr, n, i);
for (int i = n - 1; i > 0; i--)
{
swap(&arr[0], &arr[i]);
heapify(arr, i, 0);
}
}
int partition(int arr[], int b, int e)
{
int pivot = arr[e];
int index = b - 1;
for (int i = b; i < e; i++)
{
if (pivot >= arr[i])
{
swap(&arr[index + 1], &arr[i]);
index++;
}
}
swap(&arr[index + 1], &arr[e]);
return (index + 1);
}
void quick_sort(int arr[], int b, int e)
{
if (b < e)
{
int p = partition(arr, b, e);
quick_sort(arr, b, p - 1);
quick_sort(arr, p + 1, e);
}
}
void merge(int arr[], int l, int m, int r)
{
int i, j, k;
```

```
int n1 = m - l + 1;
int n2 = r - m;
int L[n1], R[n2];
for (i = 0; i < n1; i++)
L[i] = arr[l + i];
for (j = 0; j < n2; j++)
R[j] = arr[m + 1 + j];
i = 0;
j = 0;
k = l;
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 merge_sort(int A[], int b, int e)
{
if (b >= e)
return;
int m = (b + e) / 2;
merge_sort(A, b, m);
merge_sort(A, m + 1, e);
merge(A, b, m, e);
}
```

```
void selection_sort(int A[], int n)
{
for (int i = 0; i < n; i++)
{
int min = A[i], pos = i;
for (int j = i + 1; j < n; j++)
{
if (A[j] < min)
{
min = A[j];
pos = j;
}
}
if (min != A[i])
{
int temp = A[i];
A[i] = A[pos];
A[pos] = temp;
}
}
}
void insertion_sort(int A[], int n)
{
int j, key;
for (int i = 1; i < n; i++)
{
int key = A[i];
int j = i - 1;
while (j >= 0 && A[j] > key)
{
A[j + 1] = A[j]; --j;
}
A[j + 1] = key;
}
}
int main()
{
FILE *fptr1, *fptr2;
char file1[100], c;
fptr1 = fopen("file.txt", "r");
if (fptr1 == NULL)
{
printf("Cannot open file %s \n", file1);
return (0);
}
int num[50], i = 0;
```

```
char x[5];
int k = 0;
while ((c = fgetc(fptr1)) != EOF)
{
if (c != '\n')
{
x[k] = c;
k++;
}
else
{
x[k] = '\0';
num[i] = atoi(x);
i++;
k = 0;
}
}
x[k] = '\0';
num[i] = atoi(x);
printf("\nContents:\n");
for (int j = 0; j <= i; j++)
{
printf("%d ", num[j]);
}
fclose(fptr1);
int choice;
printf("\n1.Insertion sort\n2.Selection sort\n3.Merge sort\n4.Quick
sort\n5.Heap sort\n6.Exit\n");
do{
    printf("\nEnter your choice: ");
    scanf("%d", &choice);
switch (choice)
{
case 1:
printf("After Insertion sort: ");
insertion_sort(num, i + 1);
break;
case 2:
printf("After Selection sort: ");
selection_sort(num, i + 1);
break;
case 3:
printf("After Merge sort: ");
merge_sort(num, 0, i);
break;
case 4:
```

```
printf("After Quick sort: ");
quick_sort(num, 0, i);
break;
case 5:
printf("After Heap sort: ");
heapSort(num, i + 1);
break;
case 6: return;
default:
printf("Invalid choice\n");
break;
}
for (int j = 0; j <= i; j++)
{
printf("%d ", num[j]);
}
}while(choice!=6);
}
```

## 1.3 Sample Output

```
Contents:
9 2 3 5 4 6 7 8 1
1.Insertion sort
2.Selection sort
3.Merge sort
4.Quick sort
5.Heap sort
6.Exit

Enter your choice: 1
After Insertion sort: 1 2 3 4 5 6 7 8 9
Enter your choice: 2
After Selection sort: 1 2 3 4 5 6 7 8 9
Enter your choice: 3
After Merge sort: 1 2 3 4 5 6 7 8 9
Enter your choice: 4
After Quick sort: 1 2 3 4 5 6 7 8 9
Enter your choice: 5
After Heap sort: 1 2 3 4 5 6 7 8 9
Enter your choice: 6
```

Figure 1: Input and Output

## 1.4 Result

A menu driven C program was made to sort integers in ascending order in a given file using various sorting techniques like Insertion Sort, Selection Sort, Heap Sort, Merge Sort and Quick Sort by using separate functions for each.