AMRITA VISHWA VIDYAPEETHAM

CHENNAI CAMPUS

Second Year. B.TECH

(Computer Science and Engineering)

(DAA Lab Work)

Name: M CYNTHIA SHREE

RollNo.: CH.SC.U4CSE24110

Department: CSE-B

Academic Year: 2025-26

1. Merge Sort

CODE:

```
//ch.sc.u4cse24110
//merge sort
#include <stdio.h>
void merge(int arr[],int left,int mid,int right) {
int n1= mid- left+1;
int n2= right-mid;

int L[n1], R[n2];

for(int i=0;i<n1;i++) {
L[i]=arr[left+i];
}
for(int j=0;j<n2;j++) {
R[j]=arr[mid+1+j];
}

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

while (i<n1 && j<n2) {
if(L[i]<=R[j])
arr[k++]=L[i++];
else
arr[k++]=R[j++];
}

while (i<n1)
arr[k++]=L[i++];
while (j<n2)
arr[k++]=R[j++];
}
```

```
void mergeSort(int arr[], int left, int right) {
if(left<right) {
int mid= (left+right)/2;
mergeSort(arr,left,mid);
mergeSort(arr, mid+1,right);
merge(arr,left,mid,right);
}
}

int main() {
int arr[]={157,110,147,122,111,149,151,141,123,112,117,133};
int n= sizeof(arr)/sizeof(arr[0]);
mergeSort(arr,0,n-1);
printf("sorted array:");
for(int i=0;i<n;i++) {
printf("%d ",arr[i]); }
printf("\n");
return 0;
}
```

OUTPUT:

```
C:\Users\mrgns\Desktop\daa\task3>gcc mergesort.c -o mergesort

C:\Users\mrgns\Desktop\daa\task3>.\mergesort
sorted array:110 111 112 117 122 123 133 141 147 149 151 157
```

TIME COMPLEXITY: O(n log n) because array is split into 2 halves repeatedly log n times and at each level all n elements are being compared.

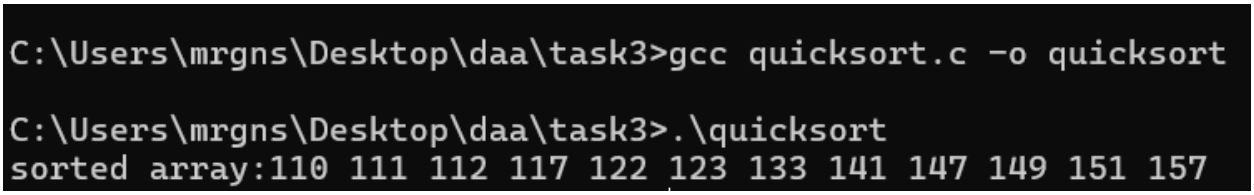
SPACE COMPLEXITY: O(n) because only left and right arrays are created.

2. Quick Sort

CODE:

```
//ch.sc.u4cse24110
//quick sort
#include <stdio.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; 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 arr[]={157,110,147,122,111,149,151,141,123,112,117,133};
int n= sizeof(arr)/sizeof(arr[0]);
quickSort(arr,0,n-1);
printf("sorted array:");
for(int i=0;i<n;i++) {
printf("%d ",arr[i]); }
return 0;
}
```

OUTPUT:



TIME COMPLEXITY: Best case is O(n log n) because array is split into 2 halves repeatedly log n times and at each level all n elements are being compared.

Worst case is O(n^2) when pivot element chosen is the smallest or the largest.

SPACE COMPLEXITY: Worst case is O(n) because of recursion stack.